Jenzabar CX

CX Implementation and Maintenance

Technical Manual

Filename: tmcximmt
Distribution Date: 02/01/2002

Contact us at www.jenzabar.com

JENZABAR, INC.
CX IMPLEMENTATION AND MAINTENANCE TECHNICAL MANUAL

TABLE OF CONTENTS

USING THIS MANUAL

Overview

Purpose of This Manual

This manual provides guidance and information on these processes:

- The general processes of implementing the CX system
- Information and procedures to guide you in the maintenance of the CX system

Intended Audience

This guide is for use by those individuals responsible for the implementation, customization, and maintenance of the product.

Product Differences

This manual contains information for using all features developed for the Jenzabar CX product. Your institution may or may not have all the features documented in this manual.

Structure of This Manual

This manual contains information for implementing and maintaining the common features of Jenzabar CX. The manual's organization follows:

Part I - Implementation

Section 1 - Preparation steps to implementation
Section 2 - Setup steps to implementation
Section 3 - Training steps to implementation
Section 4 - Go Live steps to implementation

Part II - Maintenance

Section 5 - Product Releases and Revision Control
Section 6 - Database management information
Section 7 - System administration information
Section 8 - System maintenance information
Section 9 - Customer assistance information
Section 10 - Troubleshooting

Reference information

Appendix - Jenzabar CX UNIX Commands
Index

Conventions Used in This Manual

Introduction

Jenzabar has established a set of conventions to help you use this manual. The list of conventions presented below is not exhaustive, but it includes the more frequently-used styles and terms.

Style Conventions

Technical manuals observe the following style conventions.

Boldface type
Represents text that you type into the system (e.g., Type **UNDG**), command names (e.g., **Finish**), or keys you use to execute a command or function (e.g., **<Enter>**).

Bulleted lists
Show items not ranked or without a sequential performance.

CAUTION:
Indicates a caution or warning of a potential risk or condition.

<Enter>
Represents the Enter, Return, Line Feed, or ↵ key on your keyboard.

Italic type
Is used in any of these ways:

- To represent a new or key term
- To add emphasis to a word
- To cross-reference a section of text
- To represent a variable for which you substitute another variable (e.g., substitute *filename* with an appropriate filename)

<Key name>
Represents a key that you must press.

Note:
Indicates a note, tip, hint, or additional information.

Numbered lists
Show ranking of items or sequence of performance.

Parentheses
When used around a field name, indicate the field is unlabeled. The field description includes the location of the field.

Quotation marks
Represent information written in this guide exactly as it appears on the screen.

Example: The message, "Now Running..." appears.

Jenzabar-Specific Terms

The following list identifies term conventions used in this guide.

Application
A group of one or more software programs that enables you to perform a particular procedure, such as entering student information.

Data
Specific information you enter into fields on a particular data entry screen.

Enter
To type information on a keyboard and execute by any of the following actions:

- Pressing the <Enter> key
- Clicking on the OK button
- Selecting Finish.

F key
Any of the function keys located on your keyboard (e.g., <F1>).

Hot key
The capitalized and underlined (or highlighted) letter of a command on a menu.

ID
The number assigned to each student or organization associated with your institution (e.g., 12345).

Institution
An established organization of postsecondary education that supports all operating functions (e.g., a college or university).

Parameter
A variable in the system that is given a constant value for a specific application (e.g., a date can be a parameter for producing a report).

Select
To execute a command by any of the following actions:

- Performing the keystrokes
- Pressing the hot key
- Highlighting the command or option and pressing the <Enter> key
- Clicking with the mouse

System
The Jenzabar product, CX.

Type
To press keys on a keyboard so that text or characters to appear in a specific position on the screen. To execute a command or function, you must also perform either of the following actions:

- Press the <Enter> key
- Click on the OK button
- Select Finish

Keystrokes

When you see two keys separated by a dash (e.g., <Ctrl-c>), hold down the first key (Ctrl) while pressing the second (c).

PART I - IMPLEMENTING JENZABAR CX

Overview

Introduction

Part I of this manual provides information about the phases of the implementation process. These general phases, which occur for the implementation of each CX product, are as follows:

- Preparation
- Setup
- Training
- Go Live

Each phase contains defined tasks for the institution and Jenzabar staff.

Categories in Jenzabar CX Implementation

The following lists the major categories into which CX implementation is divided.

- Letter of Intent
- Implementation Schedule
- Signed Contract
- Client Preparation
- System Installation
- Setup
- Client Training
- Consultation
- Go Live
- SMOs
- Support

Purpose of Implementation

The purpose of performing the system implementation is to ensure that the system functions as follows:

- At the performance level, standardized in the design of the CX base product.
- At the performance level, required as a result of modifying the CX base product to meet an institution's unique needs.
- A fully-integrated system.

Program Manager

Upon the contract being signed, Jenzabar assigns the institution a program manager from Implementation Services. The program manager is the daily point-of-contact between Jenzabar and the institution for the implementation of all contracted modules, and has expertise in one or more of the functional areas being implemented. The program manager does the following:

- Calls upon Jenzabar coworkers with expertise in other areas
- Contacts Jenzabar coordinator to help begin the project
- Focuses on setting up institutional values in CX, and on converting appropriate information from existing systems into CX

SECTION 1 - PREPARATION PHASE

Overview

Introduction

This section describes the general tasks in the Preparation phase of implementing CX. The Preparation phase occurs for each area of the institution that is implementing CX product(s).

Goals of the Preparation Phase

The goals of the Preparation phase of implementation are:

- To gather the specific information needed to set up CX product(s) according to the institution's needs and desires
- To determine the effects of switching from the old system to CX to the institution's offices, staffing, processes, and policies

Duration of the Preparation Phase

This phase in the implementation process begins with the sending of the Letter of Intent. The phase ends when the institution delivers a Data Conversion plan to Jenzabar. A typical duration for this phase is six to eight weeks.

General Tasks

The following lists the general tasks that the institution must complete in the Preparation phase.

- Develop and complete a data conversion plan
- Determine changes to processes
- Determine office/staffing structure
- Determine inter-office processing issues
- Determine reporting/letter writing requirements
- Determine changes to screens
- Determine the institution's readiness to continue to the Setup phase
- Provide Preparation phase information to Jenzabar

Jenzabar staff uses the information gathered from these tasks in the Setup phase of implementation.

Jenzabar Involvement

This phase involves one or more on-site visits by Jenzabar personnel, including the program manager and other Jenzabar staff, depending on the number of modules being implemented.

Training comprises the major component of the Jenzabar implementation on-site visits. Also, the account manager works closely with institutional staff to monitor the progress of the setup and conversion. To achieve this, the account manager:

- Reviews institutional needs
- Advises institutional staff on how to ensure that screen and report designs meet institutional needs
- Conducts a demonstration, providing an overview of CX to members of the customer administration and staff
- Determines the setup for the institution

Jenzabar Implementation Policy and Pre-Implementation Requirements

Policy

The institution must complete a list of requirements to prepare for system implementation. This list of requirements includes all tasks necessary to ensure the following:

- The institution provides required training to appropriate staff
- The institution identifies and lists required system modifications
- The institution tests system hardware to ensure correct functioning
- The institution reports hardware malfunctions to the Jenzabar Support Services prior to the on-site implementation visit by Jenzabar

These requirements must be completed *before* the Jenzabar on-site visit; any task not completed will critically affect the success of the implementation. If Jenzabar must spend additional hours providing training, *that was not completed before the on-site visit*, less time will be available for implementation. If such a situation occurs, the institution will be billed for any additional implementation hours.

Pre-Implementation System Modification Requirements

The implementation process requires your institution to complete procedures for customizing standard system features. These include macros, includes, parameters, menu options, screens, and reports, that are contained in the CX standard product.

During the implementation process, the Jenzabar system coordinator must identify any required changes to customize the CX standard product. If an institution requests that Jenzabar make these modifications during implementation, the institution may incur charges for the extra hours spent on customizations.

If the Jenzabar system coordinator identifies extensive system modifications (e.g., modifications to source code libraries), then Jenzabar must be notified.

Pre-Implementation System Hardware Requirements

The institution must prepare for the on-site implementation visit by ensuring that all hardware required for implementation, including personal computers, printers, and peripherals, is tested and working properly.

Customer Implementation: Suggestions for Organizing the Project

Introduction

When the customer signs a contract they make a major commitment to enhancing the way business is conducted. This commitment may be motivated by a desire to provide better service to students, to provide better reporting to internal and external constituencies, or to improve a competitive position. In a well-organized implementation project, these desires are expressed as objectives, the completion of which will result in achieving the major goals.

You must do the following to ensure the implementation's success:

- Keep senior administration's commitment throughout the duration of the project
- Maintain constant sight of project goals
- Keep all project staff apprised of the primary goals
- Keep all project staff aware of specific responsibilities

Making Project Assignments

Success is achieved when senior administrators are involved throughout the project, at each appropriate level. Jenzabar recommends that an existing executive council or cabinet establish a steering committee for the project. The committee should have the following responsibilities:

- Setting the project structure and organization
- Assigning key roles, and identifying project leaders
- Setting the overall implementation timeline
- Making major decisions about policy issues
- Ensuring regular substantive progress reports
- Taking action on progress reports
- Reviewing Jenzabar trip reports after each visit
- Acting on Jenzabar trip reports, as required

Key Positions and Roles

Jenzabar recommends that the institution identify the following implementation staff:

- Jenzabar system administrator
- Jenzabar system coordinator
- Person responsible for an institutional function
- User group
- Project team

Jenzabar System Administrator

This position requires that the individual:

- Be at the institution's executive level
- Be responsible for monitoring the contract with Jenzabar
- Be in a position to ensure that the contract terms are being met
- Will have been involved in the negotiation of the contract

Note: This placement of the role ensures that any changes requested by lower-level staff receive appropriate scrutiny.

The Jenzabar points of contact for the Jenzabar administrator are normally at the management or executive level.

Jenzabar System Coordinator

This individual is the primary point of contact between the institution and Jenzabar. This individual must:

- Be responsible for the operation of an CX product as it is configured at the institution
- Be the chair of a CX users group
- Have a wide view and understanding of the institution's operations
- Have seniority sufficient to enable effective communication and authority at all levels of staffing

Note: The Jenzabar system coordinator must have a strong technical background, must know the structure of CX in its operating environment, and must champion the use of the system by end users.

Application Coordinator

This individual:

- Is responsible for implementing a CX module or application
- Holds a technical or functional position at the institution and is able to perform implementation tasks
- Chairs the project team (key users from offices directly affected by the module or application)
- Coordinates needs assessment for a module to identify required changes in functionality
- Identifies tasks that correspond to institutional policies and operation of CX
- Assists the Jenzabar system coordinator in setting up and configuring CX
- Develops an ongoing training program for end users, based on Jenzabar training
- Represents the office of the Jenzabar users group

Jenzabar CX Users Group Representatives

Implementing CX often results in changes to customer policies and procedures. These changes, which affect several areas of operations, result from opportunities to improve policies and procedures developed for manual, paper-driven environments.

The purpose of the Jenzabar CX users group is to ensure that the causes and benefits of these changes (often referred to as cross-functional issues), and the resulting new policies and procedures, are adequately communicated to everyone who is affected.

Jenzabar strongly recommends that a user group be formed, with the following membership:

- Chair: this is the Jenzabar system coordinator
- Anyone responsible for the institutional function
- Anyone affected by a CX module or application

Representation could, for example, include the persons responsible for the institutional function for each of the following modules: Admission, Registrar, Program and Degree Audit, Financial Aid, Student Services, Student Accounting, Financial, Payroll, Alumni/Development, Academic Affairs, Academic Advising, Housing, Library, and Bookstore.

Establishing the Jenzabar CX Users Group Structure

Jenzabar System Users Group Structure

Cross-functional issues should be reviewed by the Jenzabar CX users group, which identifies policy issues, informs the steering committee of them, and establishes new procedures that expedite operation of CX.

For each module being implemented, a project team needs to be established. This team is led by the persons responsible for the institutional functions directly affected by the module, as well as other designated institutional staff.

The project team for the Registrar module might include the following:

- Person representing the Registrar's office as the chairperson
- Person representing Admissions
- Person representing Financial Aid and Student Accounts
- Person representing Academic Affairs and other academic departments

In addition, representatives from Housing, the Bookstore, and the Library might participate when the module affects their operations.

Note: When policies and procedures, which were traditionally the responsibility of the Registrar's office, have been decentralized and moved to other offices, it is important to include representatives from those operations.

Reviewing Data in Tables and Records

Introduction

After assessing features of the CX product and setting the appropriate enable macros, you must review the setup of CX tables and records.

Procedure

The following list provides the steps to review the values of the CX tables and records.

1. For each table, review the codes supplied with CX. Determine whether or not the codes meet the needs of your institution. Make updates as appropriate.
2. Review the institution's records converted from the previous system. Determine whether or not the records need to be updated to meet the needs of CX reports. Make updates as appropriate.

SECTION 2 - SETUP PHASE

Overview

Introduction

This section describes the general tasks in the setup phase of implementing CX. The Setup phase occurs for each area of the institution that is implementing a CX product(s).

CX contains database tables, macros, includes, and parameters that enable you to change the following:

- What text appears on the screens
- How text appears on the screen
- Which system options are available to you
- How the system options function

Thorough knowledge of an institution's processing policies and procedures, coupled with skill in using UNIX, ensures that you can implement and modify the tables, macros, includes, and parameters in CX.

Goal of the Setup Phase

The goal of the Setup phase of implementation is to set up the features of CX product(s) according to the institution's needs and desires.

Duration of the Setup Phase

This phase in the implementation process begins with the Project Kick-off, the first on-site implementation visit by Jenzabar staff. The phase ends when the institution agrees that CX product(s) has been set according to their needs and desires. A typical duration for this phase is four to six weeks.

General Tasks

The following lists the general tasks that Jenzabar must complete in the Setup phase.

- Installing CX and INFORMIX software on the institution's UNIX system
- Set macros and includes for CX product in accordance with the institution's desired setup
- Set the institution's desired values in CX Common tables
- Set the institution's desired values in CX product-specific tables
- Customize screens, menus, and reports in accordance with the institution's desired setup
- Set up the Communications Management product in accordance with the institution's desired setup

Installing the System

Introduction

When the hardware and operating system are in place at the institution, a Jenzabar Technology Consultant does the following:

- Configures the hardware and UNIX operating system for CX software
- Installs CX software
- Installs the INFORMIX software
- If purchased by the institution, installs the associated third party software (e.g., PowerPlay)
- Tests the system processes to ensure the system is functioning as negotiated

Jenzabar Technology Consulting also provides appropriate follow-up support for the installation.

Standard Setup

The Jenzabar Technology Consulting group installs CX with each product that the institution has purchased. Since you can customize CX, many features of the product can be enabled and disabled according to your institution's needs. Jenzabar delivers CX products in a *standard* setup where:

- Macros and includes are set with a commonly used setting
- Tables contain standard values (e.g., State table contains State codes)

During the Setup phase of implementation, Jenzabar staff may customize the standard settings for the CX product using the information gathered during the Preparation phase of the implementation.

Setting Tables and Records

Introduction

You must set up database tables in order to successfully implement and use CX. You update these tables from the CX menu system, rather than from the UNIX prompt.

This section provides the following important information for setting up tables:

- An overview of setting up tables, which includes information on selecting the table(s) to add or update, and further reference materials you can use while setting up tables
- The procedure for adding or updating tables

The data that you enter in the database tables establishes the values that will appear on the PERFORM screens and on reports.

Selecting the Tables

Select the tables to set up from the System Management: Table Maintenance Menu. While you use the table maintenance menus, be aware of the following information:

- The modules and tables listed on the table maintenance menus are not necessarily presented in the order in which you will complete them.
- You can modify or delete table entries at a later time.
- You should run the reports for the tables you set up. These reports not only provide a paper copy of each table's contents, but also allow you to review the table data for accuracy.

Reference Guides

Use the Technical Reference manual corresponding to the application to assist you in setting up the tables. The reference material contained in the following guides includes tables for a particular module or application, as well as supporting information and the table reports to assist you in setting up tables.

The Process for Setting Up Tables

The following list describes the process involved in setting up tables.

1. Review the data that currently resides in the existing tables and compare them with the institution's requirements.
2. Ensure that table macros and their table values coincide.
3. Review all tables that were added or updated by other areas of an institution during implementation, and add values to the tables as necessary.

Table Macro and Values

Step 2 of the process for setting up tables (from above) requires that you ensure that table macros and their table values coincide. This phase is very important in the process for setting up tables because you must not only add or update the tables to ensure that the required values exist, but you must ensure that there is a link between each of the table values and the table macros that exist.

Common Tables

Introduction

Common tables are tables that can be accessed from any of the applications of CX. For example, several applications use the Contact table (ctc_table).

Note: For more information about the Common tables, see *Common Tables and Records* in the *CX System Reference Technical Manual*.

Prerequisite tasks

Before you begin to modify any common tables, you must check with other departments at an institution to find out if any other Jenzabar implementation(s) has occurred, and which common tables were affected by the prior implementation(s). It is very important that you perform this checking so that you do not inadvertently negate the work performed by individuals in other areas of an institution.

Common Tables

You can find a list of the particular common tables to review or update while implementing a certain application by referring to the implementation process checksheet for the application you are implementing. It is important to update the common tables in the order in which they are listed.

Application-Specific Tables

Introduction

Application-specific tables are tables that only a particular application accesses. Any modifications you make to application-specific tables should not have an effect on how other Jenzabar applications run.

Prerequisite Tasks

Before you begin to modify any tables, you must first gather the data that will be used to populate the tables (e.g., the grades issued by the college).

When you are ready to begin modifying the tables, refer to the implementation process checksheet for the application you are implementing. The checksheet will give you the order in which you should review or update the application-specific tables.

Application-Specific Tables

You can find a list of the application-specific tables to review or update while implementing a particular application by referring to the appendix of the application you are implementing. It is important to update the application-specific tables in the order in which they are listed in the process checksheets.

Communications Management Tables

You must set the tables required for the Communications Management product. Various CX products use Communications Management to automatically create Contact records and schedule the creation of letters. For more information, see the *Communications Management User Guide*.

Adding or Updating Entries in Tables

How to Add or Update Entries in a Table

Use the following procedure to add or update entries in a table.

1. Select System Management from the Jenzabar CX College: Master Menu. The system prompts you for your password.
2. Enter your password. The System Management: Main Menu appears.
3. Select Table Maintenance. The System Management: Table Maintenance Menu appears.
4. Select the module group that contains the table you want to add or update. The System Management: Table Maintenance Menu - Modules (A-L) or Modules (M-Z) appears.
5. Select the module that contains the table in which you want to add or update entries. The table maintenance menu for the module you selected appears.
6. Select the table in which you want to add or update entries. A window that contains instructions for producing the table you selected appears.
7. Select **Finish**. The PERFORM screen for the table you selected appears.
8. Select **Query**. The cursor moves to the first field on the screen.
9. Press **<Tab>** to move to a field on which you want to query, then enter the value in that field. The cursor moves to the next consecutive field on the screen.
10. Repeat step 9 for all the fields on which you want to query. Select **Finish**.
    - If a record meets your criteria, then the data from that record appears in the fields on the screen.
    - If more than one record meets your criteria, the data from the first record appears, and a message appears on the comment line to tell you how many records were found for this query.
    - If no record meets your criteria, a message appears on the error line indicating this.
11. Do you want to add or update a table?
    - If you want to add a table to the database, select **Add** and go to step 13.
    - If you want to update the table on which you queried, select **Update** and go to step 13.

    The cursor moves to the first entry field on the screen.
12. Enter data in as many of the fields as necessary, according to the field descriptions in the corresponding Tables and Table Reports Reference guide, then press **<Esc>.** The data is added to the database.
13. Do you want to add or update another database table?
    - If yes, go to step 12.
    - If no, go to step 15.
14. Select Exit, then press **<Enter>.** The System Management: Table Maintenance Menu appears.

Setting Macros and Includes

Introduction

You must set macros and includes for a CX product to define default values in screens and to enable and disable features of the product.

Note: For more information about macros and includes, see the following in the *CX System Reference Technical Manual*:

- *CX Macros*
- *CX Includes*

Macros and Includes

Before you begin to modify any macros and includes, you must first gather the data that will be default values used in fields and know which features of the system that you want to enable or disable.

When you are ready to begin modifying the macros and includes, refer to the implementation process checksheet corresponding to the application you are implementing to find out the macros and includes that you must set.

Customizing Screens, Menus, and Reports

Introduction

This task in the Setup phase is based on the institutions desired changes to the standard *CX* screens and menus.

Screen Definition Files

To modify a standard *CX* screen, you must access and modify the screen's definition file. See *Screens and Forms* in the *CX System Reference Technical Manual* for more information on modifying screen definition files.

Menu Screens

To modify *CX* menus and menu options, you must do the following:

- Modify a menusrc file to add or delete menu options in a menu screen
- Modify a menuopt file to change a menu option screen
- Modify a menuparam file to change menu processing parameters

See *The Menu System* in the *CX System Reference Technical Manual* for more information on modifying menu option and parameter files.

Reports

To modify *CX* reports, you must use the ACE Report Writer to modify the report.

See *Reports and Output Control* in the *CX System Reference Technical Manual* for more information on modifying ACE reports.

SECTION 3 - TRAINING PHASE

Overview

Introduction

This section describes the general tasks in the Training phase of implementing *CX*. The Training phase occurs for each area of the institution that is implementing a *CX* product(s).

Goals of the Training Phase

The goals of the Training phase of implementation are:

- To create a training database
- To provide user training on the basics of *CX*
- To provide user training on locating and entering data using *CX*
- To provide training on the features and processes of the specific *CX* product

Duration of the Training Phase

This phase in the implementation process begins with the first End User Training visit by Jenzabar staff. The phase ends when the institution agrees that the training has been complete and effective. A typical duration for this phase is ten to twelve weeks.

Technical Training

Jenzabar delivers technical training after the contract is signed. The person responsible for the daily operation of *CX*, the Jenzabar system coordinator, attends Jenzabar technical training. The technical training is delivered with the following courses:

- *Fundamentals of UNIX, INFORMIX SQL and INFORMIX Online*, which teaches the client the *CX* technical operating environment
- *Database Tools and General Utilities*, which teaches the client the *CX* structure, and how to manage this structure
- *Conversion Workshop*, which teaches the client the method to converting the institution's data to *CX* format

Creating a Training Database

Introduction

To allow users the ability to practice and become skilled in using *CX* product(s), the institution must create a training database. To create the database containing test data, which users can use in training, you must perform the data conversion process using a Jenzabar-provided utility, *tpconvert*.

The Jenzabar system coordinator attends the Data Conversion course offered by Jenzabar before performing the data conversion process.

Note: A benefit to creating the training database is that you perform a dry run of the conversion process before performing the final data conversion, the last task in the Go Live phase.

Tasks for Creating the Database

The Jenzabar system coordinator does the following to create the training database:

- Creates a configuration table that defines:
  - The positions within an input record that contains pertinent data
  - What to do with that data
- Executes the tpconvert tool, specifying the configuration table
- Tests that the converted files have correct fields and information
- Tests a sampling of reports for correct functioning using the converted data

Note: For more information, see Using Tape Conversion in this manual.

Jenzabar CX Training

Introduction

The Jenzabar system coordinator provides the basic CX training the end users of the institution. This basic training occurs prior to the product-specific training visits by Jenzabar staff. The product-specific training provides instruction on using the features of the particular product area that users will use.

Creating Logins

The Jenzabar system coordinator must create user logins and establish appropriate permissions before the users can log in and use the training database. For more information, see the following in this manual:

- *Creating and Deleting User Accounts*
- *CX Groups and Permissions*

System Basics

The Jenzabar system coordinator provides the following basic training to users:

- Using terminals and/or PCs, and printers
- Logging in and out of the system
- Using CX menus
- Using electronic mail
- Using CX Query, Table Lookup, and ID-type command procedures to locate data on the system
- Using CX data entry screens and detail windows to add and update records

Note: Jenzabar recommends that the Jenzabar system coordinator provides this basic training to end users just prior to product-specific training provided by Jenzabar.

Product-Specific Training

The Jenzabar staff provides product-specific training occurs in the following ways:

- Overview training on the product's features
- Training on the processes for using the product
- Training on using the product's reports, forms, and letter producing features

Training Facility and Equipment Requirements List

Introduction

To successfully implement the CX product at an institution, the institution must meet specific facility and equipment requirements. These pages list the facility and equipment requirements necessary for Jenzabar to assist the institution in implementing CX products.

Training Facility and Equipment Requirements

The following list contains the requirements that an institution must meet *before* implementing CX:

- Cleared schedules of key implementation individuals
- Freedom from telephone calls
- A training room that is quiet and distanced from other activities
- A blackboard or flip chart (optional, but recommended)
- An overhead projector (optional)
- Terminals (preferably one terminal per trainee)

Note: Test all terminals, PCs, and printing devices for proper functionality before Jenzabar representatives arrive at the institution.

SECTION 4 - GO LIVE PHASE

Overview

Introduction

This section describes the general tasks in the Go Live phase of implementing CX. The Go Live phase occurs for each area of the institution that is implementing a CX product(s).

Goals of the Go Live Phase

The goals of the Go Live phase of implementation are:

- To determine if the institution requires additional Jenzabar assistance in:
  - Fixing problems with processing, data conversion, or end user procedures
  - Making further modifications or enhancements to CX
- To confirm that the institution is satisfied with:
  - The functioning of CX
  - The implementation project as a whole
- To perform final data conversion and begin processing *live* data using CX

Duration of the Go Live Phase

This phase in the implementation process begins with the determined Go Live date. The phase ends when the institution agrees that the institution is processing correctly using live data. A typical duration for this phase is eight to ten weeks.

General Tasks

The following lists the general tasks that the institution must complete in the Go Live phase:

- Determine if the institution needs Jenzabar assistance
- Determine if the institution needs modifications and/or enhancements made to CX
- Revise the computer system policies and procedures towards operating CX
- Confirm that CX is functioning correctly
- Review the implementation project
- Determine if the institution is prepared for final data conversion
- Perform the final data conversion

Additional Jenzabar Assistance

Introduction

Some issues, which arise on every project, are unique to the institution and cannot always be resolved according to standard implementation strategy. Jenzabar contracts contain provision for consulting with Jenzabar staff, who have expertise in areas where choices and alternatives exist.

Note: The significant background of Jenzabar staff in higher education proves highly beneficial in this area.

When an institution wants to modify the CX product, there are three options:

- Perform a local modification, with or without the assistance of Jenzabar
- Submit a Product Modification Request (PMR), requiring Jenzabar to perform the modification based on specifications, which the customer prepares.
- Submit a Product Enhancement form, requiring Jenzabar to make a specified enhancement to a CX product

Note: See your Program Manager for copies of the Product Modification Request or Product Enhancement form.

Product Modification Request Approval and Submission

A designated representative of the institution submits the PMR to Jenzabar after personnel at the institution complete, review, and approve the PMR. The personnel are the following:

- The designated representative of the institution
- Computer center staff
- Appropriate administrators
- End users of the CX product

Product Modification Request Process

The following lists the steps in the product modification request process. The client must submit all documents related to the product modification process to the Jenzabar vice president of Product Services.

1. Client obtains the PMR form from the Jenzabar account manager, completes it, and submits it to Jenzabar .
2. If Jenzabar is to develop or assist in developing the PMR, or develops a requirements document on the client's behalf, the client submits a purchase order for this cost. This is done before Jenzabar begins developing these documents.
3. Jenzabar performs the following activities within 30 working days of receiving the PMR:
   - Reviews the PMR and/or requirements document, and advises the client in writing of a decision of whether or not to proceed
   - Sends the client a written quote for the projected total cost of the project, in addition to a Requirements Acceptance Agreement
4. Client signs and submits to Jenzabar the Requirements Acceptance Agreement and a purchase order for the cost of the top-level design.
5. Jenzabar does the following:
   - Advises the client in writing of the decision of whether or not to proceed, and sends the client a top-level design schedule within 10 working days of receiving the Requirements Acceptance Agreement and the purchase order
   - Completes the top-level design

- Sends the client a design document, a revised cost quote (if one is required), and a Design Acceptance Agreement

6. Customer signs and gives Jenzabar the Design Acceptance Agreement, and the purchase order for the cost of the remainder of the project.
7. Jenzabar provides the client in writing a design schedule and product modification delivery date. This is done within 10 working days of receiving the Design Acceptance Agreement and a purchase order.

The Product Enhancement Form

An institution requesting Jenzabar to modify the CX standard product must complete a Product Enhancement form. Institutions use the Product Enhancement form to define only the general product modification requirements in the CX standard product.

After an institution completes the Product Enhancement form, the institution submits the form to the Jenzabar senior product manager. The senior product manager evaluates the form and directs the Product Enhancement form to the appropriate product manager.

Note: An institution requesting Jenzabar assistance to implement a local modification unique to the institution must complete a Product Modification Request (PMR) form.

Ensuring Customer Satisfaction

Introduction

Before Jenzabar completes the implementation process at an institution, they perform various tasks to ensure that the institution is satisfied with the functioning of CX and the implementation project.

Note: For more information on Jenzabar customer assistance, see *Customer Satisfaction* in this manual.

Confirming Correct Functioning

Jenzabar requires that the institution confirms that CX is functioning correctly. The objectives of this task are as follows:

- To ensure that CX is functioning and being used in accord with institutional policy (review the integration of the system and end user processes)
- To ensure that the user interface is correct, efficient, and effective
- To ensure that the appropriate staff are assigned to tasks
- To ensure that procedures are documented and are usable and current

If the institution determines that the system is not functioning correctly in any of the above aspects, Jenzabar will take steps to correct the situation.

Go Live-Implementation Review

Jenzabar conducts a Go Live-implementation review, based on the strong belief that the end of the implementation project is as important as all other phases.

Jenzabar conducts this review, which involves members of the Jenzabar executive management, meeting on campus with members of the customer's executive administration and project team.

In this joint review, both parties evaluate whether or not those responsible met the objectives identified and agreed upon at the beginning of the project. Also, both parties look for the best strategy to maintain the customer/Jenzabar relationship.

Final Data Conversion

Introduction

The Jenzabar system coordinator must perform the final data conversion process before the institution can use *live* data using CX. The Jenzabar system coordinator must perform the final data conversion process using a Jenzabar-provided utility, *tpconvert*.

The Jenzabar system coordinator attends the Data Conversion course offered by Jenzabar before performing the data conversion process.

Note: The Jenzabar system coordinator performs a dry run of the conversion process by creating the training database; however, Jenzabar recommends that the process be performed multiple times before the final *go live* date..

Tasks for Data Conversion

The Jenzabar system coordinator does the following to convert the institution's data:

- Creates a configuration table that defines:
  - The positions within an input record that contain pertinent data
  - What to do with that data
- Executes the *tpconvert* tool, specifying the configuration table
- Verifies that the converted files have correct fields and information
- Tests screens for correct functioning
- Tests a sampling of reports for correct functioning using the converted data

Note: For more information, see Using Tape Conversion in this manual.

PART II - MAINTAINING JENZABAR CX

Overview

Introduction

Part II of this manual provides information and procedures for maintaining CX and the UNIX machine. This part of the manual groups maintenance information in four categories:

- SMOs and Revision Control
- Database Management
- System Administration
- System Maintenance

Also provided are customer assistance and troubleshooting information.

General Maintenance

This section provides information for maintaining the common aspects of CX. For maintenance information and procedures for specific CX products, refer to the Technical Manual for the specific product.

Background Knowledge

The following list describes the necessary background information that you should know to support CX.

UNIX

Know the following about the UNIX operating system:

- Csh environment and commands
- Editor commands (e.g., vi)

INFORMIX-SQL

Know about the following INFORMIX tools:

- SQL database
- PERFORM screens
- ACE reports

Jenzabar CX database tools and utilities

Know how to use the following database tools:

- Schemas
- Macros
- Includes
- Program screens

Jenzabar CX

Know the following about the CX standard product:

- CX directory structure
- The menu processor
- The Jenzabar CX database engine

QuickMate features

Know the following about the Jenzabar CX Graphical Server:

- Client/Server processing
- Network settings
- Keyboard settings
- Mouse settings
- GUI mode commands

SECTION 5 - SMOS AND REVISION CONTROL

Overview

Introduction

This section provides information about SMOs (System Modification Orders) and the Revision Control System (RCS). Jenzabar distributes all enhancements and changes to CX by means of SMOs. A SMO is the following:

- A directory containing subdirectories and files.
- The outcome of a development activity or project, which, when installed on a client site, updates the system.
- Everything necessary for a client to update CX, including all changes to schemas, reports, documentation, screens, objects, and/or C code.

This section discusses the following about SMOs:

- Depositing the SMO
- Installing the SMO
- Resolving local customizations
- Troubleshooting
- Merging files
- Archiving
- Macros

CX uses the Revision Control System (RCS) to maintain and control all changes to files, programs, and reports.

SMO Definition

A SMO is a set of software revisions to accomplish given tasks and typically contains changes to one or more of the following:

- Schemas
- Reports
- Documentation
- Screens
- Objects
- C code
- Macros

Creation Process

The following lists the phases that occur in the creation of a SMO.

1. A project is approved and scheduled, usually as a result of an internal or client request.
2. When beginning the project, Jenzabar does the following:
   - Assigns a new SMO number to the project
   - Adds a SMO directory onto the CX development database, where all changes are checked into the SMO
3. When ending the development work of the project, Jenzabar does the following:
   - Closes the SMO
   - Finalizes the README file

   Note: READMEs are reviewed by at least three Jenzabar personnel.
4. The SMO is ported to all in-house beta databases and the responsible Quality Assurance

Manager ensures that the SMO is ready for distribution to the Beta sites.

Product Advisory

Jenzabar sends a Product Advisory when important changes in the procedures or a short term solution to an existing problem are needed. Please be sure to read all of your mail and e-mail that you receive from Jenzabar regarding SMOs. Jenzabar sends some fixes via the modem to clients when an issue warrants a quick response.

Keeping Up to Date

Jenzabar attempts to get enhancements and fixes to its clients in a timely manner. It is important that clients install SMOs and Product Advisories as soon as they receive them.

Contents of a SMO

Introduction

All SMOs reside in the following directory path: $CARSPATH/smo. To access a specific SMO, you must enter the specific directory named after the SMO number. The following describes the contents of a SMO directory and a skeleton of the SMO README file.

Mandatory SMO Files

The following lists the files that are required in a SMO.

README
Contains the installation instructions for the SMO. It also includes other important information related to the SMO installation.

Note: For an example of a README file, see *SMO README* in this section.

Revtr
Contains a list of all the files affected by the SMO. Any file listed in the Revtr is relative to $CARSPATH.

Note: The Revtr file is not used during the SMO installation and is present for Jenzabar purposes only.

Revtr.dist
Contains a list of the files, relative to $CARSPATH, that will be deposited during the installation of the SMO. The Revtr and Revtr.dist files will differ if proprietary source is contained within the SMO.

Makefile
Defines the directory to be a SMO directory structure so the appropriate *make* targets can be executed. This file must be present but should never need to be reviewed or modified.

Optional SMO Files

The following lists the files that are optional in a SMO.

Revtr.mv
Contains a list of files that will be moved during the installation of the SMO. The files listed are relative to $CARSPATH. The *smomove* make target uses this file as input and moves the files accordingly.

Revtr.rm
Contains a list of files that will be removed during the installation of the SMO. The files listed are relative to $CARSPATH. The *smoremove* make target uses this file as input and removes the files from the system.

Reinstall.tbi
Contains a list of files that must be reinstalled.

Mandatory SMO Subdirectories

The following lists the subdirectories that are required in a SMO.

RCS
Contains information for the Revision Control System.

Dist

Contains the new versions of the files distributed in this SMO, along with log files that will be appended to the RCS for each file. This subdirectory structure is identical to the directory structure relative to $CARSPATH. The new versions of the files listed in the Revtr.dist reside in the Dist directory.

Optional SMO Subdirectories

The following lists the subdirectories that are optional in a SMO.

Procedures

Contains files and scripts that are used to install the SMO. Following are two files, which may be located in the Procedures subdirectory:

Objectlist

Contains a list of any proprietary source distributed with the SMO. The *smodoproc* script uses the Objectlist as input when depositing proprietary source.

Makelist

Contains a list of make targets. The *smodoproc* script reads this file and performs the specified make target on the source listed within this file.

Objects

Contains the actual proprietary source files and other files that Jenzabar distributes as objects. The objects are copied into the install path by the *smodoproc* script, which uses the Objectlist file in the Procedures directory as input. The Objects subdirectory contains a subdirectory named for the operating system which the objects were created under; i.e., hpux, aix.

Tables

Contains the ASCII files to be loaded into the database when initializing tables or files. Scripts under the Procedures directory read the ASCII files and load them into the database.

SMO README Skeleton

The following is the skeleton file that Jenzabar uses to produce a README file for a SMO.

```
SMO#SMO_NOSMO_DESC
================================================================================
OVERVIEW:
================================================================================
INSTALLATION INFORMATION:

    1)  Number of Files.............:   0

    2)  Module(s) Affected.........:

    3)  Approximate Time Required..:   0 hours/minutes

NOTE:  The approximate time does not take into consideration time
          required to merge local customizations or time required for
       rebuilds.
================================================================================
REFERENCE:
    1)  Schema/View Files                              Build or Rebuild
-----------------------------------------------    ----------------
NONE
2)  Object Files                         Owner      Group      Mode
--------------------------------   ---------  ---------  --------
        NONE
    3)  Macro Files                                    Default Value
--------------------------------                 -------------
     NONE
    4)  Special Installation Considerations                    Action
        ------------------------------------------------   --------------
        NONE
    5)  Documentation                                    New or Updated
-----------------------------------------------   --------------
        NONE
    6)  Dependencies
-------------------------------------------------
    NONE
================================================================================
NEW FEATURES/ENHANCEMENTS:
    1)  Feature:
Benefits/Who:
================================================================================
PROBLEMS/SOLUTIONS:
    1)  Problem:
Solution:
================================================================================
INSTALLATION INSTRUCTIONS:
    Refer to the "Instructions for Installing SMOs on Client Sites" document,
    located under $DOCPATH/common/smos/clismo.doc, for general SMO
    installation information.
    1)  Pre-Deposit Steps:                                  Time:   0 minutes
-------------------------------------------------------------------------
        NONE
    2)  Deposit Steps:                                      Time:   0 minutes
-------------------------------------------------------------------------
        a)  Deposit the files.
 % cd $CARSPATH/smo/SMO_NO-ip
          % make smodeposit >& Deposit.Out
        b)  Resolve any errors in Deposit.Out file
        c)  Check in any files listed in the Revtr.CO file.  Re-run deposit
        steps (a & b) as necessary.
        d)  Verify that all files deposited by checking that the number of
            lines in the Revtr.dpst is the same as in the Revtr.dist.
% wc -l Revtr.d*
% print Revtr.LCL
        e)   Resolve any files contained in Revtr.LCL file.

    3)  Pre-Install Steps:                                  Time:   0 minutes
-------------------------------------------------------------------------
         NONE
```

```
  4) Install Steps:                                          Time:   0 minutes
-------------------------------------------------------------------------------

        Install the files.
% cd $CARSPATH/smo/SMO_NO-ip
% make smoreinstall >& Install.Out
        Resolve any errors in the Install.Out file.

NOTE:  Ignore compile warnings containing the following:
    "Runtime error is possible..."
    "Current declaration of..."
    "/* detected in comment..."
    "Optdriver: Exceeding compiler..."

    5)  Post-Install Steps:                                  Time:   0 minutes
-------------------------------------------------------------------------------
        NONE


    6)  Verification Steps:                                  Time:   0 minutes
-------------------------------------------------------------------------------
        NONE
===============================================================================
IMPLEMENTATION CONSIDERATIONS:

===============================================================================
Special In-house Installation Instructions (at CARSC ONLY):
```

SMO Naming Conventions

Introduction

Jenzabar assigns SMO numbers when the development work begins on a SMO. SMO numbers can have additional characters that describe the specific purpose of the SMO. The following describes the conventions for adding characters to SMO numbers.

Note: Since the SMO installation order is based on the development completion schedule, SMOs are not necessarily installed in numerical order. However, it is important to install the SMOs in the correct order.

General SMOs

The following naming conventions exist for general SMOs. When SMOs have multiple versions, additional characters are added to differentiate the versions. For example, a financial aid SMO can come in two versions:

- *10907* for those clients who have not purchased the Financial Aid Packaging module
- *10907M* for those clients who have purchased the Financial Aid Packaging module, but still need some of the files included in the SMO

Note: M in *10907M* above stands for *modified*.

Fix SMOs

Jenzabar can create Fix SMOs for the stages in the distribution process. The following naming conventions exist for Fix SMOs for each stage in the process.

Exceptional or Advanced Beta Fix SMO

A fix SMO sent only to the Exceptional or Advanced Beta site has the following naming conventions:

- An *A* appended to the SMO number, followed by a lower case letter to indicate the proper installation sequence of the fix(es).
- The changes in the fix SMO are placed in the original SMO, so that only one SMO, the original, is needed for proper installation on a client system.

For example, if SMO *10000* is only at an Exceptional or Advanced Beta site and a fix SMO is needed, the fix SMO will be named *10000Aa.* Any subsequent Beta fixes for this SMO will follow the sequence *10000Ab*, *10000Ac*, etc.

Beta Fix SMO

A fix SMO that is to be sent only to the Beta sites and the Advanced/Exceptional Beta site has the following naming conventions:

- A *B* appended to the SMO number, followed by a lower case letter to indicate the proper installation sequence of the fix(es).
- The changes in the fix SMO will also be placed in the original SMO, so that only one SMO, the original, is needed for proper installation on a client system.

For example, if SMO *10000* is in regular Beta testing and a fix SMO is needed, the fix SMO will be named *10000Ba*. Any subsequent Beta fixes for this SMO will follow the sequence *10000Bb*, *10000Bc*, etc.

Note: If a fix SMO has been sent to an Exceptional Beta site it will also use the naming conventions for the standard Beta Fix SMO explained above.

Pre-General Distribution

A fix SMO that is sent only to one Pre-General site to test the SMO installation order of individual SMOs that have been at various Beta sites.

- A lowercase letter appended to the SMO number to indicate the proper installation sequence of the fixes.
- The changes in the fix SMO will also be placed in the original SMO, so that only one SMO, the original, is needed for proper installation on a client system.

For example, if SMO 10000 is in regular Beta testing and a fix SMO is needed, the fix SMO will be named *10000Cc*. Any subsequent Beta fixes for this SMO will follow the sequence *10000Cb*, *10000Cc*, etc.

General Distribution Fix SMO

A fix SMO that is to be sent to all client sites has the following naming conventions:

- A lower case letter appended to the SMO number to indicate the proper installation sequence of the fix(es).
- In this case, the changes in the fix SMO are not included in the original SMO, since the clients will have already received the original.

For example, if SMO *10000* is in General Distribution and a fix SMO is needed, then the fix SMO will be named *10000a*. Any subsequent fixes for this SMO will follow the sequence *10000b*, *10000c*, etc.

Receipt of Fix SMOs

Depending on the type of site, clients can receive Fix SMOs in the following manner.

- The Advanced or Exceptional Beta site could possibly receive each of the fix SMO naming conventions (10000Aa, 10000Ba, and 10000a) explained above, if a fix is needed at each stage of the distribution.
- A Beta site can receive (10000Ba and 10000a).
- A general site can only receive the naming convention (10000a) for a fix SMO.

SMO Distribution Cycle

Introduction

Jenzabar has developed a distribution cycle of SMOs that provides for advanced testing of enhancements to CX before the general client base receives the SMO. The phases of the distribution process are described below.

Note: Jenzabar distributes SMOs in the US mail or overnight services, when necessary. Some distributions also occur via modem and the Internet. For example, distributions to Beta sites and SMOs for Financial Aid when timing is critical.

The Distribution Process

The following describes the distribution process for SMOs.

1. Exceptional or Advanced Beta Testing phase
   - The Advanced Beta site receives the SMO about eight weeks prior to general distribution.
   - If necessary, an Exceptional Beta site can enter the testing cycle at any point. This site can serve as an Advanced Beta or Beta site.
   - On occasion, a client other than the contracted Beta site can serve as an Exceptional Beta test site. This occurs when a specific client provides the best test environment for the SMO. To serve as an Exceptional Beta site, the client must have all prior SMOs installed.
   - If necessary, the README and the SMO are modified based upon the experience at the Exceptional Beta site. The site will receive these fixes in a SMO with an A in the suffix. These fix SMOs will also be sent to any other client that has already received the original SMO.
2. Beta Testing phase
   - Jenzabar has at least one Beta testing site for each supported operating system, currently HP and IBM. The Beta sites should receive the SMO about eight weeks prior to general distribution.
   - If necessary, the README and the SMO are modified based upon the experience at the Beta site. The Beta sites receive these fixes in a SMO with a B in the suffix. These fix SMOs are sent to any Exceptional Beta site that has already received the SMO.
3. Pregeneral Testing phase
   - Following beta testing and one week prior to general distribution of the SMO, the SMO is sent to a Pregeneral test site. This site installs the SMOs in the same order the general client base will. The Pregeneral site attempts to locate any hidden dependencies due to the installation order that may not have been discovered at the other test sites.
   - If necessary, the README and the SMO are modified based upon the experience at the Pregeneral site. The Pregeneral site receives these fixes in a SMO with a C in the suffix. These fix SMOs are also sent to any Exceptional Beta, and Beta site that received the original SMO.
4. General Distribution phase
   - SMO tapes are mailed to the remainder of the client population in a general distribution.
   - If problems arise that were not discovered through the testing process, then a fix SMO will be developed. The client base will receive these fixes in a SMO with a lower case alpha character suffix. These fix SMOs will also be sent to all test sites.

Advanced Beta Distributions

Jenzabar has established an Initial Beta site agreement with one of our clients. Under this agreement, the following occurs:

1. The Advanced Beta site follows the README and installs the SMO.
2. Jenzabar personnel carefully observe the installation, implementation, and end-user testing of the SMOs and are available to clarify any instructions and assist with any problems which arise.

   Note: This enables Jenzabar to learn of any errors in the SMO instructions or clarifications required in the SMO README file. If any problems occur, the problems are rectified before releasing the SMO to other beta sites.
3. The Advanced Beta site verifies that the general features in a SMO function as they should and verify that the new features do not adversely affect current operations.

Beta Distributions

Jenzabar has established at least one Beta site for each hardware vendor currently supported. If a SMO has gone to Advanced Beta, the changes are made to the SMO and then it goes to Beta.

The Beta sites also install and test the SMO. These sites report any problems detected to CX. Jenzabar works with the Beta sites to get their comments back as soon as possible. Quick response from our Beta sites allows Jenzabar to make necessary changes before the SMOs go to General Distribution.

Exceptional Beta Distribution

Enhancements can occur that cannot be thoroughly tested by the Beta sites because the features in the SMO are not relevant to those sites. To facilitate the testing of such enhancements within a SMO, Jenzabar makes arrangements with a client who will utilize the new feature. This client must become an Exceptional Beta Site and take the responsibility of testing the SMO. Jenzabar provides this client with an Exceptional Beta Agreement that they will be asked to sign and return to Jenzabar. Exceptional Beta sites must:

- Be up to date on the installation of the SMOs they have received prior to this time
- Receive and install all outstanding, completed SMOs they have not received in a General Distribution to date. These SMOs may still be at Beta.

When there are prior Beta SMOs to receive, the Exceptional Beta Site becomes an additional Beta site for these SMOs. This measure is being taken to improve the quality assurance of the SMOs in the General Distribution.

General Distribution

All clients that did not receive a SMO in the previous distribution, receive the SMO at General Distribution when it has satisfactorily passed Beta testing.

Installing a SMO

Introduction

The following provides the procedures for installing a SMO.

Installation Order

It is extremely important that SMOs be installed in the order specified by Jenzabar. Note the following:

- The SMO installation order is listed on the label affixed to the tape.
- The installation order is determined by dependencies
- The *smoorder* command lists SMOs by the order of installation. The command also displays the SMO title for reference. Following is an example of the smoorder command output:

```
DEVbetai7: /usr/local/cisc/smoorder

Installation
   Order        SMO Description
------------    -------------------------------------------------
SMO 12090       Document Imaging Release
SMO 12278       IVR Credit Card Bill Payment
SMO 12338Bc     Faculty/Student Web Access
SMO 12544       Taxpayer Relief Act 19
SMO 12538       Financial Maintenance
SMO 12517       Resource/Schedule 25 Interface
SMO 12367a      Web Registration
SMO 12367aBa    Web Registration
SMO 12519       Misc. Degree Audit
SMO 12519Ba     Miscellaneous Degree Audit
SMO 12518       Web Access - FinAid
SMO 12071       FinAid Loan Module 1.0
SMO 12100       Lead Entry/Lead Tickler
SMO 12530       RPA 1.40-Simplified Invoice Entry
SMO 12509       MHCC Enhancements
```

CAUTION: If SMOs are installed out of order, Jenzabar will not handle issues through Support Services. If any logical dependency issues occur with SMOs installed out of order, you will need to contact Jenzabar Consulting Services for assistance if needed. This assistance will be handled on a time and materials basis.

SMO Installation Rules

The following lists the rules for installing a SMO.

- Read the entire README prior to installing the SMO.
- Install all SMOs.
- Install SMOs in the proper order. Use the *smoorder* command to determine the installation order.
- Install the SMO using your normal user login. Do *not* perform any installation steps as root or super user (su) unless the README instructions specifically state to do so.
- Install the SMOs promptly.
- Redirect the output from all processes to a file. If you run the same process more than once, save the output to a different file so the original file is not overwritten.
- Examine the output for errors or informational messages.
- Finish each step before proceeding to the next step.
- No SMO is complete until the users have been informed about the new features and enhancements within the SMO.

Installing Third Party Software Upgrades

Jenzabar distributes SMOs for third party software upgrades, (e.g. the operating system).

CAUTION: Do not install the upgrade for the third party software received from the third party vendor without the SMO from Jenzabar, which contains the proper instructions concerning the upgrade.

Loading the SMO Tape

The following lists the steps for loading the SMO tape:

1. Mount the SMO tape onto the tape drive, and put the drive online.
2. Change directories to the SMO directory.
   % cd $CARSPATH/smo
3. To display the SMO tape's contents to the terminal screen, perform the following:
   % copyin -t
4. To extract the SMO tape contents, enter:
   % copyin -v

   Note: Do *not* extract the SMO tape as super user (su) or root.

Review the SMO READMEs

The following lists the steps for reviewing the SMO README file:

1. Print all the READMEs for all the SMOs contained on the tape. Each tape may contain multiple SMOs.
   % cd $CARSPATH/smo
   % print SMO#/README
2. Review the OVERVIEW section within the README. This section provides general information regarding the purpose of the SMO.
3. Review the MACROS section within the README. This section provides information on any macros in the file and their default values.
4. Review the INSTALLATION INFORMATION section within the README. This section provides information on how many files are in the SMO, which modules are affected by the SMO, and how long it should take to install the SMO.
5. Review the REFERENCE section within the README. This section provides information on which schemas will need to be built, what objects are contained in the SMO, and any special installation considerations. It will also list any documentation that the SMO will be depositing.
6. Review the NEW FEATURES/ENHANCEMENTS section. This section will describe any new features or enhancements contained with the SMO. This portion of the README should be distributed to the persons responsible for the affected modules listed in the INSTALLATION INFORMATION section.
7. Review the PROBLEMS/SOLUTIONS section. This section will describe any problems or bugs that will be fixed by the SMO. This portion of the README should be distributed to the persons responsible for the affected modules listed in the INSTALLATION INFORMATION section.
8. Review the INSTALLATION INSTRUCTIONS section. This section contains the actual steps that must be performed to install the SMO. Review the installation steps prior to starting the installation.

9. Review the IMPLEMENTATION CONSIDERATIONS section. This section provides additional information on setup requirements and instructions for new enhancements distributed in the smo.

Prepare to Start the SMO Installation

The following lists the steps for starting the SMO installation:

1. Change the name of the SMO directory to mark the SMO as in progress (-ip).
   % cd $CARSPATH/smo
   % mv SMO# SMO#-ip
2. Review the README file again. Make sure you are prepared to modify CX as stated in the README.
   % cd $CARSPATH/smo/SMO#-ip
   % print README

   Note: You can also just view (read-only) the README on the screen by typing **view README** at the prompt.

Pre-Deposit Steps

The following lists the steps for performing the README installation instructions:

1. Perform any pre-deposit steps outlined in the SMO README. Any steps in this section of the README will be explicitly stated.
2. The pre-deposit steps may include the execution of the smoremove make target. This target will read the Revtr.rm file as input and remove each file from the system. Review the Revtr.rm file prior to executing the smoremove make target. The smoremove command will create a Revtr.rmd which will contain a list of the files successfully removed. The Revtr.rm and Revtr.rmd should be identical if the smoremove was successful. To verify the smoremove, check the line counts for both files.
   % cd $CARSPATH/smo/SMO#-ip
   % make smoremove > $Remove.out
   % wc -l Revtr.rm Revtr.rmd
3. The smomove target may also be performed in this section of the README. This target will read the Revtr.mv file as input and move each file listed to its new location. Review the Revtr.mv file prior to executing the smomove make target. The smomove command will create a Revtr.mvd which will contain a list of the files successfully moved. The Revtr.mv and Revtr.mvd should be identical if the smomove was successful. To verify the smomove, check the line counts for both files.
   % cd $CARSPATH/smo/SMO#-ip
   % make smoremove > $Move.out
   % wc -l Revtr.mv Revtr.mvd

Deposit Steps

The following describes the six steps for depositing a SMO:

Step 1

Deposit the new versions of the files contained in the SMO under the Dist directory. The smodeposit make target reads the Revtr.dist file, locates the files to deposit under the Dist directory and then copies them into CX. Therefore, this target only acts upon the files listed in the Revtr.dist file.

```
% cd $CARSPATH/smo/SMO#-ip
% make smodeposit >& Deposit.out
```

The smodeposit target will create up to five output files. These five files will be located in the SMO directory and must be reviewed. Below is a description of each file and its contents, and the actions required for each file.

- **Deposit.out** - This file contains the output from the smodeposit process. This file is created when the output is redirected into it from the smodeposit. Review this file for error messages.
- **Revtr.dpst** - The dpst extension on this file stands for deposited. This file contains the list of files, relative to $CARSPATH, that were successfully deposited. As the smodeposit target reads the Revtr.dist file and copies the files from the Dist directory into CX, it appends the filename to the Revtr.dpst file. If the deposit was successful, the Revtr.dist and Revtr.dpst files should be identical except for files under the $CARSPATH/src directory ending in .c. These files are deposited at the revision level of the Revfile.c file in the same directory. However, the number of lines in both the Revtr.dist and Revtr.dpst should be identical in a successful smodeposit.
- **Revtr.CO** - The CO extension on this file stands for checked out. This file will only be created if the smodeposit target finds a checked out file. The filenames, path, and current owners of any checked out file, which this SMO was attempting to deposit, will be listed in the Revtr.CO file. Each file listed in the Revtr.CO file must be checked in so the file can be deposited. Files listed in the Revtr.CO will not appear in the Revtr.dpst unless they are checked in (or unchecked out) and have been redeposited.
- **Revtr.LCL** - The LCL extension on this file stands for local. This file will be only created if locals are found. A local is defined as any file in CX in which the latest revision has been modified at the client's site. The Revtr.LCL file contains filenames, paths, and local branch numbers of the files on which the smodeposit found a local version. Files listed in the Revtr.LCL will appear in the Revtr.dpst.
- **Revtr.LCL-mak** - The LCL-mak extension on this file stands for local on Makefile. This file will be created if a local change has been made to the previously deposited Makefile (which is now saved as .makefile).

Step 2

Examine the Deposit.out file for errors. If errors are found in the Deposit.out file, check the Troubleshooting section of this guide to see if it provides a solution for your error. If you require further assistance, call Jenzabar Support Services.

```
% print Deposit.out
```

Step 3

Resolve any files listed in the Revtr.CO file. Either check the file in or uncheckout the file, then deposit the new version of the file contained in the SMO.

```
% print Revtr.CO
```

1. To check in a file listed in the Revtr.CO file, first change directories to the path listed in the Revtr.CO file. This will preserve the changes made to the file plus it will create a local revision on the file.

   ```
   % cd $CARSPATH/ ...
   % make ci F=<file>
   ```

2. To uncheckout the file, which will delete the changes made to it and restore it to its previous version, change directories to the path listed in the Revtr.CO file and perform the following:

   ```
   % cd $CARSPATH/...
   % make unco F=<file>
   ```

3. Redeposit the new version of the files and examine the newly created Deposit2.out file for errors.

   % **cd $CARSPATH/smo/SMO#-ip**
   % **make smodeposit>& Deposit2.out**

   If few files are involved, it may be easier to deposit the new versions individually. First, recall the file version number and path from the Revtr.dist file. Then change directories to where the file is located and perform the following:

   % **cd $CARSPATH/...**
   % **make deposit F=filename:version# SMO=SMO#-ip**

   Either way of depositing the new versions will update the Revtr.dpst file.

4. Resolve all files contained in the Revtr.CO prior to resolving the files in the Revtr.LCL because the Revtr.LCL file can grow as you are resolving the Revtr.CO files.

Step 4

Resolve any files listed in the Revtr.LCL file. This file will contain files which have had local customizations made to them at the client's site. During and following implementation, screens, forms, and perhaps even programs are customized to meet the needs of users on a client site. These customizations must be evaluated when new versions of these files are released by Jenzabar.

The process of deciding what to keep and what to discard is called resolving local customizations, and is very important to updating the functionality of a system without losing the special features built into it.

1. Review the Revtr.LCL file and note the names of the files, their locations, and current revision numbers.

   % **print Revtr.LCL**

   The smodeposit target creates three files when it deposits a new version on top of a file that contains local customizations. Each of the three files carries the same base filename. The extension distinguishes one file from another.

   - <file> - This is the newly deposited version of the file.
   - <file>.lcl - This is the current local client version.
   - <file>.log - This file contains the local revision log messages tracking the changes made to the file.

2. Review the changes made to the file by using the rlog command. Identify and note the numbers for the new version, the local branch version (this will typically be listed last in the rlog output), and the last Jenzabar revision. The revision number, for a file with a local revision, will have four parts and will be in the format n.nnnnnn.nnnn.nn. The first number denotes the release, the second section denotes the Jenzabar version of the file, the third section is the client number assigned by Jenzabar, and the fourth section is the client version number. Example, 7.100003.5000.2 stands for release I and version number 100003 for CX and version 2 for client number 5000. The 7.100003 is the trunk version and the 5000.2 denotes the branch created by the client to this trunk.

   % **rlog <file> | more**

   Note: Review The Revision Control System in this section for more information on the Revision Control System and interpreting the rlog output.

   Identify the differences among the three versions of the file. The diff target compares the last two revisions of a file. The output is automatically sent to an output file whose name is <file>.out, where <file> is the base name.

   % **make diff F=<file>**

To identify the differences between any two versions of a file, pass the two version numbers to the diff target. This will still create a <file>.out file containing the differences.

% **make diff F=<file> V=version#1:version#2**

Run this command on the previous trunk version and the local version number (the Current version number from the rlog command, this is also found in the Revtr.LCL file). View what changes were made at the client's site to the previous Jenzabar version. Then diff the two Jenzabar trunk versions to see what changes were made to the file by Jenzabar. The <file>.out file contains the lines which are different among the two versions of the file. The lines with < preceding them are from the first version listed on the make diff command line, and those lines with > preceding them are from the second version listed on the make diff command line.

It is also possible to visually examine both the <file> (the new version distributed with the SMO) and the <file>.lcl (the current local version) to determine the differences between the two.

% **print <file> <file>.lcl**

3. Once the changes have been examined, one of three choices must be made. These are the choices:
   - Retain the new CX version
   - Retain only the current client version
   - Merge the two versions
4. To retain the new Jenzabar version of the file, remove the <file>.lcl file and proceed with the next file listed in the Revtr.LCL.

   Note: The <file>.log file will be automatically removed when the smoinstall or smoreinstall command is executed.

   % **cd $CARSPATH/...**
   % **rm <file>.lcl**
5. To retain the current local version of the file and discard the newly deposited version, perform the following steps:

   Keep in mind, this procedure will remove the changes Jenzabar made to the file. Therefore, review what Jenzabar has modified first to determine if the new features or enhancements are desired. Use the rlog command and the diff target to determine exactly what Jenzabar has modified prior to performing the following steps:

   % **cd $CARSPATH/...**
   % **make co F=<file>**
   % **vi <file>**

   Move cursor to the end of the Header section and delete everything to the end-of-file.

   dG <ESC>

   Read in the local version of the file.

   :r <file>.lcl

   Delete the local version Header section and save the changes and exit.

   :wq!

   % **make ci F=<file>**

   This series of steps will result in the body of the customized version being substituted for the body of the new version, and the revision control information for the customizations will be retained. When the file is checked in, a local branch number will be created.
6. To combine the two versions together, execute a merge upon the file. The merge will retain the local customizations to the file plus retain the new changes made to the file by Jenzabar. If the local and the new changes to the file have modified a common set of lines, then a <file>.mrg file will be created. The common lines are called overlaps and

the overlaps will be contained in the <file>.mrg file. The overlaps are also marked within the actual file with >>>> and <<<< marking the beginning and ending points of the overlap, with ==== between the two versions. These overlaps must be dealt with manually by editing the file and deciding which version you would like to retain.

Note: When merging program source under $CARSPATH/src never pass the make processor a F= argument. This source is treated as a whole even though several files are present within the directories. To use the merge target on program source just issue the command make merge. The make merge command checks out the file on which it acts. Therefore, all files listed in the Revtr.LCL file must be checked in before continuing with the next step in the README installation instructions. The ci, smoinstall, and smoreinstall targets automatically remove files with the following extensions; .out .lcl .mrg .log.

% **cd $CARSPATH**
% **make merge F=<file>**
% **vi <file>**

Resolve any overlaps at this time and save any changes made to the file.

:wq!
% **make ci F=<file>**

Note: If the check in process fails for program source under $CARSPATH/src due to "undefined symbols error," this is either because the program is looking for field changes that are unknown to the database at this point, or because a library that was changed in the smo has not yet been installed. After the schemas included in the smo are built, and all source code for libraries changed in the smo has been installed, the database dictionary and the source code libraries will be up to date and the program source may be checked in.

7. The smomerge target can be used if all the files listed in the Revtr.LCL file should be merged. This target will read the Revtr.LCL and perform the merge. This target will create a Revtr.co file which contains a list of all the files which it checked out. Typically, the Revtr.LCL and Revtr.co files will contain the same list of files after the smomerge target is executed.

   Note: The smomerge target checks out the file on which it acts. Therefore, all files listed in the Revtr.co file must be checked in before continuing with the next step in the README installation instructions.

   % **cd $CARSPATH/smo/SMO#-ip**
   % **make smomerge >& Merge.out**

8. If the smomerge target was used to resolve the files contained in the Revtr.LCL file then the smoci target can be used to check in the files listed in the Revtr.co file. Recall from above, the Revtr.co file contains a list of files which the smomerge target left checked out. After each file is reviewed and the overlaps are resolved, use the smoci target to check in all the files listed in the Revtr.co file at once.

   % **cd $CARSPATH/smo/SMO#-ip**
   % **make smoci >& Checkin.out**

Step 5

Review any files listed in the Revtr.LCL-mak file to determine whether these changes are still needed.

```
% cd <directory name>
% diff Makefile Makefile.LCL
```

If there are no differences that seem critical (The addition of your own library to ADDLIBS), remove the Makefile.LCL.

Step 6

After the Revtr.CO and Revtr.LCL files have been resolved, verify the deposit is complete. Recall, the smodeposit target is copying all files within the Dist directory into CX and creating the Revtr.dpst file as it works. The deposit can be checked two different ways. The two most common ways are to check the number of lines in the Revtr.dist and the Revtr.dpst files.

1. To check the number of lines in both the Revtr.dist and Revtr.dpst files, use the wc (word count) command. The -l parameter to wc will display the line counts in each file. Below is an example of the command and an example of the output. Since both files contain 19 lines, this indicates the deposit was successful.

```
% wc -l Revtr.dist Revtr.dpst
19 Revtr.dist
19 Revtr.dpst
38 total
```

2. To verify the deposit, you can use the smochkdpst target. This target will use the Revtr.dist file and verify the correct version of each file is resident in CX.

```
% cd $CARSPATH/smo/SMO#-ip
% make smochkdpst >& Chkdpst.out
% print Chkdpst.out
```

Review the Chkdpst.out file. An empty Chkdpst.out file indicates a successful deposit.

Note: This target will take some time to execute.

Pre-Installation Steps

The following lists the steps to follow prior to installing the SMO:

1. Perform any pre-install steps outlined in the SMO README. Any steps in this section of the README will be explicitly stated.

   Note: Two of the common steps found in this section of the README include schema builds and the depositing of object files.

   Note: If a schema is modified within a SMO, it will be listed in the REFERENCE section of the README plus the installation steps will be explicitly stated in the INSTALLATION INSTRUCTIONS section.

2. Prior to building a schema, use the buildn target to review the changes being made to the schema. This target will create a file with a .sql extension and it will contain the changes being made to the schema. If the merge target was used upon the schema when resolving the Revtr.LCL file, this will verify that local fields were not lost.

```
% cd $CARSPATH/schema/...
% make buildn F=<file>
% print <file>.sql
```

3. After verifying that all changes are correct and that no local fields are being deleted, use the buildy target to build the schema.

```
% cd $CARSPATH/schema/...
```

% **make buildy F=<file>**

Note: You can review all of the associated schema make targets in the SMO Make Targets section of this guide.

4. Run the smodoproc script. The smodoproc script is used to deposit any object files contained in the SMO. The smodoproc script reads the Objectlist file, found in the Procedures directory of the SMO, and performs the instructions in the file. The script will locate the actual objects in the Objects directory of the SMO and copy them into CX. This script can also perform two other activities; creating symbolic links, and performing specified make activities. Below is an example of how to execute the smodoproc script:

   Note: The README may state to execute the smodoproc script during the pre-deposit steps. Execute the script as the README indicates.

   % **cd $CARSPATH/smo/SMO#-ip**
   % **smodoproc |& tee Smodoproc.out**

   Note: The README may state to run this script as root or super user (su). If so, the README will explicitly state this. Otherwise, run the script as yourself.

5. The smoremove target may also be performed in this section of the README. This target will read the Revtr.rm file as input and remove each file listed from the system. The smoremove command will create a Revtr.rmd file which will contain a list of the files successfully removed. The Revtr.rm and Revtr.rmd should be identical if the smoremove was successful. To verify the smoremove, check the line counts for both files.
   % **cd $CARSPATH/smo/SMO#-ip**
   % **wc -l Revtr.rm Revtr.rmd**

6. The smomove target may also be performed in this section of the README. This target will read the Revtr.mv file as input and move each file listed to its new location. Review the Revtr.mv file prior to executing the smomove make target. The smomove command will create a Revtr.mvd which will contain a list of the files successfully moved. The Revtr.mv and Revtr.mvd should be identical if the smomove was successful. To verify the smomove, check the line counts for both files.
   % **cd $CARSPATH/smo/SMO#-ip**
   % **wc -l Revtr.mv Revtr.mvd**

Installing the SMO

The following lists the steps for installing a SMO:

1. Install the deposited files. This is accomplished with the smoreinstall or smoinstall target. This target will use the Revtr.dpst file as input and install each file listed within it. If the deposit was successful then the smoreinstall will install the entire SMO.
   % **cd $CARSPATH/smo/SMO#-ip**
   % **make smoreinstall >& Install.Out**
   % **print Install.Out**

2. The smoreinstall target will not create any special output files. The only file created during this step is the Install.Out file which should be checked for errors.

Post-Install Steps

The following lists the steps to follow after installing the SMO.

1. Perform any post-install steps outlined in the SMO README. Any steps in this section of the README will be explicitly stated.

   Note: This section may contain instructions for any special reinstalls that may need to take place after the SMO is installed. However, the Reinstall.tbi, which is appended and sorted to the Revtr.tbi, automates most post-install steps.

2. The smoremove target may also be performed in this section of the README. This target will read the Revtr.rm file as input and remove each file listed from the system. The smoremove command will create a Revtr.rmd file which will contain a list of the files successfully removed. The Revtr.rm and Revtr.rmd should be identical if the smoremove was successful. To verify the smoremove, check the line counts for both files.
   % **cd $CARSPATH/smo/SMO#-ip**
   % **wc -l Revtr.rm Revtr.rmd**
3. The smomove target may also be performed in this section of the README. This target will read the Revtr.mv file as input and move each file listed to its new location. Review the Revtr.mv file prior to executing the smomove make target. The smomove command will create a Revtr.mvd which will contain a list of the files successfully moved. The Revtr.mv and Revtr.mvd should be identical if the smomove was successful. To verify the smomove, check the line counts for both files.
   % **cd $CARSPATH/smo/SMO#-ip**
   % **wc -l Revtr.mv Revtr.mvd**

Verification Steps

Perform any verification steps outlined in the SMO README. Any steps in this section of the README will be explicitly stated.

Marking the SMO as Installed

Mark the SMO as installed after the SMO has been completed. The -inst suffix on the SMO directory name indicates the SMO has been installed.

% **cd $CARSPATH/smo**
% **mv SMO#-ip SMO#-inst**

Reviewing the Documents Directory

Each SMO that changes documentation, or creates new documentation, contains a Documents directory with the associated new or updated documentation. When you open the Documents directory, you will see four files:

- pclfile
- psfile
- zipfile
- README

The pclfile, psfile and zipfile contain the manuals that document the changes from this SMO in three printable formats; UNIX pcl format, postscript format, and Word zipped files. Select the print format that you want to use and download those documents. You can delete the other two files.

The README contains information about the manuals that changed and a brief description of those changes. It also contains printing instructions for each of the print formats.

Implementing the SMO Features

It is very important to educate the users during and after the SMO installation. The Jenzabar system coordinator should supply the end user with the appropriate sections of the README along with any new or updated documentation. Now that the SMO is installed, any new features or enhancements can be set up and tested. Plus, any problems or bugs fixed in the SMO should disappear.

Archiving SMOs

SMOs use a large portion of disk space; therefore, SMOs should be archived from the system after they are installed and implemented. To archive SMOs use the copyout command. This command will copy SMOs and their contents to tape. Once the tape has been loaded and the drive is online, perform the following steps. Note, enclose the SMO numbers within double quotes if more than one SMO is being archived.

% **cd $CARSPATH/smo**
% **copyout -d "SMO#1 SMO#2 SMO#3 ....."**

Once the copyout command finishes successfully, the SMOs can be removed from the system.

% **rm -rf SMO#1 SMO#2 SMO#3 .....**

Note: You may need to be root to successfully remove the SMOs.

SMO Make Targets

Introduction

This section provides information on the SMO make targets. SMO make targets use a Revtr file as input and perform the base target upon each file listed in the Revtr file. The file used as input for each target is listed below plus any output files created by the target. The SMO targets begin with smo and they must be executed from the SMO directory as follows:

% **cd $CARSPATH/smo/SMO#**
% **make <target>**

For more information on the make processor and the base targets, see *Using the Make Processor* in this document.

SMO Targets

The following lists and describes the MAKE targets for SMOs.

smodeposit
Base Target = deposit
Input File = Revtr.dist
Output File(s) = Revtr.dpst, Revtr.CO, Revtr.LCL
Description:
Checkouts the files listed in the Revtr.dist, copies the file found in the SMO Dist directory into CX, and checks the file back in. The target deposits the new file at the version level listed in the Revtr.dist.

- If the file is successfully deposited, it will list the file in the Revtr.dpst. The Revtr.dist and Revtr.dpst files should be identical unless a Revtr.CO file is present.
- If a file is found in a checked out state, it will list the file in the Revtr.CO.
- If a file contains local customizations, it will list the file in the Revtr.LCL.

smoinstall
Base Target = install
Input File = Revtr.dpst
Output File(s) = none
Description:
Installs all files listed in the Revtr.dpst file.

smomerge
Base Target = merge
Input Files = Revtr.LCL
Output File(s) = Revtr.co
Description:
Merges the local customization with the new version of the file deposited by the smodeposit target. The target leaves all the files in a checked out state; therefore, each file must be reviewed and checked in.

smomove
Base Target = move
Input File = Revtr.mv
Output File(s) = Revtr.mvd
Description:
Moves all the files listed in the Revtr.mv to the new location that is also listed in the Revtr.mv file. This target lists each file moved successfully in the Revtr.mvd file. Both the Revtr.mv and Revtr.mvd files should be identical if the smomove finished successfully.

smoremove

Base Target = remove
Input File = Revtr.rm
Output File(s) = Revtr.rmd
Description:

Removes all the files listed in the Revtr.rm file from CX. This target lists each file removed successfully in the Revtr.rmd file. Both the Revtr.rm and Revtr.rmd files should be identical if the smoremove finished successfully.

smoci

Base Target = ci
Input File = Revtr.co
Output File(s) = none
Description:

Checks in all files listed in the Revtr.co file that is created by the smomerge target.

smounco

Base Target = unco
Input File = Revtr.LCL
Output File(s) = none
Description:

Unchecks out all files listed in the Revtr.LCL file. Typically, this target would only be used after the smomerge target is executed. Keep in mind, the target restores the files back to their state immediately following the smodeposit.

smotinstall

Base Target = tinstall
Input File = Revtr.dpst
Output File(s) = none
Description:

Temporarily installs all the files listed in the Revtr.dpst file.

smoreinstall

Base Target = reinstall
Input File = Revtr.dpst
Output File(s) = none
Description:

Reinstalls all the files listed in the Revtr.dpst file.

smodelrev

Base Target = delrev
Input File = Revtr.dpst
Output File(s) = none
Description

Deletes the versions of the files deposited by the smodeposit. Only use this target in extreme cases.

Dealing with Local SMOs

Overview

This section describes the current procedures required to incorporate updates made on a client system into the standard CX. The procedures involve creating a local SMO on a client system (containing the revisions to be maintained) and then incorporating that local SMO into a regular SMO in the standard Jenzabar CX product.

The following procedures should be completed on the client system from which the revisions are to come. A SMO local to that system will be created, with the revisions to be brought back excluding any client customizations.

Steps for Incorporating Updates on Local Client Sites

The following process is required to create a local SMO:

1. Create a local SMO to capture changes
2. Check-in the revisions for the SMO
3. Close the local SMO
4. Put the SMO on tape to bring to Jenzabar
5. Integrate the local SMO into the CX
6. Extract the local SMO from the tape
7. Remove local customizations
8. Build a new Revtr file
9. Resolve version number overlaps
10. Create or determine the regular Jenzabar CX SMO to use
11. Deposit the local SMO as part of the regular SMO
12. Check for any Makefile.lcl files
13. Address files not handled by smonewrev
14. Update the README file for the regular SMO
15. Move the local SMO to the ARCH directory

Create a Local SMO to Capture Changes

You can distinguish a local SMO from a regular SMO as follows:

- Its name begins with a capital "L"
- Its next three characters are a 3-digit client number
- Its last twi characters are a 2-digit serial number

Example: L02305 for the fifth local SMO on client 23's system

You can execute the *newlclsmo* command from any location to create a local SMO directory under CARSPATH/smo:

```
% newlclsmo
>>Command: adddir
>L02305 - makeinit smo
Enter 1 line SMO Description:
Add 'db' alias for switching between databases
...thank you.
Updating '.makevar.mak'
%
```

This creates an L02305 SMO directory under CARSPATH/smo with an initial README file that can be updated with special information about the SMO.

Check in Revisions for the SMO

You can put a revision into a local SMO in one of two ways. If it has been determined ahead of time (before checking in the revision) that the changes should be put into a local SMO, the local SMO identifier can be specified on the *make ci* command line. The normal checkin of the local version will occur followed by execution of the 'lclsmo' command which will add the new revision to the Revtr.new file of the specified local SMO.

If other local customizations already exist on the branch, *lclsmo* will attempt to strip them out create a new version on a separate branch (branch 1) to include in the SMO instead. This procedure can only work properly if revisions to be brought back are not grouped together with local client customizations in the same revision.

Example 1: Unmerge client customizations

```
% cd skel
% make ci F=cshrc.s SMO=L02305 L='Add db alias'
>>Command: checkin
>cshrc.s - translate - checkin.
>>Command: lclsmo
>cshrc.s - unmerge (6.9.2300.3 - 6.9.2300.2) - creating 6.9.1.1 -
updating Revtr.new.
```

Example 2: Overlaps during unmerge

```
% cd skel
% make ci F=cshrc.s SMO=L02305 L='Add db alias'
>>Command: checkin
>cshrc.s - translate - checkin.
>>Command: lclsmo
>cshrc.s - unmerge (6.9.2300.3 - 6.9.2300.2): 1 overlap. -
skipping unmerge
>cshrc.s 6.9.2300.3 - updating Revtr.new.
```

If the SMO was not specified on the *make ci* because it was forgotten or because more control over the unmerging process is needed, the *lclsmo* command can be done separately. The following specifies that version 6.9.2300.3 is to be included in the local SMO with 6.9.2300.1 (instead of the default of 6.9.2300.2) being unmerged (or subtracted out). This might be necessary if 6.9.2300.2 also contains changes to be brought back or if overlaps occurred during the unmerge of 6.9.2300.3 and 6.9.2300.2 and it is determined that subtracting 6.9.2300.1 is better than just taking 6.9.2300.3.

Example 3 Specify version to unmerge

```
% cd skel
% make lclsmo F=cshrc.s SMO=L02305 V=Recent:6.9.2300.1
>>Command: lclsmo
>cshrc.s - unmerge (6.9.2300.3 - 6.9.2300.1) - creating 6.9.1.1 -
updating Revtr.new.
```

Close the Local SMO

Once all revisions have been checked in and added to the Revtr.new file for the local SMO, it can be closed to be brought back to Jenzabar. The procedure is the same as for closing a SMO at Jenzabar. Use *make smodist* to produce in the Dist directory a "distribution" copy of each of the files specified in the Revtr.new file.

```
% cd smo/L02305
% make smodist >& Dist.out
% more Dist.out                (check for errors)
```

Once this is done, the README file should be updated with any information that will be helpful to integrating the local SMO into the product at Jenzabar. This includes describing the enhancements and fixes that the SMO includes as well as any special installation or porting procedures.

```
% cd smo/L02305
% vi README                     (update it)
```

Put the SMO on Tape to Bring to CISC

Until *uucp* connections are working well to each client site, local SMOs must be brought back on tape. The local SMO can be combined with other files being brought back on the same tape, but the general method would be to use *copyout* as follows:

```
% cd smo
% copyout -d L02305 -v          (-v lists each file)
```

Integrating the Local SMO into the Jenzabar CX product

These procedures are not automated and are typically performed by someone on the programming staff. As with all SMO development, all work should be done in the CARSDEV system.

Extract the Local SMO from tape

```
dev: cd smo
dev: copyin -v
```

Remove Local Customizations

Determine if any of the files brought back in the local SMO contain any local customizations that shouldn't be put into the standard product and remove them.

```
dev: cd smo/L02305
dev: vi Dist/...               (remove customizations)
```

Build a New Revtr File

The branch version numbers in the Revtr.dist file must be converted to trunk versions using the 'newver' script. This command converts each branch version number to the next higher trunk version as long as that trunk version doesn't already exist.

```
dev: cd smo/L02305
dev: mv Revtr.dist Revtr.dist-o
dev: newver < Revtr.dist-o > Revtr.dist
```

Resolve Version Number Overlaps

If the new Revtr.dist file still contains some branch version numbers, those files should be taken out of the Revtr.dist and incorporated into a SMO using the normal checkout/checkin method in a later step.

```
dev: cd smo/L02305
dev: vi Revtr.dist             (remove files with branches)
```

Create or Determine the Regular Jenzabar CX SMO to Use

Revisions from a local SMO must be incorporated into a regular Jenzabar CX SMO in order to become a part of the standard Jenzabar CX product. Determine which existing open SMO into

which the revisions should be incorporated, or create a new SMO. As an example, a new SMO (10900) will be created.

```
dev: perform smo                (add a new smo - note the SMO#)
dev: cd smo
dev: make adddir F=10900
>>Command: adddir
>10900 - makeinit smo
    Enter 1 line SMO Description:
Add 'db' alias for switching between databases
...thank you.
Updating '.makevar.mak'
dev:
```

Deposit the Local SMO as Part of the Regular SMO

The revisions in the local SMO should be deposited as part of the regular SMO and then added to its Revtr.new file. The SMO# in the log messages for the revisions should be changed from the local SMO# (L02305) to the regular SMO# (10900) by editing the individual .log files before doing the deposit.

```
dev: cd smo/L02305
dev: vi Dist/...                        (change SMO# in .log files)
dev: make smonewrev >& Newrev.out       (deposit revisions)
dev: cat Revtr.dpst >> ../10900/Revtr.new
```

Check for Any Makefile.lcl Files

If any Makefile.lcl files are created by the smonewrev, be sure that the Makefile is correct for the standard Jenzabar CX product and does not include unwanted customizations.

```
dev: cd smo/L02305
dev: grep Makefile Newrev.out           (check for Makefile.lcl)
dev: cd ...                             (go to the source directory)
dev: diff Makefile Makefile.lcl         (determine which to keep)
```

Address Files not Handled by Smonewrev

If errors occurred during the *smonewrev*, resolve those problems at this time. Also, any files that were excluded from the Revtr.dist (and thus from the *smonewrev*) should be taken care of at this time by checking out the individual files and putting the revisions in by hand.

```
dev: more Newrev.out                    (check for errors)
dev: diff Revtr.dist-o Revtr.dpst       (files not deposited)
dev: cd ...                     (go to files that were not deposited)
dev: make co F=<file>
dev: vi <file>                  (put in desired revisions)
dev: make ci install F=<file> SMO=10900 L="<message>"
(include revision in the regular SMO)
```

Update the README File for the Regular SMO

As when adding any revisions to a SMO, the README file for the regular SMO should be updated with any special instructions related to the revisions just added. The README file from the local SMO may have some of these instructions in it.

```
dev: cd smo/L02305
dev: more README                (check for special instructions)
dev: smoreadme 10900            (put changes into README)
```

Move the Local SMO to the ARCH Directory

When all of the revisions brought back in the local SMO have been incorporated into a regular SMO, the local SMO should be moved to the ARCH directory to show that it is no longer needed.

```
dev: cd smo
dev: arch L02305

arch: L02305

Moving L02305 to ARCH.
dev:
```

Troubleshooting SMO Installations

Introduction

Jenzabar has attempted to provide answers for common problems that can occur during the process to install a SMO. The problem situations and corresponding responses are organized in the phases for installing a SMO, including:

- Deposit steps
- Pre-Installation steps
- Installation steps

Deposit Step Issues

The following contains the situations and responses for problems that can arise during the deposit step of the SMO installation.

CAUTION: Do *not* perform processes as root (su) when some process is not working correctly. The need to operate as root is indicative of difficulties with permissions, which should be resolved instead of being forced. The README will tell you when to become root or super user.

Situation:
The message appears: “makeinit Error code 1 - not remade because of errors is found in the Deposit.out file”

Response:
This is not a problem, continue with SMO. The makeinit process issues this error because no files are found in the newly created directory.

Situation:
The message appears: "Delta number too low, must be greater than XXX" is found in the Deposit.out file.

Response:
This is most likely the result of a SMO being installed out of order. Perform the smoorder command to verify the installation order. Determine the cause of the error before continuing. Use the rlog command and diff target to deduce if the correct version of the file is present.

Situation:
Permission is denied due to .makelist error.

Response:
Reset the permissions on the .makelist to a mode of 660 and a group of carsctrl and perform a make remake.

Situation:
The message, "updating .makelist, remake not remade because of errors" is found in the Deposit.out file.

Response:
Change directories to where the error occurred, and perform a make remake on the directory.

Situation:
You perform an rlog on a file and the recent version is not the same version as the current version.

Response:
Check the file out (co target) and then uncheck it out (unco target).

Situation:
You cannot check a file out or in.

Response:
Check the permissions on the file and update them using fileperms.

Example: % **fileperms -u <file>**

Situation:
The message from a make makedep process appears: “Need to remake module list.”

Response:
Perform a make remake in the appropriate directory to recreate the file dependencies.

Situation:
The message appears: “Deposit Touch: cannot change times on RCS.”

Response:
Check the mode and ownership on the appropriate <file>.ci files in the RCS directory. The permissions should be 660 and with a group of carsctrl.

Situation:
The error message appears: "RCS directory is not writeable".

Response:
Check the permissions and execute a fileperms -u upon the RCS directory.

Note: The permissions should be: **drwxrwx---**

Situation:
The error message appears: "RCS file is not writeable".

Response:
Check the permissions and verify the mode is 660 for RCS/<file>,v file.

Situation:
The message "Subscript out of range" occurs when depositing new program source.

Response:
Verify the Revfile.c has not already been deposited. If it has been deposited and the other files within that source directory have not been, then delete the version of the Revfile.c and redeposit.

Situation:
When checking in screen files following merging local customizations, you get the error message: “field not found in database.“

Response:
Probably a new field has been added to a schema used by this screen, but has not yet been checked in and rebuilt. Perform these actions on the schema and then check in the screen.

Pre-Installation Step Issues

The following contains the situations and responses for problems that can arise during the pre-install steps of the SMO installation.

Situation:
When rebuilding a schema with NOMAKEDEF specified (e.g., exam, tprog, billing, charge, and assessment), the system returns a message indicating this.

Response:
This is informational only. Examine the <file>.sch in the schema directory to verify that the rebuild was successful.

Installation Step Issues

The following contains the situations and responses for problems that can arise during the install step of the SMO installation:

Situation:
When installing program source, undefined structure member or type mismatch errors may occur.

Response:
This is usually because a schema was not properly built or rebuilt. Verify the build or rebuild. Execute a make makedef F=<schema> upon the schema if the build/rebuild was okay and reinstall the program source.

Situation:
The message, "No installing for schemas" is found in the Install.out file.

Response:
This is an informational message only. Schemas do not get installed, continue.

Situation:
The message, "Don't know how to make def.c" is found in the Install.out file.

Response:
Perform a make cleanup in the source directory with the problem.

Situation:
The message, "m4:menudesc:7 can't open file" is found in a menudesc.err file.

Response:
Reinstall the menuopt file in question and then reinstall the menudesc file. Continue with the SMO installation.

Situation:
You receive a warning message that the source files are empty.

Response:
You can usually ignore this message. This is the result of files in src containing only comments for future development, or containing options that are selected with ifdef statements that have been commented out in the appropriate include file. If there is a question, call Jenzabar.

Situation:

You are reinstalling or installing src, and you receive the message "undefined symbols error".

Response:

Verify the objects were deposited correctly. If the SMO contained an include file, verify it was merged correctly, checked in and installed prior to the installation of the program source. If libraries were modified by the SMO, reinstall the libraries and then reinstall the program source.

Situation:

The message appears: "system error 13" or "system error 0".

Response:

These are both permissions errors. Investigate the mode and ownership on the appropriate files or directories. Use the fileperms utility to correct any permission issues.

Situation:

You are loading a program and receive a 6005 error.

Response:

The program thinks that a field in a schema it uses no longer exists. The safe solution is to perform a make makedef on the schema. Then reinstall the program in src. If the problem persists, verify the schema matches the actual data file by executing a make build F=<schema> and reviewing the <schema>.sch file.

The Revision Control System

Introduction

Jenzabar developed a specialized version of the Revision Control System (RCS) to provide you complete control over changes to CX. With RCS, an institution has the ability to:

- Keep backup copies of all versions of a file
- Track all changes to a file, including changes from Jenzabar versus local changes
- Extract an earlier version of a file

Note: CX is heavily dependent upon the *make* processor which in turn is heavily dependent upon RCS.

Backup Copies of Files

RCS saves a backup copy of each version of a file under $CARSPATH. These versions are saved under the RCS directory. Consequently, each file in a working directory has a corresponding file in the RCS directory with a ,v as a suffix.

Example: $CARSPATH/modules/regist/reports/faclist

$CARSPATH/modules/regist/reports/RCS/faclist,v

The *<file>,v* file contains all the versions that were checked into the file via the make processor. The system creates this file when you use the *make add F=<file>* command to add a new file to CX. The make processor updates the *<file>,v* file when the working file is checked in (ci target) or checked out (co target).

Reviewing Changes to Files

Use the *rlog* command to review all the changes made to a file. This command displays:

- Each version of the file
- A description of the modification
- The user who checked in the modification.

Example: % **rlog faclist**

You can also use the *log* make target to display the same output as the *rlog* command.

Example: % **make log F=faclist**

Reviewing File Header Information

To review only the header information about a file, pass the -h parameter to the *rlog* command.

Example: % **rlog -h faclist**

Below is a sample output of the rlog -h command.

```
RCS file:       RCS/faclist,v;   Working file:    faclist
head:           7.0
branch:
locks:          ;  strict
access list:
symbolic names: Recsave: 7.0;  Active: 7.0;  Current: 7.0;
                Recent: 7.0;
                comment leader:  "      "
                total revisions: 3;
```

Note: The lines beginning with head, locks, and symbolic names provide critical information about a file.

Note: The line beginning with head contains the head version number. This number is the trunk version of the last Jenzabar release. The locks line lists the user who owns the file if the file is in a checked out state. The symbolic names line contains four separate versions numbers, which are labeled as Recsave, Active, Current, and Recent.

File Version Numbers

The following explains the four version numbers in the symbolic names line of the *rlog -h* output.

Recsave
The last version of the file that was deleted.

Active
The version last installed (not tinstalled).

Current
The version of the file in the working directory.

Recent
The version number that was last checked in.

Parts of a Version Number

The version number, for a file with a local revision, has four parts in the format: n.nnnnnn.nnnn.nn.

- The first number denotes the release
- The second number denotes the CX version number of the file
- The third number is the client number assigned by Jenzabar
- The fourth number is the client version number

For example, the version number 8.100003.5000.2 stands for:

Trunk Version
8-- Release I
100003-- Version number 100003 for Jenzabar

Branch Version
5000-- Client code assigned by Jenzabar
2-- Version 2 for client

Displaying All Versions of a File

To display all the version numbers of a file, use the qrlog command and pass the -qr parameter. This will output all the version numbers stored for the specified file.

Example: % **qrlog -qr faclist**

The output will appears as follows:

```
7.100003.5000.2
7.100003.5000.1
7.100003
7.10000
7.500
```

Extracting an Earlier Version of a File

To extract an earlier version of a file from RCS, use the co make target and pass a V= argument to the target. This will check out the specified version and place it in the working directory.

Example: % **make co F=faclist V=7.100003.5000.2**

Extracting an earlier version of a file from RCS for viewing only can also be accomplished by using the co command with a -p parameter. This command will extract the specified version and output it to the terminal.

Note: Do not leave a space after the -p and before the version number.

Example: % co -p7.100003.5000.2 faclist

To extract the specified version and output it to a printer, pipe the above command to a printer as in the following example:

Example: % co -p7.100003.5000.2 faclist|lpr

SECTION 6 – DATABASE MANAGEMENT

Overview

Introduction

This section provides information and procedures for maintaining the CX database and information in the database. The following information is provided:

- Maintaining multiple databases
- Setting up an Audit Trail database
- Setting up Detail window select and sorting features
- Managing updates to addresses

System Management Menu

The System Management: Data Dictionary menu contains the following options to assist a database administrator:

Database Administrator
Accesses the Database Administrator (dbadmin) program, which allows you to add, update, or remove user logins and to audit database system files.

> **Note:** For more information, see *Common Programs* in the *CX System Reference Technical Manual*.

Informix Tables/Columns
Accesses the Systems Tables and Columns PERFORM screen on which you can view systable and syscolumn information.

Database Files
Accesses the CX Database Dictionary Files PERFORM screen. You can query, add, update, and remove files that describe the database dictionary.

Database Fields
Accesses the CX Database Dictionary Fields PERFORM screen. You can query, add, update, and remove files that describe the database dictionary fields.

Fields By File Report
Accesses the Database Fields report (*dbfield*), which lists the fields in the database by table. You can specify the beginning and ending of a alphabetical range of table names to be included in the report.

> **Note:** You can use wildcards to specify a range of table names. For example, to specify all tables names from a to m, specify a* and m* in the parameter screen for the report.

Files By Track Report
Accesses the Database Files report (*dbefile*), which lists the tables in the database by track. You can specify the beginning and ending of a alphabetical range of track names to be included in the report. The track values you can specify include:

- A (Admissions)
- C (Common)
- D (Development and Donor Accounting)
- F (Fiscal and Accounting)
- M (Management)
- S (Student)

Fields By Track Report

Accesses the Fields By Track report (*dbetrack*), which lists the tables and fields in the database for the tracks that you specify. The track values you can specify include:

- A (Admissions)
- C (Common)
- D (Development and Donor Accounting)
- F (Fiscal and Accounting)
- M (Management)
- S (Student)

The remaining options on the menu are used with the *mergeid* program. See the section, *Merge ID Program* in the *CX System Reference Technical Manual* for information about these options.

Maintaining Multiple Databases on One Computer

Introduction

These pages describe the current procedures to be used to maintain multiple CX databases and multiple CX releases on one computer.

Multiple Complete Jenzabar CX Releases

CX consists of everything under $CARSPATH. A CX release can further be qualified as either a Complete CX Release (includes source) or as an Operational CX Release (does not include source). Multiple CX releases can exist on one computer by having multiple $CARSPATH directory trees (e.g. /usr/carsf, /usr/carstrain,...).

Creating Another Release

Multiple releases are generally set up by creating a copy of an existing CX release under a different $CARSPATH tree. Symbolic links can be used for the parts of the directory tree that are to be shared with another CX release.

```
# cpdir /usr/carsf /usr/carstrain(create CARSTRAIN)
# cd /usr/carstrain
# rm -f install/sys/lib/prtab(link to real prtab)
# ln -s /usr/carsf/install/sys/lib/prtab install/sys/lib/prtab
# cd /usr/carstrain/spool
# rm -rf lpr(remove lpr spool directory)
# ln -s /usr/carsf/spool/lpr .(create link to lpr)
# ll lpr(show the link)
lrwxrwx---   1 jim    cis   20 Nov  3 08:09 lpr -> /usr/carsf/spool/lpr
```

Switching Between Releases

Use the *setdb* command to change between CX releases. This command basically changes the CARSV environment variable to the name of the destination release (e.g., carstrain) and starts up a new csh. Since the $CARSPATH environment variable gets defined in terms of CARSV and other variables (like $MENUPATH, $BINPATH, $DBPATH, parts of PATH, cdpath, ...) get defined in terms of $CARSPATH in the cshrc file, the environment for this new csh process will be set up for the destination release.

Printenv Command

To display your environment variable settings, enter the printenv command. The following is an example of the command's output.

```
% printenv
HOME=/usr/cisids/alec
PATH=/usr/carsdevi/install/cis:/usr/ucb:/bin:/usr/bin:/usr/local/bin:/usr/carsdevi/install/utl:/u
sr/carsdevi/install/bin:/opt/informix/bin:/usr/local/cisc:/usr/games:.:/usr/cis/wp/education/File
Cabinet/schedules:.:/usr/local/cisc
LOGNAME=alec
TERM=vt100
SHELL=/usr/bin/csh
MAIL=/var/mail/alec
COLUMNS=80
LINES=24
USER=alec
MANPATH=/usr/share/man/%L:/usr/share/man:/usr/contrib/man/%L:/usr/contrib/man:/usr/local/man/%L:/
usr/local/man:/opt/upgrade/share/man/ja_JP.eucJP:/opt/upgrade/share/man/ja_JP.SJIS:/opt/upgrade/s
hare/man:/opt/audio/share/man:/opt/blinklink/share/man:/opt/ansic/share/man/%L:/opt/ansic/share/m
an:/opt/langtools/share/man/%L:/opt/langtools/share/man:/opt/CC/share/man:/opt/image/share/man:/o
pt/imake/man
```

```
TZ=EST5EDT
CARSV=carsdevi
CARSPATH=/usr/carsdevi
CARSCPATH=/usr/carsbetai
INFORMIXDIR=/opt/informix
TBCONFIG=tbconf.cars
CARSOBJ=/usr/carsdevi/objects
CARSWSD=/usr/carsdevi
CARSRCS=/usr/carsdevi
DBPATH=:/usr/carsdevi/schema/common:/usr/carsdevi/install/frm/common:.:/usr/carsf/schema/common:
WPPATH=/usr/carsdevi/wp
TXTPATH=/usr/carsdevi/text
MENUPATH=/usr/carsdevi/install/mnu
TERMINFO=/usr/carsdevi/install/sys/terminfo
TERMCAP=/usr/carsdevi/install/sys/etc/termcap
CARSIQPATH=/usr/carsdevi/iq
IQDIR=/usr/iq
PAGER=pg
SCROUTPUT=/usr/cisids/alec/scroutput
UserSource=true
CARSNAME=              CARS College
CARSADDR=         Sharonville, OH  45241
CARSPRINTER=hplpr
CARSPRINTERS=hplpr,hp4write,hp4fnaid,hp4edadv,hp4mis,hp3mail,hp3admin,hp1sys,lpt
CARSSITE=CARS
CVTPATH=/usr/cvtdir
CARSDB=devi
SACEISOL=DIRTY READ
TERMDIR=/quad/usr/qlib/files
MENUDIR=/quad/usr/qlib/qimenus:/quad/usr/qlib/help:.
QSKILL=0
QTERM=wy75
CSERVHOST=saturn
CARSMNUSD=CARS Menu
ONCONFIG=onconf.hpdev
INFORMIXSERVER=carshpdev
CARSSYS=DEV
```

Establishing the Default Release

You initially define the default CX release for each user in the skel/cshrc file under the directory where home directories are located (usually /usr/carsids). The value that CARSV is set to, if not already set when a user logs in, is whatever CARSV was set to when the skel/cshrc file was last installed. So the following would change the global default release to CARSTRAIN.

```
% setdb train
train: cd skel
/usr/carstrain/skel
train: make reinstall F=cshrc.s
>>Command: reinstall
>cshrc.s - retranslate - reinstall.
train:
```

The default release for a particular user can be changed by editing the .cshrc in their home directory. The following at the top of a .cshrc file makes CARSTRAIN the default release. The assignment of CARSV to carstrain is only made if CARSV is not already defined so that setdb can be used to change its value.

```
#
# .cshrc - Standard commands to be executed are in
#/usr/carsids/skel/cshrc
#
if (! $?CARSV) setenv CARSV carstrain
source ~/../skel/cshrc
  ...
```

Software Maintenance

To keep multiple complete CX releases up to date, all SMOs must be installed in each release. Care should be taken when installing a SMO that affects directories outside of CARSPATH.

Multiple Jenzabar CX Databases

A CX database consists of the schema and data directories under the CARSPATH directory. This includes the schemas, the database dictionary (cars.dbd), and all of the database files. The currently active CX database is determined by which schema directory is in the DBPATH environment variable. Setting up an Operational CX Release (as described later) is generally more useful than another CX database because it includes other directories like CARSPATH/text and CARSPATH/spool/forms that should be associated with the CX database in normal operations.

Creating Another Database

Multiple CX databases are usually created by making a copy of the schema and data from an existing CX database. The following creates a CX TRAIN database from the CARSF database.

```
# mkdir /usr/carstrain
# chgrp common /usr/carstrain
# chmod 750 /usr/carstrain
# cd /usr/carsf
# cpdir schema /usr/carstrain/schema
# cpdir data /usr/carstrain/data
```

Switching Between Databases

Switching to another CX database requires changing the DBPATH environment variable so that the correct database dictionary can be located. This can be done using setenv or an alias can be set up to make this easier if desired.

```
% setenv DBPATH :/usr/carstrain/schema/common
```

-- OR --

```
% alias db setenv DBPATH :/usr/cars\\!^/schema/common
% db train
```

After changing the DBPATH environment variable, all CX database accesses will go to the new database until DBPATH is changed back. The section on Operational CX Releases describes a more complete setup which allows setdb to be used to change databases.

Software Maintenance

To keep multiple CX databases up to date, all parts of a SMO that affect schema or data files must be installed. This task is easier when using an operational CX release rather than just a CX database.

Multiple Operational Jenzabar CX Releases

An Operational CX Release consists of everything under CARSPATH required for normal CX operations, which basically excludes the source. Although technically not a part of the CX database, there are a number of directories used by CX application programs that should be kept separate with each CX database. These include audit, events, text, and vchpost under CARSPATH and forms, lps, and tape under CARSPATH/spool. Other directories are not as dependent on the database and can be shared with another CX release through symbolic links. These may include the install directory (or parts of it), the wp directory, and the spool directories for the printers.

Creating an Operational Release

The following illustrates how the necessary directories can be set up to create an operational release from an existing CX release.

```
(Create the operational directory structure)

# mkdir /usr/carstrain
# chgrp common /usr/carstrain
# chmod 750 /usr/carstrain
# cd /usr/carstrain
# mkdir system
# cp /usr/carsf/system/{Bootstrap,Config} system
# cd /usr/carstrain/system
# Bootstrap dirs
*** Config: hpux 2.1, Client=1, Machine=0, Database=0, Branch=Y,
SMO=N ***
>>>Bootstrap - Creating necessary directories

(Setup the link (or make a copy) for the install directory)

# cd /usr/carstrain
# rm -rf install(remove directory created by Bootstrap)

# ln -s /usr/carsf/install .(to share)
-- OR --
# cpdir /usr/carsf/install install(to keep separate)

(Setup the link (or make a copy) for the wp directory)

# ln -s /usr/carsf/wp .(to share)
-- OR --
# cpdir /usr/carsf/wp wp(to keep separate)

(Setup the links for the spooler directories)
(These should be links rather than copies for
proper sharing of the printers between the releases)

# cd /usr/carstrain
# rm -f install/sys/lib/prtab
# ln -s /usr/carsf/install/sys/lib/prtab install/sys/lib/prtab
# cd /usr/carstrain/spool
# ln -s /usr/carsf/spool/<printer> .(do this for each <printer>)

(Create the database)

# cd /usr/carstrain
# cpdir /usr/carsf/schema schema
# cpdir /usr/carsf/data data
```

Switching Between Releases

The same method used for switching between complete CX releases can be used to switch between operational releases. The setdb command will work for an operational release because it has an install directory (via a link) whereas a CX database by itself does not.

Software Maintenance

The procedures for software maintenance in operational releases depends upon which directories are links and which are separate copies. If parts (or all) of the install directory are not linked into a complete release, then those parts will need to be updated when a SMO is installed in the complete release. Do this by doing a make reinstall of the affected files from the source directory in the complete release while in a setdb to the operational release. If the whole install directory is linked to the complete release, then the operational release will automatically have any changes installed in the complete release.

As with multiple CX databases, any changes to schema or data in a SMO must be installed into the operational release. This may involve making a copy of the Revtr.dist file that only contains the schema files affected (e.g., Revtrsch.dist) and then doing a make smonewrev Revtr=Revtrsch while in a setdb to the operational release. Any rebuilds or database loads would also have to be done.

Setting Up an Audit Trail Database

Introduction

This section describes how to set up the audit trail database used to track changes to tables, such as:

- The date and time that the change occurred
- The login name of the person who made the change
- A flag to indicate the type of change
- The specified database columns that capture the values of specific columns

Separate Database

To enable the audit trail feature, you must create a separate database using ISQL. The audit trail database can operate under your current INFORMIX database server or under any other INFORMIX server.

If your CX database has transaction logging enabled, your audit database must also have transaction logging enabled. If you have a separate database server for the audit database, you may be able to direct your transaction logging to /dev/null.

Default Database Name

You can determine the default name of the audit trail database by appending *_audit* onto the value of the CARSDB variable. For example, if you set CARSDB to *cars*, the default name of the audit trail database becomes *cars_audit*. You can override the default name for the audit trail database by defining the CARSAUDITDB variable. For example, if you set CARSAUDITDB to *data_changes*, the name of the audit trail database becomes *data_changes* and is independent of the value of CARSDB.

Audit Database Macro

You must add the following macro to the $CARSPATH/macros/custom/configure file:

CARS_DB_AUDIT

Set the macro to the new audit database name using the format: *//dbservername/dbname* or *dbname@dbservername*

Example: m4_define (`CARS_DB_AUDIT', `cars_audit@carsinformix')

Note: You must use a back quote (`) beginning the parameters and a regular quote at the end.

Building Schemas

After you create the audit trail database, you must build schemas. Enter the following commands to build the appropriate schemas; note that you must be logged in as root to execute the make build command:

Note: The following uses *carsi_audit* for the audit database name.

```
% setenv CARSDB carsi_audit
% cd schema/common
% make build F="syscolperm systabperm dbfile dbfield"
% setenv CARSDB carsi
```

Adding Audit Trails to Schemas

Do the following for those schemas that you want an audit trail.

1. Add the following to the end of the schema:
   ```
   trigger
     audit (*) grant select to (group ....)
   ```
2. Build the schema.

The system creates the schema in the audit database and creates three triggers for inserting, updating, and deleting records with the following additional fields:

- audit_timestamp
- audit_username
- audit_event

Audit Table Creation

The system creates audit tables in the default dbspace of the audit database. When initially building a schema that specifies an audit trigger, the system builds two tables with the same name. The system builds:

- The data table in the database specified by CARSDB.
- The audit table in the audit trail database.

Note: If you want to maintain an audit trail on a different server than the database, you must use the optional clause IN <audit-server-name>.

CAUTION: You must create the audit trail database and build the database's system and tables before you build schemas with audit triggers. If the specified audit trail database does not exist, your build will continue with warnings that no audit trail will be maintained. If you wish to maintain the specified audit trail, you must create the audit trail database and rebuild the schema(s) that contains the audit trigger(s).

Unnecessary Audit Trails

You can maintain two or more databases (e.g., a live database and a training database) with one set of schemas. However, you can avoid wasting disk space with an unnecessary audit trail by not creating audit trails for one of the databases.

For example, you use the same set of schemas to maintain a live database called *cars* and a training database called *train*. To avoid maintaining an audit trail on the training database:

- Create an audit trail database called *cars_audit* for the live database.
- Do not create a corresponding audit trail database (e.g., *train_audit*) for the training database.

Note: If you do not want an audit trail for certain tables within a database, before you build the tables, you can use the CARSAUDITDB variable to point those tables to a non-existent database.

Setting Up Office Permissions Checking in CX Applications

Introduction

These pages describe how to set up the office permissions checking feature in entry programs. This feature applies to programs that use records containing the ofc_add_by column. Programs can use this column to determine a user's insert, update, and delete permissions, including for holds, based on the user's office. The ofc_add_by column links the program to the Office Permissions table (ofcperm_table), which contains the office permission codes.

Note: For more information on the Office Permissions table, see Common Tables and Records in the *CX System Reference Technical Manual*.

Procedure

Do the following to set a program to perform permissions checking on a record with an ofc_add_by field.

1. Check out the program's files.
   % **make co <program files>**
2. Edit the def.c file.
   % **vi def.c**
3. In the filename[ ] array, edit the entry that references the table for which you are adding permissions checking.

   Example: { "id_rec", "ID", NULL, ENT_LOCK},

   To add permissions checking, add the flag ENT_VALUEPERM to the table's entry.

   Example: { "id_rec", "ID", NULL, ENT_LOCK|ENT_VALUEPERM},
4. In the addfld[ ] array, add an entry that fills in the ofc_add_by column with the ofc_add_by parameter passed to the program.

   Example: { "id_rec", "ofc_add_by", NULL, "ofc_added_by", PROG_BUFFER},
5. Exit from the file editor and save the def.c file (e.g., **wq:**).
6. Check in and install the program.
   % **make cii L="Add permissions checking to id_rec." <program files>**

Setting Up Select And Sort Detail Window Features

Introduction

The Library Entry programs have a feature that allows users to define the select and sort capabilities in an Entry Program detail window. A detail window with the sort feature contains the Sort command. The system links the use of the select and sort capabilities to the Permission table to deny access to table entries at user or group permission levels. CX includes the feature in all Library Entry programs, which may be modified by the institution, based on their requirements.

The system uses two functions to invoke this feature at the source code level: ENT_SCGET and ENT_SCSTART. These functions are located in the $CARSPATH/src path for each entry program. The Library Entry programs allow a user to selectively choose the type of data for review in detail windows.

For example, Library Entry programs can review multiple tickler codes, providing access to a combination of Contact records. This is done while maintaining the security of limiting the user to adding a predefined tickler, based on a series of contacts.

The Setup Process

The following describes the overall process involved in setting up the select and sort detail window features.

1. Set up the permissions macro in the file $CARSPATH/macros/user/common.
2. Set up the Permission table.
3. Set up the Entry Selection/Sort Criteria table.

Setting the Permissions Macro

To allow users to sort and select records in detail windows within entry programs, set up the permissions macro in the $CARSPATH/macros/user/common file. The value of this macro is stored in the Permissions table (perm_table) in the perm_table.ctgry field.

Follow these steps to set up the macro.

1. Enter vi common and edit the Common file.
2. Find the ENTRY_PERM_CODE macro.
3. Set this macro to the value your institution desires. The default value is ENTPERM.
4. Reinstall the macro file, then reinstall the include and source files.

Permission Table

Set up the Permission table (perm_table) to allow the users you specify to access sort and selection criteria that correspond to the ENTRY_PERM_CODE macro value.

Follow these steps to set up the Permission table.

1. Obtain a copy of the password file containing the UNIX User ID (UID) numbers of CX users.
2. Access the perm_table PERFORM screen from the Table Maintenance: Common (P-Z) menu.

3. Enter a value in the Category field, equal to the ENTRY_PERM_CODE macro value, for each UNIX group or UID number that should have permission to the sort and selection categories that you specify.
4. Specify the names of the sort and selection groups by entering their names in the Permission Code field. The names of these groups are arbitrary. The system uses the names to identify groups in the Entry Selection and Sort Criteria tables.
5. Print a copy of the table for use in creating the Entry Selection and Sort Criteria tables.

Entry Selection/Sort Criteria Tables

The Entry Selection table (entsel_table) defines the name and the database record for the sort selection. The indicated database record corresponds directly with any detail window that accesses that database record.

The Sort Criteria table (entselcrit_table) establishes how the system selects and/or sorts data in a detail window linked to the database record in the Table Name field of the entsel_table.

Define the selection and sorting criteria available for entry programs by entering data into the Entry Selection and Sort Criteria tables. To set up these tables, do the following:

1. Access the Entry Selection/Sort Criteria table (entsel_table, entselcrit_table) PERFORM screen from the Table Maintenance: Common (D-F) menu.
2. Complete the fields in both sections of the table.

Note: The Entry Selection table is the master table and the Sort Criteria table is the detail table.

Entry Selection Table Fields

The following list describes the fields in the Entry Selection table.

Default
Indicates whether or not this table entry is considered the default information. Valid values are Y and N. Enter only one Y per filename.

Note: Each entry requires one default of Y.

Description
The description of the name of the select or sort action.

Entry Selection Code
The name of the select or sort action, as determined by your institution.

Permission Code
The code (as defined in the perm_table) that a user must be a member of in the perm_table to access and use this sort.

Program Name
The name of the entry program using this select or sort.

Note: A blank in this field provides this entry to all entry programs.

Table Name
The name of the table from which the select or sort occurs (it also corresponds to the detail window as defined in the def.c file of the entry program).

Sort Criteria Table Fields

The following list describes the fields in the Sort Criteria table.

Boolean Condition
Indicates whether the select or sort is an AND or an OR condition. The system uses this feature only if there is more than one sort criteria in the sort.

Note: The program assumes the parameters have an “and” condition.

Column Name
The name of the field in the Table Name record that the select or sort criteria acts upon.

Column Value
The value that the indicated field in Column Name contains for the select or sort.

Descending
Indicates whether or not the values in the Column Value field should be sorted in descending order (Z to A). Enter Y for descending or N for ascending (A to Z).

Relational Operator
The logical condition (equal, greater than, less than, etc.) that the select or sort uses to determine relationship between the name of the field and the value that it contains (as specified in the Column Value field).

Sort Order
This indicates where this column should fall in precedence of the select or sort, if multiple column names exist in this Entry Selection Code sort (0 is the greatest).

Fields Controlling the Select and Sort Criteria

The select criteria is controlled by these fields:

- Boolean Condition
- Column Name
- Common Value
- Relational Operator

The sort order is controlled by these fields:

- Descending
- Sort Order

Selecting and Sorting in Entry Programs

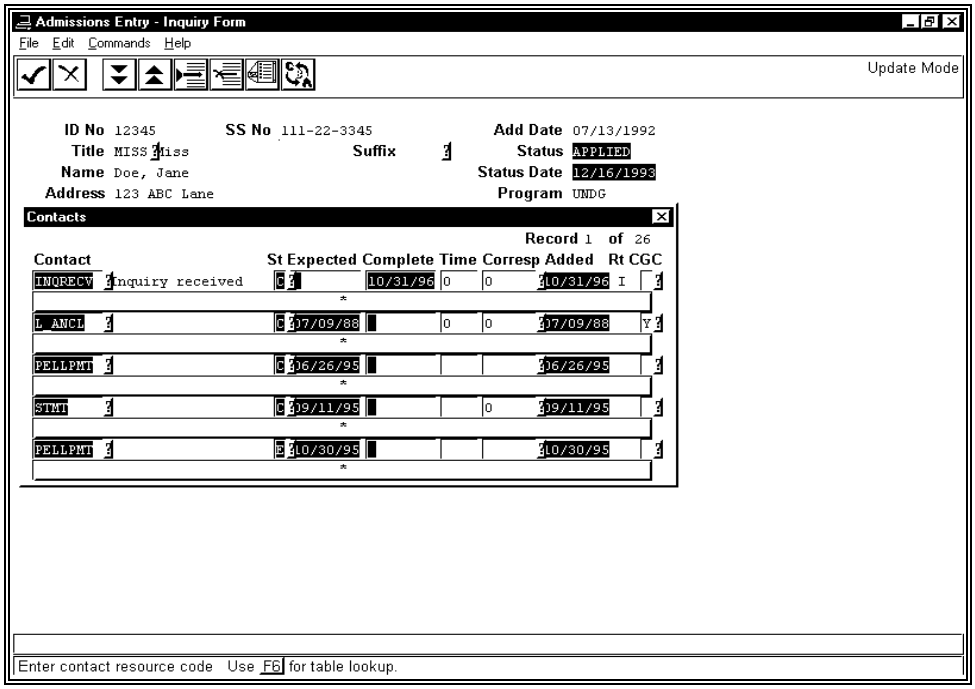
Introduction

This section provides a screen example and procedure to use when your institution sets up the select and sort screen features in CX.

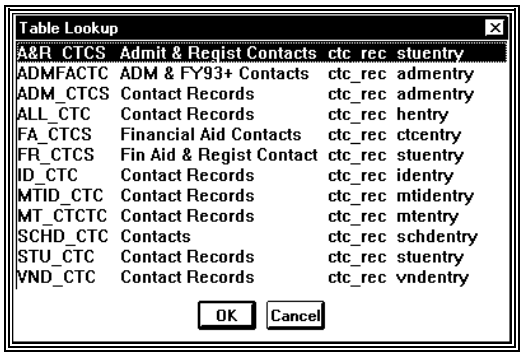
Note: The system sends no electronic mail as a result of any processing with these features.

Example Screens

The following is an example of a detail window, from which you can use the Sort command to access the select and sort detail window features.



The following example shows a pop-up window that appears when you select the Sort command. You can select a particular selection and sorting method.



How to Use Selecting and Sorting in Entry Programs

The following example lists and describes the steps to use the select and sort features in the Admissions Entry application.

1. Access the CX menu system and select Recruiting/Admissions. The Recruiting/Admissions: Main Menu appears.
2. Select Admissions Processing. The Recruiting/Admissions: Admissions Processing menu appears.
3. Select Admissions Entry, then select **Finish**. The Admission Entry menu appears.
4. Select **Application**. The Application Entry screen appears in query mode.
5. Perform a query for a sample student. Do the following:
   - Enter a student ID number.
   - Select **Finish**.

   The student's record appears on the screen.
6. Select **Scroll**. The Detail Windows window appears.
7. Select a Detail Window (e.g., contacts). The Contact detail window appears.
8. Select **Sort**. A table lookup pop-up window appears.
9. Select the desired method. The pop-up window disappears and the Contact detail window appears with the information resulting from the selected method.
10. After viewing the information, select **Finish**. You return to the Application Entry screen.

Setting Up the Automatic Address Update Feature

Introduction

CX Entry Library applications enable you to automatically update addresses for every member of a relationship when only one member's address changes. All CX entry programs (e.g., identry, stuentry, admentry, etc.) allow access to this feature.

Example: If your institution maintains a relationship for a husband and a wife, and you change the husband's address, CX Entry Library applications prompt you to automatically update the wife's address.

Note: To update addresses using the ID Entry program, see *Updating Addresses in Relationship Records.*

What Fields Do the Entry Library Applications Update?

The standard CX Entry Library applications automatically update the following fields in the ID record for each member of a relationship linked with Relationship and Secondary Relationship records:

- Street address lines 1 and 2
- City
- State
- Zip Code
- Country
- Telephone number and extension

Note: To automatically update other fields in the ID record, contact your CX account manager.

What Macros Require Setting Up?

You must set up two macros and modify the contents of CX tables and records to allow Library Entry applications to automatically update addresses for individuals in relationships.

The two macros that you must set up (ENABLE_FEAT_PREV_PHONE and AA_PREV_MAINT_CODE) accomplish the following:

- Assist Entry Library applications to create a previous Alternate Address record (aa_rec) for an individual when you use the entry applications to change an individual's address.
- Indicate if the entry applications should create an Alternate Address record if you change only an individual's telephone number (id_rec.phone) or telephone extension number (id_rec.phone_ext).

The tables and records you must set up accomplish the following:

- Define the primary and secondary relationships
- Cause Entry Library applications to prompt you to automatically update addresses for related ID numbers
- Indicate the relationships for which Entry Library applications automatically update addresses when an individual's address changes

How to Set Up the Macros

The following lists the steps to set up the macros for the automatic address update feature.

1. Access the following CX directory: $CARSPATH/macros/custom
2. Enter **vi common** to edit the macro file. The contents of the common file appear.
3. Locate the ENABLE_FEAT_PREV_PHONE macro in the common file.
4. Do you want Entry Library applications to create a previous aa_rec when only the individual's telephone number or telephone extension number changes?
   - If yes, define the macro as **Y**. You instruct CX to create a previous aa_rec whenever an address, telephone number or telephone extension number changes.
   - If no, define the macro as **N**. You instruct CX to create a previous aa_rec only when an address changes
5. Do you want to use CX default Alternate Address Maintenance code, PREV?
   - If yes, go to step 6.
   - If no, do the following:
6. Locate the macro AA_PREV_MAINT_CODE.
7. Define the macro with any four-character value. You define the previous version of the alternate address according to your institution's needs
8. Save and exit the file. You save the changes you have made.
9. Reinstall the file.

Installing Your Changes

After you define and reinstall the macros, you must reinstall all the files in the following directories:

- $CARSPATH/include
- $CARSPATH/src

How to Save Previous Addresses in the Alternate Address Record

You can set up the Alternate Address table (aa_table) to cause the Entry Library applications to create an Alternate Address record (aa_rec) for each individual in a relationship whose address changes. This feature allows your institution to retain a previous version of each individual's address in the Alternate Address record.

Note: The Maintenance field (aa_table.maint) in the Alternate Address table contains a Yes (Y) or No (N) value that controls whether the Entry Library applications create an Alternate Address record for an individual.

1. When you change an individual's ID record address using an Entry Library application, the application looks at the Alternate Address code (id_rec.aa) in the individual's ID record.
2. If the corresponding Alternate Address code in the Alternate Address table contains a Maintenance code of Y, the entry application creates an Alternate Address record that contains the individual's previous address.

   The Alternate Address code PERM is often the only code that should have its corresponding Maintenance field set to Y. Consult with the appropriate personnel at your institution to determine how you should define Alternate Address codes.

   Notes:
   - The Alternate Address code (aa_rec.aa) in previous Alternate address records contains the value of the AA_PREV_MAINT_CODE macro.
   - The setup of the relationship records and tables in the CX database determines whether or not the Entry Library applications create previous Alternate Address records for both primary and secondary individuals in a relationship.
   - If an individual's address is changed twice in one day, the Entry Library applications create only one previous Alternate Address record for that day, and it only stores the last changed address.

Example of Creating an Alternate Address

Following is an example of how the Entry Library applications create an Alternate Address record.

Example: If the Maintenance field for the PERM Alternate Address code is set to Y in the Alternate Address table, the Entry Library applications create a previous Alternate Address record for an individual if the individual's address changes and the individual's ID record contains the value PERM in the Alternate Address field (id_rec.aa).

How to Set Up the Alternate Address Table

The following lists the steps to retain a previous version of an individual's address in the Alternate Address record.

1. Access the Alternate Address table PERFORM screen, using the Table Maintenance menu option on any of the major module menus.
2. Select **Query** to locate all the table entries. The first Alternate Address table entry appears.
3. Select **Update**. The command line changes to Update mode, and the cursor moves to the first field on the screen.
4. Do you want to maintain a previous address for the code that appears in the table entry?
   - If yes, enter **Y** in the Maintenance field.
   - If no, enter **N** in the Maintenance field.
5. Select **<Esc>** to change the information. The command line changes to enable you to perform other tasks.

6. Select **Next**.
7. Does the message, "There are no more rows in the direction you are going" appear on the comment line?
   - If yes, go to step 8.
   - If no, repeat steps 3-6 until you have updated all the table entries.
8. Do the following:
   - Select **Add**.
   - Add an entry in the Alternate Address table for the name of the code you defined in the AA_PREV_MAINT_CODE macro.

   Note:
   - Enter only the code and the code description in the Alternate Address table PERFORM screen, and leave the other fields blank.
   - For more information about defining macros, refer to *How to Set Up the Macros* in this section.
9. You update the Alternate Address table to include your institution's code for previous addresses.
10. Select **Exit**. You return to the CX table maintenance menu from which you started.

Note: You can set up an alternate address for e-mail addresses. For more information, see the *Communications Management User Guide*.

How to Set Up the Relationship Tables and Records

When you change address information for an individual, the Entry Library applications attempt to change address information for any individuals who are linked to that individual by a relationship.

Example: If John and Jane Doe are linked by a husband/wife relationship, and you change John's address, then the Entry Library applications attempt to change Jane's address automatically.

The Entry Library applications automatically update addresses for more than one individual in a relationship only if the relationship between the individuals is maintained. Whether a relationship is maintained, is dependent upon how you set up the Relationship record (relation_rec), the Secondary Relationship record (relsec_rec), and the Relationship table (rel_table).

The relation_rec, the relsec_rec, and the rel_table each contain an address maintenance field called Maint. This field indicates whether or not the Entry Library applications should automatically update addresses for individuals in the corresponding relationship. You must define this field with a Yes (Y) or No (N) value in each record and table.

How to Complete the Relationship Tables and Records

The following describes how to complete the Relationship table, the Relationship Record, and the Secondary Relationship record to maintain address information automatically for relationships.

Relationship Table

1. Access the Relationship Table PERFORM screen, using the Table Maintenance menu option under any of the module areas on the CX menu. The Relationship Table PERFORM screen is located in the Common Tables area.
2. Enter **Y** in the Maintenance field for each type of relationship for which you want the Entry Library applications to automatically update addresses (e.g., enter Y for the Husband/Wife relationship, but enter N for the School/Counselor relationship).

Note: Addresses are automatically updated only if the Mnt field for both the Relationship record and the Relationship table is set to Y. For example, if the Husband/Wife relationship has a rel_table.maint value of Y, but the relation_rec.maint value of a specific husband/wife relationship in the relation_rec is N, then the Entry Library applications will not automatically update the wife's address if the husband's address changes.

Relationship Record

1. Access the First Relationship detail window for each primary individual in a relationship.
2. Enter **Y** in the Mnt field for each individual for whom you want the Entry Library applications to automatically update address information (e.g., enter Y for the Relationship record that links John Doe to his wife, Jane).

   Note: The default value for the Mnt field is Y. Make sure that the Code field on the Relationship record (e.g., HW for Husband/Wife) appears in the Relationship table with a Y in the Maintenance field.

Secondary Relationship Record

1. Access the Secondary Relationship detail window for each primary individual in a secondary relationship.
2. Enter **Y** in the Mnt field for each individual for whom you want the Entry Library applications to automatically update address information when the primary individual's address changes (e.g., enter Y for the Husband/Wife relationship, but enter N for the School/Counselor relationship).

   Note: The default value for the Mnt field is Y.

Updating Addresses in Relationship Records

Introduction

After you set up the tables, records, and macros to automatically update addresses for relationships, the Maintain Relationship window prompts you to authorize automatic address updates for each individual in a maintained relationship.

How to Update Records Automatically

To automatically update each individual's address in a maintained relationship:

1. Access any CX entry program screen that contains address information. The entry program screen appears in Query mode.
2. Perform a query to locate the ID number of the individual for whom you want to enter a change of address. The individual's name and address information appears on the screen, and the screen enters Update mode.
3. Do the following:
   - Enter the updated address information.
   - Select **Finish**.

   The Maintain Relationship window appears.
4. Select one of the following responses to the prompt in the Maintain Relationship window:
   - *Discontinue* (Changes the Mnt fields on the Secondary Relationship and Relationship records to N, and suppresses the display of the Maintain Relationship window in the future.)
   - *No* (Changes the address for the selected ID, and does not change the address for the related ID.)
   - *Yes* (Changes the address for both the selected ID and the related ID.)

   The entry screen appears in Query mode.
5. Repeat step 4 until you have completed each Maintain Relationship window.
6. Do you want to update more ID information?
   - If yes, repeat steps 2-5.
   - If no, then go to step 7.
7. Select **Cancel**, then select **Exit**. The CX menu from which you accessed the entry program appears.

Discontinued Relationships

When you select the Discontinue command from the Maintain Relationship window, the Entry Library applications change Mnt field from Y to N. Later, if you select the ID for which the relationship exists, and then change the address, the system will not display the Maintain Relationship window, and it will not update the address for the related ID.

Reinstating a Discontinued Relationship

To reinstate the maintenance of a relationship that you have discontinued with the Discontinue command in the Maintain Relationship window, access the First Relationship detail window or the Secondary Relationship detail window for each primary ID, and then change each corresponding Mnt field to **Y**.

Saving Multiple Names and Social Security Numbers

Introduction

You must set up three table values in the Configuration table to allow Library Entry programs to save previous or alternate names and social security numbers when they are changed. CX stores multiple names and social security numbers in the addressee record (addree_rec) and uses this information when performing name lookup. For information about entering and viewing alternate names and social security numbers, see the *Other Name Detail Window* in the screens section of the *Getting Started User Guide*.

Setting Up the Configuration Table

Follow these steps to set up the table values in the Configuration table to control how previous names and social security numbers are processed by Library Entry programs.

1. Access Common tables and select Configuration. You can do this using the System Maintenance option or the Utilities option on the main menu.
2. Set the ENT_ADDREE_INTERACTIVE value to Y or N indicating whether you want to activate a prompt to the user asking whether a previous name or social number should be saved in the addressee record. The default is Y which will display the prompt whenever a name or social security number is changed. The response will apply only to that name or social security number.

   If you set this value to N, previous names and social security numbers will be saved or not depending on the values in ENT_ADDREE_NAME and ENT_ADDREE_SSNO.
3. Set the ENT_ADDREE_NAME value to Y or N indicating whether you want to save previous names in the addressee record. The default is Y.
4. Set the ENT_ADDREE_SSNO value to Y or N indicating whether you want to save previous social security numbers in the addressee record. The default is Y.

Privacy Act Highlighting of Confidential Information

Introduction

The main entry screens of Entry Library programs, including Admissions Entry and Student Entry, provide the ability to highlight groups of fields. Using the Privacy field, you can specify a code to indicate a field or group of fields containing information that the student does not want released. These fields are highlighted on the screen depending on the capabilities of the terminal or PC displaying the screen.

You must modify three tables to set up the feature for highlighting confidential statuses in entry program screens. You must also ensure that each user's terminal screen is properly set up to display highlighted fields.

Privacy Table

The Privacy table (priv_table) contains a code and text description of the privacy style that you mark as "private" on screens in entry programs. Since the names of these styles (e.g., ADDR for Address information) is arbitrary, you can define them any way you prefer.

You can access the Privacy table from the Privacy Act menu option on any Table Maintenance menu located under Common Tables.

Privacy Field Table

The Privacy Field table (privfld_table) contains the database records and fields that are located in the groups from the Privacy table. Each group can contain as many records and fields as you want, but only records and fields that are accessible in entry programs are highlighted.

For example, the privacy group named ADDR may contain the addr_line1, addr_line2, and city fields from the ID record (id_rec). The system highlights these fields if any of the fields are on an entry screen for a student whose Profile Record's Privacy Code field (profile_rec.priv_code) is equal to ADDR.

Profile Record

The Privacy Code field (profile_rec.priv_code) in the Profile record (profile_rec) contains the name of a group of database fields (as defined in the Privacy and Privacy Field tables).

For example, if the Privacy Code field for a student contains the ADDR code, then any fields that are defined in this group in the Privacy Field table appear highlighted on the table. You can access this Privacy Code field on selected forms that contain Profile record information (i.e., at least the Privacy Code field). To turn off the field highlighting feature, you must blank out the code in the Privacy Code field.

Privacy Act Report

The Privacy Act report is an ACE report that corresponds to the Privacy and Privacy Field tables. The report is located on CX in two locations:

- Table Maintenance: Modules (A-L), Common (P-S) menu
- $CARSPATH/modules/common/reports/tpriv

Privacy Field

To highlight the confidential status fields for a student's records on an entry program screen, a name representing the group of fields must exist in a Table lookup for the Privacy field. You can only access the Privacy field in certain screens.

You can only assign one privacy code to a student. If you want to highlight more fields than the existing privacy code(s) allows, your computer center must define another code that contains all the fields that you want to highlight.

How to Highlight Confidential Statuses

The following example describes the steps to highlight confidential status fields for a student in the Student Entry screen.

1. Access the CX menu and select Student Management. The Student Management: Main Menu appears.
2. Select Registrar. The Student Management: Registrar Main Menu appears.
3. Select Data Entry, then select **Finish**. The Student Data Entry menu appears.
4. Select Students. The Student Data Entry screen appears in query mode.

   Note: This screen contains the Privacy field.

   Note: Other screens can contain highlighted fields, even if the Privacy field is not present.
5. Perform a query for a student. Do the following:
   - Enter a student ID number.
   - Select **Finish**.
6. The student's record appears on the screen.
7. Move the cursor to the Privacy field.
8. Select the Table lookup command. A lookup window appears.

   Note: The lookup window contains the names of the groups of fields that you can highlight on entry program screens.
9. Do you want to highlight a group of fields listed in the lookup window?
   - If yes, select a group by pressing the letter before the group name (e.g., **a**). You cause the program to highlight the fields on the entry program screen.
   - If no, select **Cancel** and press the Space Bar to leave the Privacy field blank. You do not use the confidential status feature.
10. Select **Finish**. You save the student's record with the update to the Privacy field.
11. Do you want to see the highlighted fields on the entry program screen for the student?
    - If yes, perform a second query on the student. The entry screen appears with the student's information, and the selected fields are highlighted.
    - If no, do one of the following:
      - Query on another student to process. You are ready to perform other tasks in Student Entry.
12. Select **Cancel**. You return to the Student Data Entry menu.

SECTION 7 - MAINTAINING SECURITY WITH PERMISSIONS

Overview

Introduction

Permissions are defined on several different levels in Jenzabar CX. When connecting to the CX host system from a remote PC via a network connection, the **Network** permissions based on the user's network login and remote PC's network IP address can control the access to the CX host system. If permission has been granted to allow login to the CX machine, the next level of permissions checking is the **File** permissions based on the end user's CX (UNIX) login. That will define the user's access to the directory structure and files on the CX host's operating system. Also associated with the end user's CX login is the **Database Connectivity** permission, which defines the permission to connect and make changes to the Informix database structure, and the **Database Table/Field** permissions, which control the user's access to view, update, add or remove records from the defined database tables. Commands executed when an end-user logs in control **Program** level permissions from the end user's base menu entry position that defines what options the user can access from the menu. Finally, **Data** level permissions control access to defined rows of data based on the user's primary group and/or user UNIX ID (gid/uid) number. At this level restrictions can be based on types of data in each row of a table.

Table of Permissions and Controls

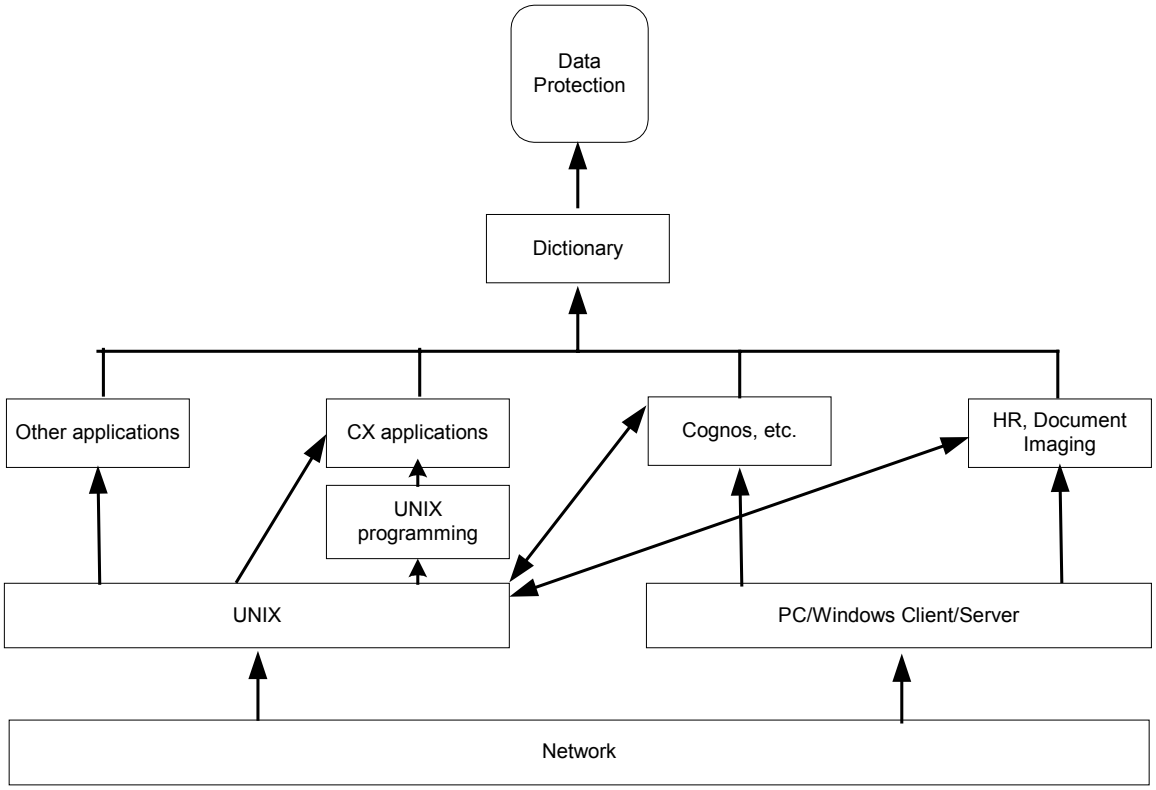
The following table shows the various types of permission levels (from the largest network to the smallest unit of data), the method by which permission is granted, and the type of control exerted by the level.

Note that the login is a key component to establishing security on your campus. For more information about creating and deleting logins, see *Creating and Deleting User Accounts* in the System Administration section of this manual.

	Level	Based on...	Permissions control...
Operating System	Network	Network login	• Connectivity to CX host machine • IP address restrictions from host/ network • Defined on host machine/remote machine basis
	File	CX (UNIX) login	• Access to files/directories on UNIX machine • Read/Write/Execute permissions • Public (Common)/Group/User access levels • Defined on individual file/directory basis
Database	DB Connectivity	CX login	• Level of control of Informix database • Connect/DBA levels • Defined on individual database basis
	DB Table/Field	CX login	• Level of access to Informix tables • Select/Update/Delete/Insert permissions • Public (Common)/Group/User access levels • Defined on per table/field basis • Controlled through schema files
Applications	Program	CX login	• Access to programs based on entry position in menu • Menu options can also have passwords (per option) • Controlled by menu command executed in .login file • Defined per user
	Data	CX login	• Define access to different types of data in same record • Used in C program (i.e., admentry, budget access) • Controlled by table entries in Informix database • Associated with group/user's UNIX ID (gid/uid) number

Diagram

The following diagram shows how the various layers of interfaces and operating systems work together to protect your data.



Description of Diagram

The following components make up the infrastructure that protects the data maintained on your CX database. Additional information about some of these components follows in this section. For information about the other components (e.g., Windows, Network, or CX or other applications), see the documentation that accompanies those components.

Network

On most campuses, this is the first access to the computer system. All network users have login names and passwords that are validated before access to any program or computer is allowed.

UNIX

The UNIX operating system maintains password files and user names to control access to CX as well as other UNIX-based applications.

PC/Windows Client/Server

The Windows operating environment requires a password for access to programs and data. In some installations, this step is bypassed because the network login provides adequate security.

UNIX programming

The UNIX program *fileperms* sets permissions for all CX files. Users with read permission can view data; users with write permission can add or update data; users with execute permission can run programs which in turn may update or display data.
For more information about *fileperms*, see the appendix to this manual.

Other applications

Other applications in this context refer to UNIX programs that interface with CX (e.g., Schedule 25). These programs rely on the same passwords and user names that provide access to CX applications.

CX applications

CX applications include all the CX programs and libraries. They are dependent on UNIX passwords and user names to control access to programs. In addition, many of the CX applications also have their own permission tables (e.g., glperms and regperms) that tailor the access of specific data to specific users. For more information about setting up office permissions checking and general use of the Permission table, see *Setting Up Office Permissions Checking in CX Applications* and *Setting Up Select And Sort Detail Window Features* in this manual.

Cognos, etc.

Cognos, as a third-party supplier of PC and internet software for use with CX, uses the UNIX passwords for validation purposes.

HR, Document Imaging

Human Resources and Document Imaging are two PC-based CX software applications. Because they are part of the CX suite of products, they use the same passwords and user names as other CX applications. An additional layer of security exists for these programs in that they must be loaded onto the individual PC before an authorized user can access them and add or view data.

Dictionary

The final line of defense in the protection of your data is the data dictionary. Created by the Informix make build process, the dictionary includes groups' and users' permissions to data at the schema level. Stored procedures, also a feature of Informix, can control data access as well. Regardless of the path taken from the network to the data, all users must satisfy the criteria of the data dictionary in order to access the database.

UNIX Groups and Permissions

Introduction

CX login groups provide the three levels of necessary permissions. A person in one group must be in all the previous groups to really have all the capabilities he needs. Other groups (i.e., staff, sys, bin, etc.) coexist as well, but the following groups are to be utilized within CX directory trees.

Home Directory Permissions

Ideally, user home directories permissions should be 700 drwx to provide the greatest security to a personal area. However, since many programs run as *carsu* and need to send output to the home directory, the user *carsu* also needs access to the user's home directory. Therefore, set up the home directories with 770 and a group in which *carsu* is a member. You can then safely remove the privileges for others on the home directories.

Common Jenzabar CX Groups

The following are the common CX user groups.

Note: Jenzabar personnel on client sites will be in all three groups: common, carsctrl, and carsprog. Computer center personnel should be put into carsctrl. No one else (except maybe the coordinator and limited staff) should be in the carsprog group.

common
All users of CX must be in the *common* group. The common group allows the execution of the menu processor, the running of reports and programs. You add all valid users to the common group because the group on the $CARSPATH directory is common and allows no permissions for others. The common group prevents non-CX users from accessing the $CARSPATH portion of the disk.

carsctrl
The carsctrl group consists of users that need access to portions of $CARSPATH/modules, such as reports, screens, etc. Use this group for computer center personnel that may not be fully involved with support of CX, but would create reports and screens occasionally.

carsprog
The carsprog group consists of trusted CX users. This group allows almost unlimited access and update capability to all aspects of CX, including program source, database structure and content, etc. The Jenzabar system coordinator (and possibly their staff) should be the only on-campus personnel in this group.

Using the Common Jenzabar CX Groups

The following permission structure utilizes the above groups. This information was extracted from the fileperms table.

Mode Name	Permission	Owner	Group
$CARSPATH	750	carsu	common
install	755	carsu	carsctrl
arc, frm,...	775	carsu	carsctrl
files	644	carsu	carsctrl

Mode Name	Permission	Owner	Group
bin	775	carsu	carsprog
files	2755	carsu	carsprog
utl	775	root	staff
files	755	root	staff
menu{src,opt}	770	carsu	carsctrl
directory	770	owner	carsctrl
files	640	owner	carsctrl
macros	770	carsu	carsctrl (M4 macros)
user,util,...	770	carsu	carsctrl
files	640	carsu	carsctrl
schema	750	database	carsprog
track	770	database	carsprog
cars.dbd	660	database	carsprog
files	640	owner	carsprog
data	750	database	carsprog
track	770	database	carsprog
files	660	database	carsprog
text	770	carsu	carsprog
directory	770	owner	carsprog
files	660	owner	carsprog
audit	775	carsu	carsprog
directory	770	carsu	group
files	660	owner	group
wp	775	carsu	carsctrl
filecabinet	770	owner	group
drawer	770	owner	group
files	660	owner	group
modules	770	carsu	carsctrl
function	770	carsu	carsctrl
files	640	owner	carsctrl
src	770	root	carsprog
module	770	owner	carsprog
progdir	770	owner	carsprog
files	640	owner	carsprog

Mode Name	Permission	Owner	Group
{util,utillib}	770	root	staff
progdir	770	owner	staff
files	640	owner	staff
include	770	root	carsprog
directory	770	carsu	carsprog
files	660	owner	carsprog

Interpreting Permissions

The permissions in the above table are interpreted as follows:

Typically, three digits specify the types of permissions on any file. The first of the three pertains to the owner of the file, the second to the group to which the owner belongs, and the third to all others. The digits themselves are computed based on what permissions should be given to each of these categories of users, and are based on the following criteria:

Read privileges	4
Write privileges	2
Execute privileges	1

By adding the digits, unique codes for any level of permission can be created.

Examples of Permissions

A 6 in the first position (the digit pertaining to the owner) means the individual can read or write to the file, but cannot execute it.

A 4 in the third position (the digit pertaining to the public) means that anyone can view the file.

A 7 in the first position means the owner can read, write, or execute the file.

A 0 in the second position (the digit pertaining to the group to which the owner belongs) means the group cannot read, write, or execute the file.

The Purpose of a Fourth Permissions Digit

Occasionally, a fourth digit will be used to set permissions on a file. When this occurs, it is the first digit of the four-character permissions code that is added. The fourth digit causes the system to interpret the owner as carsu or carsprog so the owner can have temporary permissions for a specific process that exceed his/her typical permissions. For example, in the Human Resources application, this type of permission is used to give the HR user the ability to post payroll, when ordinarily the HR user does not have access to general ledger records and processes.

The fourth digit is one of the following:

2	Pertains to owner only
4	Pertains to group only
6	Pertains to both owner and group

Other Common Groups

Other typical common groups include the following:

system, root
System manager

daemon,bin
Used by some UNIX processes

staff
System support staff. Generally computer center employees that are considered trusted users of the CX administrative system, these users are sometimes programmers for the CX administrative system.

operator
System operators. Some backup procedures will write messages to the usernames in this group.

shutdown
System shutdown username. Only the shutdown username should be in the shutdown group.

cis
Jenzabar employees are the only usernames that should be in this group.

Application User Groups

The CX application user groups include the following:

General Track Groups:
admissions, development, financial, student

Specific Application Groups:
acad_dean, acad_records, accts_pay, accts_recv, alumni, auditors, cashier, display_reg, donor_acct, fin_aid, maintenance, notes_pay, notes_recv, payroll, personnel, placement, publicity, purchasing, recruiting, registrar, research, stu_serv, stubill

Instructional System Groups

The student instructional system groups include the following: pupil, faculty, basic, cobol, fortran, pascal, clang, csmajor, instr_guest, instr_staff .

UNIX Programming Permissions

Introduction

Programming permissions within the UNIX environment include specific ways to control which users can perform specific tasks. Essential to permissions in the CX system is the *fileperms* process, which sets all the file permissions. Institutions that add processes to their CX installations must run *fileperms* to establish the accessibility of those processes.

CX utilizes all permission features of the UNIX operating system and of the INFORMIX relational database system. A shell user has access to actual UNIX files, including home directory, the CX tree (i.e., /usr/carsi), UNIX programs, etc. In addition, Jenzabar has incorporated into INFORMIX schema file permissions, groups and usernames. INFORMIX schema file permissions provide the system manager the confidence that only the authorized application user has permission to access that particular data.

Application users do not have open access to the data. Except where desired, these users cannot randomly produce adhoc updates of the database. The types of operations performed are through programs or pre-written command files. This eliminates the possibility of users changing data at will (e.g., change a General Ledger account balance without supporting detail).

Three levels of permissions are used, as shown on the chart and diagram at the beginning of this section.

1. **Operating System** – these permissions are defined in /usr/group and determine which UNIX files a user can access. Use this for home directories, programs, etc. Operating system permissions are not used for determining which tables a user can access within the database.
2. **Database** – these permissions are defined by the schemas and make processor. The database stores the information in the system tables (systabauth) and refers to those at run time.
3. **Applications** – Since most data entry and maintenance is performed by application programs, these programs are typically run with *carsu*, a database superuser. These programs control which data goes into various columns of the database tables. Not only do the programs often give a user more access than the database engine would allow, they can also restrict users based on perm tables (perm_table, glperm_table, etc.). These are used to ensure users have access to the appropriate groups of accounts, and financial and academic programs. This information is found in several perm tables.

Some overlaps exist between these levels (dbmake uses /etc/group to determine who has access to a particular table), but generally, they are maintained separately.

Access to files from outside third-party software is based on the level that a user is accessing the server. A word processor will look at files and have permissions based on the operating system; whereas, MS Access (using an ODBC) might have access to the database, but is limited to the permissions allowed by the database engine.

Additional Suggestions

Jenzabar recommends the following:

- If possible, eliminate phone-line access totally
- Limit outside calls to a local calling area as a means of pinpointing local intruders
- Keep modem telephone numbers unlisted
- Keep modem telephone numbers on a different three-digit exchange from that of the school's other numbers
- Change the modem telephone number from time to time

- Be certain that the system logs-off the user after completing a telephone call

Troubleshooting

For information about how to resolve permissions problems, see *Troubleshooting Tips for System Administrators* in this manual.

Users Permissions to Schemas in the Data Dictionary

Introduction

CX uses the UNIX permissions within each schema. These include both table and column permissions for user, group and public categories. In these permission declarations within each schema, the text user name or group name is included. When the *dbbuild* program updates the dictionary information, those names are actually stored as uid or gid (user identification numbers or group identification numbers) from the /etc/passwd and /etc/group files, as appropriate.

Changing Schema and Reassigning Permissions

Changing each schema and rebuilding those files is extremely time consuming when you use the specific application user name instead of the standard CX user names. In addition, a change in personnel results in the whole process having to be repeated. Since INFORMIX stores the identification number internally, you only need to create the new application user entry in the passwd file with the same user identification number as the standard name.

By having the new entry precede the existing standard CX entry, searches on the user identification number (to find the user name) result in locating the new user name. Therefore, when you build each schema and search the passwd file by user name, you will find a number that is the same number as the application user name. For example, the standard CX user name, coord, might have a user identification number of 340. If you want the user name of jane as the coordinator, all that is necessary is to include the jane entry before the coord entry in the passwd file (both have a user identification number equal to 340).

SECTION 8 – SYSTEM ADMINISTRATION

Overview

Introduction

This section provides information and procedures for maintaining the CX software and files. System administration issues discussed in this section include the following:

File and directory maintenance

- Using the *make* processor
- Locating, setting, and installing macros
- Setting up file transfer capability

User, permissions, and security issues

- Maintaining user accounts, groups, and permissions
- CX security tips
- Setting up slave printers for users

System/device testing issues

- Monitoring system performance
- Testing spooler devices

System data issues

- Data conversion
- Performing backups
- Transferring data from full disks
- Extracting data to tape

Maintaining Directories and Files Using the Make Processor

Introduction

CX implemented the *make* processor as an integral part of CX to simplify and maintain changes in the software.

- Used with the Revision Control System (RCS), *make* maintains a history of software modifications and allows for previous versions to be retrieved.
- Used with macros, *make* simplifies the editing of source files, and then expands the macros before translating the source.

Make also ensures that the translated versions of source are properly installed.

GNU Make Processor

CX distributes the GNU *make* processor to have a consistent version of make used across the platforms used by client institutions. The GNU *make* processor provides the following added benefits:

- Better variable manipulation
- Pattern rules and rule chaining
- Conditionals (ifeq, else, endif)

To access the documentation feature of GNU make, you enter **info -f make** when using GNU *make*.

Maintaining a History Of Changes

Make maintains a complete history of changes made to each source file in an RCS subdirectory. You should never need to access the contents of this directory outside of the *make* and RCS procedures. The history of changes for a file begins when you check in the file. Checking in a file:

- Records a copy of the file and assigns a version number to it. The name for the file that was last checked is the *Recent* version.
- Removes write permissions on the source file to avoid changes while the file is checked in.

When you want to change a file, you first check the file out; this produces a working file and locks it so only you can make the changes. You can make changes and tests to the working file. Then, you check the file in to create a new version of the file; this records the changes and the reasons for the changes. The system assigns a new version number to the changed file, and removes write permissions.

This feature of recording the revision history of source provides the ability to retrieve previous versions of files. The feature also keeps a record of client changes versus CX changes and merges new distributions of the source with client versions.

Expanding, Translating, and Installing Source Files

Expand refers to the process that expands all macros that are used in the source file before it is translated.

Translate refers to the process of producing an object file from a source file. It is this object file that the *make* processor installs.

Install refers to the process within the *make* processor that makes a file or program available for use from within the CX menu. Any translation or compilation needed on the file or program is also performed during installation.

Using *make* to translate the source file reduces the number of steps required for creating, modifying, and/or translating reports, screens, or documentation. When *make* translates a source file, it first expands all macros referenced in the file to their full definitions and then finishes the translation using the appropriate programs. Error files are saved; error messages are made available to the user.

You can use macros in any file that is translated by *make*. Using macros shortens the time you need to develop and customize source files. See *Jenzabar CX Macros* in the *CX System Reference Technical Manua* for further information on macros.

The versions of reports and programs that are executed when you install source files are located in the $CARSPATH/install directory. You can edit and test modifications to a file within the source directory without affecting the version being executed by a user. Once a version of a file is thoroughly tested and checked in to the RCS directory, it is installed and made available to users.

Separate Installed Source

The $CARSPATH/install directory contains the versions of reports, programs, etc. that the system executes. Having separate installed versions of files provides you capability to edit and test modifications to a file within the source directory without affecting the version being executed by a user.

Object Directories

The *make* processor places program *make* files into object directories. This feature makes it possible to make multiple CX releases that share the same source files.

- The variable defining access to the object directory is CARSOBJ=$CARSPATH/objects.
- The variable defining the object directory is Objdir=$CARSOBJ/$Subpath/$(Objver).
- The variable defining the object version is Objver=$(Cxxver)$(Dbgver)$(Infver).

Directory Structure Maintained by Make

Make maintains the CX directory structure. The directories under $CARSPATH/modules contain the various module directories, such as Accounting and Financial Aid. You must be familiar with the CX directory structure in order to understand how *make* maintains it. The following directories are maintained by *make*:

- $CARSPATH/include
- $CARSPATH/macros
- $CARSPATH/menuopt
- $CARSPATH/menusrc
- $CARSPATH/modules
- $CARSPATH/schema
- $CARSPATH/skel
- $CARSPATH/src
- $CARSPATH/<product>

RCS Directories

Although *make* maintains each directory level starting at $CARSPATH/modules, most *make* targets affect the files contained in the lowest directory level under the current directory, such as $CARSPATH/modules/regist/screens. These lowest level directories contain an RCS directory, used by *make* to keep a complete revision history of each file and additional information about the current status of each file.

When you execute *make* at a higher directory level, *make* passes the targets and variables down to the subdirectories where they are finally executed. This processing occurs in the $CARSPATH/modules, $CARSPATH/menuopt, and $CARSPATH/schema directories.

Make Directory Types

For each directory under *make* control, you must define the types of files or subdirectories *make* maintains. Use the directory type abbreviations listed in the following table when you initialize the directory for use by *make*; these abbreviations also serve as the extensions of the translated and installed filenames.

For example, the $CARSPATH/modules/common/reports directory uses the arc *make* type. A file in an arc type directory has an object file named <filename>.arc installed as ${CARSPATH}/install/arc/common/<filename>.arc.

The following list describes the type abbreviations and the CX directories used for the type.

Note: When multiple source directories are listed for a directory type, the second directory lists the directories used by new products developed for CX.

aplib
Application libraries
Source directory:$CARSPATH/src/Lib/
Install directory:$CARSPATH/install/lib/

aps
Application Servers (C++ programs that use Libdata.a)
Source directory:$CARSPATH/src/<module>/<service>_aps
Install directories:$CARSPATH/install/aps/<module>/<service>_aps
$CARS_ODBCPATH/$CARSV/modules (updated by thee install of the aps service)

arc
ACE reports
Source directories:$CARSPATH/modules/<module>/reports/
$CARSPATH/<product>/Reports/
Install directory:$CARSPATH/install/arc/<module>/

cgi
Web Server Scripts (m4 translation: file -> file.cgi)
Source directory:$CARSPATH/modules/<module>/cgi/....
Install directories:$(Webpath)/cgi-bin/$(subpath)
$(Webtemppath)/cgi-bin/

Note: When you want to reinstall all subdirectories under this make directory type, you can use the Dotree function. You enter: **make reinstall F=ALL Dotree=Y**

cmd
Command scripts commands
Source directory:$CARSPATH/modules/util/commands/
Install directory:$CARSPATH/install/utl/

dir
All directories for $CARSPATH/ down to the type-specific directory. There are no associated install directories.

doc
Documentation files
Source directory:$CARSPATH/modules/<module>/documents/
Install directory:$CARSPATH/install/doc/<module>/

fps

Form Production System (FPS) forms
Source directory:$CARSPATH/modules/<module>/forms/
Install directory:$CARSPATH/install/fps/<module>/

frm

PERFORM screens
Source directory:$CARSPATH/modules/<module>/screens/
Install directory:$CARSPATH/install/frm/<module>/

htm

HTML files (m4 translation: file -> file.htm)
Source directory:$CARSPATH/<product>/Html/
Install directories:$(Webpath)/htdocs/
$(Webtemppath)/htdocs/

Note: When you want to reinstall all subdirectories under this make directory type, you can use the Dotree function. You enter: **make reinstall F=ALL Dotree=Y**

inc

Program include files
Source directory:$CARSPATH/include/
Install directory:$CARSPATH/install/inc/

inf

SQL scripts (formerly called informers)
Source directory:$CARSPATH/modules/<module>/informers/
Install directory:$CARSPATH/install/inf/<module>/

lib

C source libraries
Source directory:$CARSPATH/src/Lib/
Install directory:$CARSPATH/install/lib/

ltr

WPVI letters
Source directory:$CARSPATH/modules/<module>/letters/
Install directory:$CARSPATH/install/ltr/common/

m4

M4 macro files
Source directory:$CARSPATH/macros/
Install directory:$CARSPATH/install/m4/

mnu

Menu description files
Source directory:$CARSPATH/menusrc/
Install directory:$CARSPATH/install/mnu/<track>/

mod

Module directories
Source directory:$CARSPATH/modules/

opt

Menu option files
Source directory:$CARSPATH/menuopt/
Install directory:$CARSPATH/install/opt/<module>/

oth
Modules, others, runtime macro expansion
Source directory:$CARSPATH/modules/<module>/others/
Install directory:$CARSPATH/install/oth/<module>/

prog
C programs
Source directory:$CARSPATH/src/
Install directory:$CARSPATH/install/bin/

sch
Schema files
Source directory:$CARSPATH/schema/<track>/

scp
Shell (CSH) scripts
Source directories:$CARSPATH/modules/<module>/scripts/
$CARSPATH/<Product>/Scripts/
Install directory:$CARSPATH/install/scp/<module>/

scr
Application program screens
Source directory:$CARSPATH/modules/<module>/progscr
Install directory:$CARSPATH/install/scr/<module>/

single
Single C programs
Source directory:$CARSPATH/src/common/single
Install directory:$CARSPATH/install/utl/

skl
Skeleton files
Source directory:$CARSPATH/skel/

smo
System Modification Order (SMO)
Source directory:$CARSPATH/smo/<smo#>/
Install directory:$CARSPATH/install/smo/

spl
Stored procedures
Source directory:$CARSPATH/procedures/<track>/

sys
System files
Source directory:$CARSPATH/system/
Install directory:$CARSPATH/install/sys/

util
Utilities
Source directory:$CARSPATH/src/util/
Install directory:$CARSPATH/install/utl/

Initializing a Directory: the Makeinit Command

To initialize a new directory for use by the *make* processor, use the makeinit command. The abbreviation for the type of file being maintained is passed to makeinit. For example, the following command line will set up the current directory for maintaining PERFORM screens (usually, such a directory would be named screens):

```
% makeinit frm
```

You should not need to initialize schema and module directories in the $CARSPATH directory. However, schema and module directories are available with makeinit if they are needed in another database area.

The *make* processor maintains a list of the files that are to be maintained by *make* in a .makelist file. This list of files is created during makeinit.

Note: All filenames beginning with an underscore (_), pound sign (#), period (.), dash (-), or plus sign (+) anywhere in the name are excluded from the .makelist file. However, these characters can exist in filenames that are to be kept in the directory, but not maintained by *make*. In addition, files with execute permissions are also excluded from being maintained by *make*. A capital letter as the first letter of a filename is permitted and will be used with files created by an institution.

File Names Maintained

The *make* processor uses the *.makelist* file to track the files to be maintained in a directory. The *makeinit* command creates the *.makelist file*. File names must follow certain restrictions to be maintained by *make*. For example, *make* will maintain initially capitalized file names (capital letters as the first letter). *Make* excludes the following files:

- File names beginning with an underscore (_) or pound sign (#)
- File names that contain a period (.), dash (-), or plus sign (+) anywhere in the name
- Files with execute permissions are also excluded from being maintained by *make*.

Note: Because of the above exclusions, you can keep files which you don't want *make* to maintain in a *make*-maintained directory by using these characters in the names of the file.

Using the Make Processor

Introduction

The *make* processor places a *Makefile* file in each directory the *make* maintains. The *Makefile* file provides *make* with the following information:

- The type of files to be maintained in that directory
- How to maintain the files

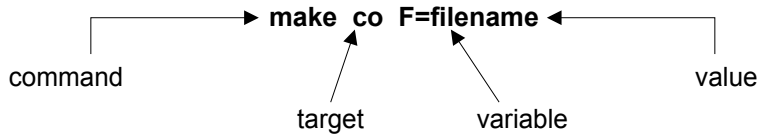
You control the actions performed by *make* by specifying *targets* and *variables* on the *make* command line. The general form of a *make* command is as follows:

Example: % **make [target] ... [variable=value] ...**

Note: Do not specify files (by using the F= variable) in any program of *make* directories, called prog.

Make Command Line Structure

The *make* processor command line has four parts to it: *command, target, variable, and value*. To illustrate the parts of the *make* command line, the following example shows what you would enter to check out a file:



This command line reads as follows: "Use the *make* processor to check out a file called *filename*."

Standard Make Targets

A *make* target is a command for *make* to perform an action on some or all of the files maintained in the current directory. See *Make Targets* in this section for a list of valid targets.

Target Naming Conventions: Prefixes

The following lists prefixes, and their meanings, that you can add to some targets.

smo
Process the files in a SMO (e.g., smodeposit)

t
Temporarily perform object (e.g., tinstall)

Target Naming Conventions: Suffixes

The following lists suffixes, and their meanings, that you can add to some targets.

n
Provide an answer of no to the prompt (e.g., rebuildn). Shows changes that would occur. Does not actually create a file or change permissions.

y
Provide an answer of yes to the prompt (e.g., rebuildy)

f

Forces a rebuild of the table. Mainly used to change dbspace for the location of the table.

Note: If you use build F, you must remake the synonyms with isql.

Make Variables and Values

Make variables allow you to specify additional information regarding the action performed by a target. If you do not specify a variable on the command line, the system either supplies a default value, or prompts you for the information. If the value you assign to a variable contains spaces, you must surround the value by single (') or double (") quotes to keep it together as one value.

The following are common *make* variables:

- (for files)
- (for a log message to be assigned to a revision)
- (for a target)
- (for a version of a file)

When you use the F variable, you must specify the file affected by *make*. To specify multiple files, list each file within quotes, with a space separating each filename. To include all the files in a directory, you can type ALL instead of listing each file.

Make Targets

The following lists each target you use with *make* in order to perform a task, at least one example command line to show the format to use with each target, and a description of each target.

Note: The directories in which you are working must be initialized for *make* before the *make* command lines are valid.

Note: If you do not specify a target on a command line, the system assigns a default target of /**translate**/.

add
adddir
addmod

The add target creates a new a skeleton file containing the revision log message header and the basic structure of the type of file you plan to create, and informs *make* that the new file is to be included in the list of files that it maintains.

Note: If you are adding a subdirectory, use the adddir target, followed by the S variable and the subdirectory name.

Example:

% **make add F=*filename***
% **make adddir S=*subdirectoryname***
% **make addmod M=*modulename***

analyze

The analyze target provides database constraint analysis. You normally run this target if an error occurs on the build target.

build

The build target creates a new database file or changes permissions on an existing database file. You must specify a value for the F variable when you use the build target. A trace of the execution is placed in a file with a .sql extension.

Note: The file must be in a checked in state (using the ci target) before you can use the build target.

Example: % **make build F=*schemaname***

ci

The ci target checks in a file. You must check in the new revision of a file before you install the file. This process updates the revision history for the file and unlocks it, making the file available for another individual to check out. In addition, *make* removes the write permissions for the file so that the next individual is reminded to check out the file (using the co target) before editing it. Each time you check in a file, you are required to provide a log message explaining the reason you revised the file.

Example: % **make ci F=myfile L=*'log message'***

cii

The cii target checks in and installs a file, combining two steps into one. See the descriptions for the ci and install targets for detailed information.

Example: % **make cii F=/*filename*/**

cleanup

The cleanup target removes all of the object files (files with a .o extension) in a source directory. The cleanup target forces *make* to recompile each individual .c source file. However, the cleanup target does not remove the installed versions of files. The cleanup target is useful for reducing the amount of disk space needed in the source directory, or for ensuring that a new object file will be created the next time the source is translated. The cleanup target must be used under the $CARSPATH/src directory.

Example: % **make cleanup**

co

The co target checks out a file so that you can modify an existing file. Checking out the file gives you exclusive write permission to the file and locks the file so that no other individual can check it out.

Example: % **make co F=*filename***

delrev

The delrev target deletes the most recent version of a file that has been checked in by error; however, delrev does not replace the working file. If the version of the file you are deleting is not the most recent version, you can use the V variable to specify the version number you want to delete.

Note: Do not specify files when you are executing the delrev target within directory type of prog.

Example:

% **make delrev F=*filename***
% **make delrev V=*version***
% **make delrev F=*filename* V=*version***
% **make delrev F=*filename:version***

diff

You can use the diff target to compare two different versions of a source file. Use the V variable to specify the two versions you want to compare, and separate the two versions by a colon (:). If you do not specify the two versions on the command line, the system prompts you to supply the versions. The output is placed in the working directory in a file with a .out extension. Do not specify files when you are executing the diff target within a prog directory type.

Example:

% **make diff**
% **make diff F=*filename***
% **make diff V=*version***
% **make diff F=*filename* V=*version***

% **make diff F=*filename* V=*version:version***

drop

The drop target deletes a database file. The schema file must have already been checked out (using the co target). A trace of the execution is placed in a file with a .sql extension. The drop target must be used in a schema directory type.

- Use the drop target, followed by the remove target, to remove a file permanently from the system.
- Use the dropn target to erase the file with an n (for No) response to the prompt that follows. The file is not checked in, and the results are saved in the .sql file.
- Use the dropy target to erase the file with a y (for Yes) response to the prompt that follows. The file is checked in, and the results are saved in the .sql file.

Example:

% **make drop F=schema L='log message'**
% **make dropn F=schema**
% **make dropy F=schema L='log message'**

exec
execdir
execmod

The exec target executes a shell command in a directory maintained by *make*. This is useful when you are passing a shell command from a higher directory level to several directories in which the command needs to be executed. Use the X variable with the exec target to define the shell command you want to execute. You can also use the execdir and execmod targets to execute a shell command within directory- and module-type *make* directories.

Example:

% **make exec X=*executable***
% **make exec X=*command***
% **make execdir X=*command***
% **make execmod X=*command***

expand (non-program types)

The expand target examines a file after any macros the file contains are expanded, but before any further translation is performed, such as SACEPREP or SFORMBLD. Use the expand target if errors occur during the testing phase of a file; the expand target expands all macro references and saves the result in a file in the working directory with a .exp extension.

Example:

% **make expand F=*filename***
% **make expand F=*ALL***

expand (program types)

The expand target for program types) passes the specified files (or all source files if none are specified) through the C pre-processor to expand all #include, #define, etc. lines. The output for each files goes onto $(objdir)/<fileroot>.i.

Example:

% **make expand F=*filename***
% **make expand F=*ALL***

expobj

The expand object, similar to expand for program types, passes the specified files (or all source files if none are specified) through the C pre-processor to expand all #include, #define, etc. lines. The target also attempts to compile the $(Objdir)/<rootfile>.i intermediate files to produce error messages with line numbers corresponding to the $ (Objdir)/<fileroot>.i files to help debug compiling errors.

Example:

% **make expobj F=*filename***
% **make expobj F=*ALL***

getprev

The getprev target retrieves a previous copy of an installed object file with an extension of .pv1, if the file was recorded by *make*. The getprev target makes the previous copy the active version and decreases the numbers of any other versions of the same type.

Example: % **make getprev F=*filename***

getsave

The getsave target retrieves a previously saved copy (which was created using the save target) of an installed object file with an extension of .sv1, if the file was recorded by *make*. The getsave target makes the previous copy the active version of the file, and decreases the numbers of any other versions of the same type.

Example: % **make getsave F=*filename***

gettemp

The gettemp target retrieves a previous copy of a temporarily installed object file (which was created using the tinstall target) with an extension of .tm1, if the file was recorded by *make*. The gettemp target makes the previous copy the active version of the file, and decreases the numbers of any other versions of the same type.

Example: % **make gettemp F=*filename***

help

The help target provides online help for a specific target. If you enter **make help**, a list of targets appears. If you enter **make help T=<target>**, a help topic about a specific target appears.

Example: % **make help T=expand**

history

The history target lists log messages for revisions made to files within a particular period of time. Use the D variable to indicate the range of dates to be included in the revision history list. Specify dates (and times) in the following formats. Separate multiple date ranges with a semicolon (;).

- d1<d2 (for all revisions created between dates d1 and d2, inclusive)
- <d (for all revisions dated d1 or earlier)
- >d (for all revisions dated d1 or later)
- d1 (for the single most recent revision dated d1 or earlier)

Example:

% **make history**
% **make history F=*filename***
% **make history F=*filename* D=*'date range'***

histweek

The histweek target lists the revision history for only the past week.

Example: % **make histweek F=*filename***

install

The install target installs the object file for general use in the CX, in the appropriate directory under $CARSPATH/install, after a file is checked in. For example, $CARSPATH/modules/common/ screens/file would be installed as $CARSPATH/install/frm/common/file.frm. You must specify the filename(s), or ALL, with the F variable.

Example:

% **make install F=*filename***
% **make install F=*ALL***

makedef

The makedef target creates the definition files for a table and places them in the appropriate $INCPATH if they have changed. You must be in a schema directory in order to use the makedef target.

Example:

% **makedef F=*schema***
% **makedef F=*ALL***

makedep

The makedep target creates or updates the dependency list for the *make* processor. You must use the makedep target only in the $CARSPATH/src directory path.

Example: % **make makedep**

merge
mergeci

The merge target makes the same changes that are made in new distributions of source files from CX to the local version at an institution. Once the new source file is checked in, the local version is merged with the distribution from CX, using the merge target, to produce a new local version. The V variable specifies the versions to be merged. The merge target also translates the source file after the versions are merged together.

CAUTION: View and test the source file carefully before you check in the new combined version. If the two versions being merged are compatible and will merge without errors, you can use the mergeci target to automatically check in the new version after merging. Do not specify files when you are using the merge target within the $CARSPATH/src directory.

Example:

% **make merge**
% **make merge F=*filename***
% **make merge F=*filename* V=*version***
% **make mergeci F=*filename* L=*'log message'***

move

The move target relocates a working file and its associated RCS to a new directory. If the move is successful, the file is reinstalled. Do not use the move target within the $CARSPATH/src directory.

Example:

% **make move F=*filename* S=*path***
% **make move F=*filename:path***

packrev

The packrev target consolidates all revisions from previous releases into the base trunk revision of the current release.

Example: % **move packrev F=*filename* V=*release***

reci

The reci target re-checks in a file. Use the reci target for instance, if you discover that further changes need to be made to a version of a file after the file has been checked in prematurely. If the file is not yet installed, you can check out, change, and re-check in the file (using the reci target). If the file is already installed, the current (incorrect) version might be in use. The reci target is most commonly used for a file that was checked in with the changes that caused new problems. The reci target is not needed if proper testing is performed on source files before they are checked in.

Example:

% **make reci**
% **make reci F=*filename***

reco

The reco target re-checks out a file if a file has been checked out (using the co target) but destroyed in some way, and the file must be restored to its initial checked out state. The reco target replaces the current working file with the last checked in version.

Example:

% **make reco**
% **make reco F=*filename***

reinstall
REINSTALL

The reinstall target installs a new copy of an object file, if it must be installed after it has already been installed. For example, use the reinstall target if you need to reinstall a file because a macro file has changed or if the installed version of a file somehow becomes corrupted.

CAUTION: If you use the reinstall target in the $CARSPATH/src directory path, everything is reinstalled except for the screens. When you are in $CARSPATH/src, use the REINSTALL target (in capital letters) instead, which allows you to reinstall a program and the associated screens, if the screens in a subdirectory called SCR.

Example:

% **make reinstall F=*filename***
% **make REINSTALL F=*filename***

remake
remakeall

The remake target re-creates the list of files that *make* maintains if, for instance, the files in the list are different from the files *make* should be maintaining. For example, use the remake target if a file is removed using the UNIX remove command (e.g., % **rm filename**) instead of the remove target, thus causing the list of files to be incorrect. You can also use the remakeall target, which uses the remake target in the current directory and then passes the remakeall target down to any subdirectories.

Example:

% **make remake**
% **make remakeall**

remove
removedir
removemod
fremovedir
fremovemod

The remove target removes a file that is no longer useful and adds a ,o extension to the RCS version of the file. The RCS files with the ,o extension are automatically removed from the system weekly by the carsweekly script, as defined in /modules/Util/scripts. However, if you decide to keep the removed file before the specified number of days has expired, you can restore the file using the restore target.

- You can remove an empty directory from a directory *make* directory type by using the removedir target. Use the fremovedir target to remove a directory with contents in it.
- You can remove an empty module from a module *make* directory type by using the removemod target. Use the fremovemod target to remove a module with contents in it.
- Do not use the remove target within a prog directory type.

Example:

% **make remove F=*filename***
% **make removedir**
% **make removemod**
% **make fremovedir**
% **make fremovemod**

rename

The rename target changes the name of a file, while preserving all previous revision history for the file. Use the F variable to specify both the old name and the new name, and separate the two names with a colon (:). Using the rename target prevents you from having to execute the add target, copy the file, and then remove the old file. Do not use the rename target within a prog directory type.

Example:

% **make rename F=*filename***
% **make rename F=*filename:filename***

restore

The restore target retrieves a file that has recently been removed using the remove target. The restore target moves an RCS/filename,o file back to an RCS/filename,v file within the RCS subdirectory and re-creates the working directory. Because only the RCS file is restored, the file must be checked out to be used again.

Note: There is no restore target for directory- and module-type *make* files.

Example: % **make restore F=*filename***

save

The save target saves a file, for instance, if you need to save the currently installed (active) version of a file for possible retrieval in the future. The save target makes a copy of the installed version with a .sv1 extension. The number in the extension of any previously-saved versions is incremented, up to the maximum number of saved versions allowed. The save target provides a method of keeping old copies of installed files, in addition to the copies kept by the install and tinstall targets. Do not use the save target within a prog directory type.

Example: % **make save F=*filename***

subs

The subs target translates the menudesc source file in a directory and its subdirectories into a complete menu object file called menudesc.mnu. In addition, you can set the T variable to the target to be executed (if other than the default target, translate). The target is executed in each subdirectory starting with the current directory and continuing to the lowest level. The default file for the menu source *make* targets is menudesc, so the F variable and filenames are not required with any of the *make* targets in the $CARSPATH/menusrc directories. You must use the subs target in the $CARSPATH/menusrc directory path.

Example: % **make subs T=*target***

tinstall

The tinstall target installs the object file for general use in the CX, in the appropriate directory under $CARSPATH/install. The tinstall target is used primarily on files that are checked out, although it will work on checked in files. In addition to installing the new object file in the $CARSPATH/install path, a copy of the previous object file is kept in the install directory and given a new extension of .tm1. Subsequent uses of the tinstall target on the file will not destroy the .tm1 file, thus keeping a copy of the object file from the previous time the file was installed.

For example, $CARSPATH/modules/common/screens/file would be installed as $CARSPATH/install/frm/common/file.frm and the previously existing copy of the file.frm would be renamed to file.tm1.

Example:

% make tinstall F=filename
% make tinstall F=ALL

translate

The translate target translates the working file into an object file for testing. The object file is the file that is installed and then accessed during the normal use of CX. For example, the filename extension of an object file for a PERFORM screen is .frm. The translate target is the default target used when no target is explicitly specified. If you do not specify an F variable, the system prompts you for specific filenames. If you want to translate all files, you can specify F=ALL on the command line. The translate target only translates those files that need to be translated, such as when a working file has been changed since the object file was created in that directory, or when the object file does not exist.

Example:

% **make translate F=*filename***
% **make F=*filename***

unco

The unco target unchecks out a file after it has been checked out (using the co target). This target puts the file back into the condition it was in when first checked out, and sets all of the revision numbers back to the original settings.

Example:

% **make unco**
% **make unco F=*filename***

Make Processor Command Quick Reference

Creating a File

You create a *make*-maintained file by entering the following:

% **make add F=<filename>**

- If the file does not already exist, the command creates the skeleton file of a type determined by the Makefile (e.g. ACE report, screen, or form).
- If the file exists, the command puts it under control of the *make* processor and adds a header, if not already present.

Checking Out a File

Before you can modify a file, you must check out the file by entering the following:

% **make checkout F= <filename>**

Note: The following is an abbreviation of the command: **make co F= <filename>**

The check out command does the following:

- Creates a working version of the source file to edit.
- Changes the ownership of the file to your login.
- Gives you, the owner, exclusive read and write permissions.

Note: You can use the breaklock script in modules/util/scripts to change ownership of a checked out file.

Translating Files

To translate your source file to an installed object file, enter the following:

% **make F=<filename>**

Note: You can also enter the following version of the command: **make translate F=<filename>**

The translate command does the following:

- Translates or compiles source files into object files.
- Translates only those source files that have been modified since the last translation.

Checking In a File

To make a file available to users, enter the following:

% **make ci F=<filename> L=<log message>**

Note: You can also enter the following version of the command: **make checkin F=<filename> L=<log message>**

The check in command does the following:

- Checks in a file which has been modified.
- Changes the permissions to read only for both owner and group.
- Automatically updates the RCS.

Note: If you enter the log on the command line, the log will be a one line version; if not, the system will prompt you, and you can enter a multi-line version or the place the log message in a <filename>.log file.

Installing Object Files

To install files for use in CX, enter the following:

% **make install F=<filename>**

The install command does the following:

- Moves new object files into their appropriate directories for use in CX.
- Current version is given the extension .pv1; other former versions are renumbered consecutively, until the maximum is reached, and those are then deleted from the system.

Note: Use *install* only the first time a file is put into place. Use *reinstall* thereafter.

Checking In and Installing Files

To combine the steps to check in and install a file, enter the following:

% **make cii F=<filename>**

The cii command combines the check in and install commands.

Command Sequence

The following lists the sequence in which you use the *make* commands for different file states.

A new file

1. % **make add F=<filename>** (creates a skeleton for the file)
2. % **vi** to create or edit the file
3. % **make F=<filename>**
4. Test the translated file
5. % **make ci F=<filename>**
6. % **make install F=<filename>**

An existing file (not previously maintained by *make*)

1. % **make add F=<filename>**
2. % **vi** to edit the file
3. % **make F=<filename>**
4. Test the translated file
5. % **make ci F=<filename>**
6. % **make install F=<filename>**

An existing file governed by *make*

1. % **make co F=<filename>**
2. % **vi** to edit the file
3. % **make F=<filename>**
4. Test the translated file
5. % **make ci F=<filename> L=<log message>**
6. % **make install F=<filename>**

An installed file version from menu (for testing)

1. % **make add F=<filename>**
2. % **vi** to edit the file
3. % **make F=<filename>**
4. % **make tinstall F=<filename>**
5. Test the file from menu
6. Edit the file, make, tinstall, test until there are no errors
7. % **make ci F=<filename>**
8. % **make install F=<filename>**

Locating Macros Within an Application

Introduction

Jenzabar has developed a tool called *applocate* that an institution can use to obtain a listing of macro definitions associated with an application.

When an institution is ready to set up and modify macros in an application, the institution can run the applocate script to obtain a listing of potential macro definitions to modify. The listing generated by applocate always reflects the current state of the system, as opposed to a hard copy listing provided by CX that would not necessarily reflect the current state of an institution's system due to macro file changes made by CX.

How to Locate Macros within an Application

There are two parameters associated with applocate script. The first parameter is the name of the application (e.g., crsent). The second parameter is a specific macro file that applocate searches (e.g., $CARSPATH/macros/custom/student). If an institution does not specify a macro file, applocate searches all of the macro files in the $CARSPATH/macros/custom and the $CARSPATH/macros/user directory paths.

The following lists the steps to follow for locating macros within an application.

1. Do you want to search for a macro in a specific macro file?
   - If yes, go to step 2.
   - If no, go to step 3.
2. Do the following:
   - Enter the following command at the prompt:**applocate *APPLICATIONNAME macrofilename* > *filename***(e.g., applocate CRSENT student > crs.student)
   - Go to step 4.In this example, the applocate script searches the $CARSPATH/macros/custom/student directory path and locates all of the macros that are applicable to the Catalog and Schedule application.The system redirects the output to a filename called crs.student. The output consists of a listing of all the macro definitions in the student macro file that CX has identified as applicable to the Catalog and Schedule application. The output also contains comments and a brief description of the other files (e.g., menu options, reports, screens, programs) that an institution must reinstall to reflect the change(s) made in the macro definition(s).
3. Enter the following command at the prompt: **applocate *APPLICATIONNAME* > *filename***

   Example: applocate CRSENT > crsent.all

   In this example, the applocate script searches all of the macro files in the $CARSPATH/macros/custom and $CARSPATH/macros/user directory paths, and locates all of the macros that are applicable to the Catalog and Schedule application.

   The system redirects the output to a filename called "crsent.all." The output consists of a listing of all the macro definitions in the custom and user macro files that CX has identified as applicable to the Catalog and Schedule application. The output also contains comments and a brief description of the other files (e.g., menu options, reports, screens, programs) that an institution must reinstall to reflect the change(s) made in the macro definition(s).
4. Do you want to know where the macros listed by the applocate script are used throughout CX?
   - If yes, use the Maclocate command.

Note: For information on how to use the Maclocate command, see *Locating All Files That Contain a Macro* in this section.

- If no, stop. You have completed this procedure.

Locating All Files That Contain a Macro

Introduction

Before modifying macros on CX, you might want to perform either of the following procedures:

- Locate all of the files that contain a macro name
- Locate all of the files that contain a specific macro name

Perform these procedures to ensure that the macro you are changing does not affect processing other than what you intend to be affected.

CAUTION: To locate all the files that contain a macro or a specific macro, the system searches every file on the system. Since this search can take several hours, make sure that you perform either of the following two procedures at night or over the weekend.

How to Locate All Files that Contain Macros

To locate all the files that contain macros, enter the following command at the prompt: **$SCPPATH/util/maclocate.scp outfile.**

Maclocate.scp is a Jenzabar-created UNIX command that searches the system for every file in which a macro is located.

"Outfile" is the filename of the Maclocate Report, which lists each macro and where that macro is used in the system.

CAUTION: A macro that appears by itself in the Maclocate Report is not currently used in the system. However, the macro might be used within the definition of another macro(s). Check the macro files before you remove a macro.

How to Locate All Files that Contain a Specific Macro

The following lists the steps to follow for locating all files that contain a specific macro.

1. Enter **vi macfile** at the prompt to create a new file called macfile.
2. Type the name of each macro that you want CX to locate.

   Example: COMMENT_ID

   CAT_DEF
3. Do the following:
   - Press **<Esc>**.
   - Enter **:wq** to exit and save the file.
4. Enter the following command line:

   $SCPPATH/util/maclocate.scp -f macfile outfile
   - The "-f macfile" variable tells the system to search for only the macros that you specified in the macfile file.
   - "Outfile" is the filename of the Maclocate Report, which lists the name of each file containing the macro that you specified in the macfile file.

   CAUTION: A macro that appears by itself in the Maclocate Report is not currently used in the system. However, the macro can be used exclusively within the definition of another macro(s). Check the macro files before you remove a macro.

Setting Up Macros

Introduction

Use the following process and procedure for all macros except for ENABLE macros. You make changes to enable macros using the Configuration table. For more information, see *Configuration Table* in *Common Tables and Records* in this manual.

The Process

The following shows the phases in the overall process of setting up and modifying macros.

1. Access the macro files located in the following directory path: $CARSPATH/macros.
2. Check out a macro file using the *make* processor.
3. Modify the macro(s) in the file using your institution's text processor (e.g., vi editor).
4. Check in the macro file using the *make* processor.
5. Install the macro file using the *make* processor.
6. Reinstall all of the files (reports, screens, programs) that use the macro(s) you modified.

How to Set Up Macros

The following lists the steps to follow to set up a macro.

1. Enter **echo $CARSDB** to find out what database you are currently working in.
2. If you are not in the appropriate database, enter the following command line: **setdb *database name*.**
3. Enter **cd $CARSPATH/macros** to access the directory containing the four macro subdirectories.
4. Enter **cd custom** or **cd user** to access the macro files you want to set up.
5. Enter **make co F=*filename*** to check out the specific file containing the macros to be set up. (e.g., make co F=student)
6. Enter **vi *filename*** to view the file containing the macros (e.g., vi student).
7. Use the text processor keys to move through the file and define every macro you want to enable and/or make changes to the macro definition as necessary.

   Note: To enable a macro, type a "Y" or enter a value for the macro definition.

   Example: m4_define(`ENABLE_FEAT_FPS', `Y')

   m4_define(`CAT_DEF', `UG9X')
8. Press **<Esc>**.
9. Enter **:wq** to exit and save the file.
10. Enter **make cii F=*filename*** to check in and install the file (e.g., make cii F=.....)
11. Use the maclocate command to identify all of the files (e.g., reports, screens, programs) that use the macro(s) you have modified.

    Note: For information on how to use the maclocate command, see *Locating All Files That Contain a Macro* in this section.
12. Reinstall all of the files (reports, screens, programs) that use the macro(s) you modified.

Note: For more information on the reinstall process, see How to Reinstall Files That Reference a Modified Macro.

Reinstalling Files That Reference a Modified Macro

Introduction

After you check out, modify, and check in all of the macro files that need customizing for an institution, you must reinstall all of the files in order for CX to recognize all of the changes you have made and for it to work properly.

When to Reinstall Files

Reinstall files that reference a macro that has been modified only after all required changes have been made to files. After you run the *maclocate* script, review the output generated by the maclocate script to determine which files you need to reinstall.

Which Files to Reinstall

After you modify a macro file, you must reinstall all other files that reference the macro you modified. Reinstalling files ensures that any files affected by the macro modifications will work properly. The following are the subdirectories in the $CARSPATH directory path that contain macros: include, macros, menuopt, menusrc, modules, src, skel

How to Reinstall Files

The following lists the steps for reinstalling files that reference a macro you have modified.

CAUTION: This procedure could take several hours. You should reinstall macro files during off-business hours to avoid disrupting normal processing using the CX.

1. Enter **cd $CARSPATH** to access the directory containing the macro files.
2. Is a file you need to reinstall located in the $CARSPATH/install directory path?
   - If yes, enter **cd include/*subdirectory*** to access the subdirectory containing the install file. Go to step 3.
   - If no, go to step 6.
3. Enter **make reinstall F=*filename*** to reinstall the include file that contains a macro you modified.

   Note: If more than one include file references the macro you modified, you can enter **make reinstall F=ALL** to reinstall all of those include files at once.
4. Enter **cd ../src** to access the directory containing source files.
5. Enter **make reinstall** to reinstall the source files that reference the macro you modified.
6. Enter **cd subdirectory/module/directorytype** to access the subdirectory containing the file that references the macro you modified (e.g., cd menuopt/regist/screens).
7. Enter **make reinstall F=*filename*** to reinstall the file that references the macro you modified.

   Note: If more than one file references the macro you modified, you can enter **make reinstall F=ALL** to reinstall all of those files at once.
8. Repeat steps 6-7 for every file you want to reinstall.

Creating and Deleting User Accounts

Introduction

CX provides tools for adding and deleting user accounts on the host system. Jenzabar has developed two commands to add and delete users, *addlogin* and *dellogin.* These commands prompt you for all necessary information to add or delete the user access, then update the files necessary for adding or deleting users, including:

- Passwd file
- Group file
- Security files, if they exist

Jenzabar has also developed standard user names to assist you in specifying a user account's access permissions. For example, you specify the *admit* login name for an Admissions office user. This standard user name contains the proper menu path access and permissions to use Admissions programs.

The password file (/etc/password) defines the primary group ID. This ID is required and is the same as the ID used for the standard user name associated with each user. The group file (/etc/group) controls any secondary groups the user is in. A user can be in as many secondary groups as you choose to assign. You use groups to control database access and UNIX file access for a user. The process that verifies whether an individual can access a file checks the ownership of the file and the permissions that are granted to all the groups.

Groups are also used in the database to control table access. The schema definitions define the permissions to tables allowed for each group. Therefore, if you want to know which groups provide update/view access to specific screens, you can look at the schema definition. However, keep in mind that most screens access more than one table and a user requires access entries for each table.

Note: A few tables exist that allow further restrictions to access permissions, especially in the registrar and financial modules.

Note: You can use the 'groups' command to view the groups an individual is in.

To complete the process to add or delete a user, you must use the *dbadmin* program to instruct the database to add or delete a user's permission to access the tables and data.

Note: For more information on the dbadmin program, see Database Administration Program in the *CX System Reference Technical Manual*.

Note: For further references, see the following sections in the UNIX manuals: *passwd(1), passwd(5), and group(5).*

User Account Requirements

CX requires that every user that accesses the system must have a user account. A complete user account includes the following on the system:

- A /etc/password file entry
- A /etc/group file entry in common group. A user can have more than one entry in the group file.
- A home directory with dot (.) files (e.g., .login, .cshrc, and .exrc files).

The user's entry in the /etc/passwd file must contain the following information:

- The login name
- An encrypted password or an encrypted password in a security file
- A user identification number (uid)

- The login group identification number (gid), which corresponds to a group in the /etc/group file
- Descriptive data, which includes the individual's name, office, and any other relevant information
- The user’s home directory path (/usr/carsids/logname)
- The login shell, including:
  - *menucsh* for menu users
  - */bin/csh* for shell users

Note: After you have added the new user to the /etc/passwd and /etc/group files, the system prompts you for the user's password.

Group Requirements

You assign users to groups in the /etc/group file. Each group in the file has a corresponding group identification number (gid) to which you add the user's login name. If a user has a group (gid) entry in the passwd file, you must also add the user's login name to the gid list in the group file. The following are the requirements for entering users in the group file:

- Add all users to the *common* group
- Add menu users to a login group
- Add users to other groups based on UNIX or INFORMIX permissions required to perform specific functions

Note: You can add a user in a multiple number of user groups, depending upon the operating system implementation.

A gid exists for every application process and access level within an application. For example: *finctrl* is for financial supervisors and *financial* is for financial entry.

Note: After you have added the new user to the /etc/passwd and /etc/group files, the system prompts you for the user's password.

Standard User Login Names

To enable optimal use of the capabilities associated with users and groups in UNIX and INFORMIX, Jenzabar has designed a set of standard login names with corresponding group and shell assignments, which provide the following benefits:

- Functional user names for menu users (e.g., admit for Admissions user) with login user and group identification numbers, home directories, and login shells.
- Each login sets the menupath for the user and any environment variables.
- Each user is entered in all the groups necessary to access INFORMIX data files.

When adding users to the system, CX requires that you create user logins that model the standard CX user logins. This ensures the assignment of the proper groups to each user.

Standard Login Names List

The following lists the standard CX user names with the corresponding login menu and login group.

Note: The home directory for each menu user and the Jenzabar system coordinator is /usr/carsids/loginname.

Note: Two logins, *backup* and *shutdown*, exist for the purpose of their implied function. The user *carsu* is a special database super user. Use it sparingly and only when CX deems that using it is necessary in order to access data files.

User name	Login menu name	Group(s)
admit	admit/menudesc.mnuadmissions	
registstudent	regist/menudesc.mnu registrar	
finaidstudent	finaid/menudesc.mnufin_aid	
stuacctfiscal	stubill/menudesc.mnustubill	
stuservstudent	stuserv/menudesc.mnustuserv	
placemntstudent	placement/menudesc.mnuplacement	
developinstdev	menudesc.mnudevelopment	
alumniinstdev	alumni/menudesc.mnualumni	
publicityinstdev	publicity/menudesc.mnupublicity	
donorinstdev	donoracct/menudesc.mnudonor_acct	
controllfiscal	menudesc.mnuaccounting	
cashierfiscal	cashier/menudesc.mnucashier	
acctspayfiscal	acctspay/menudesc.mnuaccts_pay	
acctsrecfiscal	acctsrecv/menudesc.mnuaccts_recv	
payrollfiscal	payroll/menudesc.mnupayroll	
backupstaff		
shutdownstaff		
coordcommon		
carsucommon		

Home Directory Permissions

Ideally, user home directories should be 700 drwx-----to provide the greatest security to a personal area. However, since many programs run as ‘carsu’ and need to send output to the home directory, the user carsu also needs access to the user’s home directory. Therefore, set up the home directories with 770 and a group that carsu is in. You can then safely remove the priveleges for others on the home directories.

Users Permissions to Schemas

CX uses the INFORMIX permissions within each schema. This includes both table and column permissions for user, group and public categories. In these permission declarations within each schema, the text user name or group name is included. When the *dbbuild* program updates the dictionary information, those names are actually stored as uid or gid (user identification numbers or group identification numbers) from the /etc/passwd and /etc/group files, as appropriate.

Changing each schema and rebuilding those files is extremely time consuming when you use the specific application user name instead of the standard CX user names. In addition, a change in personnel results in the whole process having to be repeated. Since INFORMIX stores the identification number internally, you only need to create the new application user entry in the passwd file with the same user identification number as the standard name.

By having the new entry precede the existing standard CX entry, searches on the user identification number (to find the user name) result in locating the user name. Therefore, when

you build each schema and search the passwd file by user name, you will find a number that is the same number as the application user name. For example, the standard CX user name, coord, might have a user identification number of 340. If you want the user name of jane as the coordinator, all that is necessary is to include the jane entry before the coord entry in the passwd file (both have a user identification number equal to 340).

Accessing Multiple Database Systems

When your institution has multiple databases for a user to access, the following occurs:

- When logging into the system, a login menu appears for the user to select the database.
- When exiting from the CX menu, the login menu appears for the user to select another database or exit from the system.

Note: Some users may require access to a training database, a corporate or main database, or another school's database. Contact your Account Manager for specific information about setting up your system for multiple databases.

The dbusers.s File

Use the dbusers.s file to add logins for users who need access to multiple databases. There are two main fields in this file:

- dbname - This field contains the name of the database.
- userlist - This field contains a comma separated list of login names. Each login name listed for an entry has access to the database contained in the dbname field.

Example:

```
Format consists of the following colon-separated fields:
#
#       dbname:userlist
#
#       Where:
#               dbname          - name of the database
#               userlist        - comma-separated list of users
#                                 who have access to database "dbname"
#
#       NOTE: "ALL" is a special "dbname" value which indicates that the
#            associated users have access to all databases in the system
#
ALL:
betah:acadrec,acctspay,acctsrec,admit,alumni,anthony,bill,brown,bryan
betah:carsu,carter,caryle,cashier,chris,cisc,comerota,controll,coord,craig
betah:dalem,darin,dave,develop,dianne,dick,donna,donor,duane,ed
betah:eldon,eric,finaid,fisher,frank,frey,gary,gene,gerry,greg
betah:hale,harold,harry,huizenga,informix,ivr,jack,james,jay,jeanne
betah:jeff,jim,john,johnj,johnp,karen,kelly,ken,kenw,kevin
betah:kim,larry,laws,mahen,mande,mary,mike,nacu,nancy,nelsen,patricia
```

A user may be specified in more than one one dbuser entry indicating access to more than one database. If a user has more than one entry in the dbusers file, he will see a login menu from which to select the desired database, as shown in the following example:

```
Please choose your database for the current
                 session from those listed below:

                 0.  LOGOUT (exit the system)

                 1.  Betai5 shared release
                 2.  Betai7 shared release
                 3.  Devi5 shared release
                 4.  Devi7 shared release

           Enter the number for your selection:
```

Adding New Users

The following lists the steps to follow to use the *addlogin* command. When you execute *addlogin*, the system prompts you for all necessary information.

Note: In order for all the sub-processes to execute properly, the coordinator must be logged in as the super user.

1. Enter at the system prompt: **addlogin**. The system displays the following prompt: "Enter new login user name to add:"
2. Enter new login user name to be added (e.g., **jane**). The system displays the following prompt: "Enter long description:"

   Note: The Initial .login, .cshrc files come from /usr/carsids/skel

   The system displays a list of 'model' users.

3. Enter the existing standard CX user name for reference (e.g., regist).

   The system displays a list of possible groups and the current groups for the user, and displays the following prompt: "Enter additional groups:"

4. Enter any additional groups into which you want to put the user.

   The system displays the following prompt: "Please enter the login menu name for <name> (<CR> for no menu):"

5. Do you want to enter the login menu name for the user?
   - If yes, enter the login menu name for the user (e.g., student/menudesc.mnu).
   - If no, press **<Enter>**.

   The system displays the following prompt: "Allow user access to the IQ Report Writer:"

6. Do you want to allow the user access to the IQ Report Writer?
   - If yes, enter **Y**.
   - If no, enter **N**.

   The system displays the following messages:

   - "About to update /etc/passwd, /etc/group, and ~Jane with the following:
   - Usernamejane
   - Groupscommon student registrar display_reg
   - DescriptionJane P. Smith
   - Login shell /usr/carsi/install/utl/menucsh
   - Login menustudent/menudesc.mnu
   - IQ accessYes
   - Build IQ dictionary:Yes"

   The system displays the following prompt: :Continue (y or n, default=n)"

7. Do you want to continue adding the user?
   - If yes, enter **Y**.
   - If no, enter **N**.

   The system displays the following prompt: "New password:"

8. Enter a password for the user. The system displays the following prompt: "Re-enter new password:"

9. Re-enter the password. The system displays the following message (example): “Now initializing user 'jane' with login group 'registrar'“

   The system displays the following message: “Now setting menu student/menudesc. mnu for user 'jane'“

   The system displays the following prompt: “Add another user (y or n) ? [y]”

10. Do you want to enter another user?
    - If yes, enter **Y**, and go to step 2.
    - If no, enter **N**.

      Note: If the user name already exists in /etc/passwd, an appropriate message displays and the process terminates.

Adding a User Needing Multiple Permissions

If a functional user needs to perform the functions of several of CX standard users, you can perform one of the following:

- Create logins to model all appropriate standard logins
- Edit /etc/group to make sure that the user is in all necessary groups.

CAUTION: Jenzabar recommends that you create logins to model all appropriate standard logins to ensure that each user is included in the proper groups for INFORMIX permissions. However, if you choose to edit /etc/group (one login, multiple functions), then call Jenzabar Support Services to make sure that the user is in all the groups necessary for data permissions.

Restricting a User’s Access to Menus

If you want to restrict a user’s access to the system to one menu, you must enter the appropriate login menu pathname (relative to the CX master menu) when executing the *addlogin* command. For example, if you are restricting jane to the student menu, enter student/menudesc.mnu at the login menu prompt. Otherwise, press the Return key at the <CR> prompt, enabling the user to have access to the system from the shell. The login name, standard CX user, and login menu path display for the purpose of verifying this information.

User Login Initialization

You must initialize the home directory files: .login, .cshrc, and other dot (.) files located in the user's home directory as the final step in the addlogin procedure. After you complete this step, the new user account is ready for use.

Adding a Super User

You must be root or SU to add a new super user. To add a user to the list of those with super user privileges, follow these steps:

1. Edit the .gurus file located in / (root directory).
2. Add the login of the new super user to the .gurus file.

CAUTION: Because of the extensive access privileges, you should restrict the number of super users to as few as possible.

Removing User Accounts

When it is possible to do so, avoid deleting users from the system. In the case of those users who access financial data, the user's login identification number is stored in the Voucher Database record (vch_rec file). GLQUERY displays the name from the passwd file with each voucher. If users are deleted, accounting audit trails become invalid. If, however, a new user login was created incorrectly, you can execute the *dellogin* command by logging on as super user.

The following lists the steps to follow to use the *dellogin* command. When you execute *dellogin*, the system prompts you for all necessary information.

1. Enter at the system prompt: **dellogin**. The system displays the following prompt: "Enter login username to be deleted:"
2. Enter login user name to be deleted (e.g., **jane**). The system displays the following prompt: "Enter username to be used as new owner:"
3. Do you want to enter a username to be used as a new owner?
   - If yes, enter the username.
   - If no, press **<Enter>**.

   The system displays the following message:

   "of files currently owned by jane or just

   CR> to indicate no file ownership changes

   r - remove home directory

   p - prompt for removal of each file

   <CR> - to not remove any files"

   The system displays the following prompt: "Enter Choice (r, p, or <CR>):"

   CAUTION: If you want to remove the home directory and also have the system remove every file and directory in the home directory, use the command rm -r. When using this command, proceed with extreme caution.
4. Do you want to remove the user's home directory, and *not* be prompted before the system deletes each file?
   - If yes, do the following:
     - Enter **r**. The system displays the following prompt: "Remove IQ directory for user 'jane' (y or n)? [n]"
     - Do you want to remove the user's IQ directory?
       - If yes, enter **y**.
       - If no, enter **n**.
   - The system displays the following message: "Username 'jane' has been successfully deleted"
   - If no, go to step 5.
5. Do you want to remove the user's home directory, and be prompted before the system deletes each file?
   - If yes, enter **p**. The system prompts you before deleting each file. The system displays the following message: "Username 'jane' has been successfully deleted"
   - If no, press **<Enter>**. The system does not remove any files.

Security for Jenzabar CX Data

Introduction

The CX database contains confidential information; therefore, you must implement security procedures to protect that data. At some point the security measures may encroach on the usability of the system. CX maintains a good balance between security and system access to valid users.

Types of Individuals Attempting Access

There are two overall categories of individuals attempting access:

- Valid users: employees or trusted individuals with a current login name and password. These users may utilize the system in normal operations.
- System abusers: individuals that are probably not assigned current login names or passwords; although, employees and other users with valid login names and passwords may become system abusers. External abusers will probably gain access to the system using a modem or by having access to a terminal in a restricted area. These system abusers can range from the curious to the vicious. System security measures intended to stop this type of system penetration include: modem restriction, password changing, login process time-out upon too many errors, and reporting login attempts.

External abusers are dangerous, but a disgruntled employee that was once trusted, could be a considerable threat. This type of system assailant is dangerous because they had valid access to the system. Data can be maliciously changed by this user within the constraints of their normal operation. Changing of passwords, menu restrictions, and database permissions are the best defense against this type of system abuse. The menu restrictions, changing of passwords, and the database permissions restrict the curious employee to the desired areas of system usage & access.

Physical Access

You must consistently restrict physical access of the system for both the computer center and each individual terminal. Having the computer center and most terminals secured is not enough. If one terminal is too easily available, the system is at risk. To restrict access ensure that users lock their offices, the computer staff locks the computer center doors, and access to backup copies of the system on tape, or disk.

Modem Access

Another method of access to your system is through a modem. Modems can be very convenient and allow CX personnel to answer questions promptly, but they can also be a point of unauthorized entry into your system. Review the activity logs watching the outside modem usage. Turn off or unplug the modem at night. If CX personnel need access to the system, they can make arrangements ahead of time.

Login Usernames and Passwords

After the physical access restrictions, login usernames and passwords are the second level of security. You assign a username to each valid user of the system. The system recognizes this name when logging on and requests the appropriate password from the user. After the user enters a password, the system validates the username and password for accuracy.

Password Maintenance

The system maintains passwords within UNIX in encrypted form only; the passwords are not kept in plain text (readable) anywhere within the system. The standard CX and UNIX usernames are generally not changed; although, you can change the individual application usernames, if desired.

Changing Passwords

You should change passwords frequently. The Jenzabar system coordinator should see that the users change their passwords as often as possible, and the Jenzabar system coordinator or system manager can assign passwords.

Passwords should never be obvious associations. They should not be names (first, last, nicknames, etc), of wives, husbands, children, etc. Users should avoid, for example, phone numbers, Social Security numbers, and street numbers (& names). Passwords can be pronounceable strings of words (or vowels & consonants) and can contain upper and lower case letters. In addition, Jenzabar recommends that you include punctuation characters.

Login Procedures

You accomplish the next level of access control by not allowing the application users to change their login procedure files: .login and .cshrc. You accomplish this by not allowing writes to those files by any user other than the system manager.

Note: The .login and .cshrc files are the initial files executed by the login shell program. By not allowing writes to these files and thus disallowing the user to change their environment at will, you control users' system access.

Menu System

CX application users are trapped in menus. This level of protection not only protects the system from malicious users but also insulates the end user from the details of the UNIX C-shell. The menu system forms a user-friendly interface for the application user. Menus have two security aspects:

- When the user logs in to a specific point in the menu tree structure, the menu does not permit the user to traverse upward in the structure.
- Passwords on various menu options help ensure that authorized user access is maintained. You can also set up menus to automatically log off users if they have not utilized the menu within a given period of time (e.g., 30 minutes). This feature eliminates the possibility of terminals being logged on overnight.

Note: If the user needs to process from a terminal, and leave it on overnight, the menu processor can lock the screen and not allow access until the correct password is entered.

Monitoring System Performance

Introduction

These pages outline steps to follow to monitor the state of the system.

The Process of Gathering System Information

When certain processes appear to be running much slower than normal or system wide performance has dropped below a reasonable level for the amount of load on the system, the following procedure should be followed to gather important information regarding the state of the system.

- At a time when no one else is on the system but the system is in multiuser mode, perform step 1 below. Label the results: **LOW ACTIVITY**
- At a time when there is moderate activity with reasonable response time (4 to 8 users), perform step 1 below. Label the results: **MODERATE ACTIVITY**
- At a time when the system does not appear to be giving a reasonable response time, perform all steps below. Label the results: **HIGH ACTIVITY**

Note: Send the results of the steps to Jenzabar.

Snap-Shot of System Activity

The following lists the steps to getting a snap shot of system activity.

1. Get a snap-shot of system activity and resource usage on a printer:

   Example: # snapsys lpr

   Note: The command produces ps, w, and vmstat listings.

2. Get a list of current system configurations on a printer:

   Example: # snapconfig lpr

   Note: The command produces listings of /usr/lib/crontab and ttys, ttytype, printers, and fstab in /etc.

3. Ask the system users if they are having unreasonably slow response time. Identify on the w or ps list each user that is having slow response time. Gather the following specific information:
   - What the user was doing
   - What part of the process was exceptionally slow
   - Classify each problem as either: Query time, Adding time, Update time, Character echo time, Spooler time, or Program load time.
   - Any additional helpful information.
4. Using the output from step 1, identify on the ps output the tty of each slow process. If the tty line is one that goes through a phone system or a port contender, indicate that fact and try to identify which phone number or port is being used.

Testing Spooled Printer Devices

Introduction

These pages provide steps for solving problems that can occur with spooled printer devices. Spooled printer problems can come from many sources and are difficult to track down. The CX spooling software is only one aspect in the process to receive printed results. For example, the operating system and hardware communication must be working properly before the spooling software can function.

To verify the operating system and hardware communication, you might need to perform some initialization before accessing the printer. The *lpinit* command provides the ability to initialize the printer communication in the same way that as the CX spooling software. Thus, by using the *lpinit* command, you can use the printer outside the spooling software.

Note: For information about creating spoolers, see *mkspooler* in the appendix.

Jenzabar CX Print Spooler

Before testing any printer device you should understand what the spooling software does to produce output. Several actions must take place to provide the ability to share a printer device between many users, which is the key function of a spooler. The following lists those actions:

Queuing a Print Job

The first step in sending output to a printer is placing the output into the printer's job queue. To perform this action, you use the *lpr* command, located in $CARSPATH/install/utl. This command takes the file, or standard input, and records the data in the spool directory for that printer.

For example, if the printer name is lpr1, the spool directory would be $CARSPATH/spool/lpr1. Also, the command links the file to the LPR program, $CARSPATH/install/utl/lpr1. This allows printer name to act as the command to send a job to that printer.

The process of queuing a job records all the information needed to send that particular data to the printer. The queued jobs will then be sent to the printer, one at a time, by the printer daemon, LPD.

The Printer Daemon

The LPR process executes the LPD process, known as the printer daemon, for the specific printer being used.

1. LPD takes jobs sequentially from that printer's spool directory and sends them to the printer device.
2. The daemon will first check to make sure no other LPD process is currently running for that printer.
   - If a runlock file exists in the spool directory, LPD immediately exits.
   - If a runlock file does not exist in the spool directory, it creates the runlock file to prevent any other LPD process from running on that spool directory.

3. The daemon will then open the printer device file located in the /dev directory. For example, for the lpr1 printer, the device would be /dev/lpr1. This file is a link to the specific device file corresponding to the hardware port to which the printer is attached. This may be either a serial or parallel device.
4. After opening the device an initialization command is executed for that printer. This command is located in the file CARSPATH/install/sys/lib/prtab (see prtab manual entry). This file contains lines of the form *printer:command*. In our example, LPD would look for a line similar to the following:

   Example: lpr1:/bin/stty 9600 ...
5. If the line contains nothing after the colon (:), no command is issued. Otherwise, everything after the colon (:) is executed as a command. For serial printers, this is normally an STTY command to correctly set the baud rate, parity, newline and carriage return settings, etc. However, this may be ANY command on the system.
6. Once the device has been initialized, the data from the first job in the spool directory is written to that device by the output filter. The default output filter is LPF.

   Note: You can specify an output filter per spool queue in the $CARSPATH/install/sys/lib/proptions file, which will be used unless specified otherwise (see proptions manual entry). To override the default, you can pass the -o option to the spooler during execution. When the first job has been completely written, the second job is sent, and so on until no jobs remain in the spool. At that point, the device file is closed, the runlock file is removed, and LPD exits.

The Lpinit Command

The *lpinit* command mimics the initialization done by LPD, allows the user to execute commands from a shell, and releases the printer. By using this command to test a printer we can be sure that the printer device is set up like it would be if the spooling software were printing the job.

1. The first thing *lpinit* does once the printer has been specified is to disable the spooling for that printer. Perform this by executing the *idle* command through LPC. Jobs may still be queued for the printer, but none will print until the idle condition is removed. In this way, no spooled jobs will conflict with the testing.
2. After the spooler is idle, the initialization is done. If the printer is a serial printer (it is connected to a tty or rt device), the device is first opened to preserve any STTY settings done in the initialization. You perform this with a very long SLEEP command. As long as the SLEEP command is running, the device remains open. The initialization command from CARSPATH/install/sys/lib/prtab is then executed for the printer device.
3. When the initialization command is finished, an interactive csh shell is executed. This allows the user to execute any command while the printer is in its initialized state.

When the user exits from the interactive shell, the SLEEP command is displayed if it was executed, and the user is prompted for verification to kill it. The verification helps prevent LPINIT from inadvertently killing a job other than the SLEEP. The user also has the option to leave the SLEEP command running. Finally, the spooler is started, removing the idle condition.

Testing a Printer Using LPINIT

LPINIT can be used to determine whether the operating system and hardware communications are working properly by following a few simple steps.

Determine the Printer Name

The first step to testing the printer is to determine the name by which CX accesses it. The printer's name is:

- Displayed by the LPC command when checking the status of the spooler.
- The name of the command used to spool jobs for the printer.

You can determine the printer's name verifying the hardware connection from the printer to the communications port, and determining which device file is associated with that port. The device file should be linked with the name of the printer. Since the device file and port have the same inode number, you can use the *ls* command to find the two devices that are linked. Do the following:

1. Execute **ls -i <device>**, where <device> is the name of the device file corresponding to the communications port for that printer. This will return results similar to the following:

   Example: % **ls -i /dev/tty1p0**

   2314 /dev/tty1p0
2. Use the **find** command to find the linked files. Execute find /dev -inum <inode> -print, where <inode> is the number listed with the ls -i command.

   Example: % **find /dev -inum 2314 -print**

   /dev/lpr1

   /dev/tty1p0

Execute Lpinit for the Printer

After you determine the printer name, you use the *lpinit* command. When executing the *lpinit* command, you pass the name of the printer on the calling line. If you omit the line or enter a valid device name, *lpinit* ask for a printer name. The following is a sample of the *lpinit* dialog:

```
Enter the printer name:  lpr1
Waiting for the spooler to idle ... done.
'lpr1' has been initialized

Starting a new shell ... enter CTRL-D when you are finished.
lpinit:
```

Write to the Printer

After you begin the interactive shell, you can send test data to the printer.

The following are tests that you can use:

- Using the *lptest* command. It prints four pages of ASCII characters.

  Example: lpinit: lptest > /dev/lpr1
- Running a file listing to the printer.

  Example: lpinit: ls /dev > /dev/lpr1
- Sending any text file on the system to the printer.

  Example: lpinit: cat filename > /dev/lpr1

If these tests do not produce the correct results on the printer, the problem lies in the operating system or hardware communication.

If these tests do produce the correct results, the problem is related to the spooling of print jobs.

Release the Printer

When you have completed the tests, enter **<Ctrl-d>** to exit from the interactive shell. If you used a SLEEP command to open the printer device, you should kill the spooler when LPINIT prompts for it. The spooler will be re-activated before LPINIT exits. Do the following:

1. Enter: **lpinit: EOD**
2. Ready to kill the following sleep command:

   PID TTY TIME COMMAND
   17615 tty3p3 0:00 sleep

   The system displays the following prompt:
   "Do you want to kill this job?"
3. Enter: **y**
   The system displays the following messages: "Starting the spooler ... done."

Setting Up a Slave Printer

Introduction

You can set up a slave printer for a dedicated terminal, which has a slave printer port, or for a PC using a terminal emulator and the local printer. You can make settings for individual logins or for all logins.

Slavecap.s File

Part of the settings you must make for a slave printer involve the *slavecap.s* file. You must make an entry for the printer that you will use as a slave for the user's terminal. The following are examples of terminal and printer entries in the *slavecap.s* file:

Note: The ^[characters in the examples below represent the **Escape** command. You enter the Escape command in vi by typing **<Ctrl-v>** then **Esc**.

Example of a terminal entry:

```
# Shareware Procomm vt100 emulation
vt100-pro:open:^[[5i
vt100-pro:close:^[[4i
```

The *open* command should be the code that tells your terminal or terminal emulator to stop displaying output on the screen and start sending it to the back port instead. The *close* command should be the code that tells your terminal or terminal emulator to stop sending output to the back port and start displaying it back on the screen.

Example of printer entry:

```
# Okidata 82 A
oki82a:6lpi:^[6
oki82a:8lpi:^[8
oki82a:condense:^]
oki82a:doublewide:^_
oki82a:pica:^^
oki82a:reset:^X^^^[6
```

The entries above are printer instructions as specified in the printer's manual (e.g., 6 or 8 lines per inch).

Slave Environment Variable

For each user who will use the slave printer, you must set the *slave* environment variable to the appropriate printer name.

Example: setenv slave oki82a

You can set the *slave* environment variable for individual users and for all menu users:

Individual users

For shell users, you set the slave environment variable in the user's .cshrc file. For menu users, you set the slave environment variable in the user's .login file.

Global menu users

You set the slave environment variable in the login.s file.

Procedure

Do the following to set up a slave printer for a terminal or terminal emulator.

1. Check out the $CARSPATH/usr/carsi/system/etc/slavecap.s file.
2. Using vi, make an entry in the *slavecap.s* file for the printer you will use as a slave and for the terminal (or a value for the TERM environment variable) to which you want to attach the slave printer. (See above for an example of entries to the slavecap.s file.)
3. Check in and install the slavecap.s file.
4. Do you want to set the variable in an individual user's login file or in the global login file for all menu users? (e.g., **setenv slave oki82a**)
   - If you want to set a shell user's login, enter the slave environment variable in the user's .cshrc file.
   - If you want to set a menu user's login, enter the slave environment variable in the user's .login file.
   - If you want to set menu users globally, go to step 5.
5. Do the following to enter the slave environment variable in the global login file.
   - Check out the $CARSPATH/skel/login.s file.
   - Using vi, enter the variable in the file.
   - Check in and install the file.

Using Tape Conversion

Introduction

The Tape Conversion program (*tpconvert*) reads its input and updates (adds, updates, or deletes records) the database based on the conversion specified by the configuration file. The program expects the input file to contain fixed length, fixed position, newline terminated records of ASCII data. The program has one required option, -c, to specify the name of the conversion definition file.

Tpconvert has varying levels of output messages but by default provides an expanded version of an unload ASCII output. This output, in addition to describing the record buffer, identifies both the file being affected and whether the action taken was a Read, Update, Add, or Delete.

Note: Prior to the availability of the Tape Conversion program, you had to do the following to move data into the database from outside:

- You needed to create temporary schema files with character type fields that exactly match the format of the data to be loaded.
- After loading the data into the temporary schemas, you used INFORMER to move and convert the data into the correct database records and fields.

Note: The Tape Conversion program mostly eliminates the need to use the temporary schema approach.

Other Uses for Tpconvert

While the CX created *tpconvert* for use in incorporating data from outside agencies into the CX database, you can use the program in many other applications as well. Some possibilities include:

- Converting data from tape such as ACT, SAT, Student Search, Financial Aid, etc.
- Converting data from an existing system to CX.
- Simplifying repetitive additions/updates of records.
- Loading outside program results (from spread sheets, for example) into the database.

Some example applications of *tpconvert* are:

- The Admissions office has a number of students to whom they want to send catalogs. The office secretary can type the list of student IDs into a text file and run *tpconvert* on this file to add the desired Contact records.
- The Dean of students runs a report listing all students who are candidates for academic probation. After eliminating certain students from the report list, the dean enters the list into a file and runs *tpconvert* on the file to add Applicable Enrollment Status and Contact records to the system.

Program Parameters

Tpconvert has several parameters that you can specify. The following is the syntax for running *tpconvert*.

Example: tpconvert -c config_file [-P config_path] [-b buffer_size]
[-o output_level] [-p] [-r] [-f]

The following lists the parameters and describes what they signify to *tpconvert*.

-c (config_file)
The name of configuration file to be used

-P (config_path)
The configuration file directory. The default is (.)

-b (buffer_size)
The maximum size of input and record buffers, default is 1024

-o (output_level)
The level of output desired (the higher the number, the greater the detail)

Note: None=0

Note: Normal level=1,3,5,

Note: Debug level=10,12,14,20,22,24

-p
Parse the config file only, do not process input

-r
Report only, do not update the database

Note: Use with -o to increase the level of detail.

-f
Update from input using previous report output

What the Configuration File Does

The configuration file tells *tpconvert*:

- The positions within an input record that contain data of interest to you
- What to do with that data, whether to do one of the following:
  - Copy that input data into a particular field
  - Perform some type of testing or conversion on the data.

What a Configuration File Looks Like

The configuration file consists of a title line optionally followed by an input command specifier. This is followed by one or more file definitions. A file definition consists of a file definition line followed by one or more field definition lines. Comments in the configuration file are delimited by the open and close brace characters. (i.e. { and }).

Configuration File Examples

Two examples of a configuration file are provided below with explanations of each of the file's components.

Simple Configuration File Example

```
{ Comments are delimited by open and close braces. }
Title:              "High School Tape Conversion Example"
File:               id_rec using ss_no update
ss_no:              trim(1-6);
name:               capword(7-36);
addr_line1:         capword(37-66);
addr_line2:         capword(67-96);
city:               capword(97-112);
state:              upshift(114-115);
zip:                117-121;
id_aa_code:         "PERM";
correct_addr:       "Y";
deceased:           "N";
mail:               "Y";
add_date:           TODAY;
File:               schtape_rec using schtape_ceeb add
schtape_ceeb:       1-6;
schtape_ceeb_ss:    1-6;
schtape_name:       capword(7-36);
schtape_addr1:      capword(37-66);
schtape_addr2:      capword(67-96);
schtape_city:       capword(97-112);
schtape_st:         114-115;
schtape_zip:        117-121;
schtape_foreign:    127-127;
schtape_f_addr:     97-126;
schtape_upd_date:   concat("06","/","13","/","89");
schtape_upd_ind:    "N";
schtape_inactive:   "N";
```

Complex Configuration File Example

```
Title:              "ACT High School Tape Conversion"
Input:              "dd if=TAPE_DEF bs=1600 | unblock 160"
File:               id_rec
Key:                ss_no
Mode:               update
                    add
ss_no:              if(store("NOT_DELETED",
                            cmpstr(134-134 != "D")),
                       1-6,
                       " ");
name:               capword(7-36);
addr_line1:         capword(37-66);
addr_line2:         capword(67-96);
city:               if(store("NOT_FOREIGN",
                            cmpstr(127-127 != "F")),
                       capword(97-112),
                       97-120);
state:              if(recall("NOT_FOREIGN"),
                       114-115,
                       "  ");
zip:                if(recall("NOT_FOREIGN"),
                       117-121,
                       121-126);
id_aa_code:         if(or(cmpstr(127-127 = "F"),
                          or(cmpstr(61-66 > "      "),
                             cmpstr(91-96 > "      "))),
                       "ABBR",
                       "PERM");
correct_addr:       "Y";
deceased:           if(recall("NOT_DELETED"),
                       "N",
                       "Y");
add_date:           if(or(cmpstr(134-134 = " "),
                          cmpstr(134-134 = "A")),
                       store("DATE",
                             if(cmpstr(128-133 = "      "),
                                TODAY,
                                concat(128-129,
                                       "/",
                                       130-131,
                                       "/",
                                       132-133))),
                       "");
ofc_add_by:         "TAPE";
last_upd_date:      if(or(cmpstr(134-134 = "U"),
```

```
                        cmpstr(134-134 = "D")),
                     recall("DATE"),
                     "");
File:              aa_rec
Key:               aa_prim
                   add    if cmpstr(getfld("id_rec.id_aa_code.1") = "ABBR")
                   update if cmpstr(getfld("id_rec.id_aa_code.1") = "ABBR")
aa_id:             getfld("id_rec.id_no.1");
aa_line1:          capword(37-66);
aa_line2:          capword(67-96);
aa_line3:          if(cmpstr(127-127 = "F"),
                     capword(97-126),
                     "");
aa_city:           if(recall("NOT_FOREIGN"),
                     capword(97-112),
                     "");
aa_st:             if(recall("NOT_FOREIGN"),
                     114-115,
                     "");
aa_zip:            if(recall("NOT_FOREIGN"),
                     117-121,
                     "");
aa_code:           "PERM";
aa_beg_date:       "00/00/00";
File:              sch_rec
Key:               sch_ceeb
                   update
                   add
sch_id:            getfld("id_rec.id_no.1");
sch_ceeb:          if(recall("NOT_DELETED"),
                     1-6,
                     "0");
sch_type:          if(cmpstr(1-2 = "00"),
                     "JRC ",
                     "HS  ");
sch_last_upd_date: recall("DATE");
```

Configuration File Definitions

The following defines the elements of a configuration file.

Note: All of the elements defined below are reserved words; therefore, they cannot be used in any manner other than as described.

Title Line

Consists of the word *Title* followed by a colon and a description string. The title line is a required element of the configuration file.

Example: Title: Sample Conversion Definition File

Input Line

Consists of the word *Input* followed by a colon and a double quoted command string.

Example: Input: "dd if=/dev/rmt/0h bs=800 | unblock 80"

The input line is optional; if you specify the line, and do not redirect the input to the Tape Conversion program on the command line, *tpconvert* uses the output from the specified command as input. If you do not specify the input line, or *tpconvert*'s input is not redirected on the command line, *tpconvert* uses *stdin* as input.

File Definition Line

Defines the database file that contains the fields to be loaded with data from the input. You can use the same database file in multiple file definitions. The file definition line has the following syntax:

```
File: DBFILENAME
              [using KEYNAME]  or [Key: KEYNAME]
              [Mode:]
              [skip [if FUNCTION]]
              [update [all] [if FUNCTION]]
              [add  [if FUNCTION]]
              [delete [all|dups] [if FUNCTION]]
Please Note: The order specified above is significant.
where: DBFILENAME is: the name of the database file to be affected.
       using      is: an indicator that the KEYNAME is specified.
       KEYNAME    is: the name of the key to be used to locate records.
                      If the KEYNAME is specified then a find
                      operation will always take place whether or
                      not the find operation is specified.  (The key
                      is optional only if the operation is 'add').  It
                      is not necessary to use the same key on the file
                      across multiple file definitions.
       Mode:      is: available only for readability.  It provides the
                      start of the action specification.
       skip       is: a reserved word providing the ability to specify
                      a condition to be executed before a database
                      find operation is performed.  This provides a
                      method of filtering out input records of no
                      interest.  If the condition is satisfied,
                      all processing on that input record is skipped.
One of the following three words is REQUIRED.
       update     is: an action described below.
       add        is: an action described below.
       delete     is: an action described below.
       if         is: an indicator that the following condition function
                      must be true before performing the action.
       FUNCTION   is: any combination of the functions described below
                      (a non-zero numeric result is required to continue).
```

Field Definition Line

Defines what database field is to be loaded with the specified data from the input. You specify any conversion in the field definition. The field definition line has the following syntax:

```
FIELDNAME:              VALUE;
where:  FIELDNAME is: the name of the field in the database.
        VALUE     is: an offset range (i.e. 1-6),
                      a double quoted string (i.e. "PERM", "Y"),
                      a specially recognized word, or
                      a function described below.
```

File Operations

The following are the file operations for *tpconvert*.

Add

Adds a new record. The Add operation does not require a key specification.

- If you specify a key, *tpconvert* adds a new record only if:
  - The search for a record matching the key value fails
  - The IF condition, if there is one, is met

  Note: *Tpconvert* does not update a record matching a specified key in any way.
- If you do not specify the key, *tpconvert* adds a new record only if the IF condition, if there is one, is met.

Update

Updates the first record found matching the specified key when the specified IF condition has been met.

- If *tpconvert* finds no record, or the condition is not satisfied, the program does not update the record.
- If you specify the All option to the Update command, *tpconvert* searches for additional records with the same key. *Tpconvert* updates these additional records assuming that the specified condition is also met.

Delete

Used to remove the first record found matching the specified key when the specified IF condition has been met.

- If *tpconvert* finds no record, or the condition is not satisfied, the program does not delete the record.
- If you specify the All option to the Update command, *tpconvert* searches for additional records with same key. *Tpconvert* deletes these additional records assuming that the specified condition is also met.
- The Dups option alters the operation of the delete command so that the first record found matching the key and satisfying the condition is *not* deleted and all subsequent records found matching the key and satisfying the condition *are* deleted.

Field Values

The following are field values that *tpconvert* searches for in the input record.

Offset Ranges

Specified as the beginning and ending positions from the input record that contain information of interest to the conversion program. The format is:

```
>>
begpos#-endpos#
For example:
     10-24   retrieves the 15 characters from position 10 through
             position 24 from the input record buffer.
    25-25    retrieves the 1 character from position 25.
```

Strings (double quoted)

Used when information is not explicitly in the input data, but may be inferred by the type of conversion being done. Often when adding records, fields may just be initialized. For example, when adding ID records, the *deceased* field should be set to N..

> **Note:** The quotes are not necessary if there are not any spaces or commas in the string, and the string is not a Null string.

Null Strings

Used when the original value of the field should not be affected. Null strings are a special version of a double quoted string. Null strings may be explicitly stated as the result of whether a specified condition has been met.

> **Note:** Null strings could be created from the trim function.

Special Words

Used as specially recognized words used just like strings. These are:

TODAY

Expands to today's date in *mm/dd/yy* format.

MONTH

Expands to the current two digit month, *mm*.

DAY
Expands to the current two digit day, *dd*.

YEAR
Expands to the current two digit year, *yy*.

UID
Expands to the system ID number for the user.

GID
Expands to the system group number for the user.

LENGTH
Expands to the length of the current database field.

Functions
Provide additional capabilities dealing with the conversion of data. All functions have been written to expect comma or space separated string parameters and return string results. A function parameter may be a numeric range (i.e., 5-12, 467-490, 45-60), a double quoted string (e.g., PERM, ABBR), or another function. Functions may be nested up to 10 levels deep at any one time.

Note: Although the functions all take string parameters, many of the parameters are expected to be numeric strings. The functions convert these parameters to numbers and use the numeric value. It is important that the correct numeric strings are passed to these functions. For example: 123ABC would evaluate to 123; but ABC123 would evaluate to 0.

Adding Functions

Note: This section is for Jenzabar internal use only.

You can add additional functions as new needs arise. To add a new function, you must modify the following files:

def.c
Make the following modifications:
- Add function type declaration
- Add function to definition array

funct.c
Place where actual function code is located

Note: Functions must allocate space for the return string and should not modify the string passed to the function.

tpgram.y
Make the following modifications:
- Add a new token definition
- Add newly added token to list of valid functions

tplex.l
Make the following modifications:
- Add new function name to be looked for
- Add a return new function token

Function Parameters

The following are the function parameters that *tpconvert* searches for in the input record.

And (cond1, cond2)
Returns either 1 or 0. If both cond1 and cond2 evaluate to non-zero, 1 is the result. Otherwise, a 0 is returned.

Capword (str1)
Upshifts the first alphabetic character of each space-separated word within the string and downshifts the remaining characters of each word. Currently, the capword function does not have any special words that are retained in uppercase. An example of words that capword would convert would be name suffixes such as: II - the second, III - the third, etc. In addition to name suffixes, state codes would also be converted.

Cmpnum (num1 op num2)
Compares two numeric string parameters. If the comparison is true, 1 is returned; otherwise, 0 is returned. The recognized comparison operators are:

- = (equal to)
- != (not equal to)
- > (greater than)
- < (less than)
- >= (greater than or equal to)
- <= (less than or equal to)

Cmpstr (str1 op str2)
Compares two string parameters. If the comparison is true, 1 is returned; otherwise, 0 is returned. The recognized comparison operators are:

- = (equal to)
- != (not equal to)
- > (greater than)
- < (less than)
- >= (greater than or equal to)
- <= (less than or equal to)

Cobol(str1, format)
Decodes a numeric field with the sign embedded in the last character of the string. For example, 123A translates to +1231 while 123J is -1231. The function is passed a numeric string along with a format string specifying how the string is to be converted. The function does the following:

- Strips Leading zeros from the numeric string unless the first character of the string is a zero.
- If the format string contains a plus sign, the function presides results with + for positive numbers and - for negative numbers.
- If the format string contains a negative sign, the function presides results with - for negative numbers and not sign for positive numbers.
- If the format string does not contain either a plus or a negative sign, the function presides results with no sign (the numeric string is considered to be unsigned)

Note: If the last character in the numeric string is a number instead of an embedded sign code, the result will be unsigned. This function should be used only if the sign is embedded in the last number of the string using the following mapping criteria:

Character	Result
{	+ 0
A	+ 1

B	+ 2
C	+ 3
D	+ 4
E	+ 5
F	+ 6
G	+ 7
H	+ 8
I	+ 9
}	- 0
J	- 1
K	- 2
L	- 3
M	- 4
N	- 5
O	- 6
P	- 7
Q	- 8
R	- 9

Concat(str1, ..., strN)

Concatenates all of its parameters together into one resultant string.

Note: This function is useful for converting a mmddyy date format on the tape. Use the function to piece together the component parts of the date along with the slash characters needed by the database date type.

Decr(num1)

Decrements the value of the numeric string passed by one and returns the resultant numeric string.

Note: This is an integer function. Any decimal portion of a number are not used or returned by this function.

Dwshift(str1)

Downshifts all alphabetic letters in a string.

Fldlen([file.]field)

Returns the length of the database field requested.

Note: Use this function only on character type fields since the length returned is the length of the field as stored in the database.

Getfld([file.]field[.buffer])

Retrieves the value of a field. The function expects one parameter string in the following format: *dbfilename.dbfieldname.buffernumber*.

Example: id_rec.id_no.1

The above example retrieves the ID number from the first ID record buffer. Assuming that only one id_rec file definition appears in the configuration file, you can leave off the buffer specification. If the dbfilename portion of the parameter is absent, then it is assumed to be the same as the current filename being processed. The field name is always required.

A set of buffers will be created for each table appearing in a File: definition within a configuration file. Each set of table buffers will begin numbering with one. Example: if id_rec has three file definitions, three buffers will be created labeled 1, 2, and 3. If ctc_rec has 5 file definitions in the same configuration file, 5 buffers will be created labeled 1, 2, 3, 4, and 5.

You can use the function in a situation similar to the following example: you are adding an ID record (id_rec) and a corresponding School record (sch_rec).

- While adding the ID record, the system serially assigns an ID number to the ID record.
- Before adding the corresponding School record, the getfld function retrieves that ID number to set the sch_id field in the School record.

If(cond1, true_str, false_str)

Allows different actions based on the result of a condition, which is the first parameter to the function. The second parameter is the action if the condition is true (non-zero), and the third parameter is the action if the condition is false (zero).

Index(str1, chr1)

Returns the first zero-relative character position in str1 that matches the character in chr1. For example, *index(ABBCD, B)* returns the character position of the first B.

Incr(num1)

Increments the value of the numeric string passed by one and returns the resulting numeric string.

Note: This is an integer function. Any decimal portion of a number is not used or returned by this function.

Leftj(str1, len)

Removes leading blanks from a string and then blank pads the string to a length specified.

Note: If the string is longer than the length specified, the string is truncated to that length.

Lookup(file.field, fld1, val1, fld2, val2, ...)

Finds the first record that meets the criteria specified for the fields: fld1, fld2, fld3, ... fldN. The function returns the value of the first specified parameter if a record is found; otherwise, the function returns a null string and prints a warning message.

- The fldn parameters should be fieldnames found in the file specified by the first parameter. The function ignores any fieldnames that are not part of the file and prints a warning message stating that one or more of the fields specified does not belong to the file.
- The valn parameters specify the values that make a record acceptable for the lookup function.

Note: Due to the special characters that can be used in the values, value parameters should be double-quoted. Value parameters of the following forms are recognized:

value	equal to value
=value	equal to value
>value	greater than value
>=value	greater than or equal to value
<value	less than value
<=value	less than or equal to value
!=value	not equal to value
value1:value2	range of valid values from value1 through value2
valpart	match the first partial value
valpart*valpart	match the first partial value and the second
#value1,value2	list of valid values

Op(num1 op num2)

Returns the result of executing the specified operation on the two numeric string parameters passed. The defined operations include:

- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)
- % **(modulo)**

Note: This is an integer function. Any decimal portion of a number will not be used or returned by this function.

Op2(num1 op num2)
Returns the result of executing the specified operation on the two decimal string parameters passed. The defined operations include:

- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)
- % **(modulo)**

Note: This is a decimal function. All decimal portions of the values will be considered and returned by this function.

Or(cond1, cond2)
Returns either 1 or 0. If either, cond1 or cond2, evaluate to non-zero then 1 is the result. Otherwise a 0 is returned.

Recall(label)
Returns the string saved under the specified label.

Note: See the store function for saving strings for use by recall.

Recno()
Returns the current input record number.

Rightj(str1, length)
Removes trailing blanks from a string and then fills in leading blanks to provide a resultant string of the length specified.

Note: If the passed string (excluding leading and trailing spaces) is longer than the destination field length, the resultant string is a truncation of the passed string after leading spaces have been excluded.

Rindex(str1, chr1)
Returns the last zero-relative character position in str1 that matches the character in chr1. For example, *rindex(ABBCD, B)* returns the character position of the second B.

Store(label, str1)
Saves the specified string and associates the label passed with that string.

Note: The saved string may be retrieved by passing the label to the recall function.

Strlen(str1)
Returns the length of the passed string.

Substr(str1, offset, length)
Returns a portion of the string passed to it.

- The offset parameter specifies the first zero-relative position of the string to be returned.
- The length parameter specifies the number of characters to be returned.

Note: If the length parameter is greater than the length of the resultant string, a string shorter than the specified length is returned.

Time()
Returns the current time of the system in a 8 character fixed position string with the following format: 11:03 pm.

Trans(str1, pattern_chars, map_chars)

Converts the characters of the passed string (first parameter) that match a character of the pattern string (second parameter) to the corresponding character of the map string (third parameter).

Example: trans(ABC, AC, CA) would return the string CBA.

Trim(str1)
Deletes leading and trailing blanks from a string. If a string consists only of blanks, a null string ("") is returned.

Upshift(str1)
Upshifts all lower-case letters of a string.

Output Levels

The following are the error message output levels that you can specify when running *tpconvert*.

Note: The program does not produce an exceptions report.

Normal Levels

The following are the error message only levels and error messages that each level displays.

Level 1

Normal line:
recno:[A |D |R |U] :filename | unload ascii of record buffer
where:

- A means Add
- D means Delete
- R means Read
- U means Update

Error line:
recno:[AE|RE|DE]:filename:error_no | unload ascii of record buffer where:

- AE means Add Error
- RE means Update Error
- DE means Delete Error
- error_no is the INFORMIX error number, the definition of which can be obtained by entering: **finderr <error_no>**

Level 2
Conversion Title String

Level 3
Error:fieldname:function_name:message

Level 4
Warning:fieldname:function_name:message

Level 5
Add|Update|Delete] filename Key:key_name Record Buffer:buffno

Level 7

- Record # skipped. Record met skip condition.
- Add skipped. Record was found for input record #.
- Update skipped. Record was not found for input record #.
- Update skipped. No fields needed to be changed.
- Delete skipped. Record was not found for input record #.

Expanded Output Levels

The following are the expanded output levels and the messages that each level displays.

Level 10

Field:fieldname

Level 12

Field:fieldname Value: !field_ret_value!

Level 14

Level:levelno Function:funct_name Return value:ret_value

Debugging Output Levels

The following are the debugging output levels and the messages that each level displays.

Level 20

Add File:filename Number of fields:num_fields Record Length:

Level 22

Add Record Buffer:buff_no File:filename Key:keyname

Level 22

Add field:fieldname

Level 24

Add funct:funct_name Level:level_num

Level 24

Add value:value Level:level_num

Level 30

[Up|Down] to level level_num

Level 50

Tape configuration file analysis done

Performing Backup Procedures

Introduction

The procedure to backup CX to tape (or disk) is completely table driven to allow easy customization by the client to directly meet their needs. To perform the normal backups for the day, you login as the user backup and respond to the questions asked.

Note: You can make backups of particular directories for special purposes using the *tar(1)* program.

CAUTION: You must perform backups (especially level 0) at a time when no one is updating the data files or else some serious inconsistencies could result if a restore is done.

Backup Dumps

The backup procedure consists of a series of incremental dumps at different levels. You make a full level 0 dump at the beginning of each month to be kept as a monthly backup. Make a level 1 dump each weekend; and throughout the week, backup the data that has changed since the previous weekend.

Note: An exception to this rule is the root file system (/) which you back up with a level 0 dump every day.

Example:

Day	Level
Mon	5
Tues	4
Wed	3
Thurs	2
Fri, Sat or Sun	1

Note: The first Friday of the month, you perform the level 0 MONTHLY dump.

Backup Script

The script in /usr/local called *backup* performs the incremental dump correctly for each day of the week (if the system date is correct). The variables at the top of the backup script must be changed to reflect the current file system mountings whenever /etc/fstab is changed. Normally, the script backs up all partitions daily (except for where /tmp is mounted) unless a partition contains a file system that does not change at all.

Backup of Logical Logs

If you have multiple tape drives, you can keep a continuous logical log backup process running during a level 0 archive.

Backup Tapes

You should keep the following tapes:

- Two sets of MONTHLY tapes for every partition used (except for /tmp if it is on a separate partition).
- Four sets of WEEKLY tapes (the MONTHLY tapes take care of the first week of each month) for every partition that will have changes at least once a month.

- One set of daily tapes (Monday, Tuesday, Wednesday, and Thursday) for each partition to be incrementally backed up every day.

Tape Labeling

You should label tapes consistently on the outside tape seal and on an adhesive label on the face of each tape. The following information should go on the tape seal:

- Process used (e.g., dump or tar)
- Special device backed up (e.g., ra0a, ra1h, hp0g, ...)
- File system mounted (e.g., /, /usr, /usr/carsb, ...)
- Day (or type) of backup (e.g., Mon, ..., Thur, WEEKLY, MONTHLY)
- Set letter (WEEKLY and MONTHLY) (e.g., A, B, C, ...)
- Volume number (as needed) (e.g., 1, 2, 3, 4, ...)

The adhesive label on the face of the tape should contain all of the above information as well as the normal level of the dump (in parentheses) on the top of the label. Under this top heading, enter the history of when and by whom the backups were actually made. If the actual level of dump performed is different from the default (as shown at the top of the label), or the mounted file system is different from the default (as shown at the top of the label), you should note this (at least on the first volume) along with the date and person making the backup.

You should add the following information to the label each time the tape is used:

- Date of backup (e.g., 10/1/97)
- Who did the backup (e.g., JRB, PNK, ...)
- Level (if different) (e.g., Lev2, Lev3, ...)
- File system (if different) (e.g., /usr/carsold, ...)

Examples of Information on Tape Seals or Outside Labels

The following are examples of labels on tape seals or outside labels:

Dump ra0h (/usr)
Monday Volume #1

Dump ra0h (/usr)
Tuesday Volume #1

Dump ra0h (/usr)
WEEKLY Set A, Volume #1

Dump ra0h (/usr)
WEEKLY Set A, Volume #2

Dump ra0h (/usr)
MONTHLY Set A, Volume #1

Dump ra0h (/usr)
MONTHLY Set A, Volume #2

Dump ra0h (/usr)
MONTHLY Set B, Volume #1

Dump ra0h (/usr)
MONTHLY Set B, Volume #2

Examples of Information on Tape Labels

The following are examples of tape labels:

```
Dump ra0h (/usr)
Monday (5)
---------------------------------
10/5/97 JRB
10/12/97 JRB
etc.

Dump ra0h (/usr)
Tuesday (4) 1
---------------------------------
10/6/97 PNK
10/13/97 JRB
etc.

Dump ra0h (/usr)
WEEKLY (1) A1
---------------------------------
10/9/97 JRB Lev 2
etc.

Dump ra0h (/usr)
WEEKLY (1) A2
---------------------------------
10/9/97 JRB
etc.

Dump ra0g (/usr/cars)
MONTHLY (0) B1
---------------------------------
10/2/97 JRB (/usr/carsold)
etc.

Dump ra0h (/usr/cars)
MONTHLY (0) B2
---------------------------------
10/2/97 JRB
etc.
```

Transferring Data Across File Systems

Introduction

A full disk can cause serious data integrity problems with the CX database. Normal daily system use can result in file systems becoming full. When a file system reaches 100% full, according to the df(1) listing, a regular user (non-superuser) will not be able to add to the file system. Examples of serious data integrity problems can include:

- Corrupted indexes
- Inconsistent data across data files
- Inconsistent transactions

To avoid a full disk, you have to move data across file systems or add a new disk drive.

Note: Because institutions can have varying system architectures, the following instructions do not cover all possibilities regarding moving data and making decisions such as, on permissions and sizes. If you have further questions, please contact your CX Account Manager.

Preparing to Move Data

To move data across file systems, do the following:

1. Determine the file system that is causing the problem.
2. Decide what file system to which to move data. This file system is usually the one with the most megabytes of space available.

   Note: If your UNIX supports symbolic links (e.g., Berkeley based HPUX), this feature allows you to utilize your disk space more efficiently. Without symbolic links, you will need to mount complete file system sub-trees on the partitions (or sections) of the disk.
3. Decide what portion of the full directory structure should be moved to the new partition.
4. Determine the size of the sub-tree using the *du(1)* command with the summary option (e.g., **du -s**) and verify that K-byte total from du(1) will fit in the target file system allowing necessary capacity for growth.

Process to Add a New Disk Drive

The following lists the phases in the process to add a new disk drive.

Note: For more information about specific steps for each phase, see the system administrator's documentation for your operating system.

1. After your hardware maintenance personnel install the new disk drive, you might need to do the following:
   - Format the disk drive

     Note: If you are using HP hardware, they provide formatting procedures for their equipment and their hardware installation personnel should complete the disk formatting for you.
   - Add the device to the kernel configuration
   - Add device entries to the /dev directory (as super-user).
2. Add (or update) the necessary partition entries for the new disk drive in the file system table (e.g., /etc/checklist under HPUX).
3. Initialize the file systems on the new disk using NEWFS.

4. Mount the new file systems as updated in the file system mount table (step 3, above) and assign correct permissions.

Steps to Moving the Data

The following lists the steps to moving data to a new location.

1. Make sure the full file system is not being used. This is best accomplished by having all users log off.
2. As superuser, change to the source directory sub-tree that is to be moved and copy the files to the target location. For example, (with symbolic links - e.g. HPUX 2.1), the following illustration assumes these file systems:

   Example: /usr/carsf 98% full 356MB used

   /mnt4 52% full 128MB used

   # mkdir /mnt4/carsdata
   # chmod 750 /mnt4/carsdata
   # chgrp carsprog /mnt4/carsdata
   # mkdir /mnt4/carsdata/financial
   # chmod 770 /mnt4/carsdata/financial
   # chgrp carsprog /mnt4/carsdata/financial
   # cd /usr/$CARSV/data
   # cpdir financial /mnt4/carsdata/financial
   # mv financial financial-o
   # ln -s /mnt4/carsdata/financial financial
3. Verify that the target data appears intact. If the copied directories include INFORMIX data files, use PERFORM or *senter2* to validate the contents. If directories were text files, use *more* to view the files and verify the contents. Also, log on as regular users and verify that they have correct access and permissions to the moved files and directories.
4. After you are satisfied the new files & directories are correct, remove the old directory structure. For example:
   # cd /usr/$CARSV/data
   # rm -rf financial-o
5. Reboot the system with the new structure up to multi-user mode. For example:
   # /etc/shutdown -i6 -g60 -y

Extracting Data to Tapes

Introduction

Periodically, it is necessary to exchange data with outside organizations, institutions, agencies, etc. via magnetic tape. Extracting data and writing tapes is easily accomplished in CX and UNIX. Exchange of data via magnetic tape is typically accomplished using 1600 BPI, 9 track format. Using INFORMIX and UNIX utilities, you can create (and read) tapes containing data for demographic studies, financial aid, letters, mailing labels, and many other purposes.

Extracting Data Using an ACE Report

You can use the ACE report writer to extract data to be written to tape. When constructing the ACE report, remember the following:

- Set all margins set to zero and create no headers.
- Each record should be one long continuous line of text.
- The criteria for the report is, of course, the selection criteria for the data that is desired to be on the tape.

For example, if you need to send out certain names and addresses for special printing, the contents of the tape record would include name, address, city, state, zip, first name, last name, and possibly some personal data. The ACE report prints all the fields in the ACE report without intermediate spacing. In this example, the total length of all these fields is 241 characters. ACE will also output the newline character at the end of a regular print statement, so the total length of the output record created is 242 characters.

Executing the Tape Record ACE Report

The following example shows the commands to execute the tape record ACE report.

```
% make add F=bldtape
% vi bldtape
% make F=bldtape
% acego -q bldtape > bldtape.out
%
```

Testing the Output

Do the following to test the ACE report output.

1. To determine the number of lines (records) in the file, use the UNIX WC command. For example:

   Example: % **wc bldtape.out**

   3093 15673 748265 bldtape.out

   %

   Note: Since ACE outputs an extra line in the file, the number of records you want to write to tape is 3092. For testing purposes, limit the number of records to a dozen or less. Write the information to a test file (or tape) to make sure.

2. To write the data and strip the newline characters from every line, use the UNIX command **dd** to test our data, our record length, etc.

   Example: % **dd if=bldtape.out of=junk ibs=242 cbs=241 obs=241 conv=block count=10**

Note: This command uses the input file (if=) and writes the data to the output file (of=). The input block size (ibs=) is the original 242 characters.

- The output block size (obs=) is the desired number of characters, without the newline.
- The conversion block size (cbs=) is also used in stripping the newline character.
- The conversion type (conv=) specifies we are creating fixed length records and the count (count=) is the number of (input) records to process.

3. Create a small file and check it with the dump program to verify the contents. (The example count equaled 10.)

   Example: % **od -a bldtape.out**

The system displays the file's output:

```
0000000    S   m   i   t   h   ,  sp   H   a   r   r   y  sp  sp  sp  sp
0000020   sp  sp  sp  sp  sp  sp  sp  sp  sp  sp  sp  sp  sp  sp  sp  sp
0000040    1   2   3   4  sp   N   o   w   h   e   r   e  sp   S   t  sp
   ...
```

Note: The OD command displays the contents of the file in a number of different formats. With the -a option, the system displays the data in ASCII. You can verify the correct record length by viewing the data and determining that no loss of characters had occurred and that no extra characters were at the end of each record.

Creating the Tape

After you have verified the output of the ACE report, you can write the output to the tape. For example, to write an unlabeled tape with the blocking factor equal to the actual record length, enter the following:

Example: % **dd if=bldtape.out of=/dev/tapedev ibs=242 cbs=241 obs=241 conv=block count=3092**

You can append additional data onto the tape after the above file, if you use the tape device /dev/tapedevnr, instead of /dev/tapedev. The tapedevnr entry does not automatically execute a rewind of the tape after the file is closed (or the program exits) as the tapedev entry does. In addition, you can use the MT command to manipulate the tape drive itself.

Note: In this discussion, tapedev is the correct designation relative to the /dev directory of the desired rewind tape device; while tapedevnr is the same for the no rewind device.

Setting Up a User's File Transmit Capability

Introduction

The Utilities: File Options menu contains the Download File option, which a user can use to transmit a file to and from CX and a PC. To transmit files from CX to the PC, the PC must have a transmit protocol running to receive the file. When using the Download File option, the user simply specifies a file name and does *not* need to know file transmit protocol commands or the directory in which to transmit the file. The Download File option uses the $CARSPATH/modules/util/xfer script, which allows the use of the following transmit protocols: FTP, Kermit, Xmodem, QuickMate, or Zmodem.

You need to be aware of three things when setting up a user's file transmit protocols.

- The macros\custom\common file and the XFER_PROTOCOL macro
- The user's *.xferrc* file
- The user's *.cshrc* file

See below for more information.

The macros\custom\common file and the XFER_PROTOCOL Macro

The macros\custom\common file contains the XFER_PROTOCOL macro which provides the default transmit protocol settings for your entire institution. The transmit protocol choices are: FTP, Kermit, Xmodem, Zmodem and QuickMate.

The *.xferrc* file

The *.xferrc* file contains the protocol settings for each specific user. Each user's *.xferrc* file is located in their home directory.

The *.cshrc* file

Each user has a *.cshrc* file. Within this file you make the following entry if you want to override the system defaults set in the XFER_PROTOCOL macro:

setenv UserSource true

This will allow the user to override the system default settings with user specific settings established in the *.xferrc* file.

CAUTION: It is important to note that this is a potential security risk. Doing this will set up this user to override the institutional settings in the XFER_PROTOCOL macro, with the settings in their *.xferrc* file.

Setting Up a User's File Transmit Protocol - FTP Settings

Settings for FTP

If you are setting up a user to use FTP as the file transfer protocol, you must do the following:

- Set three additional macros (XFER_REMOTE_DIR, XFER_REMOTE_HOST, and XFER_REMOTE_USER) in the $CARSPATH/macros/custom/common file
- Make additional settings in the *.xferrc* file
- Create the *.netrc* file

See below for more information.

Macros Used to Set Up FTP

If the user will use FTP, you must set the following macros in the $CARSPATH/macros/custom/common file:

XFER_REMOTE_DIR
Specifies the directory on the PC to receive files (e.g., /cis).

Note: When specifying the variable, use FTP syntax rather than DOS. For example, use /cis instead of c:\cis.

XFER_REMOTE_HOST
Specifies the remote server link to the PC (e.g., $LOGNAME).

XFER_REMOTE_USER
Specifies the login name of the PC user (e.g., $LOGNAME).

FTP Settings in the *.xferrc* File

If the user will use FTP, specify the following settings in the *.xferrc* for FTP.

RemoteDir
The directory on the PC to receive files.

Note: When specifying the variable, use FTP syntax rather than DOS. For example, use */*cis instead of c:\cis.

RemoteHost
The remote server link to the PC.

RemoteUser
The login name of the PC user.

Password
The user's password on the PC.

Module
The protocol used for file transfers (e.g., FTP).

Sample *.xferrc* File for FTP

The following sample specifies the use of FTP to a networked PC, named *max*, for a user login, named *maxwell*. Files are transferred to the C:\CARS directory.

```
RemoteHost=max
RemoteUser=maxwell
RemoteDir=/cis
Module=FTP
```

FTP Settings in the *.netrc* File

If the user will use FTP, specify the following settings in the *.netrc* for FTP.

login
The login name of the user. This value should match the RemoteUser variable in the *.xferrc* file.

Note: You can specify the name as *anonymous*.

machine
The auto-login value for FTP. The machine value should match the value of the RemoteHost variable specified in the *.xferrc* file.

password
The password for the remote login.

Note: If you do *not* specify a password and the system setup requires passwords, the system will prompt the user for a password during file transfers. If the user logs in as *anonymous*, the system requires no password.

Sample .netrc File

The following sample is consistent with the sample *.xferrc* file above:

```
machine max
login maxwell
password wilbur
```

Setting Up a User's File Transmit Protocol - QuickMate Settings

Settings for QuickMate

If you are setting up a user to use QuickMate as the file transmit protocol, you must do the following:

- Make additional settings in the XFER_PROTOCOL macro
- Modify the xfer script

You can optionally do the following:

- Set download location in *.xferrc* file for individual users
- Add a menuopt for QuickMate downloads

Set XFER_PROTOCOL Macro for QuickMate

To set up the XFER_PROTOCOL macro for the QuickMate option, you must set it to CSERV.

CAUTION: This will set the transfer protocol to CSERV for *all* users.

Modifying the XFER Script

You must set the Remote Directory (the PC to which you want to transfer files) for a user. To do this, add a line to the xfer script. Adding this line will allow you to add a line to a user's *.xferrc* file. Add the following line:

```
typeset -u RemoteDir   # To Uppercase
```

Note: Place this line in the last section of the script, under the following line:
typeset –u Module # To Uppercase

If you also want to permit users to upload files using QuickMate, you must make the following changes to the xfer script:

Before:

```
if [ -r "$@" ]; then
   ${Module}_${Direction} "$@"
else
   print "File does not exist."
fi
```

After:

```
if [ "$Direction" = RECV ]; then
   ${Module}_${Direction} "$@"
else
   if [ -r "$@" ]; then
      ${Module}_${Direction} "$@"
   else
      print "File does not exist."
   fi
fi
```

Changing Default Location in the *.xferrc* File

After modifying the xfer script as shown above, you can specify settings in the user's *.xferrc* file to allow users to download files to a location other than the default location specified in the XFER_REMOTE_DIR macro. Make sure the user is the owner of his/her .xferrc file, and specify the following settings:

Module = CSERV
Direction = SEND
Mode = BINARY, (or ASCII if you prefer the default transfer mode to be text)

Then, add the following line: **RemoteDir='c:\mydocu~1\downlo~1\'**

Note: The line denoting the RemoteDir location is an example for using the My Documents\Download directory as the download location. If you want to use another download location, change the "mydocu~1\downlo~1" reference accordingly.

Adding a Menuopt for QuickMate Downloads

The following menu option enables QuickMate downloads. With minor modifications (including the use of a –r parameter instead of –s), the option could be used for uploads as well.

```
{
 Revision Information (Automatically maintained by 'make' - DON'T CHANGE)
 ---------------------------------------------------------------------------
 $Header: CUqmatexfer,v 8.0.6500.3 2000/6/23 11:41:35 jdoe Released $
 ---------------------------------------------------------------------------
}

screen
{
                m4_center(DOWNLOAD A FILE USING QUICKMATE,40)

                m4_center(NOTE: This option only works within,40)
                m4_center(QuickMate and only if your CX,40)
                m4_center(account has been configured to perform,40)
                m4_center(QuickMate downloads,40)

                m4_center(The file will be downloaded to the,40)
                m4_center(location indicated in your CX,40)
                m4_center(configuration file,40)

  PP_FILE
  [PA3]
}
end

attributes

SD: optional,
   default = "Download File w/ QuickMate";

RD1: optional,
   default = "This option will transmit the specified file   ";

RD2: optional,
   default = "from the Unix host system to a PC.             ";

PR: optional,
   default = "RUN_EXPAND";

PA1: optional,
   default = "UTL_PATH/xfer";

PA2: optional,
   default = "-s";

PA3:
   comments = "Enter the name of the file to be transmitted.",
   length = 74;

end
```

Reinstalling Jenzabar CX

Introduction

You can use the reinstall_cars script to reinstall large portions of CX at one time. The script places its output in a file in /tmp called reinstall$$.out, where $$ is the process-id of the reinstall_cars.scp.

The script is located in: $CARSPATH/modules/util/commands/rein_cars. The script is installed in: $CARSPATH/install/scp/util/reinstall_cars.scp.

Script Usage

The usage of reinstall_cars.scp is as follows:

```
Usage:   [-wrld]   [Y/N]
    [-reg]    [Y/N][-aid]   [Y/N]
    [-dev]    [Y/N][-comm]  [Y/N]
    [-serv]   [Y/N][-adm]   [Y/N]
    [-matric] [Y/N][-gl]    [Y/N]
    [-apay]   [Y/N][-purch] [Y/N]
    [-bill]   [Y/N][-bgt]   [Y/N]
    [-fix]    [Y/N][-pay]   [Y/N]
    [-notes]  [Y/N]
```

Note: If you use the reinstall world (wrld) options, the script ignores the others. You must use one option.

Running the Script

You can run the script in two ways: you can reinstall the world, or you can reinstall some combination of specific modules.

- If you want to reinstall the world, pass the script a '-wrld Y'.

  Note: You must specify Y, or the script will not run.

- If you want to reinstall a combination of specific modules, pass the script the indicated abbreviation for each module you want to reinstall.

  Note: You must specify Y after each module you specify in the script.

Processing Note

Because your institution may have a unique setup, this script is not guaranteed to work in all cases. This is especially true if you run the script from a second release (e.g., carstrain) where many of the directories are linked to a different release (like carsi).

Maintaining Local Customizations

Every institution is installed with their own set of branch numbers used for maintaining local revisions. Most institutions maintain a train release and a live release using a single set of source with two install paths. This allows an institution to install a SMO only once to update both releases. It should first be installed on the train release and then tested. After testing, it can be installed on the live release. It is important to use the same branch number for both install paths.

CAUTION: You should not do a reinstall on the live release from the time a SMO is deposited until testing is completed in the train release if the source files are being shared.

Troubleshooting Customizations

If your institution uses different branch numbers for the train database and the live database, it could appear that you are losing customizations when you deposit a SMO.

Example: Assume you made local customizations on your live release using a branch number of 1100. If a new SMO is deposited and merged through the train release using a branch number of 1101, the smodeposit step tries to see if the previous trunk version of the file had a 1101 branch number. Since it does not recognize the 1100 branch number, it loses the customization.

If your institution seems to be losing customizations, check the branch numbers that are being used for the train release and the live release. Do the following to test this:

1. Under live, execute:

   ```
   % grep  _NO $M4PATH/systemConfig.m4
   ```

2. Under train, execute:

   ```
   % grep  _NO $M4PATH/systemConfig.m4
   ```

   Example results:

   ```
   m4_define('CFG_CLIENT_NO', '1')
   m4_define('CFG_MACHINE_NO', '3')
   m4_define('CFG_DATABASE_NO', '5')
   ```

If your institution is set up with a single set of source, these commands should produce the same results.

Note: A branch number is calculated using this formula:

CFG_CLIENT_NO * 100 + (CFG_MACHINE_NO * 10) + CFG_DATABASE_NO

Restoring Your Customizations

To restore your customizations, do the following:

1. Reinstall Make on both releases using a single branch number.
2. Develop and run scripts to edit the RCS files to move the branches to that single number.

SECTION 9 – SYSTEM MAINTENANCE

Overview

Introduction

This section provides information and procedures for maintaining the system that runs CX software and files. System maintenance issues discussed in this section include the following:

- Shutting down the system
- Removing bad blocks from a disk
- Adding disk space
- Reorganizing disk space

Shutting Down the System

Introduction

These pages outline the steps to shut down the system.

Shutdown Procedure

The following lists the steps to shut down the system.

1. Login as user *shutdown*.
2. Execute the system shutdown script. The system displays the following prompt: “Is this shutdown for halt, reboot, or single user mode (h/r/s, q=quit) ?”
3. Enter: **h**

   The system displays the following prompt: “Enter time for shutdown (hh:mm, +mm, now, q=quit) ?”
4. Enter the shutdown time (e.g., **+2**). The system displays the following prompt: “Enter shutdown message”
5. Enter a message for the shutdown (e.g., **Down for storm**). The system displays the following prompt: “Upon rebooting, do normal file system checks or fastboot (n/f, q=quit) ?”
6. Enter: **n**

   The system displays the following message: “About to execute: /etc/shutdown -h +2 from Shutdown script:”

   The system displays the following prompt: “Down for storm continue (y/n)”
7. Enter: **y**

   Note: After system shutdown procedure has completely finished, the console will display "Type CTRL-P to HALT"

Powering Down the System

Do the following to completely power down the system:

1. Turn the key on the front panel of the system CPU to *local*.
2. At the console, enter: **<Ctrl-p>** -

   The console displays: >>>
3. At the console, enter the following:
   - **i** to specify *initialize*
   - **h** to specify *halt*
4. Turn on Write Protect on the disk drive(s).
5. Spin down the disk(s) (e.g., Turn off Run/Stop button).
6. Turn the key on the front panel of the system CPU to the *off* position.
7. If you are shutting down due to an electrical storm, unplug the power and communication cables.

Powering Up the System

Do the following to power up the system:

1. Plug in the power and communication cables.
2. Set device switch on the front panel of the system CPU to D (for normal boot from disk).
3. Set front panel action switch to *boot*.
4. Turn the key on the front panel of the system CPU to the *local* position.
5. Spin up the disk(s) (Turn on Run/Stop button).
6. Turn off Write Protect on the disk drive(s).
7. Press white reset button to boot UNIX.
8. Turn front panel key switch to the secure position.

Available Commands

The following commands are available to su or root when not using the shutdown username.

/etc/shutdown
Orderly shutdown of system.

/etc/quickboot
Reboot and skip file system checks on startup.

/etc/quickhalt
Halt and skip file system checks on startup.

Note: For more information, see the UNIX documentation on the following UNIX commands:

- shutdown(8)
- fastboot(8)
- fasthalt(8)
- halt(8)
- reboot(8)

Managing Disk Space

Introduction

This section provides general guidelines for adding and reorganizing your disk space. For additional information, consult your Informix documentation.

Adding Disk Space

Follow these general steps to add disk space:

1. Create a new logical volume.
2. Determine the new major and minor numbers for the device.
3. Execute the 'mknod' command in the $INFORMIXDIR/dev directory.
4. Change the ownership and file modes of the new device in $INFORMIXDIR/dev.
5. Add a chunk using tbmonitor.

Reorganizing Disk Space

You may need to reorganize disk space to accomplish the following:

- Move data for more efficient disk use
- Delete unnecessary extents on tables
- Increase the size of file systems that are full

This is a complex task and requires a substantial amount of time.

Planning Steps

You should consider the following when planning for the reorganization.

1. Plan for at least two days downtime (usually a long week-end).
2. Ensure that all offices are off the system and will not be back on until you are finished.
3. Plan to reorganize the Informix database one week-end, and the file systems another.
4. Add extra disk drives to your system, or plan on spending extra time writing and reading tapes.
5. Plan where you want to move the data. Consider disk sizes and speeds, SCSI speeds, number of dbspaces you want to have, whether or not you are going to use mirroring. With Informix 7, also consider placement of temp table space and logical log space.

Performing the Reorganization

Follow these steps to perform the reorganization.

1. Clean your tape drives before you begin.
2. Lock all users off the system.
3. Write the last logical log file to tape.
4. Test the isql commands on the train database, and then drop the train database temporarily to minimize backup.
5. Take a Level 0 archive.
6. If you have a second tape drive, use it to take a second Level 0 archive.
7. Do a tbunload of the CX database.
8. Delete any old data rows you no longer need.
9. Use dbexport to unload the data.
10. Rebuild the indexes.
11. Copy cars to train.
12. Turn on buffered logging and take another Level 0 archive.
13. Build permissions on the train database.
14. Test the results.
15. Allow users back on the system.

Removing Bad Blocks from a Disk

Introduction

During the normal usage of the system, disk media can develop bad spots. When a particular sector appears in a Hard Error message on the console several times, you should remove it from use and recover the file that contained the bad sector.

The following steps are designed to assist you in moving the bad disk to an area on the disk that the system does not use. The only complete fix for bad sectors is to reformat the disk.

Note: When the root directory of the partition is mentioned below, it refers to the directory that is mounted on the partition that has the hard error (e.g., / for ra0a, /usr for ra0h, or something similar as determined with the df(1) command).

Note: The steps appearing with an example that will be different from your particular case. See the manual entry for badsect(8) for further details concerning this process.

Determine the Partition Relative Sector Number

Determine the sector number value from the hard error message on the console (or in /usr/adm/messages*). Subtract the beginning sector number of the partition from this sector number value to determine what the partition relative sector number. The following list contains the beginning sector numbers for Eagle drives and for RA drives

Example: partition relative sector number = sector number - beginning of partition

PRSN = SN - BOP

FUJITSU Eagle partitions
disk start length cyls
hp?a 0 15884 0-16
hp?b 16320 66880 17-86
hp?c 0 808320 0-841
hp?d 375360 15884 391-407
hp?e 391680 55936 408-727
hp?f 698880 109248 728-841
hp?g 375360 432768 391-841
hp?h 83520 291346 87-390

RA60 partitions
disk start length
ra?a 0 15884
ra?b 15884 33440
ra?c 0 400176
ra?g 49324 82080
ra?h 131404 268772

RA80 partitions
disk start length
ra?a 0 15884
ra?b 15884 33440
ra?c 0 242606
ra?g 49324 82080
ra?h 131404 111202

RA81 partitions
disk start length
ra?a 0 15884
ra?b 15884 33440
ra?c 0 891072
ra?d 340670 15884
ra?e 356554 55936
ra?f 412490 478582
ra?g 49324 82080
ra?h 131404 759668

Determine Which File Contains a Bad Sector

After you determine the partition relative sector number, use *icheck* and *ncheck* to determine which file contains the bad sector, and take appropriate action as recommended by your Jenzabar Account Manager. In general, do the following.

1. Use **icheck(8)** to determine the inode number of the bad sector (PRSN)

   Example: # /etc/icheck -b 1234 /dev/rra0h

   Note: 1234 is the PRSN, and ra0h is the partition.

2. Using the inode number reported by the above icheck program, use **ncheck(8)** to determine what file (if any) is associated with that inode.

   Example: # /etc/ncheck -i 106 /dev/rra0h

   Note: 106 is the inode number, and ra0h is the partition.

3. The presence of the hard error in the file can be verified by copying the file to /dev/null and checking for a hard error message on the console.

   Example: # cp filename /dev/null

4. Do the following:
   - If ncheck reports the name of a file for the inode specified, ask your Jenzabar Account Manager what is the best way to ensure that you can recover the file after it is removed.
   - If ncheck reports no file, proceed with the following steps while noting if a filename is given during the fsck process toward the end of the procedure.

Create a Link to the Bad Sector

Determining which file contains a bad sector, create a link to the bad sector using badsect. Do the following:

1. Shutdown to single user mode:

   Example: # shutdown +5 "Fix hard error problems"

2. When logged in as single user, make a BAD directory in the root directory of the partition containing the hard error:

   Example: # cd /usr

   # mkdir BAD

3. Create the link to the bad sector using badsect:

   Example: # /etc/badsect BAD 1234

Remove the File with the Bad Sector

After creating a link to the bad sector, remove the file that contained the bad sector. Do the following, still in single user mode:

1. Unmount the root directory of the partition:

   Example: **# cd /**

   # **umount** /dev/ra0h

2. Run *fsck* to remove the link from the file to the bad sector. Enter the following:

   Example: **# fsck** /dev/rra0h

   The system displays the following messages:
   "** /dev/rra0h
   ** Last Mounted on /usr
   ** Root file system
   ** Phase 1 - Check Blocks and Sizes "

   The system displays the following prompt:
   "HOLD BAD BLOCK?"

3. Enter **Y**.

   The system displays the following messages:
   "1873 DUP I=1939(or something similar...) INCORRECT BLOCK COUNT I=1939 (0 should be 2)"

   The system displays the following prompt:
   "CORRECT?"

4. Enter **Y**.

   The system displays the following messages:
   "** Phase 1b - Rescan For More DUPS 1873 DUP I=106
   ** Phase 2 - Check Pathnames DUP/BAD I=106 OWNER=root MODE=100600 SIZE=14336 MTIME=Sep 26 12:37 1997 FILE=/usr/cisids/jim/temp "

   The system displays the following prompt:
   "REMOVE?"

5. Enter **Y**.

   The system displays the following messages:
   "DUP/BAD I=1939 OWNER=root MODE=100600 SIZE=1024 MTIME=Jul 23 00:35 1997 FILE=/BAD/1234"

   The system displays the following prompt:
   "REMOVE?"

6. Enter **N**. Do not remove this one.

   The system displays the following messages:
   "** Phase 3 - Check Connectivity
   ** Phase 4 - Check Reference Counts BAD/DUP FILE I=106 OWNER=root MODE=100755 SIZE=14336 MTIME=Sep 26 12:37 1997"

   The system displays the following prompt:
   "CLEAR?"

7. Enter **Y**.

   The system displays the following message:
   "FREE INODE COUNT WRONG IN SUPERBLK "

   The system displays the following prompt:
   "FIX?"

8. Enter **Y**.

   The system displays the following messages:
   "** Phase 5 - Check Cyl groups 7 BLK(S) MISSING BAD CYLINDER GROUPS"

   The system displays the following prompt:
   "SALVAGE?"

9. Enter **Y**. The system displays the following messages:
   - "** Phase 6 - Salvage Cylinder Groups 362 files, 5714 used, 1715 free (91 frags, 203 blocks) "

Reboot the System

After removing the file with the bad sector, reboot the system. During the reboot, *fsck* may get some errors, but it should correct the problems and continue with the reboot.

Example: # sync

reboot

Note: After rebooting the system, a file should exist in the BAD directory for the bad sector. This does not fix the bad sector, but instead puts the sector in a safe place where the system will not use it. The Hard Error messages may still appear during a level 0 dump of the system, because the dump program will be attempting to read the bad sector.

SECTION 10 – CUSTOMER ASSISTANCE

Overview

Introduction

This section provides information on the ways that Jenzabar provides customer assistance. To provide ongoing assistance, Jenzabar maintains a close relationship with its customers. The following are aspects of Jenzabar's relationship to its client:

- Unconditional guarantee
- Partnership with client
- Account Management that remains in process (sales representative becomes ongoing contact)
- Program management from Professional Services group (employees have educational backgrounds)
- Support for programs
- Program Review
- NACU
- Client satisfaction/retention

Jenzabar Services

The following lists the services that Jenzabar provides to its customers:

- Six month implementation
- Application consulting
- Technical consulting
- Re-engineering (process reorganization)
- Support (up to 7 days/24 hours a day)
- Education and training (current clients/ongoing training)
- Documentation

Corporate Commitments

The following lists the Jenzabar corporate commitments, which affect customer assistance:

- Focus on strategic Business issues
- Empowering users
- Product area specialists
- Continual improvement of documentation/training
- More time on campus (Jenzabar comes to you)
- Regional support
- Comprehensive education program (ongoing/aggressive)
- Improved maintenance agreement
- Product advisories and focus groups

Quality Customer Service: How Jenzabar Delivers Support

SMO and Revision Control System (RCS)

Jenzabar provides continuing support and enhancement for CX. This is one of the major benefits of contracting with Jenzabar. In addition, the product includes the Revision Control System (RCS), to track modifications and ensure that the customer's version is current. Some types of modifications include:

- Fixes to bugs
- Changes to meet the requirements of external agencies, (e.g., financial aid requirements)
- Enhancements suggested by customers
- Enhancements suggested by Jenzabar staff

Support Services

Jenzabar Support Services provides continuing support to customers who have completed the implementation as part of the maintenance agreement. Staffed with experts in all areas of the CX products, Support Services is the customer's around-the-clock access to the corporation.

Support Services logs calls into an internal tracking system, and prioritizes them according to their level of urgency. Jenzabar follows strict standards for response time to ensure minimum disruption to customers' daily operations.

Note: Institutions usually find it most cost-effective to designate the Jenzabar system coordinator to officially channel calls to Support Services.

For information about contacting Support Services, see the *Client Support Services User Guide*.

Quality Assurance Survey

Jenzabar employs a full-time staff person, who follows up on all contact with customers, completing research for a Customer Satisfaction Evaluation form. The customer is given this opportunity to rate the quality of the service provided by Jenzabar.

Note: The results of these evaluations are used by individual Jenzabar staff and by their managers to ensure that quality and timeliness of response to customers remain a top priority.

REACH

Jenzabar follows up with institutions using the CX product through the REACH (review, educate, analyze, care, help) program. The REACH program includes the following goals:

- Visits to the customer by a Client Services representative and Jenzabar executive management, to assess directly how CX is performing
- Assessment of concerns and questions of CX users and administrators to help in evaluating performance of the CX
- Identification of the future direction that the institution plans to take, and comparison to that planned by Jenzabar
- Assurance that the investment by the customer and Jenzabar has provided a solid basis for a continuing relationship

National Association of Jenzabar Users (NACU)

Jenzabar supports NACU. However, Jenzabar is not under its direct control. Jenzabar customers formed NACU, creating a forum in which members share the most effective means of using CX. Also, NACU serves as a vehicle through which customers communicate collectively with Jenzabar. NACU holds an annual meeting and invites Jenzabar to present items of interest to NACU.

Note: The size of the Jenzabar user base has made possible the formation of special interest groups within NACU to deal with specific product or other issues.

The National Association of CX Users (NACU)

Introduction

The National Association of CX Users (NACU) is an organization of CX users. Although independent of Jenzabar, NACU and Jenzabar work together in many ventures.

CX users become members of NACU their first year free of charge; in subsequent years, each institution must pay yearly dues in order to maintain membership in NACU.

Steering Committee

The governing body of NACU is the Steering Committee. The Steering Committee is a vehicle for communication between Jenzabar and its clients. The Steering Committee keeps in touch with NACU members through periodic mailings, and the yearly national conference.

Members of the Steering Committee are elected yearly by NACU members at the annual conference. The Steering Committee appoints elected individuals to each position.

Annual Conference

The NACU annual conference usually takes place in June. In cooperation with the Steering Committee, Jenzabar typically offers several optional seminars and workshops for attendees.

Software Exchange

The Software Exchange program was established for the following purposes:

- To encourage CX users to share and benefit from the innovative software developments at other institutions
- To reduce unprofitable duplication of effort at institutions
- To provide CX users with economical access to existing software
- To encourage institutions to create innovative uses for administrative software

Although software is distributed *as is*, CX requests that only software packages that have been used, tested, and revised as necessary are submitted; the Software Exchange does not test software. Submitted materials must conform to a standard format and must be accompanied by a release form.

The Software Exchange charges no fees for software contributed by its members. Only shipping and handling charges are charged by the distributor to cover the cost of magnetic tapes, mailers, shipping charges, and other costs of handling. Jenzabar posts all entries on the clients-only Web page.

Software Contest

Any CX user who is also a member of NACU is eligible to enter the software contest. The purposes of the software contest are as follows:

- To encourage CX users to develop and share software
- To recognize contributors of the software contest

Monetary prizes are awarded in the following categories; winners are announced at the NACU conference.

- Accounting/Financial
- Admissions/Recruiting
- Institutional Advancement
- Financial Aid
- Registrar/Records/Student Affairs
- System Utilities/Management

Each submission must be accompanied by a README file describing the entry value, documentation, a release form, and an entry form.

A minimum of three entries must be submitted in any category for an award to be made. If there are not enough entries in a category to make an award, the entries in that category are carried over to the following year's contest. However, entries that compete in one year's contest and do not win are not eligible for the next year's contest.

Call for Papers

Jenzabar encourages clients to deliver papers relating to or discussing products or services provided by Jenzabar at established national organizations. Papers must be original material but may be published or submitted for publication elsewhere. Papers should report concrete results or be survey or tutorial papers that include synthesis and evaluation. Papers that make recommendations should be based on more than the author's opinion.

Each CX user who presents a paper at an approved national organization (and who provides Jenzabar with a written copy for distribution) will receive reimbursement of reasonable expenses incurred by the presenter and his or her spouse while attending the conference at which the paper was presented. In addition, the presenter is recognized with a plaque at the next NACU conference.

SECTION 11 - TROUBLESHOOTING

Overview

Introduction

This section provides a crash recovery procedure and troubleshooting tips for a system manager.

Product-specific Troubleshooting

This section has troubleshooting tips for the common system. For troubleshooting information for a specific CX product area, refer to the technical manual for the product area.

Crash Recovery Procedure

Core Dump Recovery

The following list describes the steps to recover from a core dump of an entry program.

1. Access the program screens directory for the entry program.

   Example: % cd modules/regist/stuentry/progscr

2. Reinstall each program screen file.

   Example: % make reinstall F=<filename>

   Note: You can also reinstall all of the screens by entering the following:

   Note: % make reinstall F=all

3. Attempt to execute the entry program. Did the reinstall of the program screens fix the error?
   - If yes, you are done.
   - If no, go to step 4.

4. Access the source code directory of the entry program.

   Example: % cd src/regist/stuentry

5. Reinstall the source code for the entry program.

   Example: % make reinstall

6. Attempt to execute the entry program. Did the reinstall of the program source code fix the error?
   - If yes, you are done.
   - If no, go to step 7.

7. In the source code for the entry program, delete the old compiled code for the entry program.

   Example: % make cleanup

8. Reinstall the entry program source code.

   Example: % make reinstall

9. Attempt to execute the entry program. Did the deletion of the old code and reinstallation of the program source code fix the error?
   - If yes, you are done.
   - If no, go to step 10.

10. Review the libraries for the entry program. In the source code for the entry program, review the file, Makefile. In the file, search for the parameter, ADDLIBS, which identifies the libraries that you must reinstall.

    Example: % vi Makefile

    /ADDLIBS

11. Reinstall the libraries for the entry program and reinstall the source for the entry program.

    Example: % cd <to appropriate library>

    % make reinstall

    % cd src/regist/stuentry

    % make reinstall

    Note: You must reinstall the source program to include any library changes.

12. Attempt to execute the entry program. Did the reinstallation of the libraries for the entry program fix the error?
    - If yes, you are done.
    - If no, call Jenzabar Support Services.

Troubleshooting Tips for System Administrators

Introduction

The following are troubleshooting tips for system administrators.

User(s) Cannot Login

Below are three possible problems:

Permissions problem

This problem can result when you have changed the owner, group, and permissions of all files that start with a '.' in a user's home directory. If you entered **chown xxx** to change the owner of all files that start with a dot, you also changed the dot-dot (..) file. Since this file represents the directory above the current directory, the above command can affect the parent directory and/or all the other home directories. The command could prohibit all other users from being able to login because they are no longer the owner of their home directory.

To fix the problem, you must change the owner, group and permissions back on the parent directory and all other home directories. The /home/carsids directory should look like the following line:

```
drwxr-xr-x  156 root     common      3072 Jan  3 14:36 /home/carsids
```

The individual user home directories should be owned by the individual user. Each user should have read, write, execute (rwx) permissions in their own directory. The *group* and *other* permissions should also be rwx to allow programs, which run as *carsu-carsprog*, to write to the real user's home directory.

Missing mail file

If the login process is only failing for certain users, check to see if those users have a mail file in /var/mail (/var/spool/mail on AIX). If the failing logins do *not* have a mail file in existence, the problem lies in /etc/csh.login. The standard way HP checks for mail in their standard /var/csh.login file will not work for users logging into the menu who have no mail file in existence. CX modifies this file upon installation to change the mail test to one that works. If you have lost the modifications, contact Jenzabar.

Backup and Shutdown user problem

If the login process only fails for the user *backup* and the user *shutdown*, the problem lies in $CARSPATH/skel/login.s. The following line, near the bottom of the file, should *not* be executed for the user *backup* or *shutdown*:

```
source ~/../$SKEL/env_set
```

Users Get Errors and Return to Menu or Login Prompt

When reaching new peaks of usage, the system may stop allowing new processes to start up. While this condition is occurring, users all over the system can experience a variety of errors. The users should *not* hang under these conditions but only get error messages and be returned to the menu or login prompt. The following are several limits that the system can reach:

Memory limits

The system has only so much memory and swap space. During a time when all memory and swap is in use, a variety of errors can occur. HP clients can monitor swap space usage by running *swapinfo* as root. Repeated occurrences of this problem might indicate the need for more memory or swap space

Kernel limits

HP systems have a kernel configuration file that limits the maximum number of files and users that can be in use on the machine at any one point in time. If you are hitting the maximum number of user limit, use the **ps** command to check for orphaned processes. These would be processes with a parent-id of 1.

Both HP and AIX systems limit the number of processes a single user can run. Since many of the CX processes run as the user *carsu*, it is possible that *carsu* could be denied any more processes. In this case, all other processes should work except for processes that run as *carsu*. Use the following command to see all processes running as carsu: **ps -ef | grep carsu**

Semiphores are additional resources in the kernel that are of limited supply. If programs start getting errors that reference semiphores, or contain a code that starts with SEM, try running the *cleansem* program.

Informix limits

Informix has a configuration file that limits the number of processes that can connect to the database. Informix also limits the number of open files and locks at any point in time. Enter the command **tbstat -p** will display statistics. The third line of statistics lists the number of times these limits have been reached since the Informix engine was started up.

INFORMIX can also reach a disk space limit. Informix needs space to store temporary tables and sort those tables. If all available temporary space is used up, database processes needing such space will receive errors and abort. Use the *dbspace* command to monitor Informix disk space usage. Clients running Informix 5.x can also use the **bdf** command to monitor free space in /tmp. Under 5.x, Informix uses /tmp to store temporary sort files.

dbadmin

If you ran *dbadmin* to add a new user or update the permissions of an existing user during production time, this program can cause most of the existing users to get errors and be thrown out of the programs they were running.

Users/Shell Commands are Hanging

If logins and shell commands hang or take too long to process, you might have a runaway process. This can be a single process that consumes all machine time, but it is more likely to be a process that is creating child processes as fast as it can. If you are unable to find and kill the offending process(es), you might have to reboot the computer.

If logins and shell commands work but the CX menu and processes that access the database hang, you have an Informix problem. The most common Informix problem is that the informix Logical Logs are full. You can usually clear up this problem by backing up the logical logs. Here are some issues with logical logs:

- If an Informix archive is running and it reaches the end of tape, the whole database will stop and wait for the tape to be changed.
- If a single long transaction is holding most of the logical logs, the informix engine may suspend all other processes and attempt to undo the transaction before the remainder of the logs fill up. In this case, backing up the logical logs will not help. Furthermore, bringing the engine down or rebooting the system can make the situation worse. You should wait and see if the system will clear itself out before it fills all the logs. If it does not, call Informix to correct the problem.

The $INFORMIXDIR/Logs/cars.log file (or similar file on your system) may contain messages to help diagnose these and other informix problems. The latest messages are located at the end of the file.

User/Program Permissions Problems

Below are two possible solutions:

Users Cannot Insert/Update id_recs

Starting with Product Release 12292, the id_rec will grant insert/update permissions to the group *addid*. You can add those users who understand US Postal requirements and your institutional name and address standards to the *addid* group.

Running Programs as *carsu*

Some programs require more permissions than are set for an individual user. To enable a program to do more than the user, the program can run as the user *carsu*. The $CARSPATH/system/lib/fileperms.s file controls which programs run as *carsu*. The following line from $CARSPATH/system/lib/fileperms.s indicates that voucher and vchrecover will run as user *carsu* and group *carsprog*:

```
./bin/{vchrecover,voucher}      6775    carsu   carsprog
```

To set a program to run as *carsu*, you can add an additional line to $CARSPATH/system/lib similar to the line above.

Note: The order of lines in fileperms.s is important. Add the new line next to similar lines. Do *not* add the line to the bottom or top of the file.

To stop the program from running as *carsu*, you can remove the 6 from the 6775 setting.

CAUTION: Be aware that removing the 6 from 6775 can prohibit the affected program from performing critical processing. CX assistance in fixing problems caused by changes to the $CARSPATH/system/lib/fileperms.s file is billed on a time & material basis.

File Installs with Different than Expected Permissions

When you install a file, the *make* processor runs fileperms to determine the correct permissions that the installed version of the file should have. The fileperms process searches $CARSPATH/system/lib/fileperms.s for an entry specific for the file being installed. If fileperms does not find a specific entry naming this file, it searches for the entry that provides the generic permissions for the directory into which the file is being installed.

For example, you want to install $CARSPATH/modules/common/reports/tst. *Make* will install such a file into $CARSPATH/install/arc/common. Fileperms will search $CARSPATH/system/lib/fileperms.s for an entry that deals specifically with $CARSPATH/install/arc/common /tst.arc. Failing to find one, fileperms will search for a generic entry for the $CARSPATH/install/arc/common directory. Failing to find one, fileperms will use the generic entry for everything under $CARSPATH/install/arc. The entry will be in $CARSPATH/system/lib/fileperms.s that starts with: ./arc/*/?*

Fsck Errors That Reoccur

When a system crashes and reboots, it tries to repair file system inconsistencies using the *fsck* program. If errors occur in the root (/) section of the disk, *fsck* sometimes appears to fix the problem; but when the system automatically reboots, the errors are still present causing an infinite loop of reboots.

The problem is that *fsck* fixes the file system directly, bypassing the system buffer pools which still have the bad version of the file system. This problem does not occur if the file system is unmounted when the *fsck* is done, but the root file system is never unmounted. The fix for this situation is to reset the system after the *fsck* fixes the root file system, but before the disks are synced. This is easiest in single user mode.

Locally Added Detail Window Causes Core Dump

If you created a detail window to appear in a Entry Library program, and you did not define the scrolling region with the same number of elements for each field, the following message, followed by a core dump, appears when you attempt to access the window:

```
_dmm_adjust: i=not_initialized 1=1 u=0 a=0 c=-1
_dmm_check: Garbled: i=not_initialized l=1 u=0 a=0 c=-1
```

Make sure that every field in the scrollgroup has the same number of elements and that the number of elements displayed on the screen match.

Print Jobs Sent to Spooler Do Not Print

When print jobs do not go through a spooler, this can result from problems from a previous print job that halted unexpectedly. To clear the spooler, do the following:

1. Enter the following commands to reset and restart the spooler:
   - **lpreset <spooler>**
   - **lpc -P <spooler> -s**
2. Did the above commands fix the problem?
   - If yes, you are done.
   - If no, go to step 3.
3. If the above commands did not fix the problem, use *grep* to locate the printing process, then enter the following:
   - **kill -9 <process>**
4. Enter the following commands to reset and restart the spooler:
   - **lpreset <spooler>**
   - **lpc -P <spooler> -s**

APPENDIX – CX UNIX COMMANDS

Overview

Introduction

This appendix contains an alphabetically organized list of CX-specific UNIX commands. Jenzabar provides these commands in addition to the standard set of commands provided with the UNIX operating system.

Descriptions of Commands

Information about the commands in this appendix is provided in the following blocks:

Purpose
A quick reference summary of the command.

Synopsis
The command's syntax including all available parameters and options.

Description
A detailed description of the command and its features.

Options
A detailed list of the command's options

Files
A list of the pathnames and files associated with the command

See Also
A listing of associated commands, files, or documents that provide more information

Troubleshooting
A list of tips to keep in mind when processing the command

addlogin

Purpose

The addlogin command creates new user logins.

Synopsis

addlogin

Description

The addlogin command is an interactive process used to add new users to CX. To ensure that all the sub-processes execute properly, you must have a *carsroot* or *super-user (su*) login when executing the command.

The addlogin command prompts you for the following:

- The new user name to be added
- The standard CX user name for reference
- The login menu (if one is applicable) for the new user

The system displays the data entered for your review before continuing to add the user to the group(5) and passwd(1)(8) files. After the system adds the user to the passwd file, the system requests that the user enter a password in order to initialize the password. Lastly, the system initializes the user's home directory by adding *.login*, *.cshrc*, and similar files to the user's home directory.

> **Note:** To delete an improperly added user, execute the dellogin command.

Files

${CARSPATH}/install/utl/addlogin
Creates new user logins

${CARSPATH}/install/cis/dellogin
Deletes user logins

mnew
Initializes menu users' home directories

new
Initializes non-menu users' home directories

See Also

dellogin(CARSC)
passwd(1) passwd(5) group(5) vipw(8)

apstat

Purpose

The apstat command provides the status of an application server for Web applications.

Synopsis

apstat

Description

The apstat command provides the following information about the application server and its license limits and usage:

- The number of users
- The version number
- The total number of application servers used or registered to the system
- The maximum limit of the number of concurrent users
- The number of users currently using the application server
- The number of maximum concurrent users that have used the server
- The total number of users that have used the server
- The number of requests that have been denied due to exceeding the concurrent limit
- The number of licenses in use

Processing Notes

As processes run they obtain a license and release the license when terminated normally. However, errors can be caused by abnormal termination or loss of the PC connection and the license is not released. The apstat command reviews the licenses being held to ensure that they are still actively in use. It frees any licenses that have been allocated but are no longer in use.

apsetkey

Purpose

The apsetkey command allows you to register a machine for Web application use after you have obtained a license key from CX.

Synopsis

apsetkey licensed-users cpu-id apserver-name

Description

You run the apsetkey command from the shell to register your application server with the license key you have received from CX. For additional information about registering your app server, refer to the *License Manager* section in the *CX System Reference Technical Manual*.

Options

licensed-users

The number of licensed users. 00 indicates an unlimited number.

CPU ID

The CPU ID argument specifies your machine's identification number that will be registered with your license key.

apserver-name

The apserver name specifies the app server being licensed.

Sample

00883b9da234528a39adr_aps

catat

Purpose

The catat command concatenates scheduled process files.

Synopsis

catat [organization_file] [time]

Description

The purpose of the catat command is to concatenate jobs queued in the AT spool directory, (e.g., /usr/spool/cron/atjobs or /usr/spool/at). When you use the command, the jobs queued at the time of the command's issuing do not conflict for processing time and swap space. The command does not affect the execution of files created by other owners, nor does the command affect the order of jobs in a queue.

Without option arguments, catat concatenates all AT processes into one file to begin execution immediately. The two optional option arguments allow jobs to be concatenated into multiple queues based on a grouping of users, and to begin execution at the time specified.

Options

organization_file

The organization_file argument is a file used to determine organization of the job queues. The default is /dev/null, or 1 queue. The system searches the organization file each time the owner of an AT file is determined. The system adds the AT file to the job queue for the queue of which the owner is a member. If the owner is not in any of the specified job queues, the system adds the AT file to a "default" job queue. The format of the organization file is as follows (similar to /etc/group):

```
queue1:user1,user2,user3,...
queue2:user6,user7,user8,...
   ...
queuen:user100,user101,usern,...
```

You can use as the queue name any alpha-numeric string that does not conflict with user names. The user names are specified in the /etc/passwd file.

As a special case, you can use the /etc/group file as the organization file in which case the users are categorized by their login group.

time

The time argument specifies the time in which to start all the job queues. The default is immediately. Enter it in 24 hour time.

> **Note:** If you use the time argument, you must first specify an organization_file argument.

Processing Notes

The catat command moves the original AT files to $CARSPATH/spool/catat and creates new AT files in the AT spool directory for each job queue to be run concurrently. The system moves the original AT files to $CARSPATH/spool/catat/past when they are executed and saves any output or error messages that are not redirected in '.out' files in that directory.

When run on a daily basis by root or in CRONTAB, the catat command removes files from $CARSPATH/spool/catat/past that are over 1 day old. The Jenzabar system coordinator should review the /usr/spool/catat/past directory periodically.

Sample

Below is a sample organization_file followed by the command line from the shell for execution of CATAT.

```
boff:cashier,payroll,payable,controll
admt:admstaff
regr:regist,regstaff
comp:jones,smith,smythe
faid:fastaff
devl:develop,alumni
cis:jim,joe,mary
```

```
$SCPPATH/util/catat.scp $CARSPATH/spool/catat/groups 2000
```

The above command line create the job queues based on the organization_file $CARSPATH/spool/catat/groups, which will begin execution at 2000 hours, or 8:00 p.m.

Files

/usr/spool/at
The at command spool directory

$CARSPATH/spool/catat
The catat job queue directory.

/usr/groups
The system file for user groups.

/etc/passwd
The system file of valid login user.

$UTLPATH/sush
The su and execute a shell script.

$SCPPATH/util/catat.scp
The catat script.

See Also

at(1), cut(1), groups(5), passwd(1), passwd(5)

cgrep

Purpose

The cgrep command searches a file for a pattern.

Synopsis

cgrep [option] ... [string] [file] ...

Description

The cgrep command searches the input files for lines matching a pattern. The system reads the specified file name from the standard input, or if you specified no files, the system reads the standard input. The system usually copies each line found to the standard output.

The cgrep command is designed to allow more efficient searches than the standard grep family of utilities. Its pattern is a fixed string (only one string allowed).

Options

The following options are recognized:

-v

All lines but those matching are printed.

-x

(Exact) only lines matched in their entirety are printed.

-c

Only a count of matching lines is printed.

-number

Only the specified number of matches are found before quitting.

-l

The names of files with matching lines are listed (once) separated by new lines.

-q

Quick searches are used on the files (must be sorted).

-n

Each line is preceded by its relative line number in the file.

-l

The case of letters is ignored in making comparisons (e.g., upper and lowercase are considered identical).

-s

Silent mode. Nothing is printed (except error messages). This is useful for checking the error status.

-w

The expression is searched for as a word (as if surrounded by `<' and `>', see **ex (1).)**

-f

The string is taken from the file.

CAUTION: When using wild card characters $ ^ and \ in a string, remember that these characters are also meaningful to the Shell. To avoid problems, enclose the entire string argument in single quotes (‘).

Special Characters

The cgrep command accepts the following special characters:

****

The character \ followed by a single character other than new line matches that character.

^

The character ^ matches the beginning of a line if used at the beginning of the string.

$

The character $ matches the end of a line if used at the end of the string.

See Also

ex(1), sh(1)

Troubleshooting

- Lines are limited to 512 characters; longer lines may be matched incorrectly.
- The command’s Exit status equals:
  - 0 if any matches are found
  - 1 if no matches are found
  - 2 if syntax errors are found

clocate

Purpose

The clocate command locates filename(s) within CX.

Synopsis

clocate [string]

Description

The clocate command builds and searches a list of filenames from specific directories in CX and displays the locations that match the target string. The string argument may contain regular expression metacharacters in the format for grep(1). In addition to providing the location of the file(s), clocate attempts to give a description of the file based on its location.

Examples

The first example searches for filenames that contain only 'id'. The second example searches for filenames that begin with 'idpr' and are followed by anything.

You enter:
% **clocate id**
Looking for: id

Results:
data/common/id.dat is a data file in the common track under data
data/common/id.idx is an index file in the common track under data
$SCRPATH/Cislib/libtbl/id.scr is a program screen file
schema/common/id is a file in the common track under schema

You enter:
% **clocate 'idpr.*'**
Looking for: idpr.*

Results:
idpr.*
$FRMPATH/common/idprofile.frm is a PERFORM screen file
$FRMPATH/personnel/idprofpers.frm is a PERFORM screen file
$OPTPATH/common/screens/idprofile.opt is a menu option file
$OPTPATH/personnel/screens/idprofpers.opt is a menu option file
%

Troubleshooting

- The message “No Files were Found for id“ indicates that the system could not find a matching filename.
- The message “Building New File List - Please Wait” indicates the system is rebuilding the name list
- The command’s Exit status equals:
    - 0 if no errors occurred
    - 1 if errors occurred

Copyin

Purpose

The copyin command copies a file into the system.

Synopsis

copyin [-h -m -l -n -Z -v -t -u -x] [-f source] [-o opts] [-p patterns]

Description

The copyin command specifies various parameters for data being copied into the system from tape or disk. You can use the -t option to list the files instead of copying them.

Options

The following options are recognized:

-h
High density on tape drive.

-m
Medium density on tape drive.

-l
Low density on tape drive.

-n
No rewind on tape drive.

-Z
Input is compressed.

-v
Verbose listing of files.

-t
List instead of copying the files.

-u
Force an update regardless of modify times.

-x
Display the actual CPIO commands executed.

-f <source>
Specifies the source of the Input. Default is /dev/rmt/0m; '-' indicates stdin.

-o <opts>
Specifies additional options to CPIO.

-p <patterns>
Specifies the CPIO patterns. The default is *.

See Also

copyout

Troubleshooting

On IBM AIX systems, if a DAT tape will not read, but for other functions such as backup and archiving, the tape drive works fine, check the blocksize parameter for the tape drive.

```
% mt -f /dev/rmt0  status
```

You can use the “smit” command to reset the blocksize value. Reset the value to 0. This should not affect any other tape functions.

Copyout

Purpose

The copyout command copies data from the system to a file or disk.

Synopsis

copyout [-h -m -l -n -Z -v -x] [-f dest] [-o opts] [-C cddir] [-d finddir]

Description

The copyout command specifies various parameters for data being copied from the system to tape or disk.

Options

The following options are recognized:

-h

High density on tape drive.

-m

Medium density on tape drive.

-l

Low density on tape drive.

-n

No rewind on tape drive.

-Z

Compress the output.

-v

Verbose listing of files.

-x

Display actual CPIO commands executed.

-f <dest>

Output to specified destination. The default is /dev/rmt/0m; '-' indicates stdout; -x option void.

-o <opts>

Specifies additional options to CPIO.

-C <cddir>

Change to the specified directory before the next -d option.

-d <finddir>

Copy all files from 'find <finddir>'. finddir may be a quoted list of blank separated directories. A list of files comes from stdin by default.

See Also

copyin

cpdir

Purpose

The cpdir command copies directory contents to a new location.

Synopsis

cpdir source target

Description

The cpdir command copies the contents of one directory subtree to another location. If the target directory does not exist, cpdir attempts to create it and set the same permissions, group, and owner as the source directory.

The cpdir command uses the tar(1) UNIX command to do the copying.

CAUTION: You can ensure permissions preservation on all files if you use cpdir as root (superuser).

Troubleshooting

The command's Exit status equals:

- 0 if no errors occurred
- 1 if errors occurred

ctail

Purpose

The ctail command copies the named file to the standard output beginning at a designated place.

Synopsis

ctail [-qfsN] [-N | +N] [file]

Description

The ctail command copies the named file to the standard output beginning at a designated place. If you don't specify a file, the system reads the standard input. Copying begins at the distance +number from the beginning, or -number from the end of the input. Number is the number of lines.

Options

f

Causes ctail to not quit at end of file, but rather wait and try to read repeatedly in hopes that the file will grow.

q

Causes ctail to not output error messages, but operate in quiet mode.

s

Causes ctail to use the specified size as the internal block size for reading/saving. The default (and minimum) is 32000 bytes.

See Also

dd(1), tail(1)

cutsheet

Purpose

The cutsheet command pauses after every page of output.

Synopsis

cutsheet [-n lines] [-p printer] [-m message] [file] ...

Description

The cutsheet command copies its input files to the printer and pauses after every lines number of lines. The system reads the specified file name from the standard input, or if you specified no files, the system reads the standard input.

If you do not specify a printer, the system copies each line to the standard output. The default number of lines is 66. You can change the pause prompt by specifying a message. If you do specify a printer, the system disables the spooler while cutsheet uses the device. The printer device will also be opened with exclusive-use mode.

The system writes prompt messages to the tty device and reads the responses from the tty device. The system writes all errors to standard error.

Examples

The first example prints the prepared file of letters.out to the printer known as nec2. The second prints the rolodex output to the printer spinw.

```
% cutsheet -p nec2 letters.out
% cutsheet -n 22 -p spinw rolodex.out
```

Troubleshooting

- Lines are limited to 1024 characters; longer lines will be counted incorrectly.
- The command's Exit status equals:
    - 0 if no errors occurred
    - 1 if errors occurred

dbmmanage

Purpose

The dbmmanage command allows you to view and perform some maintenance on authentication files.

Synopsis

dbmmanage

Description

The dbmmanage command allows you to display, add, and update users in the Web User Authentication files. Separate authentication files are created for students and faculty to allow web access. These files are located in:

/opt/apache/carsi-*/var/authdb/

See Also

Setup_web_dbm

dbreport

Purpose

The dbreport command prints database dictionary reports.

Synopsis

dbreport [-stracp] datbase_name

Description

The CX Database Report (dbreport) command provides an accounting of the current status of the INFORMIX database dictionary

The dbreport command reads the INFORMIX data dictionary file to produce a listing of all database files and file names, relations, attributes, composite keys, and permissions in the combination specified by the options passed for the database_name specified on the command line. If you do not pass options, the system provides detailed listings for each category.

Options

-s

Provides a summary listing of the database dictionary categories. The default is a summary listing of each category. If other options are passed, -s will only summarize the areas not specified.

-t

Creates the detailed string table listing of filenames, file names, and the location of each.

-r

Creates the detailed listing of relations: file names.

-a

Creates the detailed listing of attributes: field names, their length and location.

-c

Creates the detailed listing of composite key indexes.

-p

Creates the detailed listing of permissions for each file.

Files

- ${CARSPATH}/install/utl/dbreport (data dictionary report)
- database_name.dbd (INFORMIX data dictionary)

dbsu

Purpose

The dbsu command allows programs to run with additional permissions than those granted to users.

Synopsis

dbsu [-l]

Description

The dbsu command allows programs to run with additional permissions. For example, Cashier is granted permissions to add and update the general ledger that you would not give directly to an end user.

Running a program setuid sets the effective user ID but keeps the real user-ID the same. Since Informix identifies the user that is accessing the database from the real user-ID, dbsu sets the real user-ID to the effective user-ID of the setuid program, and then runs the program as the setuid user. This achieves the setuid effect in a way that Informix can use.

Options

The -l argument sets the real user-ID to what the getlogin() call returns.

Files

${CARSPATH}/opt/carsi/install/utl/dbsu (creates additional permissions)

dellogin

Purpose

The dellogin command deletes CX user login names.

Synopsis

dellogin

Description

The dellogin command is an interactive process used to remove users from CX. To ensure that so all the sub-processes execute properly, you must have a *carsroot* or *super-user (su*) login when executing the command.

CAUTION: Use the dellogin command with care and sparingly so that audit trails in the accounting system are not destroyed.

The dellogin command prompts you for the user name to be deleted. The command also prompts for the name of the user who will own any files that currently owned by the user to be deleted. You can choose to have all files removed from the user's directory, and the system prompts you for the removal of each file.

Files

- ${CARSPATH}/install/cis/dellogin (deletes user logins)
- ${CARSPATH}/install/utl/addlogin (creates new user logins)
- /bin/rm (removes system files)

See Also

addlogin(CARSC)

fileperms

Purpose

The fileperms command maintains proper permissions.

Synopsis

fileperms [-auctfSve] [-F user] [filenames...]

Description

The fileperms command is used to maintain the necessary file permissions for CX. The permissions are listed in the fileperms table discussed below.

Options

The actions of the fileperms command are governed by these options:

-a

Assign (update) all permissions based on the fileperms table entries. This option is used to update the permissions for the entire CX product. (same as -tu)

-u

Update permissions as specified in the fileperms table. This option must be specified in order for any permissions to be modified.

-c

Check the permissions for the filenames specified against the appropriate fileperms table entries and print the comparison if there is a difference (this is the default action).

-t

Expand the fileperms table entries to obtain a filename list. This option is useful when it is not appropriate to specify the filename list on the command line or as standard input (stdin). It is typically used with the -u and/or -S options.

-f

Force the file owner to be the user id (UID) of the user running fileperms. This option is only useful if there is an asterisk '*' in the owner field of the fileperms table and the -u option is specified.

-F

This option is similar to the -f option, but allows the new owner to be specified as user. Any user with a user id (UID) of 0, such as root, is invalid with this option.

-S

Only do setuid root operations. This option is typically used with the **-a** option so that only the setuid root entries in the fileperms table will be updated.

-v

Give verbose output of progress.

-e

Output internal error messages. (useful for debugging)

Processing Notes

If you do not specify the -t option, fileperms gets the list of filenames from the command line. If you do not specify a file name on the command line, the system reads the standard input (stdin) to obtain the list.

For security purposes, fileperms will not allow a non-root user to modify the permissions of a setuid root file. If you do not specify the **-**t option, and the system asks fileperms to process a setuid root file, and notifies the user with an error message. If specify the -t option, the system suppresses the error message.

Permissions

The fileperms table defines the permissions for CX. Its source location, $CARSPATH/system/lib/fileperms.s, is only accessible by root. The table is installed as $CARSPATH/install/sys/lib/fileperms with a mode of 4400, an owner of root, and a group of carsprog. Any change to the table's permissions will cause fileperms to not recognize the table as valid. Each entry in the table consists of four white-space separated fields: filename, mode, owner, and group.

The filename field may contain environment variables, such as ${CARSPATH}, and use csh(1) filename metacharacter notation. If you specify the filename as an absolute path of a directory, fileperms considers it to be the current directory, and considers subsequent relative filenames as relative to this current directory. A trailing slash '/' on a filename containing metacharacters causes the filename to only match directories. If the specified filename matches two or more entries in the fileperms table, the system uses the last entry. The mode field is the numeric representation, such as 775, of the UNIX permissions for the file. The owner field is a valid UNIX username as specified in /etc/passwd, and the group field is a valid UNIX group as specified in /etc/group. If an asterisk '*' appears in either the mode, owner, or group fields, the system considers the value to be irrelevant.

Examples

In order to set the permissions for the entire CX product, type the following command:

```
% su
Password:
# fileperms -a
# exit
```

Note: If you do not run this command as root, the systems skips the setuid root entries in the fileperms table.

This example shows how to use **fileperms** to set the permissions of just the setuid root files in CX. This is useful if a non-root user has just done a global reinstall of $CARSPATH/src or $CARSPATH/modules.

```
% su
Password:
# fileperms -aS
# exit
```

To update the permissions on a single file, such as $CARSPATH/install/utl/menu, follow this example.

```
% fileperms -u $CARSPATH/install/utl/menu
```

Files

- ${CARSPATH}/system/Make/user/Make/src (location of the fileperms C source code - only accessible by root)
- ${CARSPATH}/install/sys/lib/fileperms (table containing CX permissions)

See Also

chown(1), chgrp(1), chmod(1), csh(1), passwd(4), group(4).

Troubleshooting

The fileperms table must contain a reference to the symbolic link as well as the physical location with identical relative paths under each one.

findstring

Purpose

The findstring command finds a string of characters in files.

Synopsis

findstring [-v] string-to-find [directory-path] [nodot|nodoto] [norcs]

Description

The findstring command uses find(1) and grep(1) (or `egrep(1))' to locate patterns in files within a specified directory tree.

The findstring command accepts various options on files or directories to ignore and with the -v will output various progress messages. With the nodot flag, the command does not check files with a '.' (dot) in their name and with nodoto specified, it will not check files ending in '.o'. The norcs flag indicates that RCS directories should be skipped.

lnspooler

Purpose

The lnspooler command links previously defined spooler definitions from one release to another. For example, from your production release to a train release.

Synopsis

lnspooler (entered with interactive prompting)

Description

Determines defined printers of a release based on the prtab file from the production release. Alternate CX releases should have the prtab symbolically linked to the production release. Alternate releases get their printer lists from this prtab file.

lpc

Purpose

The lpc command reports on the status of the print spooler.

Synopsis

lpc [-P printer] [-isk] [-f [form]] [-o [owner]]

Description

The CX Line Print spooler system sets up files for printing by another process. This frees up the current process to continue other tasks. Various functions provide the ability to remove files spooled, list spooled files, set particular form types, etc. The spooling system also logs spooling usage in pages, lines, and characters by owner name, if desired.

The lpc command, without any parameters, or lpq, reports on the status of the print spooler. Each entry in the queue is printed showing the owner of the queue entry, the type of form the job is to be printed on, an identification number, the size of the entry in characters, the percentage printed or asterisk (this field is non-blank only on the file being printed), the number of copies to be printed, and the file which is to be printed. The id is useful for removing a specific entry from the printer queue using lprm.

Options

The lpc command has several options to control the printer.

-i

Causes printing of queued jobs to stop after the current job. Print jobs can still be added to the print queue but they will not be printed.

-k

Causes printing of queued jobs to stop immediately(within 15 seconds). Print jobs can still be added to the print queue but they will not be printed.

-s

Starts (or restarts) the spooler. Allows the jobs queued to beginning printing after an idle or kill.

-f

Specifies that a particular form has been loaded in the printer and that only jobs with that form type should be printed. This command can only be used when the printer is inactive. If no form type is given, then any print job with no form type may be printed.

-f IGNORE_FORMTYPE

Prints any file on the printer and ignores any form type. If you use this option, ensure the paper in the printer is acceptable for all output.

Note: If users will be able to enter this option using the Load Form Type menu option, you must make the form type parameter longer than 12 (as defined in menuopt) in menuopt/utilities/others/lpc.form.

-o

Specifies that only jobs for a particular owner may be printed. This command can only be used when the printer is inactive. If no owner is given, then any print job from any owner may be printed.

Files

- ${CARSPATH}/install/sys/util/lpd (printer daemon)
- ${CARSPATH}/install/sys/util/lpf (printing filter; banner, underline...)
- ${CARSPATH}/install/utl/lpr (main spooler program)
- ${CARSPATH}/install/utl/lpcf (change formtype)
- ${CARSPATH}/install/utl/lpcn (change number of copies)

- ${CARSPATH}/install/utl/lpmv (move to another spooler)
- ${CARSPATH}/install/cis/lpc (report on the status of the print spooler)
- ${CARSPATH}/install/cis/lpr (lpr script to process proptions file)
- ${CARSPATH}/install/cis/lpreset (reset the spooler)
- ${CARSPATH}/install/cis/lprm (remove from spooler)
- ${CARSPATH}/spool/{spooler name} (spool directory)
- /usr/adm/{spooler name}acct (spooler accounting files)
- /dev/{spooler name} (spooler device)
- ${CARSPATH}/install/sys/lib/prtab (printer initialization file)
- ${CARSPATH}/install/sys/lib/proptions (printer options file: filters, banner)
- ${CARSPATH}/install/utl/mkspooler (create new spooler)
- ${CARSPATH}/install/utl/rmspooler (remove existing spooler)

See Also

pr(1), mail(1), write(1), printers(5), mkspooler(1), rmspooler(1), lpracct(1)

lpcf

Purpose

The lpcf command changes the formtype on a spooled file.

Synopsis

lpcf [-P spooler_name] formtype id_number ...

Description

The CX Line Print spooler system sets up files for printing by another process. This frees up the current process to continue other tasks. Various functions provide the ability to remove files spooled, list spooled files, set particular form types, etc. The spooling system also logs spooling usage in pages, lines, and characters by owner name, if desired.

The lpcf command is the csh script used to change the formtype on a spooled file.

Options

-P

Requires a spooler name. If no spooler_name is given, the spooler defaults to 'lpr'.

formtype

The formtype value passed to lpcf should be 10 characters or less in length.

Example: lpcf narrow mari wide

id_number

You can pass multiple ID numbers or any other string value (owner of files, existing formtype) to accept the new formtype value.

Example: lpcf wide 10241 17521

Files

- ${CARSPATH}/install/sys/util/lpd (printer daemon)
- ${CARSPATH}/install/sys/util/lpf (printing filter; banner, underline...)
- ${CARSPATH}/install/utl/lpr (main spooler program)
- ${CARSPATH}/install/utl/lpcf (change formtype)
- ${CARSPATH}/install/utl/lpcn (change number of copies)
- ${CARSPATH}/install/utl/lpmv (move to another spooler)
- ${CARSPATH}/install/cis/lpc (report on the status of the print spooler)
- ${CARSPATH}/install/cis/lpr (lpr script to process proptions file)
- ${CARSPATH}/install/cis/lpreset (reset the spooler)
- ${CARSPATH}/install/cis/lprm (remove from spooler)
- ${CARSPATH}/spool/{spooler name} (spool directory)
- /usr/adm/{spooler name}acct (spooler accounting files)
- /dev/{spooler name} (spooler device)
- ${CARSPATH}/install/sys/lib/prtab (printer initialization file)
- ${CARSPATH}/install/sys/lib/proptions (printer options file: filters, banner)
- ${CARSPATH}/install/utl/mkspooler (create new spooler)
- ${CARSPATH}/install/utl/rmspooler (remove existing spooler)

See Also

pr(1), mail(1), write(1), printers(5), mkspooler(1), rmspooler(1), lpracct(1)

lpcn

Purpose

The lpcn command changes the number of copies on a spooled file.

Synopsis

lpcn [-P printer_name] number_copies id_number ...

Description

The CX Line Print spooler system sets up files for printing by another process. This frees up the current process to continue other tasks. Various functions provide the ability to remove files spooled, list spooled files, set particular form types, etc. The spooling system also logs spooling usage in pages, lines, and characters by owner name, if desired.

The lpcn command is a csh script that will change the number of copies on a spooled file. It follows the same format as lpcf except that it requires the number of copies instead of a new formtype name.

Example: lpcn 3 10241

lpcn 1 mari

Files

- ${CARSPATH}/install/sys/util/lpd (printer daemon)
- ${CARSPATH}/install/sys/util/lpf (printing filter; banner, underline...)
- ${CARSPATH}/install/utl/lpr (main spooler program)
- ${CARSPATH}/install/utl/lpcf (change formtype)
- ${CARSPATH}/install/utl/lpcn (change number of copies)
- ${CARSPATH}/install/utl/lpmv (move to another spooler)
- ${CARSPATH}/install/cis/lpc (report on the status of the print spooler)
- ${CARSPATH}/install/cis/lpr (lpr script to process proptions file)
- ${CARSPATH}/install/cis/lpreset (reset the spooler)
- ${CARSPATH}/install/cis/lprm (remove from spooler)
- ${CARSPATH}/spool/{spooler name} (spool directory)
- /usr/adm/{spooler name}acct (spooler accounting files)
- /dev/{spooler name} (spooler device)
- ${CARSPATH}/install/sys/lib/prtab (printer initialization file)
- ${CARSPATH}/install/sys/lib/proptions (printer options file: filters, banner)
- ${CARSPATH}/install/utl/mkspooler (create new spooler)
- ${CARSPATH}/install/utl/rmspooler (remove existing spooler)

See Also

pr(1), mail(1), write(1), printers(5), mkspooler(1), rmspooler(1), lpracct(1)

lpinit

Purpose

The lpinit command initializes a spooled device for testing.

Synopsis

lpinit [printer_name]

Description

The CX Line Print spooler system sets up files for printing by another process. This frees up the current process to continue other tasks. Various functions provide the ability to remove files spooled, list spooled files, set particular form types, etc. The spooling system also logs spooling usage in pages, lines, and characters by owner name, if desired.

The lpinit command is a csh script used to initialize a spooled device for testing.

Example: lpinit lpr3

Files

- ${CARSPATH}/install/sys/util/lpd (printer daemon)
- ${CARSPATH}/install/sys/util/lpf (printing filter; banner, underline...)
- ${CARSPATH}/install/utl/lpr (main spooler program)
- ${CARSPATH}/install/utl/lpcf (change formtype)
- ${CARSPATH}/install/utl/lpcn (change number of copies)
- ${CARSPATH}/install/utl/lpmv (move to another spooler)
- ${CARSPATH}/install/cis/lpc (report on the status of the print spooler)
- ${CARSPATH}/install/cis/lpr (lpr script to process proptions file)
- ${CARSPATH}/install/cis/lpreset (reset the spooler)
- ${CARSPATH}/install/cis/lprm (remove from spooler)
- ${CARSPATH}/spool/{spooler name} (spool directory)
- /usr/adm/{spooler name}acct (spooler accounting files)
- /dev/{spooler name} (spooler device)
- ${CARSPATH}/install/sys/lib/prtab (printer initialization file)
- ${CARSPATH}/install/sys/lib/proptions (printer options file: filters, banner)
- ${CARSPATH}/install/utl/mkspooler (create new spooler)
- ${CARSPATH}/install/utl/rmspooler (remove existing spooler)

See Also

pr(1), mail(1), write(1), printers(5), mkspooler(1), rmspooler(1), lpracct(1)

Troubleshooting

The lpinit command does not work well with host initiated connections on DECServers.

lpmv

Purpose

The lpmv command moves a spooled job from one spooler to another.

Synopsis

lpmv id_number from_queue to_queue

Description

The CX Line Print spooler system sets up files for printing by another process. This frees up the current process to continue other tasks. Various functions provide the ability to remove files spooled, list spooled files, set particular form types, etc. The spooling system also logs spooling usage in pages, lines, and characters by owner name, if desired.

The lpmv command is a csh script used to move a spooled job from one spooler to another. It requires the specific id number of the file to be moved, the queue it is to be moved from, and the new queue.

Example: lpmv 10241 lpr nec

Files

- ${CARSPATH}/install/sys/util/lpd (printer daemon)
- ${CARSPATH}/install/sys/util/lpf (printing filter; banner, underline...)
- ${CARSPATH}/install/utl/lpr (main spooler program)
- ${CARSPATH}/install/utl/lpcf (change formtype)
- ${CARSPATH}/install/utl/lpcn (change number of copies)
- ${CARSPATH}/install/utl/lpmv (move to another spooler)
- ${CARSPATH}/install/cis/lpc (report on the status of the print spooler)
- ${CARSPATH}/install/cis/lpr (lpr script to process proptions file)
- ${CARSPATH}/install/cis/lpreset (reset the spooler)
- ${CARSPATH}/install/cis/lprm (remove from spooler)
- ${CARSPATH}/spool/{spooler name} (spool directory)
- /usr/adm/{spooler name}acct (spooler accounting files)
- /dev/{spooler name} (spooler device)
- ${CARSPATH}/install/sys/lib/prtab (printer initialization file)
- ${CARSPATH}/install/sys/lib/proptions (printer options file: filters, banner)
- ${CARSPATH}/install/utl/mkspooler (create new spooler)
- ${CARSPATH}/install/utl/rmspooler (remove existing spooler)

See Also

pr(1), mail(1), write(1), printers(5), mkspooler(1), rmspooler(1), lpracct(1)

lpr

Purpose

The lpr command queues files to be printed.

Synopsis

lpr [-Bnmrcw] [-b banner] [-f form] [-o filter] [file ...]

Description

The CX Line Print spooler system sets up files for printing by another process. This frees up the current process to continue other tasks. Various functions provide the ability to remove files spooled, list spooled files, set particular form types, etc. The spooling system also logs spooling usage in pages, lines, and characters by owner name, if desired.

The lpr command causes the named files to be queued for printing. If no files are named, the standard input is read and copied. In general, files are linked to for spooling. By linking, disk usage is reduced. If the linking fails (i.e.,. spool directory on a separate file system), or if the -c option is specified, a copy is made of the file for printing. This allows a snapshot to be taken of the file if it is expected that the file might change during the printing process.

Options

-m
Causes notification via mail (1) to be sent when the job completes.

-w
Causes notification via write (1) to be sent when the job completes.

-r
Removes the file after printing it.

-c
Makes a copy of the file to be printed before returning to the user.

-f
The file will be printed only when the form is loaded in the printer.

-b
Causes the banner message to appear on the banner page.

-B
Causes the owner's name (login name) to be used as the banner.

-n
Causes the specified number of copies to be printed.

-o
Specifies the (output) filter to be used.

Files

- ${CARSPATH}/install/sys/util/lpd (printer daemon)
- ${CARSPATH}/install/sys/util/lpf (printing filter; banner, underline...)
- ${CARSPATH}/install/utl/lpr (main spooler program)
- ${CARSPATH}/install/utl/lpcf (change formtype)
- ${CARSPATH}/install/utl/lpcn (change number of copies)
- ${CARSPATH}/install/utl/lpmv (move to another spooler)
- ${CARSPATH}/install/cis/lpc (report on the status of the print spooler)
- ${CARSPATH}/install/cis/lpr (lpr script to process proptions file)
- ${CARSPATH}/install/cis/lpreset (reset the spooler)
- ${CARSPATH}/install/cis/lprm (remove from spooler)

- ${CARSPATH}/spool/{spooler name} (spool directory)
- /usr/adm/{spooler name}acct (spooler accounting files)
- /dev/{spooler name} (spooler device)
- ${CARSPATH}/install/sys/lib/prtab (printer initialization file)
- ${CARSPATH}/install/sys/lib/proptions (printer options file: filters, banner)
- ${CARSPATH}/install/utl/mkspooler (create new spooler)
- ${CARSPATH}/install/utl/rmspooler (remove existing spooler)

See Also

pr(1), mail(1), write(1), printers(5), mkspooler(1), rmspooler(1), lpracct(1)

Troubleshooting

The -r option only applies if the "linking" failed. It is not functional if you also use the -c option.

lpracct

Purpose

The lpracct command displays the status of the spooler accounting files.

Description

The system updates spooler accounting files (e.g.,. /usr/adm/lpracct) each time the spooler is used with owner name and usage statistics. If the file does not exist for a given spool queue, the system will not write accounting information by the spooling software.

The format of the accounting files consists of one line per entry (each job) containing owner name, number of pages printed, number of lines printed and number of characters printed.

Each spooler configured in the system has a separate accounting file. The accounting files reside in the directory /usr/adm and are named with the spool queue name followed by acct (e.g., /usr/adm/lpracct, /usr/adm/laseracct, etc).

These files are updated by the standard filter lpf and are typically created by the mkspooler command, although they can be created by the system administrator using the touch command as well.

See Also

lpr(1) lpc(1) mkspooler(1) rmspooler(1)

lpreset

Purpose

The lpreset command checks a spooler queue for jobs that may be printing.

Synopsis

lpreset [printer_name]

Description

The CX Line Print spooler system sets up files for printing by another process. This frees up the current process to continue other tasks. Various functions provide the ability to remove files spooled, list spooled files, set particular form types, etc. The spooling system also logs spooling usage in pages, lines, and characters by owner name, if desired.

The lpreset command is a csh script used to check a spooler queue for jobs that may be printing. If there are no 'lpd' processes for the spooler, the system removes any existing locks and starts the spooler.

Example: lpreset nec1

Files

- ${CARSPATH}/install/sys/util/lpd (printer daemon)
- ${CARSPATH}/install/sys/util/lpf (printing filter; banner, underline...)
- ${CARSPATH}/install/utl/lpr (main spooler program)
- ${CARSPATH}/install/utl/lpcf (change formtype)
- ${CARSPATH}/install/utl/lpcn (change number of copies)
- ${CARSPATH}/install/utl/lpmv (move to another spooler)
- ${CARSPATH}/install/cis/lpc (report on the status of the print spooler)
- ${CARSPATH}/install/cis/lpr (lpr script to process proptions file)
- ${CARSPATH}/install/cis/lpreset (reset the spooler)
- ${CARSPATH}/install/cis/lprm (remove from spooler)
- ${CARSPATH}/spool/{spooler name} (spool directory)
- /usr/adm/{spooler name}acct (spooler accounting files)
- /dev/{spooler name} (spooler device)
- ${CARSPATH}/install/sys/lib/prtab (printer initialization file)
- ${CARSPATH}/install/sys/lib/proptions (printer options file: filters, banner)
- ${CARSPATH}/install/utl/mkspooler (create new spooler)
- ${CARSPATH}/install/utl/rmspooler (remove existing spooler)

See Also

pr(1), mail(1), write(1), printers(5), mkspooler(1), rmspooler(1), lpracct(1)

Troubleshooting

The lpreset command does not determine whether or not the Letter Production System (LPS) or Form Production system (FPS) programs are currently using the spooler.

lprm

Purpose

The lprm command removes an entry from the line printer queue.

Synopsis

lprm [-P printer] [id] [file] [owner] [form] ...

Description

The CX Line Print spooler system sets up files for printing by another process. This frees up the current process to continue other tasks. Various functions provide the ability to remove files spooled, list spooled files, set particular form types, etc. The spooling system also logs spooling usage in pages, lines, and characters by owner name, if desired.

The lprm command removes an entry from the line printer queue. The id, filename, form type, or owner should be reported by the lpq command. All appropriate files will be removed. The id of each file removed from the queue will be printed. The optional printer name argument may be specified for print queues other than the default. (The default printer will be the first one listed by lpq.)

Files

- ${CARSPATH}/install/sys/util/lpd (printer daemon)
- ${CARSPATH}/install/sys/util/lpf (printing filter; banner, underline...)
- ${CARSPATH}/install/utl/lpr (main spooler program)
- ${CARSPATH}/install/utl/lpcf (change formtype)
- ${CARSPATH}/install/utl/lpcn (change number of copies)
- ${CARSPATH}/install/utl/lpmv (move to another spooler)
- ${CARSPATH}/install/cis/lpc (report on the status of the print spooler)
- ${CARSPATH}/install/cis/lpr (lpr script to process proptions file)
- ${CARSPATH}/install/cis/lpreset (reset the spooler)
- ${CARSPATH}/install/cis/lprm (remove from spooler)
- ${CARSPATH}/spool/{spooler name} (spool directory)
- /usr/adm/{spooler name}acct (spooler accounting files)
- /dev/{spooler name} (spooler device)
- ${CARSPATH}/install/sys/lib/prtab (printer initialization file)
- ${CARSPATH}/install/sys/lib/proptions (printer options file: filters, banner)
- ${CARSPATH}/install/utl/mkspooler (create new spooler)
- ${CARSPATH}/install/utl/rmspooler (remove existing spooler)

See Also

pr(1), mail(1), write(1), printers(5), mkspooler(1), rmspooler(1), lpracct(1)

make

Purpose

The make command (standard make targets & variables) maintains changes in the software.

Synopsis

make [targets & variables]

Description

The make processor manages and controls the configuration of the CX product from the application source code level to handling source for ACE reports, PERFORM screens, scripts, and utility programs. A number of CX standard make targets manage and control the configuration..

Standard Targets

The following list is a quick reference for CX make targets.

co F=?

Checks out the current version of the specified file(s).

ci install F= L=

Checks in the working version of the specified file(s) with the given log message and proceeds to install the appropriate file(s).

reinstall F=?

Installs the appropriate file(s) for the current version regardless if there is a previously install copy or not. This is only available if the file(s) are checked in.

reco F=?

Checks out file(s) that are already checked out and does not retain any changes. All changes are lost and a new copy of the current version is placed in the directory.

reci install F=? L=?

Checks in all changes as additional changes to the previous version with the given log message appended, and the appropriate file(s) installed.

Special Targets

expand F=?

Executes initial macro expansion on the file and leaves the copy in the filename.exp.

print

Prints the source files to the printer with banner page using the pr(1)filter.

lint

Executes lint(1)on C source files and produces a listing on stdout that can be consolidated using conslint.

analyze

Executes various analysis programs on a document file using spell(1,) style(1,) diction(1),etc.

depend

Rebuilds the make variable file (.makevar.mak.)

remake

Rebuilds the list (.makelist) of make files and executes the depend target.

def.o

Compiles the def.c, builds the dec.h, and rebuilds the make dependency file (.makedep.mak.)

mkspooler

Purpose

The mkspooler command creates new spooling devices.

Synopsis

mkspooler tty_name spooler_name [spooler_type]
mkspooler /dev/lp0 spooler_name parallel
mkspooler ip_address service_no spooler_name lan
mkspooler FILE spooler_name

Description

The mkspooler command is the procedure for creating new CX print spoolers. To ensure that all the sub-processes execute properly, you must have a *carsroot* or *super-user (su*) login when executing the command. The mkspooler command can create spoolers for serial printers, parallel printers and network printers. The command can also create a special spooler that sends its output to a file on disk instead of a printer. You should use lptest to test each new spooler.

Serial Printers

The mkspooler command uses three arguments to create a spooler for a serial printer.

- The first argument is the tty_name the printer is connected to.
- The second argument is spooler_name you wish to give this spooler.
- The third argument, spooler_type , is not required, but does help configure the spooler for the specific printer type. See the file ${CARSPATH}/system/etc/prtypes for a list of valid printer types.

Example: The following command line creates a spooler named "bop" (e.g., the business office printer) configured for a spooler type of "p300" (Printronix P300 - a basic line printer) and the second example creates the spooler 'lpr' with type of 'oki84'.

```
# mkspooler  ttyh4   bop  p300

# mkspooler  tty5p2  lpr  oki84
```

Parallel Printers

The mkspooler command uses three arguments to create a spooler for a serial printer.

- The device file for the parallel port of the computer /dev/lp0 is used instead of tty_name for the first argument.
- The second argument is the spooler name.
- A special printer type of Parallel has been set up for the third parameter. This type has no configuration settings. You can use the hp-ux command /usr/bin/slp to configure various options of the parallel port. See the man page on slp for more information.

Example: The following command creates a spooler for lpr for a parallel printer

```
# mkspooler  /dev/lp0  lpr  parallel
```

Network Printers

The mkspooler command requires four arguments to create a spooler for a network printer. The four arguments are:

- A specific IP (Internet Protocol) address (must be a unique number)
- A service number (a calculated number based on how you are connecting the printer)
- The spooler name
- The spooler type of lan

HP DTC

For printers being connected over an HP DTC, you can use the following formula to calculate the service number: (256 * ((32 * board_number) + port_number + 1)) +23

ANNEX

For printers being connected with a xylogics ANNEX, calculate the service number as follows: 7000 + port #

Lantronics Terminal Servers

For printers being connected with Lantronics terminal servers, calculate the service number as follows: 2000 + port #

HP JetDirect card

If an HP JetDirect card is installed in your printer, and the printer is connected to an Ethernet network, do the following:

Configure the JetDirect with an unique ip_address, then use a service number of 9100. Once you have connected the printer and figured out the ip_address and service number, you should be able to telnet to the ip_address and service number and what you type should appear on the printer. Type the following:

- telnet (ip_address) (service_number)

You should receive a message about connection established and have no prompt. Then whatever you type should appear on the printer. If this does not occur, you have a network problem that must be resolved before you can set up a spooler.

Example: If you can "telnet (ip_address) (service_number)" and what you type appears on the printer, then you can try to create a network spooler with the following command.

mkspooler (ip_address) (service_number) (spooler_name) lan

Update Environment Variables

After you create a print spooler, update the CARSPRINTER and CARSPRINTERS environment variables in $CARSPATH/skel/userprtab.s. $CARSPRINTER defines the first printer in the list. This is the default printer. $CARSPRINTERS defines all printers in the available printer list after the first printer. This file can limit a printer to a person or group and can limit access to a printer on a specific CX release to a specific person.

(carsroot) # cd $CARSPATH/skel
(carsroot) # make co F=userprtab.s
(carsroot # vi userprtab.s (add printers)
(carssroot) # make cii F=userprtab.s L='*log msg'*

Note: Some CX applications (such as Transcript) will not accept printer names that exceed six characters. Therefore, you should keep your printer names at six characters or less. The first printer created is usually named lpr.

Note: The CARSPRINTERS variable has a maximum of 80 characters.

Network Spooler Debugging

If after creating your spooler, it is not printing, try the following test:

telnet (spooler_name) (spooler_name)

Notice that this test does not specify the ip_address and service number. The system does not look up the address and number. If the above telnet test succeeds and the second test fails, you most likely have some kind of domain name server running, and the domain name server does not know about (spooler_name). The other possibility is that the spooler name appears more than once in each of the following two files: /etc/hosts, /etc/services.

File Output

The mkspooler command requires two arguments to create a spooler which sends its output to a file on disk.

1. The first argument is the literal keyword **FILE**
2. The second argument is the **spooler_name**

Note: The spooler will append its output to a file in /dev named spooler_name. Jenzabar recommends that you cp /dev/null /dev/spooler_name to clean out the file to avoid unchecked growth.

All Spoolers

Mkspooler will prompt you with the following:

- Enter special input filter commands if needed:
- If banner is always required, enter 'y' or string for banner:
- Enter special output filter command if needed:
- Do you wish to initialize Spooler Accounting for (spooler_name)?

Typically the filter 'pul' will be used as an input filter for printers that only handle the traditional concept of underlining with carriage returns and sending the buffer with underscores in it, then sending the line feed (e.g., Printronix P300). Most other printers will not need input filters.

If the system administrator wants banner pages printed for all jobs on a particular spool queue, it can be set up when prompted.

When the output filter is prompted for, the system administrator can specify a filter to handle special capabilities or options. For example, the HP2563 printer requires a carriage return and new line combination as the end of line sequence. The spooler_type configuration information can configure serial spoolers to handle this. The hpux slp configuration can handle this for parallel printers, but the output filter of ${CARSPATH}/install/sys/util/lpf -t is needed to accomplish this for network spoolers.

Files for Spooling

The following files are used for spooling:

- ${CARSPATH}/install/utl/mkspooler (create spooling devices on CX)
- ${CARSPATH}/install/sys/etc/prtypes (template for printer lines in /etc/printers)
- ${CARSPATH}/install/sys/lib/prtab (file of active printers)
- ${CARSPATH}/install/sys/lib/proptions (options file for specified printers)
- ${CARSPATH}/install/utl/lpr (spooling program)
- ${CARSPATH}/install/utl/queue_name (spooling program for queue_name)
- ${CARSPATH}/spool/queue_name (spooling directory in for spooler)
- /usr/adm/queue_nameacct (spooling accounting file for queue_name)

Related Scripts

For scripts to delete spooling devices from CX, see the following:

- rmspooler
- ${CARSPATH}/install/utl/rmspooler

newlogin

Purpose

The newlogin command creates new system logins.

Synopsis

newlogin newname [oldname [-u]] [-pnq] [-igdhsa opt_arg]

Description

The newlogin command is a utility program that simplifies the process of creating new logins on CX. The various options and option arguments provide flexibility to the process.

The newlogin command requires at minimum a user name to create. It will add the new user to both the /etc/passwd and /etc/group files.

Options

The first set of options allow the new user login to mimic an existing user login. Other options are used for the degree of prompting the operator for additional verification before continuing or for the degree of initialization of the new user.

new_name

The new user name to add the password and group files.

old_name

The example user name from the password file.

-u

Causes the program to use the example user name id number.

-p

Prompt interactively on each group approval for inclusion in that group.

-n

Do not call /bin/passwd for password assignment.

-q

Do not output progress messages.

-l

Requires the specified user id number as opt_arg.

-g

Requires a specified group id number or group name as opt_arg.

-d

Requires a description string for the password file as opt_arg. For example, the description for user "coord" might be "Jenzabar system coordinator: Joe Smith". Follow conventions already established in /etc/passwd. What is entered for the description string is displayed with the finger and f UNIX commands.

-h

Requires the home directory string as opt_arg. If not used, directory path for the old_name (reference name) user will be copied, replacing old_name with new_name.

-s

Requires a specified login shell name as opt_arg. Menu users need "${CARSPATH}/install/utl/menucsh". Non-menu users, or shell users need /bin/csh.

Note: If you want menu user to be able to FTP from the CX system to their PCs you have to change the src/common/single/noshell.c to include FTP as a filter.

-s

Requires one or more additional groups for the user in addition to any previous groups, each separated by commas.

Files

- /etc/passwd (password file)
- /etc/group (group file)
- /bin/passwd (add new password process)
- /bin/csh (csh for login shell name)
- /usr/ucb/finger (user data lookup program)
- /usr/ucb/f (another form of user data lookup program)
- ${CARSPATH}/install/utl/menucsh (menu "no-shell" login shell name)

See Also

- addlogin(CARSC), dellogin(CARSC)
- passwd(5), passwd(8), group(5), csh(1), finger(1)

printmenu, pmsort

Purpose

The printmenu command creates an outline of the CX menu structure.

Synopsis

printmenu [-btn] [-sX] [-iX] [-mN] [menu_file_name]
printmenu [-at] [-mN] [menu_file_name]

Description

The printmenu and pmsort commands provide the user with an outline of the CX Menu structure. Each process can either be run on the entire menu structure or a specific section. printmenu can also be run to provide the CX Acceptance Report.

The printmenu command creates the menu structure outline. If no menu_file_name is specified, it begins relative to the user's ${MENUPATH}, which in the majority of cases is ${CARSPATH}/install/mnu.

Options

-b
Inhibits printing of blocked menu options.

-t
Inhibits printing of the title line in the output.

-n
Inhibits printing of the menu statistics.

-sX
Allows the operator to specify the initial string (X) that precedes the menu option letter and short description. The default is a tab. Any argument (X) should immediately follow the -s option without spaces and be enclosed in single quotes.

-iX
Allows the operator to specify the indent string (X) between menu levels. The default is '. ' (a period and 4 spaces). Any argument (X) should immediately follow the **-i** option without spaces and be enclosed in single quotes.

-mN
Allows the operator to specify the maximum number of nesting levels (N) desired. The default is to include all levels.

-a
Creates the acceptance report of CX based on the menu structure. This can also be run for a specific area of the menu system. The output includes not only each menu option, but also its type (Menu, Report, Screen, Program, etc.).

menu_file_name
The path of the desired menu for the output relative to ${MENUPATH} including the menudesc.mnu file.

Files

- ${CARSPATH}/install/utl/printmenu (printmenu program)
- ${CARSPATH}/install/utl/reordpm (reorder program)
- ${CARSPATH}/install/utl/pmsort (printmenu sort script)
- /usr/bin/sort (UNIX sort utility)
- ${MENUPATH}/*.mnu (translated menu files)

See Also

- sort(1)
- Y command of the MENU processor

pmsort

Purpose

The pmsort command creates an alphabetically sorted listing of all CX menu options under the current menu.

Synopsis

pmsort [-btn] [-mN] [menu_file_name]

Description

The printmenu and pmsort commands provide the user with an outline of the CX Menu structure. Each process can either be run on the entire menu structure or a specific section.

The pmsort command creates an alphabetically sorted listing of all menu options with the path to each option relative to ${MENUPATH}. A specific menu_file_name , relative to ${MENUPATH} , can be passed to PMSORT as a parameter to create the listing for one segment of the menu system. Other parameters that can be passed to pmsort include the options to exclude blocked options (-b), exclude the title line (-t), exclude the menu statistics (-n) and specify the maximum number of menu nesting levels (-mN).

Files

- ${CARSPATH}/install/utl/printmenu (printmenu program)
- ${CARSPATH}/install/utl/reordpm (reorder program)
- ${CARSPATH}/install/utl/pmsort (printmenu sort script)
- /usr/bin/sort (UNIX sort utility)
- ${MENUPATH}/*.mnu (translated menu files)

See Also

- sort(1)
- Y command of the MENU processor

prtab and proptions

Purpose

The prtab and proptions commands provide current configuration information about spooled printers.

Description

The file ${CARSPATH}/install/sys/lib/prtab contains descriptive information about the various spooled printers on a system. The prtab and proptions files are modified by the mkspooler and rmspooler commands and are used to initialize a spooled printer device and spooling options by the lpr.

The format of the prtab file consists of one line for every printer on the system with the name of the printer, a colon, and an initialization statement, typically using the stty command to set the mode of the printer. The format of the proptions file consists of one line for every printer on the system with special options. Each line contains colon separated fields with the printer first, followed by the input filter(s), banner field, and output filter. If any of the 3 fields is missing, that portion is ignored. The banner field can be empty (equivalent to 'N') or consist of 'Y' or 'N' a specific banner string.

These files are read by various printer spooling programs to determined the default printer which should be the first printer described.

Files

- $CARSPATH/install/sys/etc/prtypes
- $CARSPATH/install/sys/lib/prtab
- $CARSPATH/install/sys/lib/proptions

See Also

lpr(1) lpc(1) mkspooler(1) rmspooler(1)

qp

Purpose

The qp command queries the process list.

Synopsis

qp

Description

The qp command is a menu driven command that allows a user to query the process list for information. These queries can be by command, parent process id number, process id number, username, etc. In addition, qp will display both the lineage & ancestry of a process. This can be very useful for determining where a process is coming from or what are the child processes of one in question. Also, qp can send signals to a process or list of processes that have been queried.

Troubleshooting

The command's Exit status equals:

- 0 if no errors occurred
- 1 if errors occurred

See Also

ps(1)

rmspooler

Purpose

The rmspooler command deletes existing spooling devices.

Synopsis

rmspooler spooler_device

Description

The rmspooler command simplifies the procedure for deleting existing spooling devices.

The rmspooler command requires only the spooler_device name as the command line argument. To ensure that so all the sub-processes execute properly, you must have a *carsroot* or *super-user (su*) login when executing the command.

Example: The following is a sample command line for deleting the **spooler_device** bop (business office printer).

```
# rmspooler bop
```

Files

- ${CARSPATH}/install/utl/rmspooler (delete spooling devices from CX)
- ${CARSPATH}/install/lib/prtab (file of active printers /etc/ttys terminal initialization data)
- ${CARSPATH}/install/utl/queue_name (spooling program for queue_name)
- ${CARSPATH}/spool/queue_name (spooling directory in for spooler)

See Also

- mkspooler(CARSC)
- ${CARSPATH}/install/utl/mkspooler (create spooling devices on CX)

senduucp

Purpose

The senduucp command sends files to remote hosts.

Synopsis

senduucp [-l] [-n] destination_host filename...
senduucp [-l] [-n] destination_host... -f filename...

Description

The senduucp script compresses filename(s) using compress(1), unless you specify the -n (no compression) flag, and transmits them to the destination_host via uucp(1C). If you use the -f (filenames follow) flag, the system assumes that previous arguments are hosts to which all the specified files will be sent. When you do not specify the -f flag, a single destination_host is implied. The -l flag indicates that the transmission of these files are low priority. By default, the transmission is attempted immediately. If the files to be transmitted are of "low" priority (e.g.,. could be sent at night), the at(1) command is utilized to send the files at a later time (currently, this is 11:10 p.m.). If you use this flag, it must be the first argument to senduucp.

To improve efficiency, each file is compressed only once and the compressed version is then sent to each named host rather than compressing each file for each host separately.

Upon completion of the transmission(s), the system sends mail to the user.

See Also

uucp(1C), compress(1), at(1), `UUCP' Implementation Guide

Troubleshooting

- When using uucp(1C), if the first attempt to transmit is unsuccessful, the files will remain in the "queue" until a uupoll(1C) or a uucico(1C) command is executed.
- The command requires a *recvuucp* compliment for use on the remote sites.

setdb

Purpose

The setdb command sets the current database to the one you specify.

Synopsis

setdb { dbname }

Description

The setdb command provides the shell user with the ability to change effective databases easily. This facilitates converting to a new release when on the client's site and allows multiple databases at Jenzabar.

The setdb command works in conjunction with the contents of ~/.cshrc file used by CSH. It changes the CARSV value based upon the database name supplied and the information in the dbtab and reltab files. The system then uses the CARSV value to define all necessary pathnames within the ~/.cshrc file for environment variables. The system also changes other environment variables using information in ~/../skel/dbtab and ~/../skel/reltab. These variables include INFORMIXDIR and TBCONFIG for identifying the database engine, as well as others.

In addition, setdb changes to the appropriate directory if the current directory has a counterpart in the target release. After this, the system starts a new CSH. Once you execute a setdb, you can move between environments using shell built-in commands. For example, you can suspend a shell with the *suspend* command to return to the previous environment, or you can restart a child shell with the *fg* command.

To terminate the setdb command (the new CSH), type a control-D or use the shell built-in command exit.

Files

- ~/.cshrc (CSH initialization file)
- ~/../skel/dbtab (Database information table)
- ~/../skel/reltab (Release information table)

See Also

csh(1)

setup_web_dbm

Purpose

The setup_web_dbm command regenerates either the student or faculty authentication files.

Synopsis

setup_web_dbm user-id

Description

The setup_web_dbm command generates the DBM file used by Apache for faculty web server authentication. To activate a faculty member for web server access, a pin # must exist in the profile_rec.password column, and an entry must exist in the userid table. Before entering the setup_web_dbm command, you must enter the password for the faculty member.

Example: % **SU csh**

% Password for user: password

 # setup_web_dbm user-id

 # exit

See Also

dbmmanage

slave

Purpose

The slave command filters output to slave printer.

Synopsis

slave [-t term] [-p printer] [-c commands] ... [file] ...

Description

The slave command is a filter (operates on specified files) that passes the necessary character sequences to the current CRT to output to an attached slave printer.

The slave command accepts an optional -t argument to override the default terminal type. In addition, specifying -p allows you to specify the printer type, and specifying the -c option processes special command words as they appear in the slavecap file.

Files

${CARSPATH}/install/sys/etc/slavecap slave capability database

SU

Purpose

The CX SU (superuser) command is a simple and relatively safe way for you to become a superuser and execute various commands as needed. You can simply type SU at the shell prompt followed by the command you wish to execute. You will then be required to enter your password.

Example: % **SU <command>**

% **Password for <user>**

Note: This uppercase SU command is different from and an enhancement to the UNIX lowercase su command.

When the command is complete and has finished executing you will automatically be returned to your original login.

When you invoke the SU command and type your password, the system will perform a lookup in the **.gurus** file. If your login name is not listed in the .gurus file, you will not be granted SU permissions. Typically, employees like the Jenzabar Coordinator, and other computer center personnel are included as determined by the Jenzabar Coordinator. Make the decision regarding which users should have their login name in the .gurus file carefully.

GURUlog File

An entry is logged in a log file, GURUlog, for each use of SU. This log file is located in /var/adm. It gives the time, date, user and the command for each instance of using SU. If a user who does not have an entry in the .gurus file attempts to use the SU command an entry indicating that his attempt was denied will appear in the log file. The /var/adm/GURUlog file must exist in order to support the SU command. If it is absent, the SU command will function but entries will not be logged. The file should have a mode of 644 and ownership of root to prevent users from modifying the file. Below is an example of entries in the GURUlog file:

```
2001/08/15 17:18 jsmith attempted SU su
2001/08/16 08:39 sbrown attempted SU csh
2001/08/16 10:08 sbrown attempted SU uidlist
2001/08/17 13:51 sbrown failed SU uidlist
2001/08/17 13:55 jlyons attempted SU kill -9 4157
```

The example indicates that jsmith used the SU command to become root and was successful. Sbrown attempted to use the SU command and failed. Jlyons stopped a process in progress.

up2low

Purpose

The up2low command converts characters from upper to upper/lower case.

Synopsis

up2low -r informix_file_name -f informix_field_name1 informix_field_name2 ...

Description

The up2low command converts each word of the field(s) you specify by making the first character as upper-cased and the rest of the word as lower-case. The command has the following defined to separate words: space, comma, period, dash, quote, backquote, slash, and colon. Words defined to not be made lower-cased include: HS, II, III, IV, SIP, and SW.

CAUTION: You cannot reverse changes programmatically.

Troubleshooting

- The system sends error messages for unrecognized files or fields. If the system does not locate all specified fields in the file, the conversion does not occur.
- The command's Exit status equals:
    - 0 if no errors occurred
    - 1 if errors occurred

updstats

Purpose

The updstats command updates statistics for the database.

Synopsis

updstats [-d database] [-f file] [-t table]

Description

The updstats command runs update statistics statements for Informix Dynamic Servers, version 7 and higher. If you do not specify any parameters, updstats updates the statistics for all tables, except synonym tables, in all databases except the Informix System databases (sysmaster and sysutils).

Options

-d

Identifies the database for which statistics will be updated.

-f

Specifies a file to output the commands to instead of having the system execute them. You can then peruse the file to determine what was done.

-t

Identifies a table for which statistics will be updated. If you specify a table, you must also specify a database and a file.

vt

Purpose

The vt command sets up a virtual terminal to another system.

Synopsis

vt [-s speed] [-d device] [-p parity] [-hvzn] [-t type] [-l dialog]

Description

The vt command sets up a connection to another +1 system, a terminal, or possibly a non-+1 system. It manages an interactive conversation with possible transfers of files.

Options

-s

Speed gives the transmission speed (110, 150, 300, 1200, 4800, 9600); 1200 is the default value. Most of our modems restrict us to choose between 300 and 1200. Directly connected lines may be set to other speeds.

-d

Value may be used to specify the device name for the communications line device. The default device is "/dev/outgoing".

-h

The line should not be hung-up when vt exits.

-p

Designates the parity to be generated for data sent to the remote. Valid parities are even, odd, mark, and space.

-t

Specifies the remote machine type. This is only needed to do file transfers. Possible machine types are UNIX (VAX & others), mibs (GA), and uniflex (SWTPc). The default is UNIX.

-l

Specifies a login dialog. This dialog can be used as commands to an autodial modem and/or perform the login sequence on the remote machine. The specification of the dialog is similar to the login sequence in uucp.

-n

Used with the dialog option and specifies that the process is be run as a background task (no terminal is associate with vt). If the dialog is unable to complete, vt will terminate.

-v

Enables verbose messages about the status of vt.

-z

Enables debugging output.

Processing Notes

After making the connection, vt runs as two processes. The transmit process reads data from the standard input and, except for lines beginning with a `~,' passes that data on to the remote system. The receive process accepts data from the remote system and, except for lines beginning with a **`~,'** passes that data on to the standard output. Normally, an automatic XON/XOFF protocol is used to control input from the remote so the buffer is not overrun. This is handled by the communications hardware (modem, controllers, etc). Lines beginning with ~ have special meanings.

Transmit
The transmit process interprets the following:

~
Terminates the conversation.

~!
Escapes to an interactive shell on the local system.

~! cmd\.\.\.
Executes the shell with cmd line on the local system.

~$ cmd\.\.\.
Executes the shell with cmd line locally and send its output to the remote system.

~% cd dir
Changes the current working directory on the local system.

~% take from [to]
Copies an ASCII file from the remote system to the local system. If to is omitted, the from argument is used in both places.

~% put from [to]
Copies an ASCII file from the local system to the remote system. If to is omitted, the from argument is used in both places.

~% recv from [to]
Transfers a file (using ftx) from the remote system to the local system. If to is omitted, the from argument is used in both places.

~% send from [to]
Transfers a file (using ftx) from the local system to the remote system. If to is omitted, the from argument is used in both places.

~#
Sends a break to the remote system.

~~ .\.\.
Sends the line ~ ".\.\." to the remote system.

Receive
The receive process normally copies data from the remote system to its standard output. A line from the remote that begins with ~. terminates the session. A line from the remote that begins with ~>initiates an output diversion to a file. The complete sequence is the following, with zero or more lines to be written to file.:

Example: ~>[>]:file

The system diverts (or appends, if **>>** is used) data from the remote to file. The trailing ~> terminates the diversion.

The ~%take and ~%put commands use standard UNIX commands to do the file transfer and may have no flow control and should only be used for ASCII character files. The use of ~%put requires stty (1) and cat (1) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

The use of ~%take requires the existence of echo (1) and cat (1) on the remote system. Also, stty tabs mode should be set on the remote system if tabs are to be copied without expansion.

In use of ~%recv and ~%send requires that both systems have the ftx(CARSC) program.

Files

- /usr/lib/dialog
- /usr/spool/uucp/+1..(tty-device)

See Also

cat(1), echo(1), stty(1), uucp(1C), tty(4), ftx(1).

Troubleshooting

The command's Exit status equals:

- 0 if no errors occurred
- 1 if errors occurred

An artificial slowing of transmission by vt occurs during the *~%put* operation to reduce the possibility of data loss.

INDEX

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z