Jenzabar CX

# Communications Management

JENZABAR

Technical Manual

Filename:  tmcommgt
Distribution date:  10/09/2000

Contact us at www.jenzabar.com

JENZABAR, INC.
COMMUNICATIONS MANAGEMENT TECHNICAL MANUAL

TABLE OF CONTENTS

# SECTION 1 - USING THIS MANUAL

## Overview

### Purpose of This Manual

This manual provides technical information required to install, customize, and maintain the CX Communications Management product.

### Intended Audience

This manual is for use by those individuals responsible for Jenzabar CX installation, customization, and maintenance.

### How to Use This Manual

If you are not familiar with the processes and features of the Communications Management product, read the manual for:

- Detailed reference information about how the product works
- Procedures for customizing and maintaining the product

If you are familiar with the processes and features of the Communications Management product, and just need specific reference information or a procedure, look through the table of contents or index and refer to the pages you need.

### Product Differences

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

### Structure of This Manual

This manual contains both general reference information and procedures for installing and maintaining the Communications Management product.  The organization of the manual is as follows:

**Overview information**
Section 1 - Information about using this manual
Section 2 - Overview information about the product

**Product reference information**
Section 3 - Tables used in the product
Section 4 - Macros and Includes
Section 5 - CX program files
Section 6 - 15 - Programs used in the product
Section 16 - Menus, screens, scripts, and reports

**Product procedures**
Section 17 - 20 - Procedures to install and customize your processes

**Error reference/Recovery procedures**
Section 21 - A reference of fatal and serious errors, recovery procedures, and debugging information for program and report errors
Section 22 - Recovery and debugging procedures for letter writing problems

**Reference information**

Index

## Related Documents and Help

The following resources are also available to assist you in installing, customizing, maintaining, and using the Communications Management product.

**QuickMate online help**
*Using QuickMate*
*Getting Started User Guide*

**UNIX-based help**
Help command (**<Ctrl-w>**) in screens and menus

**User guides**
*Using Communications Management*
*Getting Started User Guide*

## 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

Jenzabar CX 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 list**
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 designate a program name (e.g., *identry*) within paragraphs
- 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.

**Percent symbol**
Indicates the standard UNIX prompt. Your prompt may vary.

**Pound symbol**
Indicates the standard UNIX superuser prompt indicating the user has additional permissions. Your prompt may vary.

**Quotation marks**
Represent information written in this manual exactly as it appears on the screen

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

## Jenzabar-Specific Terms

Some terms used in this manual may be unfamiliar to you, either because they are terms you have not used before or because Jenzabar has assigned a slightly different meaning to a familiar term.  The following list identifies and explains the most common Jenzabar-specific terms:

**Application**
One or more software programs that enable you to perform a particular procedure, such as producing letters.

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

**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).

**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 **<Enter>**
- Clicking on the icon or button with the mouse

**System**
The Jenzabar product, CX.

## 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>**).

# SECTION 2 - COMMUNICATIONS MANAGEMENT PROCESSES

## Overview

### Introduction

This section provides information on the purpose and process flow of Communications Management.

### Purpose of Communications Management

The primary purpose of CAR's Communications Management product is to enable an institution to organize, create, and track communication with organizations and individuals. This communication can include both outgoing and incoming correspondence in a variety of media (e.g., telephone calls, faxes, letters, applications, and personal visits). Communications Management records each such communication in the CX database through the use of Contact records.

Every day, the CX letter production process can produce each letter or document scheduled for that day. It retrieves data for each prospect from the database and merges it with the appropriate form letter, joining names and selecting alternate addresses and salutations, as required. It also inserts data and text as appropriate, formats the resulting personalized text and routes the completed letter to your printer's spooler. Users can then print any letters or documents in the printer's spooler at their convenience.

The standard CX Communications Management product includes some common reports (e.g., the Tickler Structure report that helps users understand and correct their Tickler strategies) and some standard ACE reports that collect data used in letters. In addition, you can use the INFORMIX query and report language tools to create other customized reports as desired.

### Purpose of Tickler Systems

A Tickler system is a means causing communications to occur at the desired time. The CX Tickler can carry out the institution's communications strategy by automatically generating Contact records. A set of Tickler tables defines the parameters needed to generate the Contact records.

A Tickler system can be created for any administrative area that requires a means of tracking and/or controlling contacts with individuals. Institutions use Tickler systems most extensively in the Admissions, Development, and Financial Aid offices.

Users in the Development Office, for example, could use a Tickler system for reaching constituents in a current capital campaign. In this tickler, the office could have a different strategy for cultivating each audience, such as trustees, alumni and friends. The office could have a second tickler for the annual fund. Any individual can be included in multiple ticklers. For example, an alumnus could be tracked by both the capital campaign and the annual fund ticklers.

Strategies are highly flexible. The simplest is a fixed sequence of documents. Strategies typically depend on communications received (or not received) from correspondees. For example, a recruiting strategy would send an application acknowledgment to a prospective student only if an application were received. A campaign strategy would suspend further appeals and send an acknowledgment if the donor submits a pledge. For more flexibility, strategies can be further subdivided. Tactics may be optional or required, have beginning and completion dates, and be processed concurrently with other tactics. The strategy can control the minimum and maximum time between outgoing contacts to each addressee, and can eliminate unintentional duplicates.

Each office has complete control over its ticklers, and can manually cancel scheduled documents and add new ones, modifying or terminating any strategy.

## Background Knowledge

The following list describes the necessary background information that you should know to implement and support the Communications Management product.

**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:

- ISQL database
- PERFORM screens
- ACE reports

**Jenzabar CX database tools and utilities**

Know how to use the following database tools:

- MAKE processor
- Schemas
- Macros
- Includes
- Program screens
- SMO process
- *libentry* programs

**Jenzabar CX**

Know the following about the Jenzabar CX standard product:

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

**QuickMate features**

Know the following about the CX Graphical Server:

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

**C Programming**

If you want to modify any CX programs to meet unique needs at your institution, you must know how to use the C programming language and have an in-depth knowledge of the CX code.

**Communications Management policies and procedures**

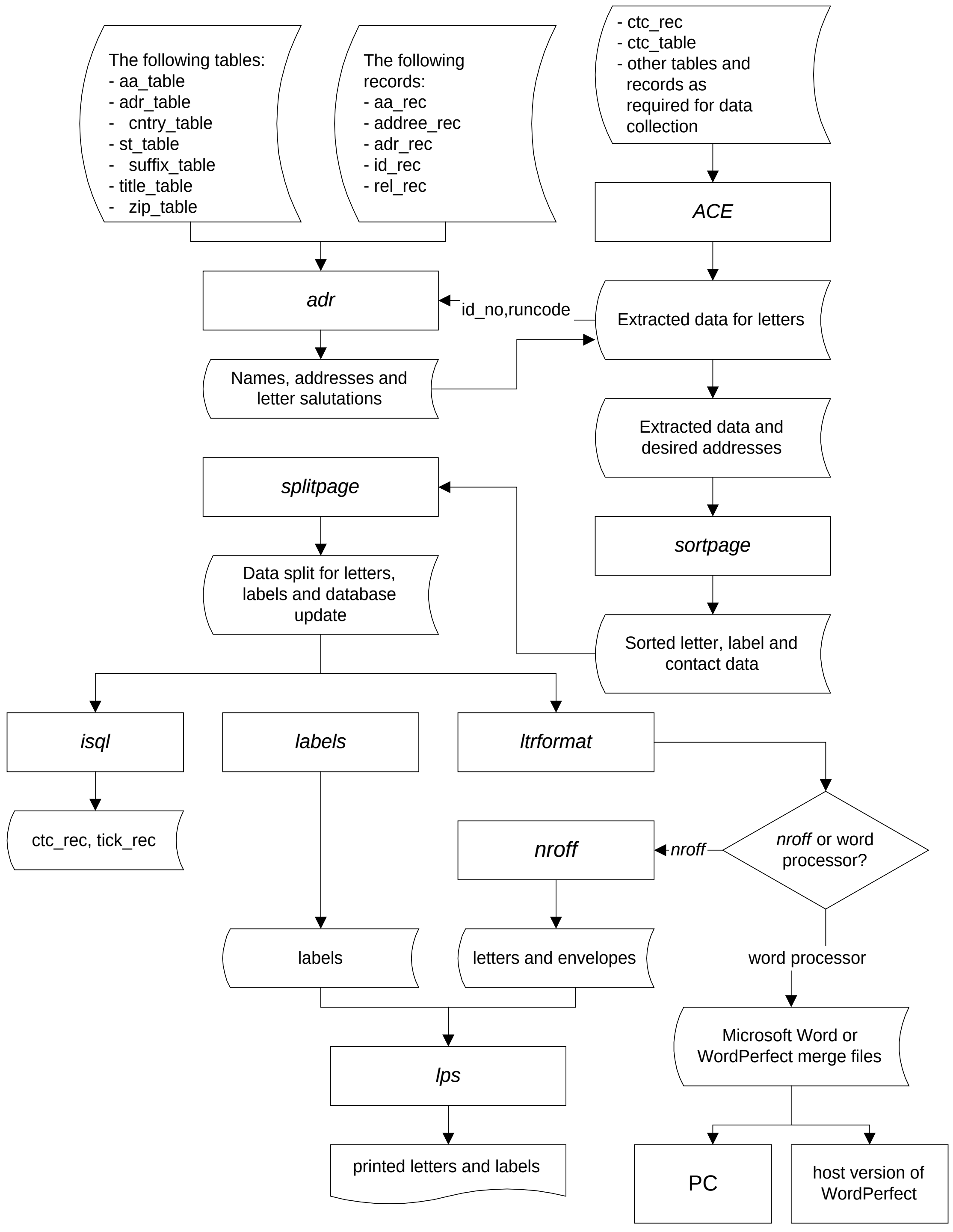
Know answers to the following questions:

- What is the institution’s plan for corresponding with leads, applicants, students, employees, alumni, and others?
- Who composes letters?
- Who prints letters?

## Process Flow

### Diagram

The following diagram shows the process flow of the Communications Management product. The overall process is normally controlled by the CX script *ltbrun*.



## Contact Record Process Description

The first process in Communications Management is to review Contact records, identifying those contacts scheduled for completion. This overall process follows:

1. Reviews each individual who has a Contact record.
2. Scans the Contact records for a status code of (E)xpected and the specified Due Date (i.e., a date that is earlier than or the same as the current date).
3. Identifies the contact to be made (the letter code, which is the same as the contact resource).
4. Groups that individual with others receiving the same contact(s).

After the contacts are grouped and sorted, Communications Management performs the following two general steps. For an expanded description of these general steps, see *Overall Process Description* in this section.

1. Merges the information from the database with the form letter with the same name as the contact, or creates a word processing merge file.
2. Produces an output file for printing as scheduled by the institution.

## Overall Process Description

The following list describes the processes involved with the Communications Management product.

1. The *ACE* process extracts information for letters, and passes ID numbers to *adr*.
2. The *adr* program extracts information for the selected ID numbers, and outputs names, addresses and salutations.
3. The *sortpage* program evaluates the data, sorting it by the fields sent to it by the *ACE* report.
4. The *splitpage* program divides the data, sorting it into the following types of information so it is usable input for subsequent processes:
   - Letter contents
   - Label contents
   - Database update
5. The *ltrformat* program formats the letters and envelopes.

   **Note:** The operation of the *ltrformat* program depends on the type of merge file requested by the parameter STDLPS. A value of *stdlps* yields an *nroff* file, while values of *wordp* or *rtf* yield word processing merge files for WordPerfect or Microsoft Word, respectively.

   If an *nroff* merge file is used, then *ltrformat* bundles the merge records (2000 at a time) and passes them to *nroff*. The output from *nroff* is then placed in the *lps* spooler directory. If either *wordp* or *rtf* is used, then *ltrformat* itself creates the correctly formatted merge file and places it in the Merge directory of the proper WP (word processing) drawer.
6. The *labels* program takes the row label information from *splitpage* and produces formatted labels which are then stored in the *lps* spooler.
7. The *isql* process updates Contact records and Tickler records, changing statuses and dates.
8. The *lps* program prints the letters, envelopes and labels that were sent to the *lps* spooler.

## WPVI Drawer Types

### Introduction

CX supports a variety of letter types, and uses the WPVI *FileCabinet* system to organize them. Drawers (which actually are UNIX subdirectories that can be considered as separate compartments in file cabinets) exist for each functional area of CX (e.g., admissions, alumni, and registrar). Under each of these drawers, users can maintain different types of letters in subdrawers. CX processes the letters in different ways, depending on the subdrawer in which they exist. This section provides descriptions of the following standard drawers and the types of letters contained in each.

- common
- letters
- wpreports

**Note:** For more information about producing all the supported letter types, see the CX guide, *Using Communications Management.*

### Contents of the *common* Drawer

The *common* drawer for each functional area primarily contains single letters (i.e., letters that you do not send frequently, or to a large number of recipients). The command line of the *common* drawer provides the *nroff* command that expands *nroff* codes in the letters, a feature used extensively in single letters. The following letters are the type that you store in the *common* drawer.

**Single Letter, One Time Use Only**
This letter type is a memo or a personal letter that you do not intend to use again. You create it in the same form in which it will be sent, without using special formatting commands or database records to supply the name and address. CX does not maintain a computer record of this type of correspondence. You can save the letter in the *common* drawer, or delete it after printing.

**Single Letter, May be Used Again**
This letter type meets a specific need that may recur; however, you expect it to be infrequent, and do not require multiple copies at a given time. This letter is similar to the one-time letter, but the user saves it so that its name and address may be changed for the next person to whom you address it. CX does not maintain a computer record of this correspondence.

**Letters Merged with a User-Created Data File**
This type of letter is a form letter with an associated merge file for a particular mailing. The merge file may consist of one of the following:

- A list typed into a merge file by a user.
- A list obtained from some other computer (typically, a purchased list (e.g., ACT Student Search names). Normally, you send this type of letter to a group that does not yet exist in the database (e.g., prospects). You must format this letter using *nroff's* dot (.) commands for insertion of the name, address, date, and salutation text in the proper places, and must therefore store it in the letters drawer where the *nroff* command line option is available.

### Contents of the *letters* Drawer

The *letters* drawer contains the letters that you use most frequently. The letters in this drawer work with Contact records (ctc_rec) to track the recipients, and can contain variable information extracted from the CX database by ACE reports. The following letters are the types that you store in the *letters* drawer.

**Single Letters, Used Frequently**

This letter type fills a routine need, and you use it frequently in day-to-day activities. It has the same text, regardless of the addressee. Because you use it frequently and routinely, it must be easy and accurate to process.

If you address the letter to a person who is not in the CX database, you can format the letter interactively. If you format the letter interactively, you can store it in any drawer in the WPVI structure. If, however, you intend to schedule the letter with a Contact record, it must be in the *letters* drawer within WPVI.

If you address the letter to a person who is in the CX database, you can produce it most efficiently by adding a Contact record.

**Letters or Labels for a Frequently Used Mailing List**

The following two types of letters fall under this category:

- Previously defined mailing list

  A Subscription record identifies a specific group of constituents in the database as part of a particular mailing list. The program locates individuals with the proper subscription code, and prepares the form letters for them.
- Ad hoc defined mailing list

  A user-created ACE report selects an ad hoc list of names and addresses according to the *where* clause in the report, and prepares a file of the individuals it selects. Alternatively, you can manually create a data file of names and addresses. The program merges the information into the form letter.

**One Letter, Using INFORMIX Database**

This type of letter uses database information to fill the name and address requirements of the form letter, and records that the communication has taken place by entering a record in the Contact record for each person receiving the letter. The letter is initiated by entering a scheduled or (E)xpected Contact record (status E) for the individual, or by adding a group of Contact records for an ad hoc defined group of individuals.

Within each administrative module, an ACE report exists that locates the contacts that are due, and selects the appropriate data for the individuals for whom the letter is intended.

**Letters with Text or Value Insertion**

Some of the supported letters may require that a user enter text (single words, groups of words, or entire paragraphs) or values (fields from the database) into the body of the letter to provide additional personalization.

This type of letter requires the use of an ACE report to extract the needed data. You must set up the letter using the text or field macros that correspond to the data extracted by the ACE report.

Most of the commonly used fields have been defined in the CX database. The data in these fields may be retrieved by an ACE report. These reports meet the needs of specific administrative areas and are available as menu options in those areas.

This type of letter usually requires the maintenance of Contact records for the communication. The ACE report looks for Contact records with a status of (E)xpected and an appropriate Due Date, selects the individuals' data, and then prepares the data for insertion into the form letter.

These letters provide the means to prepare highly personal communications, and require minimal maintenance after they are set up. For *nroff* style merge letters, the letter will exist in the letters drawer of a WPVI file cabinet, a location easily accessible for every user.

Tickler systems can use these letters, fitting into a prescribed sequence of scheduled or response-dependent communications strategies.

## Application Relationships

### Related Jenzabar Applications

The Communications Management product interacts with several other applications and products in CX.  The following list describes the interrelationships.

**Admissions**
Users of the Admissions application use Communications Management to track and prepare correspondence with inquiries and applicants.

**Alumni Association**
Users of the Alumni Association application use the Tickler process within Communications Management to schedule contacts for recent graduates to encourage them to remit alumni association membership dues.

**Development**
Users of the Development application use Communications Management to produce pledge reminders and acknowledgment letters.

**Financial Aid**
Users of the Financial Aid application use Communications Management for document tracking.  Specifically, Communications Management can automatically schedule FASTAT (Financial Aid application status) letters at specified intervals of time until all supporting documents have arrived.

**Human Resources**
The Personnel department uses Communications Management to contact employees and former employees about training opportunities and expiration of benefits, and to track interviews and other personnel events.

**Registration**
Users of the Registration application use Communications Management to track and prepare correspondence with students, including Dean’s List letters and probation notifications.

**Student Billing**
Users of the Student Billing application use Communications Management to send dunning letters to students as required.

# SECTION 3 - COMMUNICATIONS MANAGEMENT TABLES AND RECORDS

## Overview

### Introduction

This section provides you with reference information about each table and record associated with the Communications Management product.  The following tables appear in this section:

- Alternate Address table (aa_table)
- Address table (adr_table)
- Communications table (comm_table)
- Contact table (ctc_table)
- State table (st_table)
- Step table (step_table)
- Suffix table (suffix_table)
- Tickler table (tick_table)
- Title table (title_table)

The following records appear in this section:

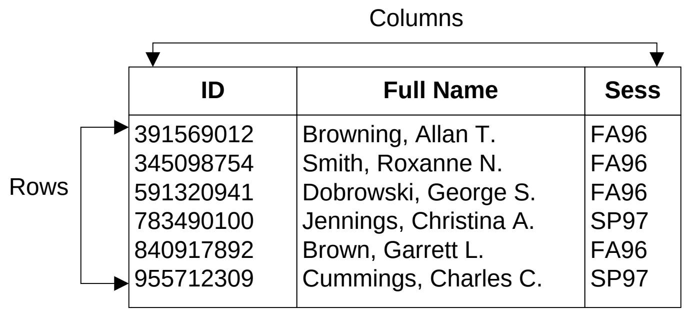
- Alternate Address record (aa_rec)
- Addressee record (addree_rec)
- Address record (adr_rec)
- Contact blob (ctc_blob)
- Contact record (ctc_rec)
- Contact Detail record (ctcdetl_rec)
- Relationship record (relation_rec)
- Step record (step_rec)
- Step Contact record (stepctc_rec)
- Step Object record (stepobj_rec)
- Step Requirement record (stepreq_rec)

### Alphabetical Organization

The tables and records appear in alphabetical order in this section.

### What Is an SQL Table?

In a relational SQL database, a table is an organized set of any kind of data, regardless of its purpose for validation or information maintenance.  The basic unit of organization of a table is a column, that is, a category of data.  A table can have multiple columns and multiple rows of data.



| ID | Full Name | Sess |
|---|---|---|
| 391569012 | Browning, Allan T. | FA96 |
| 345098754 | Smith, Roxanne N. | FA96 |
| 591320941 | Dobrowski, George S. | FA96 |
| 783490100 | Jennings, Christina A. | SP97 |
| 840917892 | Brown, Garrett L. | FA96 |
| 955712309 | Cummings, Charles C. | SP97 |

## What Is a Jenzabar CX Table?

CX makes name distinctions in the use of database tables. A *table* in CX contains information that remains static and is denoted with the *_table* extension. For example, the State table, named *st_table,* contains the list of the states in the United States of America. On the CX menu, you can access most tables in Table Maintenance menus. In Communications Management, however, you can add and update Tickler table entries through interrelated program screens.

## What Is a Jenzabar CX Record?

CX makes name distinctions in the use of database records. A *record* in CX is a table containing information that changes on a regular basis and is denoted with the *_rec* extension. For example, the Alternate Address record, named *aa_rec*, contains any other addresses where students can be contacted, such as a summer address. You access records in CX program screens, scroll screens, and PERFORM screens.

## Common Tables

Communications Management uses several common tables throughout CX. The tables are listed below.

**Note:** Documentation for the following common tables that are used in the Communications Management product appears in this section.

- Alternate Address table (aa_table)
- Address table (adr_table)
- Contact table (ctc_table)
- County table (cty_table)
- Country table (ctry_table)
- Denomination table (denom_table)
- Ethnic table (ethnic_table)
- Facility table (facil_table)
- Handicap table (hand_table)
- Hold table (hold_table)
- Occupation table (occ_table)
- Office table (ofc_table)
- Relationship table (rel_table)
- State table (st_table)
- Tickler table (tick_table)
- Title table (title_table)
- User ID table (userid_table)
- Zip table (zip_table)

## Shared Records

Modules in the Communications Management product use several records that are shared throughout CX. The following lists these records and the modules that use them.

**Note:** If you make changes to schemas for the following records, you must reinstall each associated product or module.

**Address record (adr_rec)**
All entry programs

**ID record (id_rec)**
All products

**Relationship record (rel_rec)**

All entry programs

**Required Tables and Records**

The following records are required to run the features of the Communications Management product. Each letter recipient must have the following records:

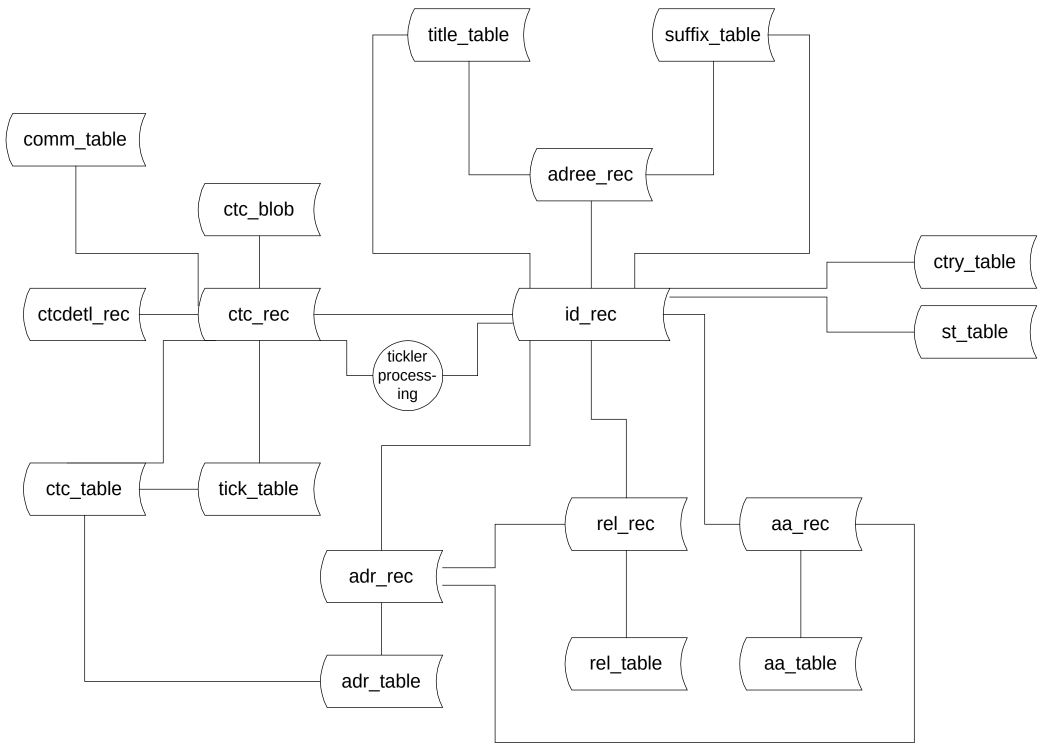
- ID record (id_rec)
- Address record (adr_rec)
- Contact record (ctc_rec)

In addition, if the institution uses Tickler to schedule contacts, recipients must also have Tickler records (tick_rec).

## Table and Record Relationships

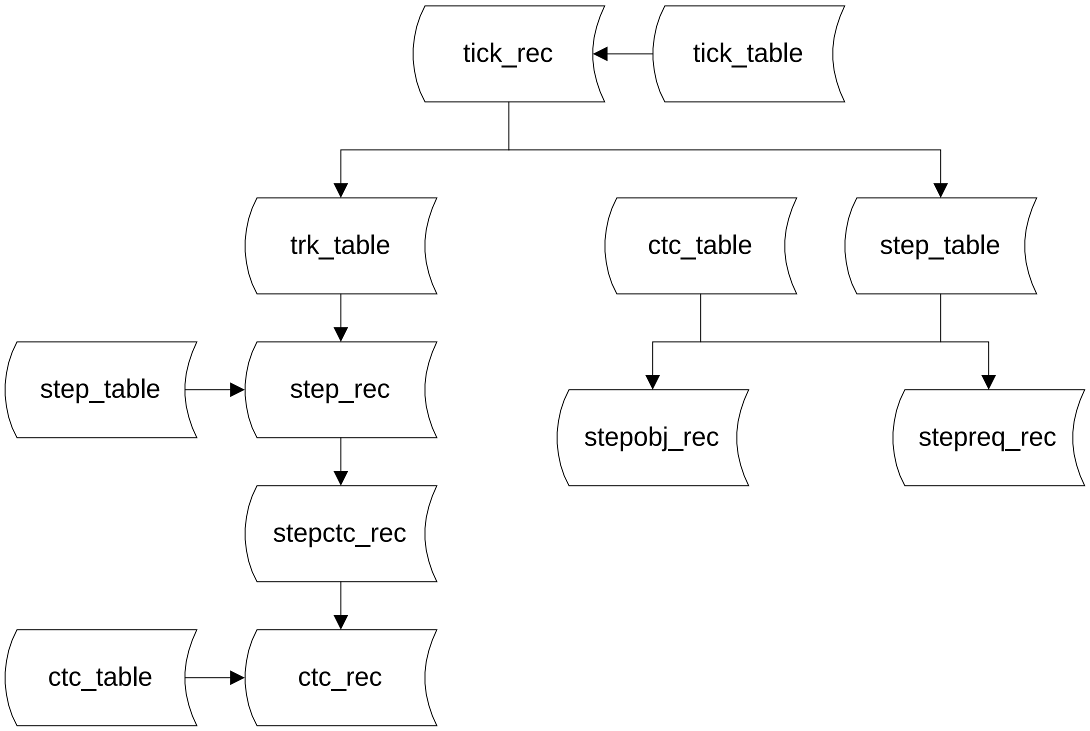
### Entity Relationship Diagram for Communications Management

The following diagram shows the relationship between the tables and records used by the Communications Management product:



### Entity Relationship Diagram for Tickler

The following diagram shows the relationship between the tables and records used by the Tickler component of the Communications Management product.


tick_rec
tick_table
trk_table
ctc_table
step_table
step_table
step_rec
stepobj_rec
stepreq_rec
stepctc_rec
ctc_table
ctc_rec

## Communications Management Schemas

### Introduction

Schema files define the structure of database files and associated fields in the CX data dictionary.  You can access schema files associated with the Communications Management product in the following directory path:  $CARSPATH/schema/common

### File Naming Conventions

CX makes name distinctions in the naming of schemas.  For schema files containing definitions of CX tables, the UNIX file name begins with the letter *t* followed by characters describing the English name of the table (e.g., *tst* for the State table).  For schema files containing definitions of CX records, the UNIX file name describes the English name of the record (e.g., *id* for the ID record).

The first line in a schema file, after revision information, specifies the INFORMIX database table that the schema defines.  For example, *st_table* (State table) is specified in the *tst* schema file.

### Field Descriptions

Schema files contain descriptions of each field defined in a table or record.  You can view descriptions of fields in Communications Management tables and records by accessing the schema files.  You can also obtain information about fields that you complete during implementation in *Customizing the Communications Management Processes* in this manual.

# Communications Management Tables and Records

## Introduction

The following list identifies the tables and records that relate to the Communications Management product, including Tickler. The list includes the filenames, purpose, location, and association of each table and record with programs and other tables and records.

**Note:** The *Program interrelationships* in the list are included in the Communications Management product. The *Product interrelationships* in the list are not included in the Communications Management product.

**Address record**

Defines the type of addressing to perform for the runcode, alternate address, and individual/organization specified.

*UNIX filename:* adr

*Informix filename:* adr_rec

*Schema location:* $CARSPATH/schema/common

*Program interrelationships: adr*

*Product interrelationships:* none

*Table/record interrelationships:* adr_table, id_rec

**Address table**

Defines the valid runcodes.

*UNIX filename:* tadr

*Informix filename:* adr_table

*Schema location:* $CARSPATH/schema/common

*Program interrelationships: adr*

*Product interrelationships:* none

*Table/record interrelationships:* adr_rec

**Addressee record**

Defines the preferred salutation and/or name line for an individual for a particular case.

*UNIX filename:* addree

*Informix filename:* addree_record

*Schema location:* $CARSPATH/schema/common

*Program interrelationships: adr*

*Product interrelationships:* All entry programs

*Table/record interrelationships:* id_rec, suffix_table, title_table

**Alternate Address record**

Maintains alternate address information for an individual or organization.

*UNIX filename:* aa

*Informix filename:* aa_rec

*Schema location:* $CARSPATH/schema/common

*Program interrelationships: adr*

*Product interrelationships:* All entry programs

*Table/record interrelationships:* aa_table, adr_rec, id_rec

**Alternate Address table**

Defines valid alternate address codes (e.g., SUMR, PERM) and sets priorities for using alternate addresses.

*UNIX filename:* taa

*Informix filename:* aa_table

*Schema location:* $CARSPATH/schema/common

*Program interrelationships: adr*

*Product interrelationships:* All entry programs

*Table/record interrelationships:* aa_rec

**Communications table**

Defines the valid types of communication in which an institution engages (e.g., BROC (brochure), PHON (phone call), LETT (letter and envelope), LABL (label only), or LTLB (letter, envelope, and label)).

*UNIX filename:* tcomm

*Informix filename:* comm_table

*Schema location:* $CARSPATH/schema/common

*Program interrelationships:* letter writing ACE reports

*Product interrelationships:* none

*Table/record interrelationships:* ctc_rec

**Contact blob**

Contains the text field associated with a Contact record.

*UNIX filename:* bctc

*Informix filename:* ctc_blob

*Schema location:* $CARSPATH/schema/common

*Program interrelationships:* none

*Product interrelationships:* Entry programs (where available from detail windows)

*Table/record interrelationships:* ctc_rec

**Contact Detail record**

Provides the handling requirements for letter contacts.

*UNIX filename:* ctcdetl

*Informix filename:* ctcdetl_rec

*Schema location:* $CARSPATH/schema/common

*Program interrelationships:* letter writing

*Product interrelationships:* *daentry*

*Table/record interrelationships:* ctc_rec

**Contact record**

Records the sending or receipt of correspondence with another individual or organization.

*UNIX filename:* ctc

*Informix filename:* ctc_rec

*Schema location:* $CARSPATH/schema/common

*Program interrelationships:* *adr, ctcbatch, tickler*

*Product interrelationships:* *admstat*, *faentry*, *grdrpt*, *regist*, *stmt*, *trans*

*Table/record interrelationships:* ctc_blob, ctc_table, id_rec

**Contact table**

Defines the valid types of communications that the institution uses to interact with another individual or organization (e.g., ACPTLTR (an acceptance letter) or ADVISOR (an advisor's letter)).

*UNIX filename:* tctc

*Informix filename:* ctc_table

*Schema location:* $CARSPATH/schema/common

*Program interrelationships:* *adr, ldtickler, tickent, tickler, most letter ACE reports*

*Product interrelationships:* *admstat*, *faentry*, *grdrpt*, *regist*, *stmt*, *trans*

*Table/record interrelationships:* comm_table, ctc_rec, enrstat_table, tick_table

**ID Contact record**

Serves as a temporary table before creating actual Contact records within *ctcbatch*. Its purpose is to provide backup records. CX uses the backup records if an error occurs that causes the Contact records to not be created. If this type of error occurs, the user can use the menu option Rerun Contact Batch Entry to retrieve the ID Contact records and generate the desired Contact records.

*UNIX filename:* idctc

*Informix filename:* idctc_rec

*Schema location:* $CARSPATH/schema/common

*Program interrelationships:* *ctcbatch*

*Product interrelationships:* none

*Table/record interrelationships:* ctc_rec

**Relationship record**

Identifies two individuals and the relationship between them.

*UNIX filename:* rel

*Informix filename:* rel_rec

*Schema location:* $CARSPATH/schema/common

*Program interrelationships: adr*

*Product interrelationships:* All entry programs

*Table/record interrelationships:* adr_rec, id_rec, relation_table

**State table**

Defines the states and the codes associated with them.

*UNIX filename:* tst

*Informix filename:* st_table

*Schema location:* $CARSPATH/schema/common

*Program interrelationships: adr*

*Product interrelationships:* All entry programs

*Table/record interrelationships:* id_rec

**Step Contact record**

Defines the contacts to be generated for a given step, track, and Tickler strategy.

*UNIX filename:* stepctc

*Informix filename:* stepctc_rec

*Schema location:* $CARSPATH/schema/common

*Program interrelationships: tickent, tickler*

*Product interrelationships:* none

*Table/record interrelationships:* ctc_rec, step_rec

**Step Object record**

Defines the objective for a particular step in a track within a defined Tickler system.

*UNIX filename:* stepobj

*Informix filename:* stepobj_rec

*Schema location:* $CARSPATH/schema/common

*Program interrelationships: tickent, tickler*

*Product interrelationships:* none

*Table/record interrelationships:* ctc_table, step_table

**Step record**

Defines the steps to be taken for a given Tickler system and track.

*UNIX filename:* step

*Informix filename:* step_rec

*Schema location:* $CARSPATH/schema/common

*Program interrelationships: tickent, tickler*

*Product interrelationships:* none

*Table/record interrelationships:* stepctc_rec, step_table, trk_table

**Step Requirement record**

Defines the requirements to activate a step for a specified Tickler system and track.

*UNIX filename:* stepreq

*Informix filename:* stepreq_rec

*Schema location:* $CARSPATH/schema/common

*Program interrelationships: tickent, tickler*

*Product interrelationships:* none

*Table/record interrelationships:* ctc_table, step_table

**Step table**

Defines the steps to be taken for a given Tickler system and track.

*UNIX filename:* step

*Informix filename:* step_rec

*Schema location:* $CARSPATH/schema/common

*Program interrelationships: tickent, tickler*

*Product interrelationships:* none

*Table/record interrelationships:* stepobj_rec, stepreq_table, tick_rec

**Suffix table**

Defines the suffixes for names (e.g., CPA and M.D.), and the text associated with each suffix.

*UNIX filename:* tsuffix

*Informix filename:* suffix_table

*Schema location:* $CARSPATH/schema/common

*Program interrelationships: adr*

*Product interrelationships:* All entry programs

*Table/record interrelationships:* addree_rec, id_rec

**Tickler record**

Contains information needed to place an individual or an organization in a specific Tickler system, and specifies a completion date for the expected communication.

*UNIX filename:* tick

*Informix filename:* tick_rec

*Schema location:* $CARSPATH/schema/common

*Program interrelationships: tickent, tickler*

*Product interrelationships:* none

*Table/record interrelationships:* step_table, tick_table, trk_table

**Tickler table**

Defines the valid Tickler systems and the codes for the systems.

*UNIX filename:* ttick

*Informix filename:* tick_table

*Schema location:* $CARSPATH/schema/common

*Program interrelationships: tickent, tickler*

*Product interrelationships:* none

*Table/record interrelationships:* tick_rec

**Title table**

Defines the titles used in addressing an individual (e.g., Captain, Mr. and Mrs., and Dr.) and a code for each of the titles.

*UNIX filename:* ttitle

*Informix filename:* title_table

*Schema location:* $CARSPATH/schema/common

*Program interrelationships: adr*

*Product interrelationships:* All entry programs

*Table/record interrelationships:* adree_rec, id_rec

**Track table**

Defines alternate paths through the same Tickler system and the code associated with the path.

*UNIX filename:* ttrk

*Informix filename:* trk_table

*Schema location:* $CARSPATH/schema/common

*Program interrelationships:* tickent, tickler

*Product interrelationships:* none

*Table/record interrelationships:* step_rec, tick_rec

# SECTION 4 - COMMUNICATIONS MANAGEMENT MACROS AND INCLUDES

## Overview

### Introduction

This section provides reference information about macros that set up the Communications Management product.

### The Relationship Among Macros, Includes, and C Programs

CX offers several methods to allow institutions to implement or define their customized use of features. In much of CX, m4 macros located under $CARSPATH/macros include the definitions of features that have been designed to be modified for each institution. Source code, however, is not passed through the m4 processor because the C compiler has its own preprocessor - cc. Macros in C programs are handled by this preprocessor.

To provide the same apparent functionality to C programs, a section in the include source directory has been allocated for a type of include file that acts as an intermediary between the m4 processor and the cc preprocessor. In operation, m4 macros are defined whose output is a valid cc macro. These m4 macros are placed in the include files. Then the include files are translated and the appropriate cc macro is placed in the include file. The installed include file is included by the C compiler at compile time so that the result of the compilation is influenced by the m4 macro.

### General Installation Procedures

See the *CX System Reference Technical Manual* for general procedures on setting and installing changes to macros and includes.

### Configuration Table

The Configuration table (config_table) will eventually replace macros for every CX product; however, the current Communications Management product does not access the Configuration table to obtain processing values.

## Communications Management Macros

### Introduction

CX contains macros that define specific values used throughout the Communications Management product. The macros and includes enable you to change the available options and functionality of the Communications Management product without having to modify C code. By modifying macros, you can customize your implementation of the Communications Management product and make the product easier to maintain.

### Definition and Function

A macro is an instruction that causes the execution of a pre-defined sequence of instructions in the same source language. A macro consists of uppercase letters and underscores, and is used in place of a text string within source files. CX expands the macro to the longer text during the installation process for a file. CX uses the following kinds of macros:

- Enables - Allow you to enable a CX feature
- DBS_COMMON - Allows you to define database values in screens
- Periodic - Allows you to make changes on a periodic basis (e.g., quarterly)

Macros can perform one of the following functions:

- Define defaults on a screen (_DEF)
- Define valid values in a field (_VALID or _INCL)
- Enable system modules (ENABLE_MOD)
- Enable system features (ENABLE_FEAT)
- Establish a valid value for a C include file

### Macro Usage in Communications Management

In addition to the standard functions for macros within CX, Communications Management uses macros to minimize setup effort and to maximize letter flexibility. The macros that perform these special purposes within Communications Management are called WP (Word Processing) macros.

### How to Locate Macros

Communications Management macros reside in different files, depending on their purposes. The following list contains directory paths for the different types of macros that you may need to customize.

#### Enable macros

*Purpose:* Implement Communications Management features.
*Location*: $CARSPATH/macros/custom/common and $CARSPATH/macros custom/admissions

#### Periodic macros

*Purpose:* Define valid date values that can change over time.
*Location*: $CARSPATH/macros/custom/periodic

#### Tickler macros

*Purpose:* Define valid codes that relate to Tickler processing.
*Location*: $CARSPATH/macros/custom/admissions

#### Alumni/Development macros

*Purpose:* Automate receipt production.
*Location*: $CARSPATH/macros/custom/develop

**Word Processing (*nroff*) macros**

*Purpose:* Provide user-friendly names for variables in letters.
*Location*: $CARSPATH/macros/custom/ltrwp

**Other Word Processing macros**

*Purpose:* Define filename extensions for WordPerfect and Word for Windows merge files, and implement some word processing features.
*Location*: $CARSPATH/macros/custom/common

## Applocate Program

You can also locate macros using the *applocate* program. This program checks the descriptions of macro files for the product you specify (in this case, the Communications Management product is part of the CX *common* area), and lists each file that it locates in a separate file in your home directory.

Follow these steps to run the *applocate* program.

1. Select Utilities from the CX menu. The Utilities: Main menu appears.
2. Select File Options. The Utilities: File Options menu appears.
3. Select Locate Macro Values. The Locate Macro Values screen appears.
4. Select **Table Lookup** in the Macro Category field. A list of module names appears in a Table Lookup box.
5. Select a module name (e.g., COMMON), and click **OK**. The Table Lookup box disappears.
6. Select **Finish**. The Output Parameters window appears.
7. Enter the following:
   - In the Time field, enter **NOW**
   - In the Background field, enter **Y**
8. Select **Finish**.

The system creates the file, *applocate.out*, and sends it to your home directory.

**Note:** Since *common* macros can relate to other aspects of CX, you must review the applocate.out file to locate the macros that directly impact Communications Management.

## Enable Macros

During Communications Management implementation, you may need to modify the following enable macro, located in the $CARSPATH/macros/custom/common file.

**m4_define(`ENABLE_FEAT_FPS,`Y')**
Defines whether or not to display the menu options dealing with the Forms Production System. The macro default setting, Y, specifies that the FPS menu options appear.

To enable Tickler processing, you may need to modify the following enable macro, located in the $CARSPATH/macros/custom/admissions file.

**m4_define("ENABLE_FEAT_ADM_TICKLER","Y")**

Defines whether the institution will use *tickler* to schedule Contact records in the Admissions office.  The macro default setting, Y, maintains the *tickler* program and tables in the system.

**m4_define(“ENABLE_FEAT_AUTO_DOC_TRACK”,”Y”)**
Defines whether the system will call the SQL statement in $CARSPATH/admit/informers/admdocone to automatically evaluate and add document control contacts based on the adm_rec.add_doc field.  Enable this macro only after the SQL statement has been modified and the Contact table has been created to evaluate the correct table values against each applicant.

**m4_define(“ENABLE_FEAT_PREV_INTERACTIVE”,”Y”)**
Defines whether the system will automatically maintain previous addresses for individuals, businesses, or organizations with a change of address.  If the macro value is Y, programs will prompt users about maintaining previous addresses when they change an address; if the macro value is N, the system will automatically save the previous address.

**m4_define(“ENABLE_FEAT_PREV_PHONE”,”Y”)**
Defines whether the system will maintain previous telephone numbers for individuals, businesses, or organizations with a change on number.

### Periodic Macros

The following lists the Communications Management periodic macros located in $CARSPATH/macros/custom/periodic.  You must update these macros annually to ensure that you are using current dates.

**m4_define(‘ADM_TICK_CMPL_DATE’,’09/01/1997’)**
Defines the date on which all Admissions Tickler systems become complete.

**m4_define(‘FA_TICK_DEF’,’FY97’)**
**m4_define(‘FA_TICK_VALID’,’FY94,FY95,FY96,FY97,FY98,FY99’)**
**m4_define(‘FA_TICK_INCL’,’include=(FA_TICK_VALID), upshift’)**
**m4_define(‘FA_TICK_NEXT’,’FY98’)**
Define the valid dates and default values for Financial Aid Tickler systems.

### Tickler Macros

The file $CARSPATH/macros/custom/admissions contains the following macro:

**m4define("ADM_TICKLEVEL_DEF","M")**
**m4define("ADM_TICKLEVEL_VALID","A,M,Z")**
**m4define("ADM_TICKLEVEL_INCL","include=(ADM_TICKLEVEL_VALID), upshift")**
Define the codes for tickler levels.  In the standard CX product, level A designates the most frequently contacted individuals (e.g., the most qualified applicants), level M designates most individuals (e.g., the average applicants), and level Z designates the least frequently contacted individuals (e.g., the marginal applicants).

**m4define("ADM_TICKTRACK_DEF","HS  ")**
Defines the default tickler track for Admissions.

### Alumni/Development Macro

The file $CARSPATH/macros/custom/develop contains the following macro:

**m4_define(‘ENABLE_FEAT_RECEIPT_LETTERS’. ‘N’)**

Causes gift receipts to be included in the acknowledgment letter.

### Contact Code and *adr* Macros

The file $CARSPATH/macros/user/common contains a variety of macros that impact contact and *adr* code values.

**CAUTION:** Do not modify the macros in this file, as they work with values in C programs.

### Word Processing (*nroff*) Macros

The file $CARSPATH/macros/custom/ltrwp contains the following types of macros::

**WP_DEF('WP_ACAD_STAT', 'TA')**
**WP_DEF('WP_ACCT_BAL', 'BL')**
Provides a readable name for a variable to be included in a letter, and the corresponding two-character code that *nroff* uses when formatting letters. The above two examples are representative of more than 200 *nroff* macros in the standard CX product, and your institution can add more macros if desired. When adding *nroff* macros, you must ensure that the two character value for the macro (e.g., TA or BL) is unique.

### Other Word Processing Macros

The following lists the Communications Management macros located in the $CARSPATH/macros/custom/common file.

**Note:** A variety of other macros are used within the body of customized letters, and do not contain the same types of global instructions that the macros in the macros/custom directory provide. For more information about using other word processing macros, see *Customizing the Communications Management Processes* in this manual.

**m4_define('MERGE_WORDP_FILE_EXT','mrg')**
Defines the filename extension for WordPerfect merge files. These are the files placed in the Merge drawer (e.g., wp/admissions/FileCabinet/Merge) in WPVI when generating merge data files instead of *nroff* letter files.

**m4_define('MERGE_WORDW_FILE_EXT','doc')**

Defines the filename extension for Word for Windows merge files. These are the files placed in the Merge drawer (e.g., wp/admissions/FileCabinet/Merge) in WPVI when generating merge data files instead of *nroff* letter files.

## Communications Management Includes

### Introduction

The Communications Management product contains includes that determine the features enabled in the product. An include can either be a compile option that enables or disables a feature, or that defines default values for a feature.

To enable a feature in the Communications Management product, you must define an include in $CARSPATH/include/common. To disable an include, comment out the include in the same file. See the *CX System Reference Technical Manual* for more information on enabling and disabling includes. By modifying includes, you can customize your implementation of the Communications Management product and make the product easier to maintain.

### Purpose

An include allows you to activate or deactivate features in C programs without explicitly changing the C code.

### Macro Dependency

Includes have a dependency on macros. Normally, you do not directly modify includes for the product. You must modify a corresponding macro value and then reinstall the include. For some parts of CX, a user must modify includes directly, but for others, a user must modify an associated macro and does not directly modify includes. Reinstalling includes is dependent on macro values.

### How to Locate Includes

To locate a Communications Management include, access the following file: $CARSPATH/include/util/adr.

**Note:** For more information about the MAKE processor and modifying includes, see the *Jenzabar CX Technical Manual*.

### Associated Includes

The following modification may be required for the Communications Management includes:

**#define ADR_THE_LASTNAME_S**
Controls the formatting of last names in case of missing title codes. Default formatting of formal joint labels using title codes is Mr. and Mrs. *firstname initial lastname* (e.g., Mr. and Mrs. John J. Smith). If one or both of the title codes is blank or invalid, the default format is to use the informal addressing style (e.g., John and Mary Smith). This include changes the formatting to use the last name only (e.g., The Smiths). If the institution prefers the informal style, mark the include in the file as a comment.

# SECTION 5 – JENZABAR CX PROGRAM FILES

## Overview

### Introduction

This section provides reference information about the files that relate to most CX programs. By understanding the file structure and the contents of the files, you can locate most of the information you need about any program.

### Program Files Detailed

This section contains details about the following files:

**Note:** All other files for each CX program are standard C programming files with standard components and structure.

**def.c**
The def.c file contains the definition of external variables (including structures) that must be available to all source files in the program. These variables can also be initialized in this file. As with other C source files, the files also contain comments. The **makedec** command uses the def.c file to create the dec.h file. In the dec.h file, all definitions of variables are converted to external declarations.

**mac.h**
The mac.h file contains preprocessor include and define statements, typedef statements, and structure template definition statements. The file also contains macro substitution defines and declarations of structures. This file is included in all source files during compilation through use of the dec.h file.

### Definition File

Every program uses a definition (def.c) file, located in the $CARSPATH/src directory path.

The *def.c* file for a screen-oriented program can contain the following information:

- Includes for a mac.h file
- Declaration of global variables and structures used throughout the program
- Structure and non-structure screen binds (i.e., program buffer to screen buffer binds)
- Ring menu definitions
- Prompt line information
- Program parameters
- Declarations of dynamic memory (dmms, dmls, and dmlts) in relation to functionality within libdmm (the dynamic memory management package)
- Form pointers that provide the location for forms
- Sqlda pointers that bind the data structure to the form
- Screen pointers

The def.c file for a non-screen-oriented program can contain the following information:

- Includes for a mac.h file
- Global program variables
- Includes for schema files def.c files
- Form pointers that provide the location for forms
- Sqlda pointers that bind the data structure to the form
- dmm, dml, and dmlt definitions
- Program parameters
- Declarations of functions so the compiler can handle a call of that function

- Screen pointers

### Example of a def.c File

The following is an edited excerpt from the def.c file for the Communications Management program *tickler*. It appears here to illustrate the common components of a standard CX def.c file.

**Note:** The legend for the file contents directly follows the example.

```
#include "mac.h"                                                                      1

#include <util/getparam.h>
#include <schema/common/iddef.c>                                                      2
#include <schema/common/ctcdef.c>

extern void             exit();

int alter_tick = TRUE;  /* Allow tick modification */
int alter_date = TRUE;  /* Allow date modification */                                 3
int dump_only = FALSE;  /* Only do dump */

long          today;                        /* INFORMIX date */

char          mprompt[2][81];               /* Main prompt lines */                   4
char          mail_user[9];                 /* User to mail messages to */
char          *scr_errm();

SCREEN        *ctc_scr=NULL;                /* Contact edit screen */
SCREEN        *output_scr=NULL;   /* Output Screen */                                 5
SCREEN        *scr_load();

void          tk_close();
void          tk_dumpstep();                                                          6
void          tk_outstep();

/* -----
    Program parameters
----- */
struct param_type prog_params[] =
   {
              /* ---- PROGCVT WARNING ---- */                                         7
/* The field name "tick_code", in the following line, */
{ 'r', (char *)&review_only, PRM_LOGICAL, 0, PRM_TRUE, "review_only",
        "review only" },
    { 'o', (char *)&dump_only, PRM_LOGICAL, 0, PRM_TRUE, "dump_only",
        "dump only" },
int max_params = (sizeof(prog_params)/sizeof(struct param_type));

/* -----
    Structure Definitions
----- */                                                                              8
struct tick_type        tick_rec; /* Tickler file structure */
struct ttick_type       tick_tb;  /* Tickler table structure */
struct id_type                    id_rec;             /* Id file structure */
struct ctcuse_type      ctcuse;             /* Contact usage structure */

/* -----
    DMM Structure Setup
----- */
DMLT_TYPE_DEF(ctc_dmlt, struct ctclev_type);          /* Contact level */             9
DMLT_TYPE_DEF(req_dmlt, struct reqlev_type);          /* Requirement level */
DMM_DEF(tctc_dmm, struct tctc_type);                  /* Contact table dmm */
DMM_DEF(ttrk_dmm, struct ttrk_type);                  /* Track table dmm */
```

**Legend for the def.c file:**

1. mac.h include
2. schema file def.c's
3. external variable declarations
4. global program variables
5. screen pointer definition
6. global variable declarations
7. program parameters
8. structure definitions
9. dmm and dml structure definitions

### mac.h Files

Every program uses a macro header (mac.h) file, located in the $CARSPATH/src directory path.

The *mac.h* file can contain the following information:

- Includes related to system header files
- Includes related to the CX System library and other application processes
- Includes for schema type mac.h files
- Program constant definitions (i.e., *#define* statements)
- Structure definitions

**Example of a mac.h File**

The following is an edited excerpt from the mac.h file for *tickler*. It appears here to illustrate the common components of a standard CX mac.h file.

**Note:** The legend for the file contents directly follows the example.

```
#include <sys/types.h>
#include <string.h>                                                          1
#include <util/ids.h>
#include <schema/common/idmac.h>
#include <schema/common/ttrkmac.h>                                           2

/* Return statuses */

#define     TK_OK                          0
#define     TK_FATAL         -1

/* INFORMIX Values */

#define DB_LOCKED            107
#define DB_DEADLOCK          6022

/* Message routine values */

#define     MSG_GROUP                      1

/* Record edit action flags */

#define     EDIT_ADD          1
#define     EDIT_UPD          2
#define     EDIT_DEL          3

/* Tickler statuses */                                                       3

#define     TK_ACTIVE                      'A'
#define     TK_INCOMING                    'I'

/* Conditionals */

#define OTHER_OUTPUT 1                /* Allow extra options in output */
#endif

/* Screens */

#define SCREEN_DIR           "commgmt/tickler"
#define     TICK_SCR          "commgmt/tickler/tick"

/* Function-type macros */

#define TSTEP_LEV            dml_position(&tick_dmlt, 0)
#define CTC_GETP(p)  (p = (struct ctc_type *) dmm_getp(&ctc_dmm))
#define CHECK_SPAN_LIMITS(p) \
   if (p > tick_tb.max_ctc_span)              \
       p = tick_tb.max_ctc_span;              \
   else if (p < tick_tb.min_ctc_span)\
       p = tick_tb.min_ctc_span

/* Structure declarations */

struct steplev_type
    {
    struct step_type         step;            /* Step record structure */
    long                     beg_date;        /* Date to begin step */
    long                     exp_date;        /* Date step to be completed */     4
    char                     stat[2]; /* Step status */
    long                     date;            /* Status date */
    };
```

**Legend for the mac.h file:**

1. includes for standard header files
2. includes for schema files
3. program constant and macro definitions
4. structure definitions

# SECTION 6 - ADDRESSING

## Overview

### Introduction

This section provides reference information about the Addressing (*adr*) program. The Communications Management product uses *adr* to extract address and salutation information from a variety of source records.

Most of *adr*'s functionality comes not from the code in the program itself, but from the library *libadr*. This partitioning allows some programs (e.g., *regent* in the Registrar application) to access adr functionality. Therefore, the *adr* program itself primarily performs front-end processing that enables other processes (e.g., letter writing) to access the CX address logic rules. By supplying resources to processes such as letter writing, *adr* minimizes the need to call other programs (e.g., *regent*).

### Program Features Detailed

This section contains details about the following features of the *adr* program:

- Process flow
- Parameters
- Program screens

### Program Files

All the program files for *adr* appear in the following directory: $CARSPATH/src/common/adr

The *adr* program is also linked with the file $CARSPATH/src/Lib/libadr

### Program Screens

Because *adr* is a background process, it does not use program screens or windows.

### Tables Used in the Program

The *adr* program uses the following tables and records:

**aa_rec**
The Alternate Address record that contains free-format, expanded addresses, or addresses that vary with the time of year.

**aa_table**
The Alternate Address table that defines the valid types of alternate addresses the institution wants to track.

**addree_rec**
The Addressee record that contains former names and social security numbers for individuals or organizations whose information has changed.

**adr_rec**
The Addressing record that defines the style and joining relationships to use for correspondence.

**adr_table**
The Addressing table that defines valid runcodes.

**ctry_table**

The Country table that defines valid country codes.

**id_rec**
The ID record that contains name and address information for an individual or organization.

**hold_act_table**
The Hold Action table that defines the processes to disallow when a particular hold is in effect.

**hold_rec**
The Hold record that contains information about the holds that have been placed on an individual.

**rel_rec**
The Relationship record that defines the relationship between two specified IDs.

**state_table**
The State table that defines the valid state codes.

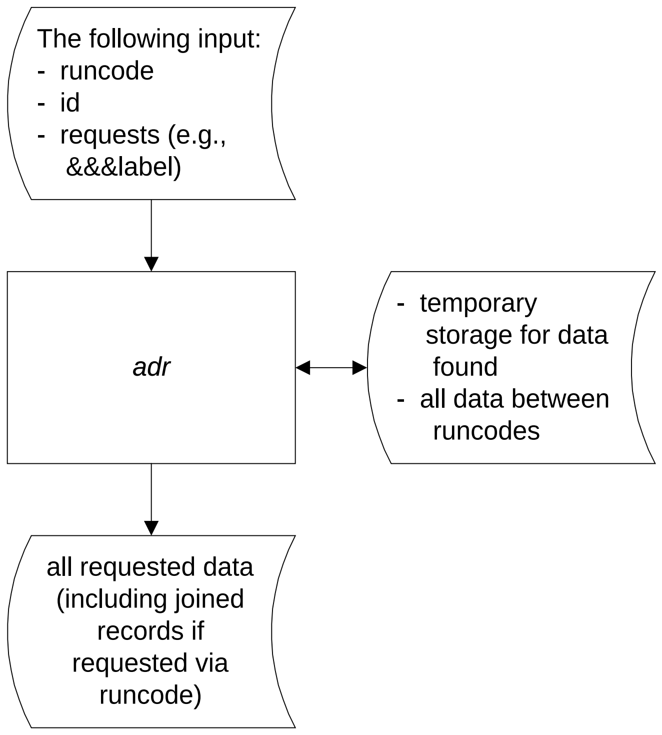
**suffix_table**
The Suffix table that defines the valid suffixes to use with names (e.g., ESQ. or M.D.).

**title_table**
The Title table that defines the valid titles of authority to use with names (e.g., MR, MRMS, CAPT).

## Process Flow

### Diagram

The following diagram shows the flow of data in the *adr* program.


The following input:
- runcode
- id
- requests (e.g., &&&label)
*adr*
- temporary storage for data found
- all data between runcodes
all requested data (including joined records if requested via runcode)

### Data Flow Description

The following describes the data flow in the *adr* program.

1. The data stream entering *adr* consists of various commands or requests (e.g., &&&label). The commands request, for example, a label for the current ID, using information in the appropriate addressing tables as specified by the runcode.
2. If the runcode calls for records to be joined, then *adr* routes the data to temporary storage until the data is complete or a new runcode is detected.
3. When the data is complete, *adr* joins records and resolves duplicates as directed by the runcode.
4. Requests in the data stream are replaced by their values and the data is sent to the standard output of the *adr* program.

**Note:** The primary purpose of *adr* is to serve as a front-end for the library *libadr*. Most of the addressing functionality resides in the library. The *adr* program ensures the data can be passed to the code in *libadr*, and provides for temporary storage when the runcode calls for joined records. The entry of a new runcode resets the process and causes the output of all accumulated data. The *adr* program runs as a UNIX filter type of program, in that both input and output are in standard form.

**Program Relationships**

The following programs use *adr*.

- The letter writing process *ltbrun* takes the output from an ACE report, which may contain requests for address and/or name components. These requests are input to *adr*, which returns the appropriate information.
- The *sortpage* program may use extracted names and addresses from *adr*.

## Parameters

### Introduction

CX contains parameters and compilation values for executing the *adr* program. You can specify parameters to compile *adr* in a specified manner at the time of execution.

**Note:** You can also specify compilation values with the includes for the Communications Management product that affect the *adr* program.

### Parameter Syntax

You can display *adr* parameters by entering the following: **adr -,**

The following is the correct usage for running the *adr* program from the UNIX shell:

**adr [-d date] [-r run_code] [-u user_id] [-b] [-e] [-i] [-n]**

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

### Parameters

The following lists the parameters for running *adr*.

**-d date**
Optional - Specifies the date that you want to use to select the correct address.

**-r run_code**
Optional - Specifies the run code that you want to use to join records and select salutations.

**-u user_id**
Optional - Specifies the ID number of the user for those cases in which the selected salutation varies by user (e.g., a letter from the president may address some donors by their first names, while all other correspondence from the institution uses surnames for the donors).

**-b**
Optional - Joins two records connected by an appropriate Relationship record if both are present in the data stream.

**-e**
Optional - Selects IDs for a deceased person. The default is to not select IDs for the deceased.

**-i**
Optional - Selects IDs for which the address is flagged as incorrect. The default is to not select IDs for incorrect addresses.

**-n**
Currently not in use - designed to use daemon mode, causing contacts to refer to the flags (correct_addr and decsd) in the id_rec.

# SECTION 7 - ADDRESS TEST

## Overview

### Introduction

This section provides reference information about the Address Test (*adrtest*) program. The *adrtest* program enables users to test how a particular set of table entries will cause an address or salutation to be formatted. This process helps users verify that the correct alternate address has been selected. The process uses the same logic that *adr* uses for address selection (i.e., the library *libadr*), so users can ensure that the final printed letters, labels and envelopes will be correct.

### Program Features Detailed

This section contains details about the following features of the *adrtest* program:

- Process flow
- Parameters
- Program screens

### Program Files

All the program files for *adrtest* appear in the following directory:
$CARSPATH/src/common/adrtest

The *adrtest* program is also linked with the file $CARSPATH/src/Lib/libadr

### Tables Used in the Program

The *adrtest* program uses the following tables and records:

**aa_rec**
The Alternate Address record that contains free-format, expanded addresses, or addresses that vary with the time of year.

**aa_table**
The Alternate Address table that defines the valid types of alternate addresses the institution wants to track.

**addree_rec**
The Addressee record that contains former names and social security numbers for individuals or organizations whose information has changed.

**adr_rec**
The Addressing record that defines the style and joining relationships to use for correspondence.

**adr_table**
The Addressing table that defines valid runcodes.

**ctry_table**
The Country table that defines valid country codes.

**id_rec**
The ID record that contains name and address information for an individual or organization.

**hold_act_table**

The Hold Action table that defines the processes to disallow when a particular hold is in effect.

**hold_rec**
The Hold record that contains information about the holds that have been placed on an individual.

**rel_rec**
The Relationship record that defines the relationship between two specified IDs.

**state_table**
The State table that defines the valid state codes.

**suffix_table**
The Suffix table that defines the valid suffixes to use with names (e.g., ESQ. or M.D.).

**title_table**
The Title table that defines the valid titles of authority to use with names (e.g., MR, MRMS, CAPT).

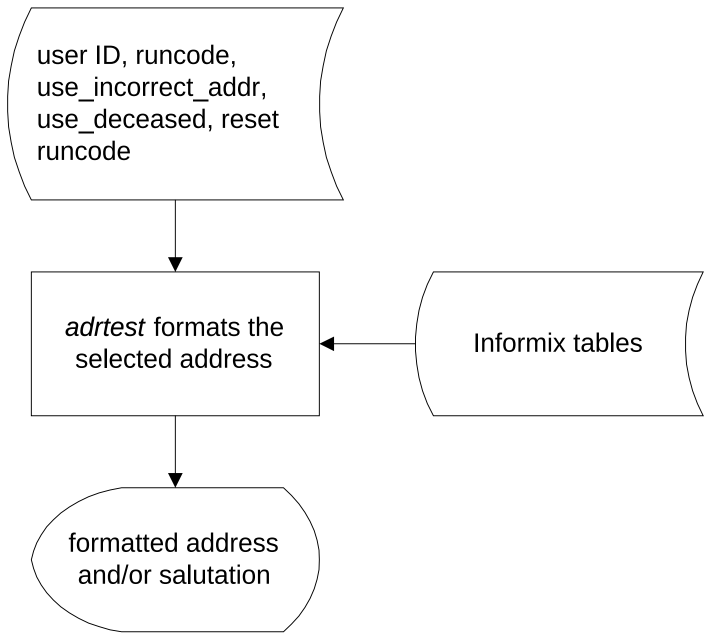
## Program Parameters

The *adrtest* program does not require any parameters.  However, if you use a Type of **S** on the Test ADR screen, *adrtest* produces tests of both the salutation and the address.

## Process Flow

### Diagram

The following diagram shows the flow of data in the *adrtest* program.


user ID, runcode, use_incorrect_addr, use_deceased, reset runcode
*adrtest* formats the selected address
Informix tables
formatted address and/or salutation

### Data Flow Description

The following describes the data flow in the *adrtest* program.

1. The *adrtest* program reads the Alternate Address record (aa_rec) and the Addressing record (adr_rec), and obtains the Priority from the Alternate Address table.
2. The *adrtest* program creates an address and, if desired, a salutation based on the address criteria in the records, and displays it on the screen.
3. The user reviews the address to ensure that *adr* will select the correct alternate.

**Note:** The *adrtest* program is useful for testing table setup and obtaining immediate feedback about the impact a particular table entry will have on *adr* output. As users change table values, they can verify how *adr* will respond to the changes without producing a complete run of letters.

### Program Relationships

The *adrtest* program is a stand-alone program to test address generation. It does not interact with other Communications Management programs.

## Program Screens

### Introduction

The *adrtest* program uses one screen on which the user enters test information.

### Access

The screen file is located in the following directory path:
$CARSPATH/modules/common/progscr/adrtest

### Screen Files and Table/Record Usage

The *adrtest* program screen appears in the following file and uses the indicated table and record:

**testadr**
Contains the Test ADR screen.

*Tables/Records:* aa_rec, id_rec

# SECTION 8 - CONTACT BATCH ENTRY

## Overview

### Introduction

The Contact Batch Entry program (*ctcbatch*) allows you to identify a contact resource code, status, tickler, and due date, and link as many student IDs as necessary to update the related Contact records (ctc_rec). When you execute the program, the system attempts to add or update the contacts in the Contact record that correspond with the IDs you have entered. You can enter the IDs manually in the ID List screen, or you can create a list of IDs in a file and then use the Filename option to preload the process.

### Program Features Detailed

This section contains details about the following features of the *ctcbatch* program:

- Process flow
- Parameters
- Program screens

### Program Files

All the program files for *ctcbatch* appear in the following directory:
$CARSPATH/src/common/ctcbatch

### Tables Used in the Program

The *ctcbatch* program uses the following tables and records:

**ctc_rec**
The Contact record that tracks scheduled and completed correspondence.

**idctc_rec**
The ID Contact record that provides for batch updates to the Contact record.

**Note:** This table is a temporary holding file.

**id_rec**
The ID record that provides name and address information for an individual or organization.

## Process Flow

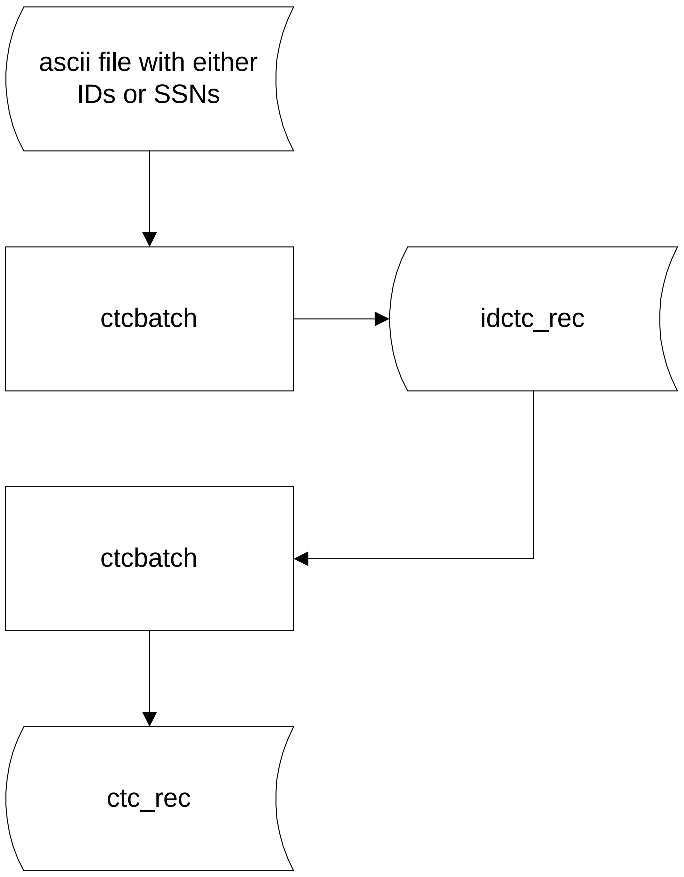
**Methods of Running Contact Batch Entry**

Users can run *ctcbatch* in the following two ways:

- Interactively, where users manually enter student IDs or social security numbers
- In batch mode, where users create an input file containing a list of ID numbers and/or social security numbers

**Diagram**

The following diagram shows the flow of data in the *ctcbatch* program.



**Data Flow Description**

The following describes the data flow in the *ctcbatch* program.

1. The user provides the following input:
   - Processing parameters
   - Data entered on a query screen if the parameter list does not provide all the required information

     **Note:** This input includes resource code, status, and date to use on the ctc_recs created.
   - Standard text file that contains a list of either ID numbers or social security numbers, arranged one per line.

**Note:** This input can be created manually or by using an ACE report to extract the data.

2. The *ctcbatch* program stores the IDs in the temporary holding record, idctc_rec. This record stores the IDs until each ID can be processed.
3. The *ctcbatch* program uses the idctc_rec as input to create the desired ctc_recs.

**Example:** If you specify a status of C, *ctcbatch* attempts to add or update contacts with a status of C for the students in the ID list. For example, if ID 14232 currently has a Contact record that matches the input parameters (e.g., the specified tickler and resource) and that Contact record has a status of E, then *ctcbatch* updates that Contact record to a status of C (assuming the Status field was C). If an ID number currently does not have a Contact record that matches the input parameters, *ctcbatch* adds a Contact record with the status of C.

When *ctcbatch* successfully updates or creates a Contact record, the program removes the ID from the idctc_rec.

**Note:** Because each ID remains in the idctc_rec until *ctcbatch* processes the ID successfully, you have the option to re-run the process for each ID that did not process. This ensures data integrity throughout the program processing.

## Contact Record Logic

The following describes the logic of how *ctcbatch* adds and updates Contact records for each student in the ID list.

**If the Status field contains C (for Completed)**
The *ctcbatch* program updates any matching Contact records with a status of E to C. If no matching Contact records exist, then *ctcbatch* creates a Contact record with a status of C.

**Note:** The process only adds completed Contact records when it does not locate any Contact records with an expected status to update to a completed Contact record status.

**If the Status field contains E (for Expected)**
The *ctcbatch* program always adds a Contact record with a status of E.

**Note:** The process only adds Contact records with an expected status if an expected status does not exist for the specified resource, tickler, and ID or social security number.

The process does not add duplicate expected contacts.

**If the Status field contains V (for Voided)**
The *ctcbatch* program updates any matching Contact records with a status of E to a status of V. If a matching Contact record is found, the system does not add another Contact record with a status of V. If a matching Contact record with a status of C is found, the system does not update the Contact record.

**Note:** The *ctcbatch* program only voids Expected contacts.

## Menu Access for Contact Batch Entry

The following lists the menus from which you can access *ctcbatch*. The following options appear in each menu:

- Contact Batch Entry
- Re-run Contact Batch Entry

**Recruiting/Admissions**

Recruiting/Admissions:  Communications Management menu

**Alumni/Development**
Alumni Association:  Communications Management menu
Development:  Communications Management menu

**Support Services**
Support Services:  Communications Management menu

**Financial Aid**
Financial Aid:  Document Tracking menu

## Program Relationships

The *ctcbatch* program creates ctc_recs that serve as input to *ltbrun*, the CX letter writing process.

## Parameters

### Introduction

CX contains parameters and compilation values for executing the *ctcbatch* program.

**Note:** You can also specify compilation values with the includes for the Communications Management product that affect the *ctcbatch* program.

### Parameter Syntax

You can display *ctcbatch* parameters by entering the following: **ctcbatch -,**

The following is the correct usage for running the *ctcbatch* program from the UNIX shell:

Following is the correct usage for running the *ctcbatch* program.

**ctcbatch [-r *resource*] [-t *tickler*] [-s *stat_code*] [-d *due date*] [-f *filename*] [-R]**

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

### Parameters

The following parameters are valid for the *ctcbatch* program.

**-r resource**
Optional - The contact resource code to use.

**-t tickler**
Optional - The contact tickler code to use.

**-s stat-code**
Optional - The contact status code to use.

**-d due date**
Optional - The contact due date to use.

**-f filename**
Optional - The filename of the text ID file to use for running the program in batch mode. See *Contact Batch Entry Interactive Processing* in this document for more information.

**-R**
Optional - A switch that indicates no entry is required. Useful in restarting *ctcbatch* if it creates idctc_recs and then stops. This parameter causes *ctcbatch* to run in batch mode and processes the rows in the idctc_rec to create ctc_recs.

## Program Screens

### Introduction

The *ctcbatch* program uses two screens for the features described in this section.

### Access

The screen files are located in the following directory path:
$CARSPATH/modules/common/progscr/ctcbatch

### Screen Files and Table/Record Usage

The *ctcbatch* program screens appear in the following files and use the indicated tables and records:

**ctc**
Contains the Parameters screen.

*Tables/Records:* ctc_table, tick_table

**id**
Contains the ID List screen.

*Tables/Records:* id_rec, idctc_rec

## Mail Messages

### Introduction

The *ctcbatch* program sends you e-mail messages after the program successfully completes or when errors occur in processing. The mail usually contains helpful summary information such as how the IDs were processed.

### Example Messages

The following is an example of a message indicating a successful conclusion:

**Example:** Message Group: Batch Contact Entry

```
Batch contact update complete.
    1 requested contact change(s).
    0 contact(s) updated.
    1 contact(s) inserted.
/usr/carsi/install/bin/ctcbatch -R
```

The following is an example of an error message from *ctcbatch*:

**Example:** Message Group: Error Messages

```
ctcupdate1 LOCK failed on idctc_rec errno=-289.
Unable to complete batch update. Select rerun batch to try again.

/usr/carsi/install/bin/ctcbatch  -t  DEV  -r  ACK  -s  E  -d 04/21/1994  -f
```

**Note:** If you receive the above error message, see *Re-running Contact Batch Entry* in this section for more information on how to re-run *ctcbatch*.

## Re-running Contact Batch Entry

### Introduction

The *ctcbatch* program may encounter database errors or other problems when processing ID numbers. For example, if a student's Contact record is locked at the time *ctcbatch* attempts to update the record, the system cannot process that student's *ctcbatch* request for contact update. The program keeps the *ctcbatch* request in a holding area, and sends the program operator e-mail indicating that the request could not be processed for that student.

Users have the following two options for re-running the *ctcbatch* process.

### Contact Batch Entry

If you entered ID numbers (or social security numbers) that *ctcbatch* did not process because of errors, you can process those IDs the next time that you run *ctcbatch* for new IDs. When you access *ctcbatch* again, enter any new IDs. Do *not* re-enter the IDs that did not process earlier because of errors. When you execute the new list of IDs, *ctcbatch* processes the new list and any previously unprocessed ID numbers.

### Re-running Contact Batch Entry

If users entered ID numbers (or social security numbers) that *ctcbatch* did not process due to errors, they can select the Re-run Contact Batch Entry menu option from one of the specified menus. This menu option does not require any information, nor does it allow the entering of any IDs interactively. This menu option is a background version of the *ctcbatch* program that attempts to re-process any IDs that the system did not process previously due to errors.

# SECTION 9 - LABELS

## Overview

### Introduction

This section provides reference information about the Labels (*labels*) program. The Communications Management product uses *labels* to arrange blocks of data to the user's specifications. The specifications include label size, spacing between labels, and row/column ordering.

### Program Features Detailed

This section contains details about the following features of the *labels* program:

- Process flow
- Parameters
- Program screens

### Program Files

All the program files for *labels* appear in the following directory: $CARSPATH/src/util/labels

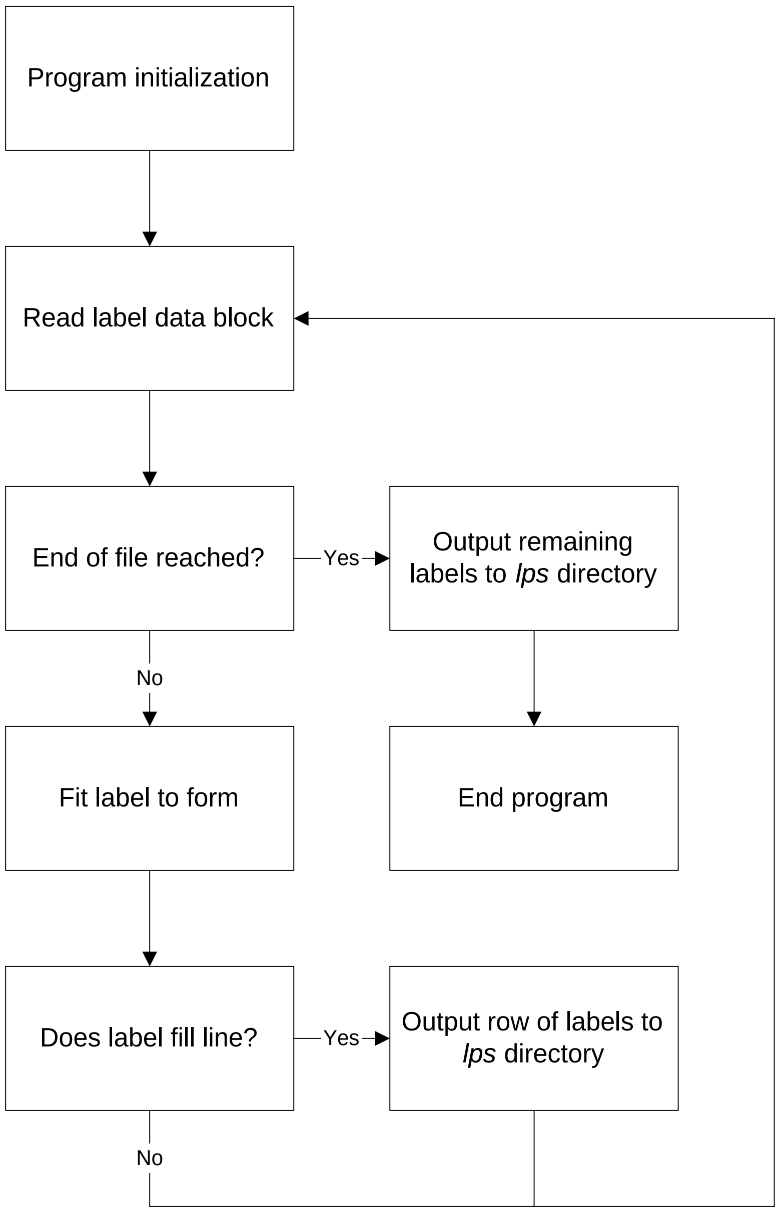
### Tables Used in the Program

The *labels* program does not access the CX database itself, but instead serves as a UNIX filter that accepts a data stream as standard input and directs its output to the *lps* directory.

## Process Flow

### Diagram

The following diagram shows the flow of data in the *labels* program.



### Data Flow Description

The following describes the data flow in the *labels* program.

1. After program initialization, the *labels* program sets the read pointer into the file for column output, if necessary.
2. The *labels* program reads the row labels one at a time until it encounters a line consisting

only of a carriage return.

3. The *labels* program places the label on the first form (of those passed as arguments) that is large enough for the label and updates counters for the number of labels on each form and the number of labels printed to the error file.

   **Note:** If the label is too large for the specified form, then *labels* truncates the label to fit on the last form specified or writes the label to an error file in the user's home directory.

4. If the last label position for the row on the form is filled and you are using multi-column labels, then *labels* prints the row.
5. The *labels* program routes the row of labels (or the incomplete row) to the *lps* file corresponding to the argument passed for the output filename, and generates the commands that *lps* requires.

   **Note:** In the case of the letter writing process, *ltbrun* uses the hardcoded filename of L*resource*, where *resource* is the contact resource code used for the letter run.

6. When the *lps* files are complete, *labels* generates and sends appropriate mail messages and closes input and output files.

**Program Relationships**

The following programs use *labels*.

- The *lps* program prints output from *labels*.
- Data obtained from *ACE* and under the control of *ltbrun*, but processed through *splitpage* serves as input to *labels.*

## Creating Input for the *labels* Program

### Input Data Specifications

The data for a label is expected to be one label line per line with a blank line (consisting only of a carriage return) separating lines of one label from the next.

**Example:**
Mr. James R. Doe, Jr.
23 New Street
Cincinnati, OH  45240

**Example:**
Ms. Susan B. Clarke
2345 Walden Glen
Apt. 26
Cincinnati, OH  45230

**Example:**
Susan and Walter Clarke
New Kensington Avenue
Georges Town, NY 12345-2345

### Using WPVI to Create Input

You can easily create a file according to *labels*'s specifications by using the WPVI program interface to UNIX's vi text processor. Use this approach when you must send letters to a person for whom records do not exist on your database, or to a group of people who are so diverse that creating an ACE report to extract the information is less efficient than typing the names and addresses into a file.

### Using ACE

You can use or create an ACE report to collect and format the data for the labels. Set left, top and bottom margins to 0. Use the following basic format:

```
on every record
   if (title_text > "  ") then
       print title_text clipped, 1 space;
   print _full_name(lead_name)
   if (lead_addr_line1 > "  ") then
      print lead_addr_line1
   if (lead_addr_line2 > "  ") then
      print lead_addr_line2
   if (lead_country = "USA" or lead_country = "   ") then
       print lead_city clipped, ", ", lead_state, 2 space, lead_zip
   else
       begin
       print lead_city clipped, 2 space, lead_zip
       print ctry_text clipped
       end
```

### Using ACE with ADR

You can create an ACE report to collect and format the data for the labels. Left, top and bottom margins should equal 0. Use the following basic format:

```
        first page header
            ADR_DATE(passdate)
            ADR_RUN_CODE("JOINT")

        on every record
            ADR_PRIM_ID(id_no)
            ADR_LABEL
            skip 1 line
            ADR_END
```

Pipe the output from the ACE report to *adr*. The output will be in the format required by the *labels* program.

## Parameters

### Introduction

CX contains parameters and compilation values for executing the *labels* program.

**Note:** You can also specify compilation values with the includes for the Communications Management product that affect the *labels* program.

### Parameter Syntax

You can display *labels* parameters by entering the following: **labels -,**

The following is the correct usage for running the *labels* program from the UNIX shell:

**labels -l scrfile1 [scrfile2 ... ] -t 'text' [-f inputfile] [-r] [-c] [-s] [-n] [-C]**

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

### Parameters

The following lists the parameters for running *labels*.

**-l scrfiles(s)**
Required - One or more label format files that determine the size of each label and the number of labels across each row. If you provide multiple format files, then *labels* will place each label in the *lps* output file corresponding to the first form on which the label fits. Format files are actually screen files and each line of each label is defined as a field in the screen file. The *labels* program expects files in the following locations:

- Installed .scr files - $CARSPATH/install/scr/util/labels
- Source files - $CARSPATH/modules/util/progscr/labels

**-t text**
Required - Text description (of 24 characters or less) within single quotes (‘ ’) that is used as a file name root in the *lps* directory for printing the labels.

**-f inputfile**
Optional - Specifies the file to open for input. If this option is not specified, labels will expect standard input.

**-r**
Optional - Explicitly states that labels are to be horizontally arranged. Horizontal arrangement is the default.

**-c**
Optional - Causes labels to be arranged by column.

**-s**
Optional - Forces all labels to be printed. If a label does not fit one of the available label form files listed with the -l option, extra lines and/or long lines will be truncated to the size of the last form file. If the user does not pass this parameter, *labels* writes oversized labels to an error file in the user's home directory.

**-n**
Optional - Explicitly states that labels are not to be crowded into a form. This option is a default.

**-C**

Optional - Causes the labels to be capitalized.

## Program Screens

### Introduction

The *labels* program uses 23 screens for the features described in this section. Most of the screens define the label forms.

### Access

The screen files are located in the following directory path:
$CARSPATH/modules/util/progscr/labels

### Screen Files for Label Types

The *labels* program screens that relate to labels and envelopes appear in the following files:

**Note:** The standard naming convention for the screen files follows:

- The first number (e.g., 1 as in the first file) indicates number of labels per row.
- The second number (e.g., 11 in the first file) indicates the number of lines on the label.
- The third number (e.g., 40 as in the first file) indicates the number of characters per each line of the label.

Your institution can create and use other screen files and naming conventions.

- 1up11x40
- 1up25x94
- 1up5x35
- 1up5x40
- 1up5x50
- 1up8x35
- 1up8x40
- 2up5x35
- 2up8x35
- 3up5x35
- 3up5x40
- 3up8x40
- 4up5x30
- 4up5x33
- 4up5x35
- env
- env5x7
- env5x7ret
- env6x9
- env6x9ret
- envret

### Label Format Screen Definition

The label format file is a screen file that defines the dimensions of a particular type of label stock. Typically, one label consists of three or more fields. Each field may be a different size, but labels within the same screen should have the same dimensions. Each field must be identified as follows: an *a* corresponds to label 1, *b* is label 2, and *c* is 3. In addition, *0* is field one, *1* is field two, *2* is field three.  You can have as many labels and fields as required by continuing the same numbering and lettering scheme.

When you combine these label and field designations, the results are names as follows:

- *a4* is label 1-field 5
- *c6* is label 3-field 7

### Label Format Screen Sample

A sample screen follows:

```
---------------------------------------------------------------------------------
screen
{
[a0                      ] [b0                      ] [c0                      ]
[a1                      ] [b1                      ] [c1                      ]
[a2                      ] [b2                      ] [c2                      ]
[a3                      ] [b3                      ] [c3                      ]


}
end

attributes

end
---------------------------------------------------------------------------------
```

Within a data file to be passed to the label program, the first group of data will be placed in a0 through a3 of the above screen, the last group of data being placed in locations c0 through c3.

The above screen describes 3-up labels consisting of four printed lines and two blank lines between labels.  Should a label contain only three fields then the fourth field (line) of the label would be blank.

### Processing Screen Files and Table/Record Usage

The *labels* program screens appear in the following files:

**backupids**

**backuplbls**

# SECTION 10 - LETTER PROCESSING SYSTEM

## Overview

### Introduction

This section provides reference information about the Letter Processing System (*lps*) program. The Communications Management product uses *lps* to print letters, envelopes and labels.

### Program Features Detailed

This section contains details about the following features of the *lps* program:

- Process flow
- Program screens

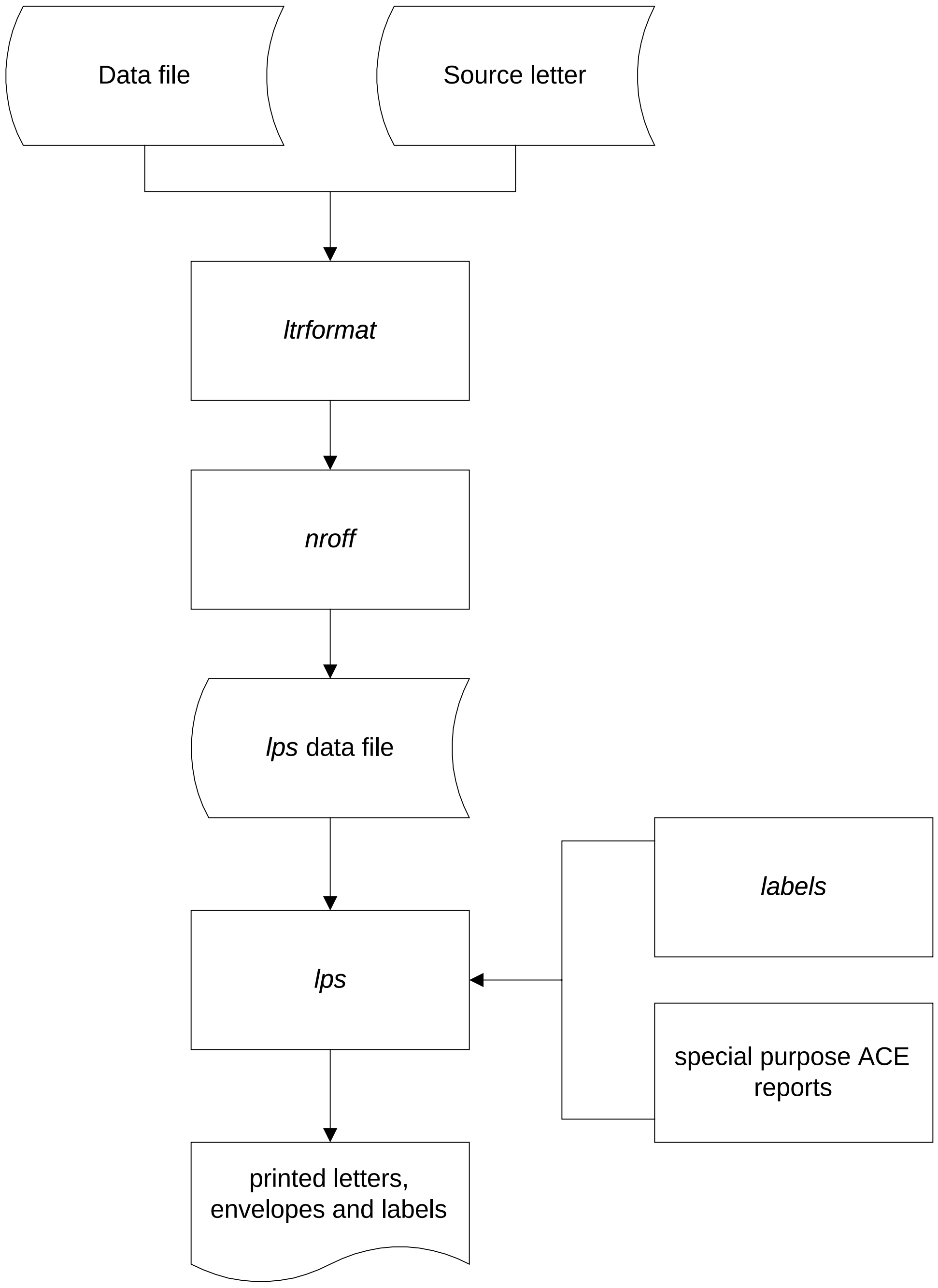
### Parameters

The *lps* program does not require any processing parameters.

## Process Flow

### Diagram

The following diagram shows the flow of data in the *lps* program.


Data file
Source letter
ltrformat
nroff
lps data file
labels
lps
special purpose ACE reports
printed letters, envelopes and labels

### Data Flow Description

The following describes the data flow in the *lps* program.

1. The user runs the letter writing process or an ACE report to produce a data file for the

letter(s).

2. For letters, the data file merges with the source letter (through *nroff*) to produce an *lps* data file.
3. If the operator requested labels, the *labels* program creates the labels.
4. The *lps* program prints the letters, envelopes and labels.

**Program Relationships**

Output from the *ltrformat* and *labels* programs serves as input to *lps*. Some C programs (e.g., the Registrar program *trans*) also put output directly into the *lps* directories.

**Input**

The names of the files in the *lps* directory have two parts: a base name followed by a period (.), then a three character extension. The format of the base name depends upon the source of the file. For letterwriting, a letter file is placed into the *lps* directory with a name equal to the resource code used for the letterwriting run. In contrast, a labels file from letterwriting receives a name L*resource* where *resource* is the contact resource code.

**Example:** If a contact resource for an acceptance letter were ACCEPTED, then the files would have the names ACCEPTED.l1 and LACCEPTED.l1 for the first letter and label file of this type.

The extension provides information about the status or type of file. The extension is of the form .*ann* where *a* represents one of six characters (I, l, t, d, r, or k) and *nn* represents a serial number from 1 to 99. Therefore, up to 99 files with the same name can exist before a *lps* detects a duplicate filename problem.

The six possible characters (and examples of each) are:

**.i15**
An input file while it is being created (e.g., ACCEPTED.i15).

**.l15**
An input file in *lps* (e.g., ACCEPTED.l15).

**.t15**
The tracking file showing progress of printing for the base filename and number. For example, if an operator has a file containing 100 letters, opens it in *lps*, prints 10 letters and exits *lps*., then at that point, the .l15 file remains but in addition there is a .t15 file which shows that 10 letters have been printed (e.g., ACCEPTED.t15).

**.d15**
A tracking file for a file that has been completely printed (e.g., ACCEPTED.d15).

**.r15**
A tracking file for a file that the user, while in *lps*, has indicated may be removed (e.g., ACCEPTED.r15).

**.k15**
A tracking file which is locked (e.g., ACCEPTED.k15).

## Program Screens

### Introduction

The *lps* program uses five screens for the features described in this section. The five screens display and track the letter, envelope and label files that are ready to be printed.

### Access

The screen files are located in the following directory path:
$CARSPATH/modules/util/progscr/lps

### Screen Files and Table/Record Usage

The *lps* program screens appear in the following files and use the indicated tables and records:

**entry**
Contains the File to be Printed screen.

*Tables/Records:* none

**files**
Contains the Files to be Printed screen.

*Tables/Records:* none

**log**
Contains the Print Log screen.

*Tables/Records:* none

**subs**
Contains the Subtypes to be Printed screen.

*Tables/Records:* none

**track**
Contains the Current Status screen.

*Tables/Records:* none

# SECTION 11 - LETTER FORMAT

## Overview

### Introduction

This section provides reference information about the Letter Format (*ltrformat*) program. The Communications Management product uses *ltrformat* to control the process of creating letters or word processor merge files, and to format the letters and envelopes just before printing.

CARS originally created the *ltrformat* program to enable users to produce an unlimited number of letters using merge records that could be passed to nroff in a single file. Later, *ltrformat* was enhanced to handle the production of WordPerfect and Microsoft Word merge files.

The *ltrformat* program is a UNIX filter type of program. It accepts a data stream from standard input and creates standard output. The program requires the operator to pass a parameter that names the letter control file; in the data stream, it requires a line that provides the pathname of the letter itself.

With WordPerfect or Microsoft Word merge files, the letter control file determines how *ltrformat* handles the data stream. For *nroff* letters, it serves as a template that controls the appearance of the letter. For word processor merge files, the output is placed in the Merge drawer of the FileCabinet which contains the letter. For *nroff* letters, the output goes to the *lps* directory under the control of the *runletters* script.

### Program Features Detailed

This section contains details about the following features of the *ltrformat* program:

- Process flow
- Parameters
- Program screens
- Program commands

### Program Files

All the program files for *ltrformat* appear in the following directory:
$CARSPATH/src/common/ltrformat

### Tables Used in the Program

Since *ltrformat* only uses and creates data files, it does not use any CX tables.

### Program Parameters

The *ltrformat* program requires the name of the letter control file. The standard CX product provides the following three letter control files in the directory:
$CARSPATH/modules/common/letters:

**rtf**
Produces a merge file compatible with Microsoft Word.

**stdlps**
Produces an *lps* file with standard size letters.

**wordp**
Produces a merge file compatible with WordPerfect.

**Note:** The letter control file and the letter file itself are not the same.

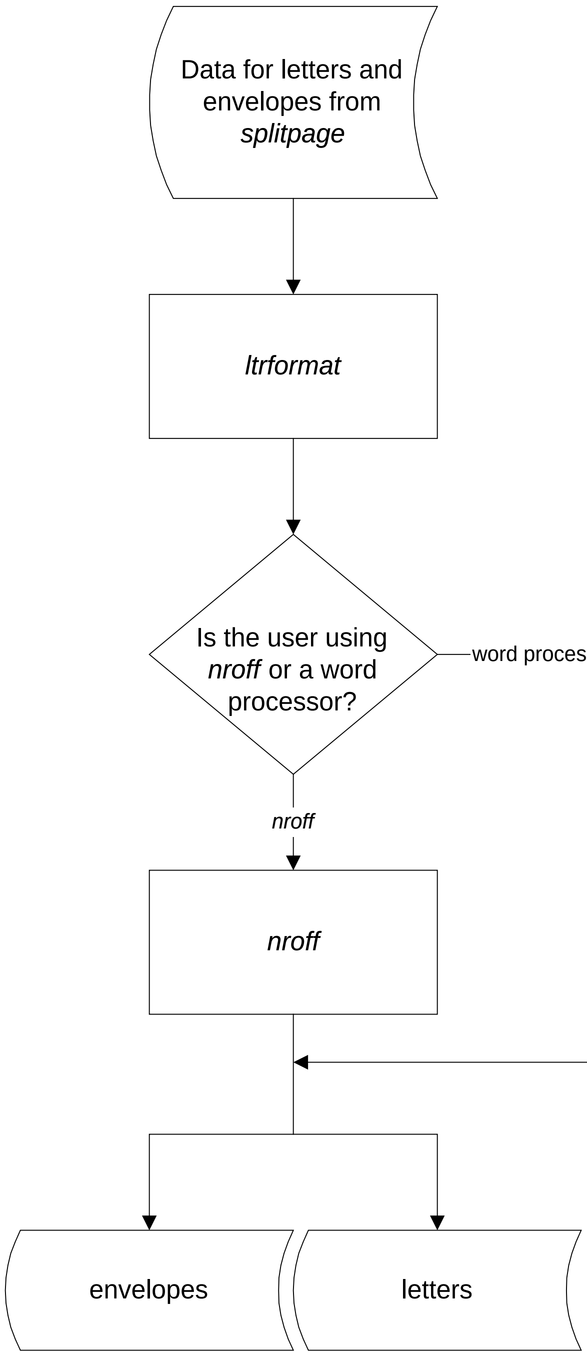
**Program Screens**

Because *ltrformat* is a background process, it does not require any program screens.

## Process Flow

### Diagram

The following diagram shows the flow of data in the *ltrformat* program.


Data for letters and envelopes from *splitpage*
*ltrformat*
Is the user using *nroff* or a word processor?
word processor
merge file
*nroff*
*nroff*
envelopes
letters

### Data Flow Description

The following describes the data flow in the *ltrformat* program when driven by the standard letter writing process.

1. Output from *splitpage* that relates to letters and envelopes serves as input to *ltrformat*. This data stream consists of header information for *lps*, and macros and their values to be merged into the letters. One of these macros (WP_LETTER) must be defined for *ltrformat* to operate.

   **Note:** The *ltrformat* program does not process letters that explicitly do their own reads (i.e., use the *nroff* command .rd). If your institution uses letters that contain the .rd *nroff* command, you must disable the *ltrformat* program to process the letters. For more information, see *Disabling the Letter Format Program in a Menu Option* in this manual.

2. The *ltrformat* program takes the merge records (in groups) and routes them to *nroff* for letter creation.
3. The formatted output files from *ltrformat* and *nroff* are placed in the *lps* directory to serve as input to *lps*.

**Program Relationships**

The following programs use *ltrformat*.

- The *splitpage* program provides input to *ltrformat* in the standard letter writing process.
- The *lps* program uses output from *ltrformat* to print letters and envelopes.

**Program Commands**

The @ commands are used in the *stdlps* file. This file, located in $CARSPATH/install/ltr/common/stdlps.ltr, serves as input for *ltrformat*. The *ltrformat* program interprets output from the ACE report and the *stdlps* file, then passes information to *nroff*. The *stdlps* file is processed until all letters are complete.

**@nroff**
Instructs *ltrformat* that it is going to use *nroff* as the text formatter.

**@dnl**
Removes all input from the source file up to and including the end of the line. Its purpose is to prevent extra blank lines from appearing in the output file.

**@read_single**
Causes the program to process one macro at a time. A macro is a line with a two (2) character macro name, followed by a blank line, followed by the value for the macro (which may be multiple lines), followed by another blank line. The blank line cannot contain any spaces or <Tab> keystrokes.
The following is an example of how *ltrformat* would interpret the @read_single command:
**ACE report:**

```
\!CN

\!Jane Doe
```

**Passed to *nroff* by *ltrformat*:**

```
.ds CN Jane Doe
```

**@read_many**
Handles multiple macro definitions up to the end-of-record indicator (i.e., two pound signs (##)).
The following is an example of how *ltrformat* would interpret the @read_many command:

**ACE report:**

```
\!ID

\!12345
```

```
\!LA

Jane Doe
123 Main Street
Springfield, OH  12345-6789

##
```

**Passed to nroff by *ltrformat*:**

```
.ds ID 12345
.di LA
Jane Doe
123 Main Street
Springfield, OH  12345-6789
.di
```

# SECTION 12 - SORTPAGE

## Overview

### Introduction

This section provides reference information about the Sortpage (*sortpage*) program. The Communications Management product uses *sortpage* to sort output from the *adr* program, since *adr*'s selection of the correct address can cause records to lose their original sort sequence. The *sortpage* program uses the UNIX sort utility, using values supplied by ACE to perform the sort.

### Program Features Detailed

This section contains details about the following features of the *sortpage* program:

- Process flow
- Parameters
- SRT macros
- Commands

### Program Files

All the program files for *sortpage* appear in the following directory:
$CARSPATH/src/common/sortpage

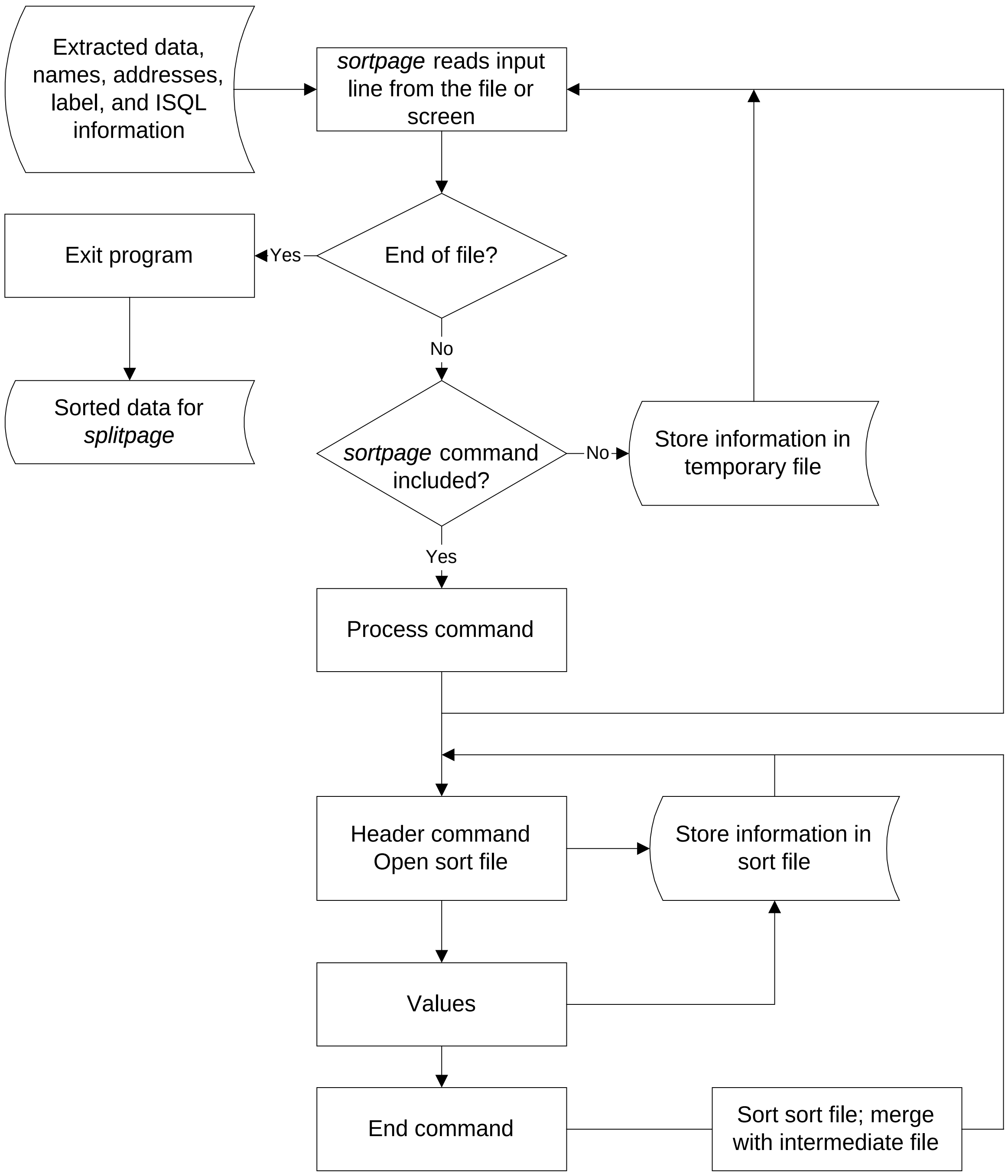
### Program Screens

Because *sortpage* is a UNIX filter process, it does not require any program screens. It can be used as a filter in any process that generates *sortpage* commands as part of the data stream.

## Process Flow

### Diagram

The following diagram shows the flow of data in the *sortpage* program.


Extracted data, names, addresses, label, and ISQL information
sortpage reads input line from the file or screen
End of file?
Yes
Exit program
No
Sorted data for splitpage
sortpage command included?
No
Store information in temporary file
Yes
Process command
Header command Open sort file
Store information in sort file
Values
End command
Sort sort file; merge with intermediate file

## Commands

The *sortpage* program continues to receive information until it reaches the end of the file. At that time, the intermediate file is closed. The intermediate file is not sorted at this time unless *sortpage* has detected an End command. If a line is not a *sortpage* command (*sortpage* commands must be on a separate line from the other data lines), the line is put in a temporary file for later merging with sorted information.

**Process Command**
The Process Command Step is executed if a *sortpage* command is detected.

**Header Command**
If the Header command is detected, the sort file is opened and the lengths for each of the sort fields is defined.

**Value Command**
If the Value command is found, the value for each of the sort fields is defined for this record and placed in the sort field.

**End Command**
If the End command is reached, *sortpage* sorts the sort file and merges the temporary file with the sorted information, then passes it to the output file or device.

**Note:** If the end of the file has not been reached, sortpage continues to process until all lines have been read and processed.

## Program Relationships

The following programs use *sortpage*.

- The *adr* program output provides names, addresses and salutations to *sortpage*.
- The ACE report writer extracts data for *sortpage*.
- The *sortpage* program output serves as input to the *splitpage* script.
- The *ltbrun* script most commonly controls sortpage.

## Parameters

### Introduction

CX contains parameters for executing the *sortpage* program. You can specify parameters to compile *sortpage* in a specified manner at the time of execution.

### Parameter Syntax

You can display *sortpage* parameters by entering the following: **sortpage -,**

The following is the correct usage for running the *sortpage* program from the UNIX shell:

**sortpage [-b] [-d]**

Parameters that appear in brackets are optional.

### Parameters

The following lists the parameters for running *sortpage*.

**-b**
Optional - causes *sortpage* to sort addresses in accordance with bulkmail regulations. This option is no longer in use because of changes in bulk mailing postal regulations.

**-d**
Optional - causes *sortpage* to print debugging messages, a feature that helps correct errors in a client-generated process that creates the data stream.

## Macros

### Introduction

The standard CX product includes SRT macros that allow users to easily incorporate *sortpage* into any ACE report to be used for letter/label production. The macros are as follows:

- SRT_DEFINE
- SRT_SORT_BY
- SRT_HEADER
- SRT_VALUES
- SRT_SORTBREAK
- SRT_END

### SRT_DEFINE Macro

The SRT_DEFINE macro defines the variables needed for the ACE report to function using *sortpage*. The following is the expanded version of SRT_DEFINE:

```
function _dbtype
function _dbsize
variable _srt_type      type integer
variable _srt_long      type long
variable _srt_double    type double
```

If you use *sortpage*, SRT_DEFINE must be in the define section of ACE. The macros are structured so the SRT_DEFINE macro can be included at the top, even if you do not use *sortpage*. The key macro for inclusion is the following macro, SRT_SORT_BY.

### SRT_SORT_BY Macro

The SRT_SORT_BY macro indicates the fields *sortpage* must use in sorting. This macro may be used in addition to (or in place of) the ACE sort clause. The format of the macro is:

```
SRT_SORT_BY(VALUE1 [descending],...,VALUEn)
```

The optional word *descending* reverses the order; ascending order is the default. An example of using a SRT_SORT_BY clause is:

```
SRT_SORT_BY(ADR_ZIP_VALUE,ADR_NAME_VALUE)
```

The macro parameters ADR_ZIP_VALUE and ADR_NAME_VALUE are keywords to the SRT_SORT_BY macro. These are replaced by *adr* commands to include the named values from the output of *adr* on the ID number processed.

**Example:** The *adr* program is processing a record with the ID number of 100, and the zip code from the ID record of 45056. The *adr* program searches for and locates an alternate address for that ID and replaces the address with one with a zip code of 89000. The macro ADR_ZIP_VALUE will pass the zip code supplied by *adr*, 89000, instead of the home address zip code, 45056.

The SRT_SORT_BY macro is used by the SRT_HEADER and SRT_VALUES macros.

The *sortpage* macros within ACE are set up to automatically include everything needed by *sortpage* if the macro SRT_SORT_BY is found in the ACE report. Even if the macro is commented out, the sort information will still be included since the macro pre-processor does not check for ACE comments.

### SRT_HEADER Macro

The SRT_HEADER macro, automatically included with the LTR macros, expands to create the necessary first page header information needed by *sortpage*. The macro defines the number of sort fields and their lengths. This macro also begins the sorting process, which continues until the macro SRT_END is reached. The process can subsort within one ACE report output, by using multiple combinations of SRT_HEADER and SRT_END. The actual ACE commands generated are:

```
print "$$$", "fields", ":";
print "10", "", "";
print ",";
print "32", "", "";
print
print "$$$", "names", ":";
print "zip", "", "";
print ",";
print "name", "", "";
print
```

These commands produce the following output:

```
$$$fields:10,32
$$$names:zip,name
```

### SRT_VALUES Macro

The SRT_VALUES macro expands to put the actual values to pass to *sortpage*. Using the example of a sort by zip code and name, the macro expands to the following ACE commands:

```
print "$$$", "values", ":"
print "&&&","value",":","zip";
print
print "&&&","value",":","name";
print
```

which would produce the following output:

```
$$$values:
&&&value:zip
&&&value:name
```

The *adr* program interprets the &&&value and replaces it with the actual values passed back from *ADR*. In the zip code/name sort example, the output passed to *sortpage* is:

```
$$$values:
89000
Smith, John R.
```

### SRT_SORTBREAK macro

The SRT_SORTBREAK macro instructs *sortpage* that a group of data has ended. The data are sorted, then printed. Text from this point until the next values command is copied directly to standard output. The actual ACE commands generated are as follows:

print "$$$", "sortbreak", ":"

### SRT_END Macro

The SRT_END macro, also automatically included by LTR macros, marks the ending point of sorting.  Another SRT_HEADER macro can then be executed to start another sorting run, which could sort differently than the previous sort.  The actual ACE commands generated are:

print "$$$", "end", ":"

The commands generate the following output:

$$$end:

### Sample ACE Report Using *sortpage* Macros

The following is an example of an ACE report using SORTPAGE macros:

```
define
        .
        .
        SRT_DEFINE
    end
    .
    .
    read into a
    .
    .
    end

    SRT_SORT_BY(ADR_ZIP_VALUE,ADR_NAME_VALUE)

    format                   -------------
        first page header                |
            .                            |
            .                            |
            SRT_HEADER                   |
                                         | ----->  May be combined into one
        on every record                  |         macro for adr
            .                            |
            .                            |         LTB_FORMAT
            SRT_VALUES                   |
        .                                |
                            --------------
        on last record      --------------
            .                            |
            .                            |
            SRT_END                      | ----->  Combined into macro
            .                            |
            .                            |         LTB_LAST_REC
    end                     --------------
```

# SECTION 13 - SPLIT PAGE

## Overview

### Introduction

This section provides reference information about the *splitpage* program. The letter writing process uses *splitpage* to divide a data stream into several destination files. In the CX letter writing process, the script *ltbrun* calls the *splitpage* process to create the following substreams from the input data stream:

- Merge file information to *ltrformat*
- Labels information to *labels*
- SQL information to ISQL
- Bulk mail information to the Merge drawer in the users' FileCabinet

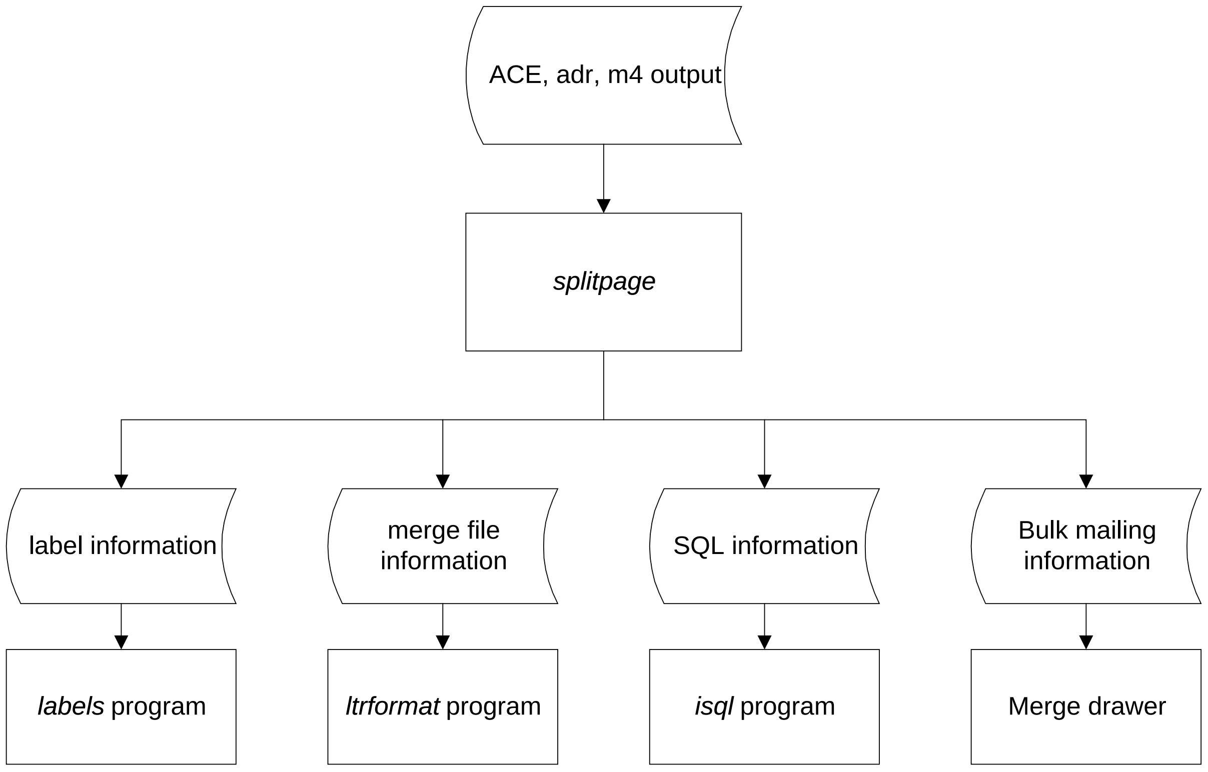
### Program Screens

Because *splitpage* is a filter process, it does not require any program screens.

## Process Flow

### Diagram

The following diagram shows the flow of data in the *splitpage* program.



### Data Flow Description

The *splitpage* program is a UNIX filter. It receives a data stream on its standard input. The following describes the data flow in the *splitpage* program.

1. The *splitpage* program accepts a data stream from its input sources (ACE, *adr* and m4).
2. The *splitpage* program processes the stream, delimiting the sections.
3. The outputs from *splitpage* route to the appropriate processes for further processing.

### Example of *splitpage* Data Flow

Assume a data stream in which most of the output relates to the program *to_process*. Embedded in this data stream, however, are sections that relate to the program *sub*. In this case, the data source, *from_process*, will delimit these special sections by single lines containing the characters %%%. For this example, assume the command line was:

```
from_process | splitpage +”%%%” subfilename | to_process
```

In this case *splitpage* splits the datastream from *from_process* into two substreams. The first, consisting of all lines of subtype, is directed into the file *subfilename*. The second, consisting of all lines of main type, will appear on the standard output of *splitpage* and according to the command line, will be passed to the standard input of *to_process*.

**Note:** For more information about delimiting characters, see *Parameters* in this section.

Before *splitpage* processes the data stream, it will resemble the following:

| Input Stream | Substream to *subfilename* | Substream to *to_process* |
|---|---|---|
| main type line 1 | subtype line 1 | main type line 1 |
| main type line 2 | subtype line 2 | main type line 2 |
| . | . | . |
| . | . | . |
| . | . | . |
| main type line *n* | subtype line m | main type line n |
| %%% | | main type line n+1 |
| sub type line 1 | | main type line n+2 |
| sub type line 2 | | |
| . | | |
| . | | |
| . | | |
| sub type line *m* | | |
| %%% | | |
| main type line *n*+1 | | |
| main type line *n*+2 | | |

Note that it is possible to have the substreams overlap in the input stream. Consider the following input stream (in *infile*).

```
line 1
line 2
%%%
line 3
line 4
-------
line 5
line 6
%%%
line 7
line 8
-------
line 9
line 10
```

If the command line to process this datastream was:

```
splitpage +”%%%” pfile.out +”---” dfile.out <infile > ofileout
```

then the contents of the three output files would be:

| ofile.out | pfile.out | dfile.out |
|---|---|---|
| line 1 | line 3 | line 5 |
| line 2 | line 4 | line 6 |
| line 9 | line 5 | line 7 |
| line 10 | line 6 | line 8 |

## Program Relationships

The script *ltbrun* controls the use of the *splitpage* program for letter writing.

## Parameters

### Introduction

CX contains parameters for executing *splitpage*. Ordinarily, *splitpage* is a process run under the control of the *ltbrun* script, and the *ltbrun* script directs how *splitpage* will work by determining the parameters passed to it. However, if you run *splitpage* manually, you specify the parameters that cause the program to process a datastream as desired.

### Parameter Syntax

The *splitpage* program accepts parameters in the following format:

**splitpage [-[c][k]] patt_parm_seq [patt_parm_seq ...] < input file   > output file**

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

The format where the call to splitpage is embedded in a pipeline in a script is:

```
from_process | splitpage [-[c][k]] patt_parm_seq  [patt_parm_seq ...]  | to_process
```

### Parameters

The following lists the parameters for running *splitpage*:

**-ck**

Optional - The parameters “c” and “k” may appear directly as shown above or may appear in one or more of the pattern parameter sequences. If they appear immediately after splitpage, then they have a global effect on all of the pattern parameter sequences. Either one or both “c” and “k” may appear in any permissible location.

The c (copy) parameter instructs *splitpage* to copy the contents of the designated substream to the standard out (output file). If the c parameter being considered is part of a pattern parameter sequence, then only the substream associated with that pattern is copied to standard out. If the c parameter is a global one, then all substreams will be copied to standard out. The default action is to not copy the substream output to standard out.

The k (keep) parameter instructs *splitpage* to keep the contents of the line containing the pattern . If the k parameter being considered is part of a pattern parameter sequence, then only the pattern(s) associated with that pattern parameter sequence are kept. If the k parameter is a global one, then all patterns will be kept and sent to the appropriate substream output. The default action is to not keep the pattern sequence lines in the input stream. One use of the k parameter is to keep the pattern lines when the pattern is part of the actual data instead of an arbitrary pattern inserted just to delimit a substream.

***patt_parm_seq***

Required. This is a pattern parameter sequence (i.e., a sequence of possible parameters for each desired substream). At least one pattern parameter sequence is required in the call to *splitpage*. The format of a pattern parameter sequence is:

```
+beg_patt  [-end_patt]  [-[c][k]]  [patt_file_name]
```

The parts of a pattern parameter sequence are:

**+beg_patt**

Required - This parameter is a string of characters up to 127 characters. This string delineates the beginning of the substream associated with this pattern. There is almost no restriction placed on this string by *splitpage*. However, there are problems associated with using some special characters in the command line to *splitpage* because of the special interpretation placed on them by the shell. This parameter is the only part of a pattern parameter sequence that is always required. The filename is only required in the last one.

There are three special cases for the formation of beg_patt. They involve special characters used to match the beginning and ending of an input line. These characters are currently: "^" for beginning match and "$" for ending match. These characters specify the location of the pattern on the line. If they are not used, the pattern will match no matter where it occurs on the input line. If the "^" is used to indicate a beginning of line match, it must be the first character of the string beg_patt. When used, it indicates the pattern is to cause a match only if it occurs as the first characters of the input line. If the "$" is used to indicate an end of line match, it must be the last character of beg_patt. When used, it indicates the pattern is to cause a match only if it occurs as the last characters of the input line. If either character is used in any position other than the indicated one, it has no special significance and acts as any other pattern character. Note that when used in these fashions, the actual pattern searched for does not include either character. Finally, both special characters may be used in the same beg_patt string. In this case the contained pattern must be the only characters on the line (excluding new line character) if a match is to occur.

**-end_patt**

Optional - This parameter is a string of characters up to 127 characters. This string delineates the end of the substream associated with this pattern. If this parameter is not present in a particular pattern parameter sequence, the end pattern is assumed to be the same as the beginning pattern. In the CX letter writing process only a +beg_patt parameter is used. The same considerations that apply to using a +beg_patt string apply to an -end_patt string.

**-ck**

Optional - This part of a pattern parameter sequence acts in the same manner as that described above except that it only applies to the substream determined by the pattern parameter sequence in which it occurs.

**patt_file_name**

Optional - This parameter is a valid system file name in which the output of the substream with which it is associated is stored. Two special considerations apply to this parameter. (1) If a given pattern parameter sequence contains no filename, the substream for this pattern will be directed to the filename in the next pattern parameter sequence which does contain a filename. This feature will not, however, cause a duplication of lines in a filename if the patterns for two substreams overlap. (2) The program correctly handles the output if the same filename is used in two pattern parameter sequences. Because of the ability to have more than one substream directed to a given filename, only the last pattern parameter sequence is required to have a filename.

***< input file***

Required - Indicates that the input to the splitpage is directed into the standard input of the program This may be done in two ways. The first as shown is by redirecting the input from a file containing the input data stream. The second is that splitpage may operate as a filter so the output of some other process supplies the data stream to the standard input of splitpage via a pipe.

***> output file***

Required - Indicates the location where the direct output from *splitpage* is to be directed. This output will be any lines in the input stream which do not belong to one of the substreams determined by the pattern parameter sequences, along with any lines in the

input stream which do belong to one of the substreams but for which there is an active c
parameter indicating that the substream is to be copied to the standard out.

# SECTION 14 - TICKLER ENTRY

## Overview

### Introduction

This section provides reference information about the Tickler Entry (*tickent*) program. The Communications Management product uses *tickent* to ease the entry and maintenance of Tickler tables. After users access *tickent* to create and maintain table entries, the Interactive Tickler (*tickler*) program maintains the status of individuals being monitored by that Tickler system.

### Program Features Detailed

This section contains details about the following features of the *tickent* program:

- Process flow
- Parameters
- Program screens

### Records and Tables Used

The *tickent* program uses the following records and tables:

**ctc_rec**
The Contact record that records the sending or receiving of communication with an individual or organization.

**ctc_table**
The Contact table that contains the code and description for the types of communication in which the institution engages (e.g., PHON, LABL, LTLB, or LETT).

**step_rec**
The Step record that defines the steps for a given tickler and track.

**step_table**
The Step table that defines the valid step codes for a given tickler.

**stepctc_rec**
The Step Contact record that defines the contacts to generate for a given step, track and tickler.

**stepobj_rec**
The Step Objective record that defines the objective for a particular step in a track within a defined tickler.

**stepreq_rec**
The Step Requirement record that defines the requirements to activate a step for a specified track and tickler.

**tick_rec**
The Tickler record that contains information required to place an individual or organization on a specific tickler.

**tick_table**
The Tickler table that defines the valid ticklers and the codes for each.

**trk_table**

The Track table that refers to alternate paths through the same tickler, and defines a code associated with the path.

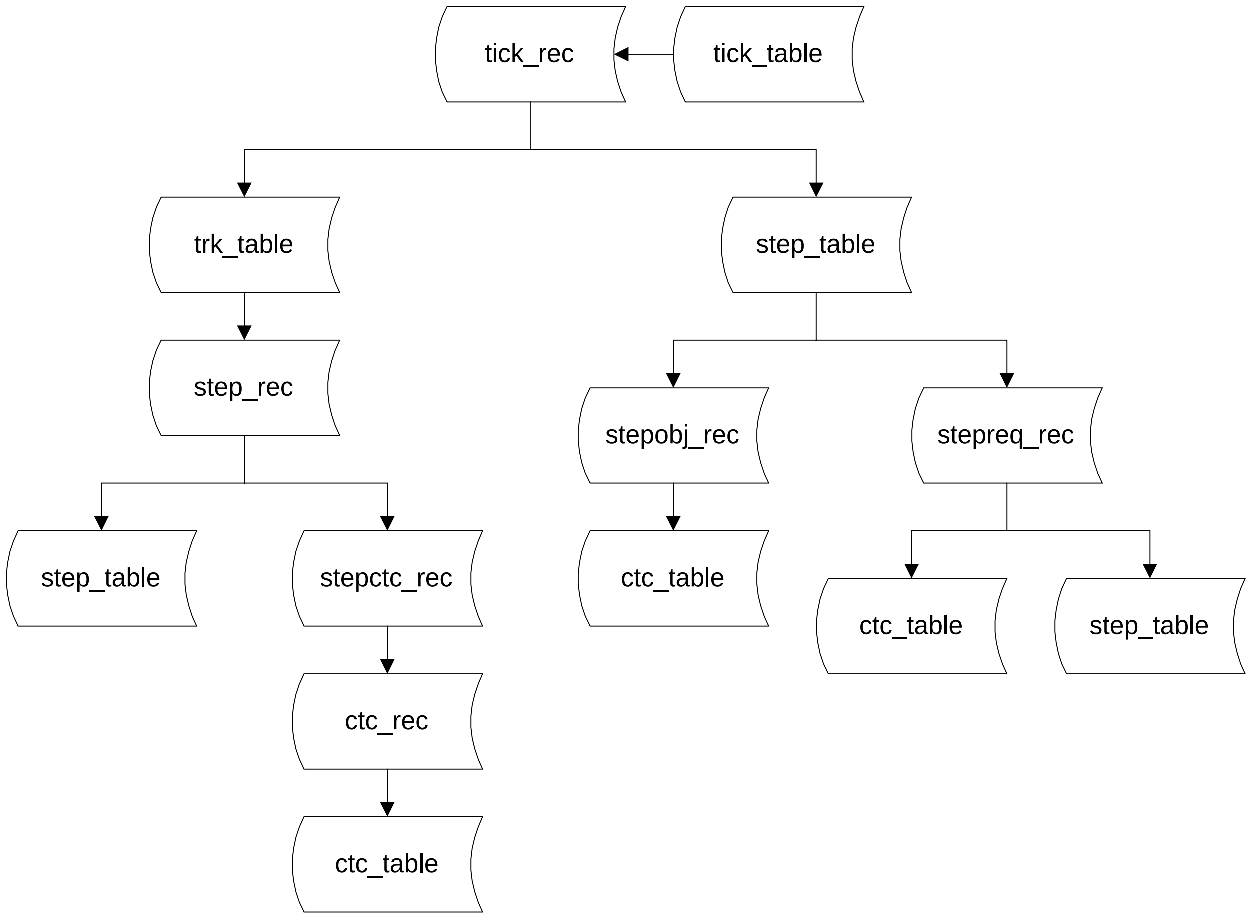
## Program Relationships

The *tickent* program updates the tables that define the institution's various Tickler strategies.  It therefore operates as a table entry program.  Data entered in *tickent* provides information to *tickler* about the timing and requirements for creating contacts.

## Data Interrelationships

### Diagram

The following diagram shows the interrelationship of the tables used in the *tickent* program.



## Parameters

### Introduction

CX contains parameters and compilation values for executing the *tickent* program. You can specify parameters to compile *tickent* in a specified manner at the time of execution.

**Note:** You can also specify compilation values with the includes for the Communications Management product that affect the *tickent* program.

### Parameter Syntax

You can display *tickent* parameters by entering the following: **tickent -,**

The following is the correct usage for running the *tickent* program from the UNIX shell:

**tickent [-f] [-d] [-D debug_level] [-t tick] [-r track_code]**

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

### Parameters

The following lists the parameters for running *tickent*.

**-f**
Optional - Causes the program to load quickly, skipping the consistency check of all Tickler systems.

**-D debug_level**
Optional - Sets the debugging level when modifying or maintaining the program. Valid values are:

- 1-2 echo[/pause] on delete
- 3-4 echo[/pause] on add/update tick: specify code of Tickler system to limit changes to track_code: specify track code to be used, default is “HS.”

**-t tick**
Optional - Specifies the specific Tickler system that you want to access (e.g., ADM).

**-r track_code**
Optional - Specifies the specific track that you want to access. If you use the -t parameter, you must also use the -r parameter.

## Program Screens

### Introduction

The *tickent* program uses a variety of screens. Many of the screens provide help for using *tickent*, and several others allow data entry.

### Access

The screen files are located in the following directory path:
$CARSPATH/modules/commgmt/progscr/tickent

### Screen Files and Table/Record Usage

The *tickent* program screens appear in the following files and use the indicated tables and records:

**Note:** Unless otherwise noted, the screen files appear in the directory $CARSPATH/modules/commgmt/progscr/tickent

**stepctc**
Contains the Step Contact screen.

*Tables/Records:* adr_table, comm_table, ctc_table, enr_stat_table, stepctc_rec, tick_table

**stepobj**
Contains the Step Object screen.

*Tables/Records:* comm_table, ctc_table, enr_stat_table, stepobj_rec, tick_table

**stepreq**
Contains the Step Requirement screen.

*Tables/Records:* adr_table, comm_table, ctc_table, enr_stat_table, stepreq_rec, tick_table

**stepsum**
Contains the Step Summary screen.

*Tables/Records:* ctc_table, step_rec, step_table, stepctc_rec, stepobj_rec, stepreq_rec, tick_table

**steptbl**
Contains the Tickler System Files screen.

*Tables/Records:* comm_table, ctc_table, step_rec, step_table, stepctc_rec, stepobj_rec, stepreq_rec, tick_table, trk_table

**Note:** The file for this screen appears in the directory $CARSPATH/modules/commgmt/screens

**tickstep**
Contains the Tickler Entry screen.

*Tables/Records:* step_rec, step_table, tick_table, trk_table

# SECTION 15 - TICKLER

## Overview

### Introduction

This section provides reference information about the Tickler (*tickler*) program. The Communications Management product uses *tickler* to create Contact records. The Contact records, in turn, generate letters or other mailing or correspondence according to a strategy that the institution defines in Tickler tables.

**Note:** Correct Tickler table setup is crucial to successful *tickler* program usage. For detailed examples and information about setting up Tickler tables, see *Customizing the Tickler Processes* in this manual.

### Program Features Detailed

This section contains details about the following features of the *tickler* program:

- Process flow
- Parameters
- Program screens

### Program Files

All the program files for *tickler* appear in the following directory:
$CARSPATH/src/commgmt/tickler

### Tables Used in the Program

The *tickler* program uses the following tables and records:

**ctc_rec**
The Contact record that tracks scheduled and completed communications.

**ctc_table**
The Contact table that defines valid types of communications.

**id_rec**
The ID record that provides name and address information for an individual or organization.

**step_rec**
The Step record that defines the characteristics of one step of a tickler track.

**step_table**
The Step table that defines each step of each tickler.

**stepctc_rec**
The Step Contact record that defines the contacts associated with a step of a tickler track.

**stepobj_rec**
The Step Objective record that defines the objective contact(s) of a tickler step, one of which must be met to complete the step.

**stepreq_rec**
The Step Requirement record that defines the requirements of a tickler step, all of which must be met before beginning the step.

**tick_rec**

The Tickler record that defines the appropriate tickler(s) for each ID.

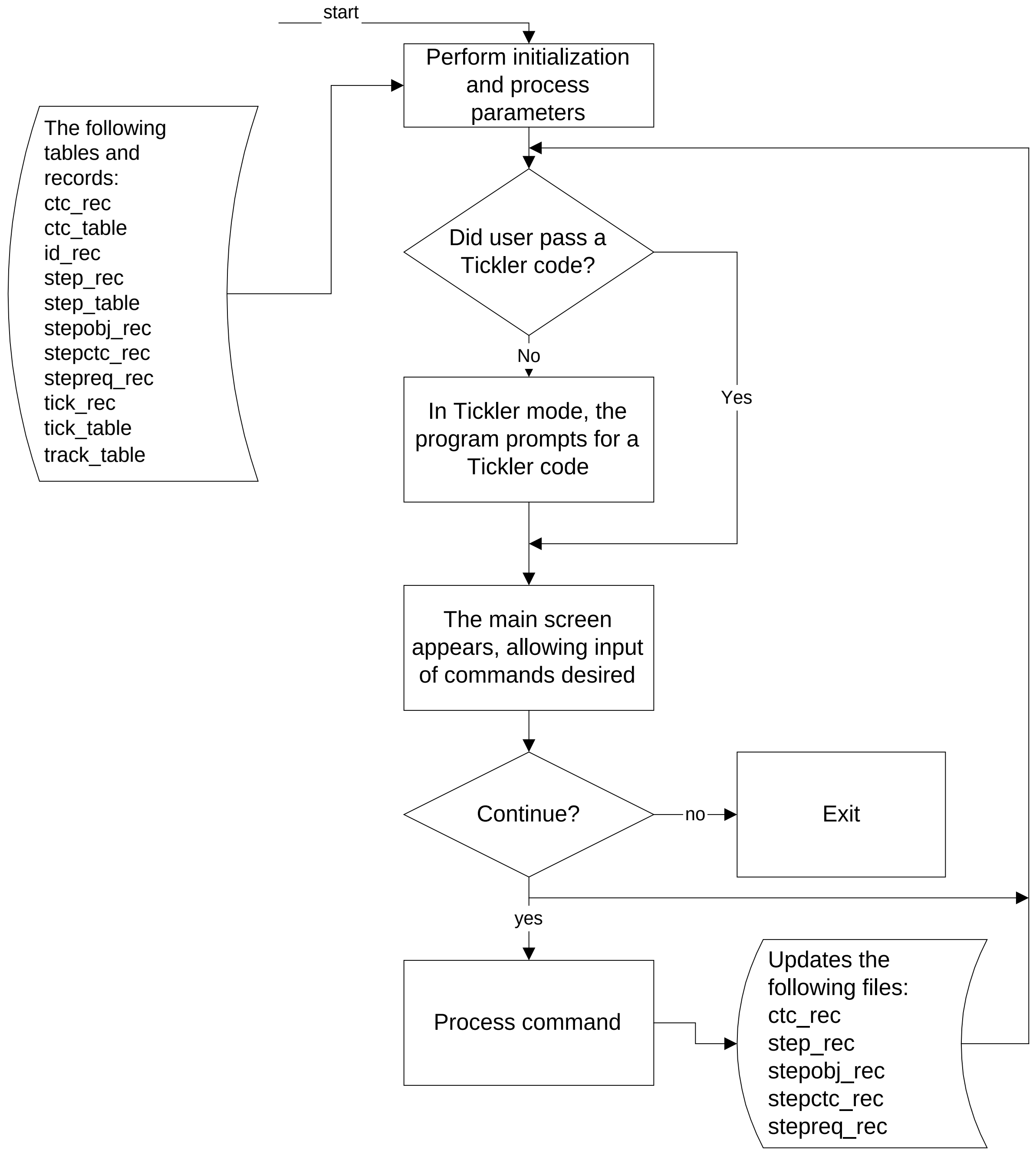
**tick_table**
The Tickler table that defines all the ticklers.

**trk_table**
The Track table that defines each track of each tickler.

## Process Flow

### Diagram

The following diagram shows the flow of data in the interactive portion of *tickler*.



start
Perform initialization and process parameters
The following tables and records: ctc_rec ctc_table id_rec step_rec step_table stepobj_rec stepctc_rec stepreq_rec tick_rec tick_table track_table
Did user pass a Tickler code?
No
Yes
In Tickler mode, the program prompts for a Tickler code
The main screen appears, allowing input of commands desired
Continue?
no
Exit
yes
Process command
Updates the following files: ctc_rec step_rec stepobj_rec stepctc_rec stepreq_rec

## Data Flow Description

The following describes the data flow in the *tickler* program.

1. The program performs initialization and processes any parameter passed to Tickler.
2. The program prompts you for the Tickler code to be used during this session. A valid Tickler code must be passed before you may access the next step.
3. From the main screen, the user selects a process command, and *tickler* executes the command.

## Program Relationships

The following programs use *tickler.*

- Table entries created in *tickent* serve as input to *tickler*.
- The *tickler* program updates Contact records that serve as input to ACE.

## Using Cron to Run Tickler Review

The *admprocess* script that runs *tickler* as a background process each night is located in $CARSPATH/modules/admit/scripts. The standard CX product is delivered with this process turned off. To activate the background running of *tickler*, remove the pound sign (#) before the BIN_PATH variable (i.e., BIN_PATH/admstats -p ADM_PROG_PROGRAM -t ADM_PROG_TICK).

## Parameters

### Introduction

CX contains parameters and compilation values for executing the *tickler* program. You can specify parameters to compile *tickler* in a specified manner at the time of execution.

**Note:** You can also specify compilation values with the includes for the Communications Management product that affect the *tickler* program.

### Parameter Syntax

You can display *tickler* parameters by entering the following: **tickler -,**

The following is the correct usage for running *tickler* from the UNIX shell:

```
tickler [-t tick] [-d today] [-r] [-o] [-a] [-f]
```

Parameters that appear in brackets are optional.

### Parameters

The following lists the parameters for running *tickler*.

**-t tick**
Optional - defines the tickler code that you want to process. If you do not enter a tickler code, *tickler* will select all codes.

**-d today**
Optional - specifies the date that you want to use as if it were the current day.

**-r**
Optional - causes *tickler* to review all individuals whose Next Review Date has arrived. This option is usually controlled by the Cron processor and runs as a background process every night. If you use the -r parameter, you must also use the -t tick parameter to specify a particular tickler system to review.

**-o**
Optional - creates an output of the Tickler strategy for the specified Tickler code. If you use the -o parameter, you must also use the -t tick parameter. The output goes to standard output (e.g., the terminal screen), but can be redirected to a file or piped to a printer if desired.

**-a**
Optional - enables you to alter the date for setup and testing.

**-f**
Optional - causes *tickler* to bypass consistency checks of your Tickler systems, enabling the program to load more quickly.

## Program Screens

### Introduction

The *tickler* program uses 65 screens for the features described in this section. Most of the screens (i.e., those with an “h” in the first position of the filename) contain help information for menu users. Information about the remaining screens appears in this section.

### Access

The screen files are located in the following directory path:
$CARSPATH/modules/commgmt/progscr/tickler

### Screen Files and Table/Record Usage

The *tickler* program screens appear in the following files and use the indicated tables and records:

**ctc**
Contains the Contact screen.

*Tables/Records:* ctc_table, ctc_rec, tick_table

**output**
Contains the screen on which users direct their output from *tickler*.

*Tables/Records:* none

**step**
Contains the Step screen.

*Tables/Records:* step_rec, step_table

**stepctc**
Contains the Contacts screen.

*Tables/Records:* ctc_rec, stepctc_rec

**stepdtl**
Contains the Step/Step Detail screen.

*Tables/Records:* step_rec, step_table

**stepobj**
Contains the Objectives screen.

*Tables/Records:* stepobj_rec

**stepreq**
Contains the Requirements screen.

*Tables/Records:* stepreq_rec

**tick**
Contains the Tickler screen.

*Tables/Records:* tick_rec, id_rec, tick_table, trk_table

# SECTION 16 - MENUS, SCREENS, SCRIPTS, AND REPORTS

## Overview

### Introduction

This section provides reference information on the following features of the Communications Management product:

- Menu source files
- Menu option files
- PERFORM screens
- SQL scripts
- Csh scripts
- ACE reports

### Directory Locations

The features detailed in this section are located in the following directory paths:

**Menu source files**
$CARSPATH/menusrc/utility
$CARSPATH/menusrc/admit
$CARSPATH/menusrc/instdev/alumni
$CARSPATH/menusrc/instdev/develop
$CARSPATH/menusrc/instdev/publicity

**Menu option files**
$CARSPATH/menuopt/utilities
$CARSPATH/menuopt/common
$CARSPATH/menuopt/commgmt

**PERFORM screens**
$CARSPATH/modules/commgmt/screens
$CARSPATH/modules/common/screens

**Scripts**
$CARSPATH/modules/util/scripts
$CARSPATH/modules/commgmt/scripts
$CARSPATH/modules/common/scripts

**ACE reports**
$CARSPATH/modules/common/reports
$CARSPATH/modules/admit/reports
$CARSPATH/modules/alumni/reports
$CARSPATH/modules/develop/reports
$CARSPATH/modules/finaid/reports
$CARSPATH/modules/finaudit/reports

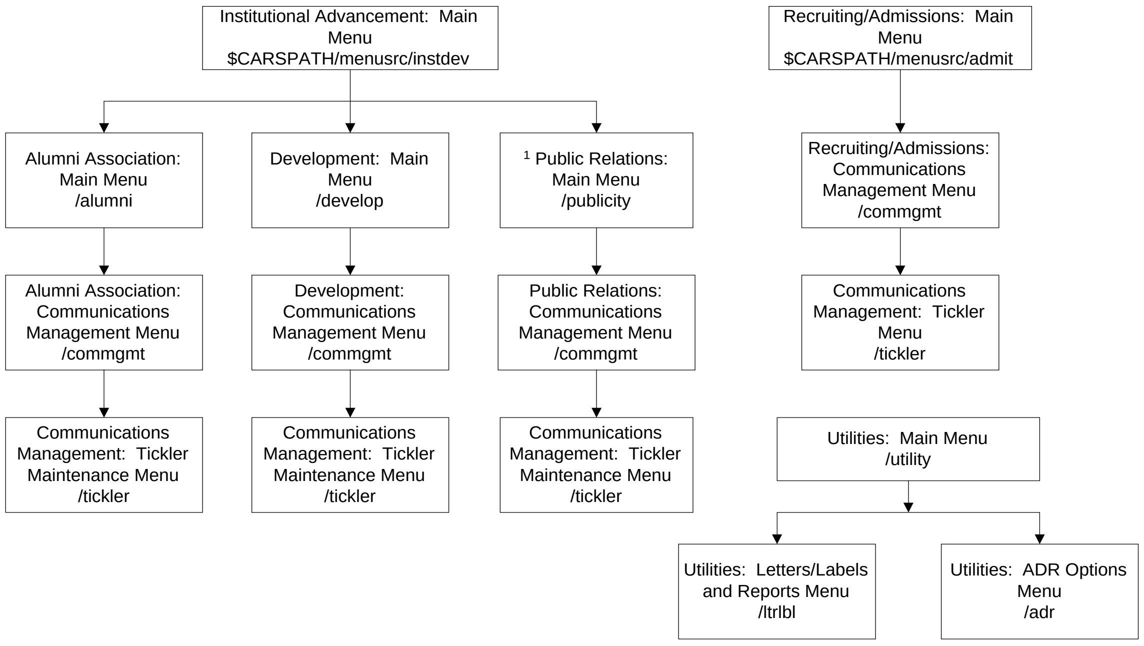
## Menu Structure

### Introduction

The CX menu source (menusrc) directory path contains definitions of the CX menu structure. Specifically, the $CARSPATH/menusrc/util, $CARSPATH/menusrc/instdev and $CARSPATH/menusrc/admit directory paths contain definitions for the Communications Management menus.

### Communications Management Menu Structure

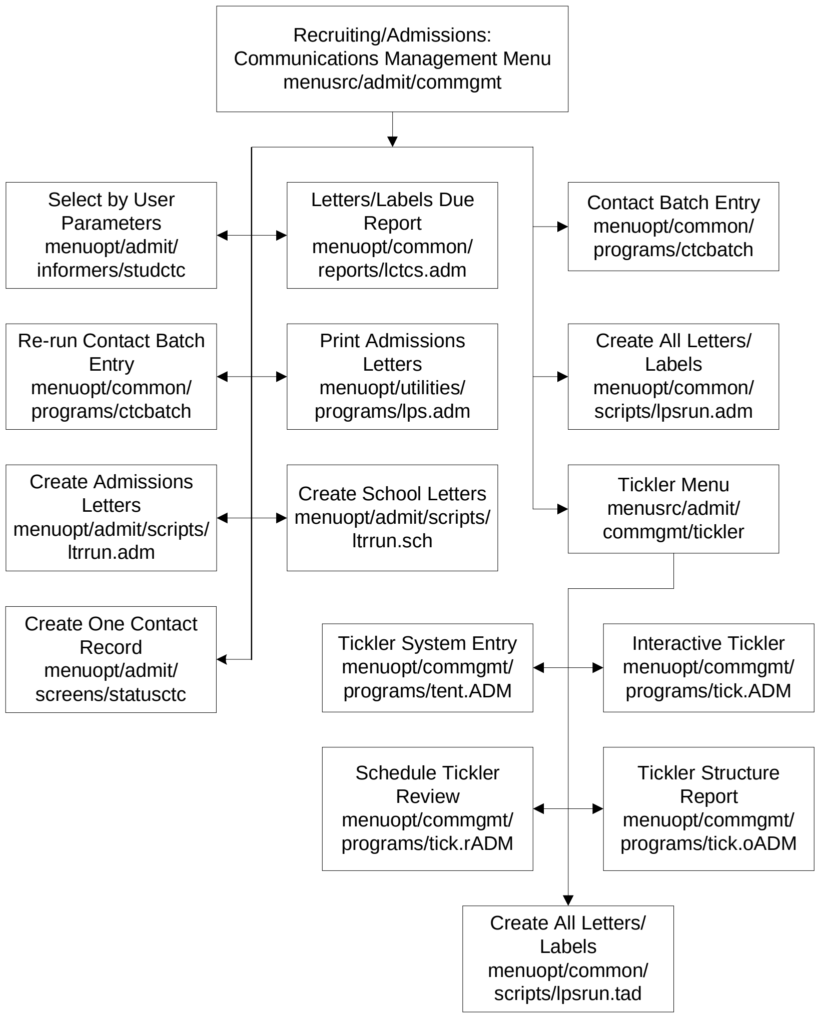
The following diagram illustrates the Communications Management menu structure and includes the corresponding menu source (menusrc) directory paths corresponding to the Communications Management submenus on CX. Note that Communications Management relates to several different CX product areas. Each directory and subdirectory contains a *menudesc* file, which specifies what menu options appear in a menu. Specific menu options, however, are defined in the menu option (menuopt) directory path. For more information, see *Menu Options* in this section.


Institutional Advancement: Main Menu $CARSPATH/menusrc/instdev
Recruiting/Admissions: Main Menu $CARSPATH/menusrc/admit
Alumni Association: Main Menu /alumni
Development: Main Menu /develop
1 Public Relations: Main Menu /publicity
Recruiting/Admissions: Communications Management Menu /commgmt
Alumni Association: Communications Management Menu /commgmt
Development: Communications Management Menu /commgmt
Public Relations: Communications Management Menu /commgmt
Communications Management: Tickler Menu /tickler
Communications Management: Tickler Maintenance Menu /tickler
Communications Management: Tickler Maintenance Menu /tickler
Communications Management: Tickler Maintenance Menu /tickler
Utilities: Main Menu /utility
Utilities: Letters/Labels and Reports Menu /ltrlbl
Utilities: ADR Options Menu /adr

```
Legend:
1.  This submenu is available if the macro ENABLE_FEAT_PR is set to Y.
```

### Admissions: Communications Management Menu Structure

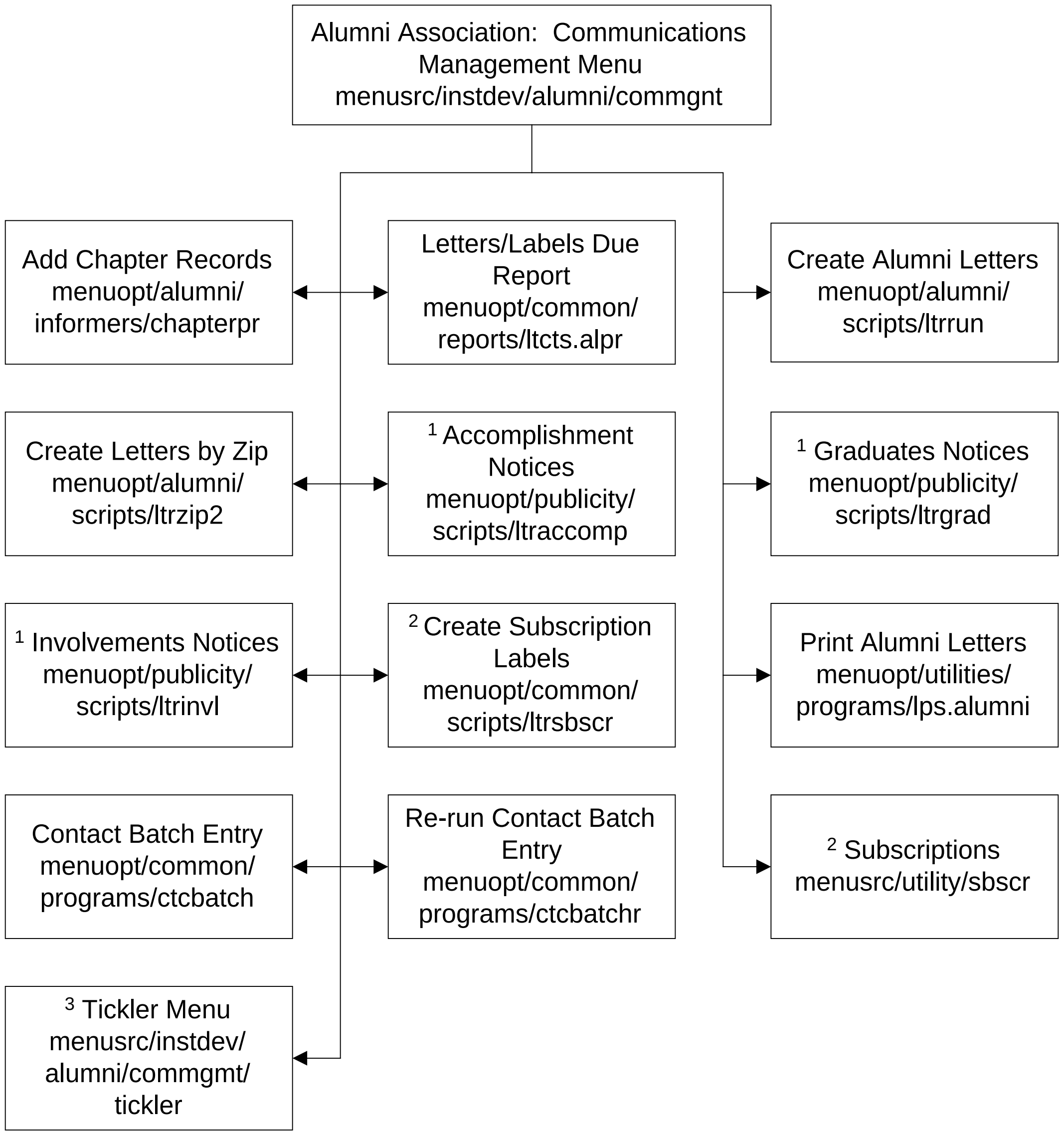
The following diagram illustrates the Admissions: Communications Management menu structure and includes the corresponding menu option (menuopt) directory paths for the programs, reports, and scripts it calls.


Recruiting/Admissions: Communications Management Menu menusrc/admit/commgmt
Select by User Parameters menuopt/admit/ informers/studctc
Letters/Labels Due Report menuopt/common/ reports/lctcs.adm
Contact Batch Entry menuopt/common/ programs/ctcbatch
Re-run Contact Batch Entry menuopt/common/ programs/ctcbatch
Print Admissions Letters menuopt/utilities/ programs/lps.adm
Create All Letters/ Labels menuopt/common/ scripts/lpsrun.adm
Create Admissions Letters menuopt/admit/scripts/ ltrrun.adm
Create School Letters menuopt/admit/scripts/ ltrrun.sch
Tickler Menu menusrc/admit/ commgmt/tickler
Create One Contact Record menuopt/admit/ screens/statusctc
Tickler System Entry menuopt/commgmt/ programs/tent.ADM
Interactive Tickler menuopt/commgmt/ programs/tick.ADM
Schedule Tickler Review menuopt/commgmt/ programs/tick.rADM
Tickler Structure Report menuopt/commgmt/ programs/tick.oADM
Create All Letters/ Labels menuopt/common/ scripts/lpsrun.tad

```
Legend:
1. This submenu is available if the macro ENABLE_FEAT_ADM_TICKLER is set to Y.
```

## Alumni Association:  Communications Management Menu Structure

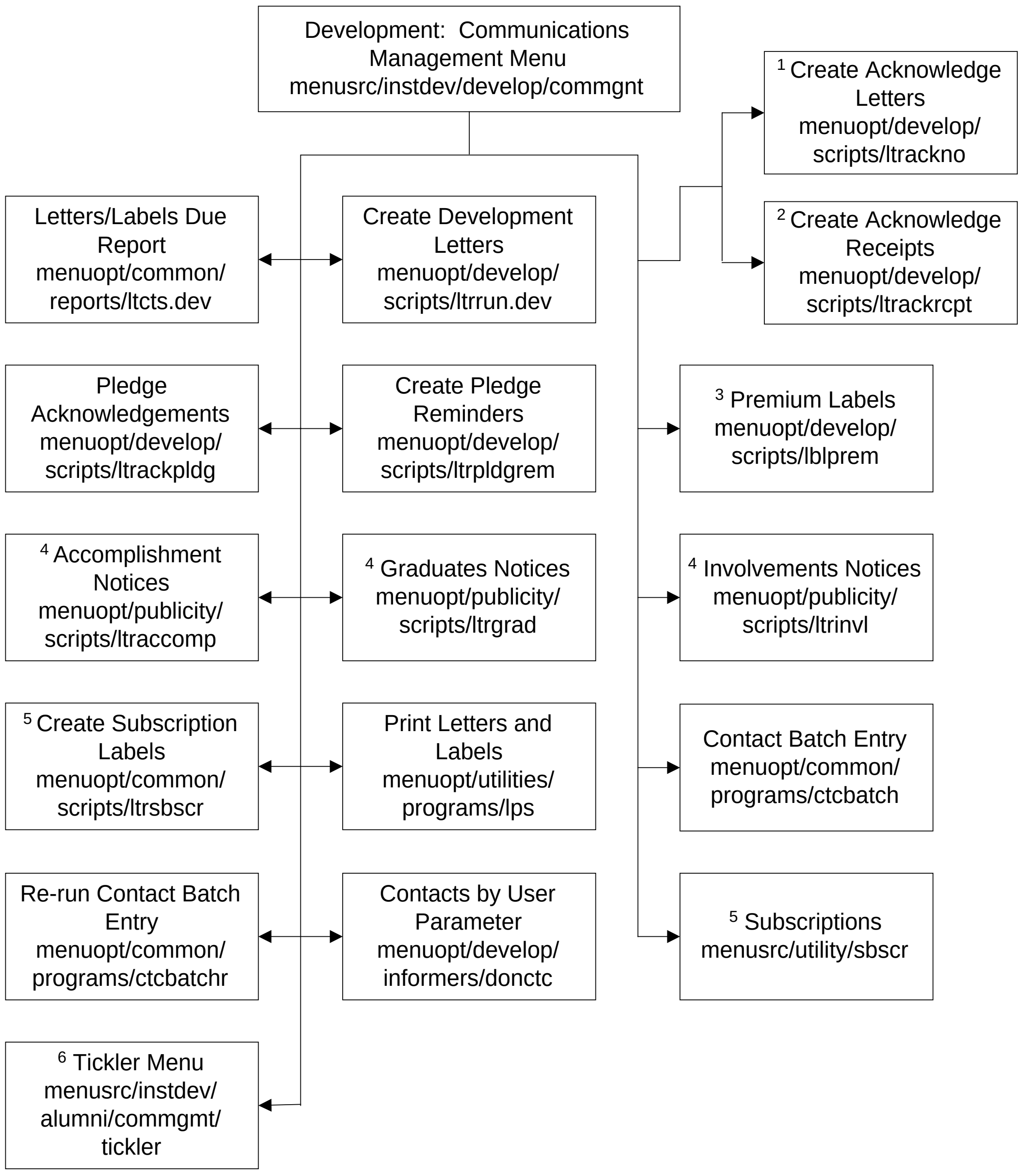
The following diagram illustrates the Alumni Association:  Communications Management menu structure and includes the corresponding menu option (menuopt) directory paths for the programs, reports, and scripts it calls.


Alumni Association: Communications Management Menu menusrc/instdev/alumni/commgnt
Add Chapter Records menuopt/alumni/ informers/chapterpr
Letters/Labels Due Report menuopt/common/ reports/ltcts.alpr
Create Alumni Letters menuopt/alumni/ scripts/ltrrun
Create Letters by Zip menuopt/alumni/ scripts/ltrzip2
1 Accomplishment Notices menuopt/publicity/ scripts/ltraccomp
1 Graduates Notices menuopt/publicity/ scripts/ltrgrad
1 Involvements Notices menuopt/publicity/ scripts/ltrinvl
2 Create Subscription Labels menuopt/common/ scripts/ltrsbscr
Print Alumni Letters menuopt/utilities/ programs/lps.alumni
Contact Batch Entry menuopt/common/ programs/ctcbatch
Re-run Contact Batch Entry menuopt/common/ programs/ctcbatchr
2 Subscriptions menusrc/utility/sbscr
3 Tickler Menu menusrc/instdev/ alumni/commgmt/ tickler

```
Legend:
1. This submenu is available if the macro ENABLE_FEAT_ALUMNI_PR is set to Y.
2. This submenu is available if the macro ENABLE_FEAT_ALUMNI_SUBSCR is set to Y.
3. This submenu is available if the macro ENABLE_FEAT_ALUMNI_TICKLER is set to Y.
```

**Development: Communications Management Menu Structure**

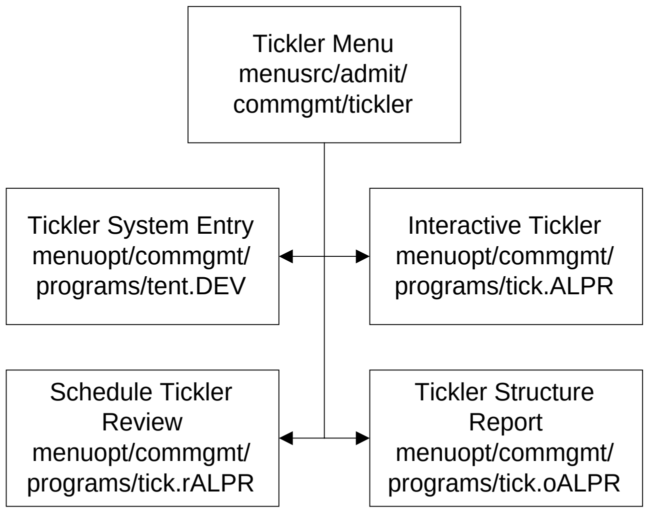
The following diagram illustrates the Development: Communications Management menu structure and includes the corresponding menu option (menuopt) directory paths for the programs, reports, and scripts it calls.

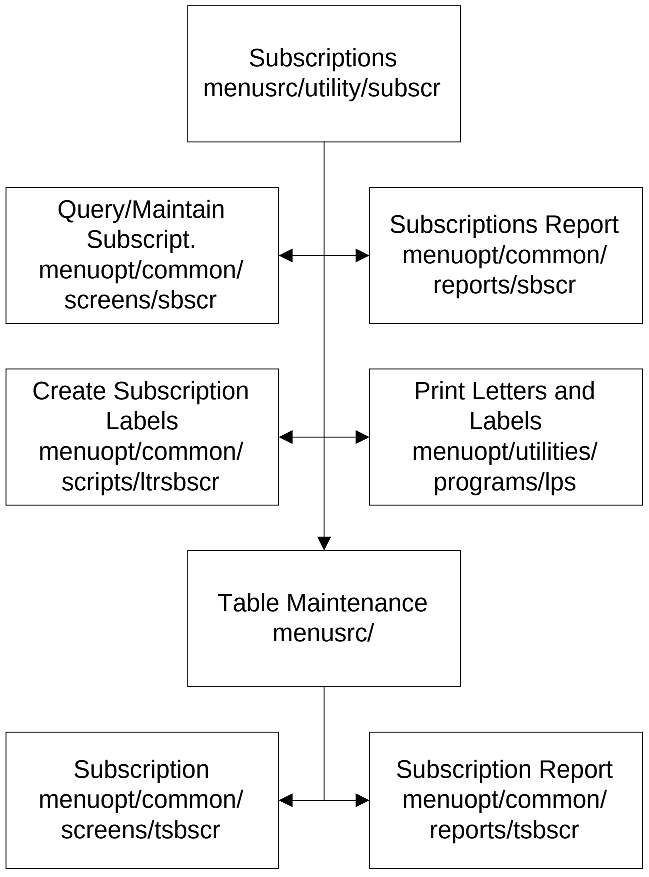

Development: Communications Management Menu menusrc/instdev/develop/commgnt
[1] Create Acknowledge Letters menuopt/develop/scripts/ltrackno
[2] Create Acknowledge Receipts menuopt/develop/scripts/ltrackrcpt
Letters/Labels Due Report menuopt/common/reports/ltcts.dev
Create Development Letters menuopt/develop/scripts/ltrrun.dev
Pledge Acknowledgements menuopt/develop/scripts/ltrackpldg
Create Pledge Reminders menuopt/develop/scripts/ltrpldgrem
[3] Premium Labels menuopt/develop/scripts/lblprem
[4] Accomplishment Notices menuopt/publicity/scripts/ltraccomp
[4] Graduates Notices menuopt/publicity/scripts/ltrgrad
[4] Involvements Notices menuopt/publicity/scripts/ltrinvl
[5] Create Subscription Labels menuopt/common/scripts/ltrsbscr
Print Letters and Labels menuopt/utilities/programs/lps
Contact Batch Entry menuopt/common/programs/ctcbatch
Re-run Contact Batch Entry menuopt/common/programs/ctcbatchr
Contacts by User Parameter menuopt/develop/informers/donctc
[5] Subscriptions menusrc/utility/sbscr
[6] Tickler Menu menusrc/instdev/alumni/commgmt/tickler

```
Legend:
1. This option is available if the macro ENABLE_FEAT_RECEIPT_LETTERS is set to N.
2. This option is available if the macro ENABLE_FEAT_RECEIPT_LETTERS is set to Y.
3. This option is available if the macro ENABLE_FEAT_CTC_PREMIUMS is set to Y.
4. This option is available if the macro ENABLE_FEAT_DEVELOP_PR is set to Y.
5. This submenu is available if the macro ENABLE_FEAT_DEVELOP_SUBSCR is set to Y.
6. This submenu is available if the macro ENABLE_FEAT_DEVELOP_TICKLER is set to Y.
```

## Alumni Association, Development and Utilities Communications Management Submenu Structure

The following diagram illustrates the submenus from both the Alumni Association/Communications Management and Development/Communications Management menus and includes the corresponding menu option (menuopt) directory paths for the programs, reports, and scripts it calls.



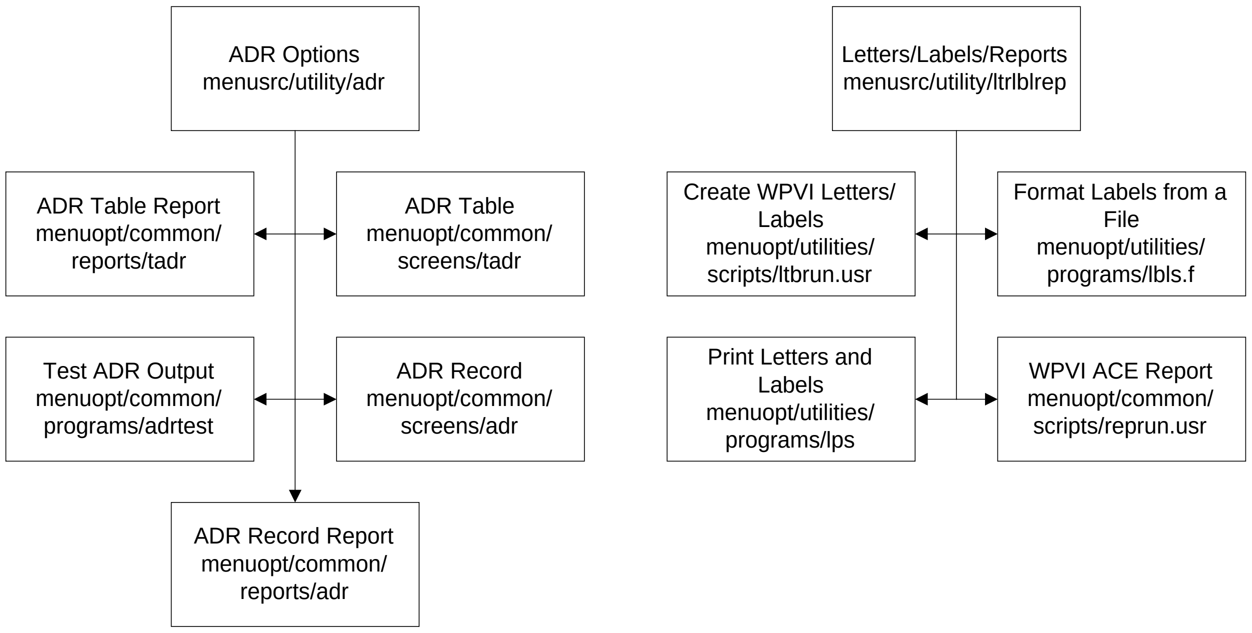


**Utilities: Communications Management Menu Structure**

The Utilities: Main Menu offers access to a variety of system-wide features. The following three menu options relate to Communications Management:

- ADR Options
- Letters/Labels/Reports
- Subscriptions

The following diagram illustrates the ADR Options and Letters/Labels/Reports submenus, and includes the corresponding menu option (menuopt) directory paths for the programs, reports, and scripts they call.  For the Subscriptions menu structure, see *Alumni Association, Development and Utilities Communications Management Submenu Structure* in this section.



## Menu Options

### Programs, Screens, and Scripts

The following list associates each Communications Management program, screen, and script menu option and corresponding menuopt file and identifies the menuopt locations and what the menu option accesses.

**Note:** The following menus and options are listed in the order in which they appear on the standard CX menu structure. Italicized parameters indicate those that a user can enter or change.

**Recruiting/Admissions: Communications Management Menu**

**Select by User Parameters**

*Accesses:* $CARSPATH/modules/admit/informers/studctc

*File:* $CARSPATH/menuopt/admit/informers/studctc

*Parameters Passed:*

- *-DPROG (program)*
- *-DSESSION (session)*
- *-DYEAR (year)*
- *-DENRYR (enrollment year)*
- *-DMAJOR (major)*
- *-DADMCL (classification)*
- *-DSTATUS (enrollment status)*
- *-DADMDEC (admittance decision)*
- *-DHSGTYPE (housing preference)*
- *-DCOUNSELOR (counselor)*
- *-DSCH_ID (school ID)*
- *-DSTATE (state)*
- *-DCOUNTY (county)*
- *-DZIP1 (beginning zip code)*
- *-DZIP2 (ending zip code)*
- *-DCOUNTRY (country)*
- *-DSEX (sex)*
- *-DETHNIC (ethnic code)*
- *-DDENOM (religious affiliation)*
- *-DACCOMP (accomplishment code)*
- *-DINTEREST (interest code)*
- *-DINVOLVE (involvement code)*
- *-DCONTACT1 (contact code the student has received)*
- *-DCONTACT2 (contact code the student has not received)*
- *-DTICKLER (tickler code)*
- *-DDUEDATE (contact due date)*
- *-DRESOURCE (contact resource code)*

**Create One Contact Record**

*Accesses:* $CARSPATH/modules/admit/screens/statusctc

*Menuopt File:* $CARSPATH/menuopt/admit/screens/statusctc

*Parameters Passed:* *statusctc* form

**Contact Batch Entry**

*Accesses:* $CARSPATH/src/common/ctcbatch

*Menuopt File:* $CARSPATH/menuopt/common/programs/ctcbatch

*Parameters Passed:*
- -t tickler (tickler code)
- -r resource (letter name)
- -s status (resource status; default is (E)xpected)
- -d due date (date contact to complete)
- -f file (name of file containing IDs for which contacts are to be created)

**Re-run Contact Batch Entry**

*Accesses:* $CARSPATH/src/common/ctcbatch

*Menuopt File:* $CARSPATH/menuopt/common/programs/ctcbatchr

*Parameters Passed:* -R rerun option

**Create All Letters/Labels**

*Accesses:* $CARSPATH/modules/common/scripts/lpsrun

*Menuopt File:* $CARSPATH/menuopt/common/scripts/lpsrun.adm

*Parameters Passed:*
- Tickler code for admissions (ADM)
- Drawer location (admit)
- Letter date
- Selection date

**Create Admissions Letters**

*Accesses:* $CARSPATH/modules/common/scripts/ltbrun

*Menuopt File:* $CARSPATH/menuopt/admit/scripts/ltrrun.adm

*Parameters Passed:*
- Letters/Labels/Both/None (desired output)
- Tickler (tickler track)
- Report (ACE report to use for data extraction)
- Resource (name of letter or contact)
- Date (date to print on letters)
- Date (contact selection date)
- Format (file format for output (e.g., Standard LPS, WordPerfect, or Rich Text Format)
- Bulk mail (default is N)

**Create School Letters**

*Accesses:* $CARSPATH/modules/common/scripts/ltbrun

*Menuopt File:* $CARSPATH/menuopt/admit/scripts/ltrrun.sch

*Parameters Passed:*
- Letters/Labels/Both/None (desired output)

- Tickler (tickler track)
- Resource (name of letter or contact)
- Date (date to print on letters)
- Due Date (contact selection date)
- Format (file format for output [e.g., Standard LPS, WordPerfect, or Rich Text Format])
- Bulk mail (default is N)
- Relationship Code (code used to identify a relationship between a counselor and a a school)
- Default Addressee (phrase to use instead of counselor name if a name is not available [e.g., Guidance Counselor])

### Print Admissions Letters

*Accesses:* $CARSPATH/src/util/lps

*Menuopt File:* $CARSPATH/menuopt/utilities/programs/lps.adm

*Parameters Passed:* none

## Communications Management: Tickler Menu

### Tickler System Entry

*Accesses:* $CARSPATH/src/commgmt/tickent

*Menuopt File:* $CARSPATH/menuopt/commgmt/programs/tent.ADM

*Parameters Passed:* -t tick (tickler code)

### Interactive Tickler

*Accesses:* $CARSPATH/src/commgmt/tickler

*Menuopt File:* $CARSPATH/menuopt/commgmt/programs/tick.ADM

*Parameters Passed:* -t tick (tickler code)

### Schedule Tickler Review

*Accesses:* $CARSPATH/src/commgmt/tickler

*Menuopt File:* $CARSPATH/menuopt/commgmt/programs/tick.rADM

*Parameters Passed:*
- -r (review mode)
- -t tick (tickler code)

### Tickler Structure Report

*Accesses:* $CARSPATH/src/commgmt/tickler

*Menuopt File:* $CARSPATH/menuopt/commgmt/programs/tick.oADM

*Parameters Passed:*
- -o (output)
- -t tick (tickler code)

### Create All Letters/Labels

*Accesses:* $CARSPATH/modules/common/scripts/lpsrun

*Menuopt File:* $CARSPATH/menuopt/common/scripts/lpsrun.tad

*Parameters Passed:*
- Tickler code for admissions (ADM)
- Drawer location (admit)
- Letter date
- Selection date

## Alumni Association: Communications Management Menu

### Add Chapter Records

*Accesses:* $CARSPATH/modules/alumni/informers/chapterpr

*Menuopt File:* $CARSPATH/menuopt/alumni/informers/chapterpr

*Parameters Passed:*
- -DCHAPTER *chapno* (chapter number visited)
- -DVISITDATE *today* (date of visit; defaults to current date)

### Create Alumni Letters

*Accesses:* $CARSPATH/modules/common/scripts/ltbrun

*Menuopt File:* $CARSPATH/menuopt/alumni/scripts/ltrrun

*Parameters Passed:*
- *PP_LTB_SELECT (letters, labels, both or none)*
- *PP_CTC_RESRC (drawer location alumni/ltralum)*
- *PP_DATE (date to print on letters*
- *PP_DATE (contact selection date)*
- *Letter processing type (stdlps)*
- *Bulk mail flag (default is N)*

### Create Letters by Zip

*Menuopt File:* $CARSPATH/menuopt/alumni/scripts/ltrzip2

*Parameters Passed:*
- *PP_LTB_SELECT (letters, labels, both or none)*
- *PP_CTC_RESRC (resource code)*
- *PP_DATE (date to print on letters)*
- *PP_DATE (contact selection date)*
- *PP_ZIP_RAN (beginning zip code of range)*
- *PP_ZIP_RAN (ending zip code of range)*
- *PP_RUNCODE (runcode to use for letters)*

### Accomplishment Notices

*Accesses:* $CARSPATH/modules/common/scripts/ltbrun

*Menuopt File:* $CARSPATH/menuopt/publicity/scripts/ltraccomp

*Parameters Passed:*
- *PP_LTB_SELECT (letters, labels, both or none)*
- *PP_DATE (date to print on letters)*
- *PP_ACCOM (accomplishment code)*
- *PP_SESS (session)*
- *PP_YEAR (year)*

- *PP_RUNCODE (runcode to use for letters)*

### Graduates Notices

*Accesses:* $CARSPATH/modules/commgmt/scripts/ltbrun

*Menuopt File:* $CARSPATH/menuopt/publicity/scripts/ltrgrad

*Parameters Passed:*
- *PP_LTB_SELECT (letters, labels, both or none)*
- *PP_DATE (date to print on letters)*
- *PP_SESS (session)*
- *PP_ACAD_YR (academic year of graduation)*
- *PP_RUNCODE (runcode to use for letters)*

### Involvements Notices

*Accesses:* $CARSPATH/modules/commgmt/scripts/ltbrun

*Menuopt File:* $CARSPATH/menuopt/publicity/scripts/ltrinvl

*Parameters Passed:*
- *PP_LTB_SELECT (letters, labels, both or none)*
- *PP_DATE (date to print on letters)*
- *PP_DATE (contact selection date)*
- *PP_INV (involvement code)*
- *PP_RUNCODE (runcode to use for letters)*

### Create Subscription Labels

*Accesses:* $CARSPATH/modules/common/scripts/ltbrun

*Menuopt File:* $CARSPATH/menuopt/common/scripts/ltrsbscr

*Parameters Passed:*
- *PP_LTB_SELECT (letters, labels, both or none)*
- *PP_SUBSCRIP (subscription code)*
- *PP_DATE (date to print on letters)*
- *PP_DATE (date for subscription and alternate address verification)*
- *PP_FILECABINET (file cabinet name for optional letter)*

### Print Alumni Letters

*Accesses:* $CARSPATH/src/util/lps

*Menuopt File:* $CARSPATH/menuopt/utilities/programs/lps.alumni

*Parameters Passed:* none

### Contact Batch Entry

*Accesses:* $CARSPATH/src/common/ctcbatch

*Menuopt File:* $CARSPATH/menuopt/common/programs/ctcbatch

*Parameters Passed:*
- *PP_TICK (tickler code)*
- *PP_CTC_RESRC (resource code)*
- *PP_STATUS (contact status)*
- *PP_DATE_DUE (contact due date)*

- *PP_FILE (file location for IDs)*

### Re-run Contact Batch Entry

*Accesses:* $CARSPATH/src/common/ctcbatch

*Menuopt File:* $CARSPATH/menuopt/common/programs/ctcbatchr

*Parameters Passed:* -R Rerun option

## Tickler Menu

### Tickler System Entry

*Accesses:* $CARSPATH/src/commgmt/tickent

*Menuopt File:* $CARSPATH/menuopt/commgmt/programs/tent.DEV

*Parameters Passed:* -t (tickler code)

### Interactive Tickler

*Accesses:* $CARSPATH/src/commgmt/tickler

*Menuopt File:* $CARSPATH/menuopt/commgmt/programs/tick.ALPR

*Parameters Passed:* -t (tickler code)

### Schedule Tickler Review

*Accesses:* $CARSPATH/src/commgmt/tickler

*Menuopt File:* $CARSPATH/menuopt/commgmt/programs/tick.rALPR

*Parameters Passed:*
- -r (review ticklers for Next Review Date)
- -t tick (tickler code)

### Tickler Structure Report

*Accesses:* $CARSPATH/src/commgmt/tickler

*Menuopt File:* $CARSPATH/menuopt/commgmt/programs/tick.oALPR

*Parameters Passed:*
- -o (output)
- -t tick (tickler code)

## Subscriptions Menu

### Query/Maintain Subscript.

*Accesses:* $CARSPATH/modules/common/screens/sbscr

*Menuopt File:* $CARSPATH/menuopt/common/screens/sbscr

*Parameters Passed:* none

### Create Subscription Labels

*Accesses:* $CARSPATH/modules/common/scripts/ltbrun

*Menuopt File:* $CARSPATH/menuopt/common/scripts/ltrsbscr

*Parameters Passed:*
- *PP_LTB_SELECT (letters, labels, both or none)*
- *PP_SUBSCRIP (subscription code)*
- *PP_DATE (date to print on letters)*
- *PP_DATE (date for subscription and alternate address verification)*
- *PP_FILECABINET (file cabinet name for optional letter)*

### Print Letters and Labels

*Accesses:* $CARSPATH/src/utilities/lps

*Menuopt File:* $CARSPATH/menuopt/utilities/programs/lps

*Parameters Passed:* none

## Table Maintenance (Subscriptions)

### Subscription

*Accesses:* $CARSPATH/ modules/common/screens/tsbscr

*Menuopt File:* $CARSPATH/menuopt/common/screens/tsbscr

*Parameters Passed:* none

## Development: Communications Management Menu

### Create Development Letters

*Accesses:* $CARSPATH/src/utilities/lps

*Menuopt File:* $CARSPATH/menuopt/develop/scripts/ltrrun.dev

*Parameters Passed:*
- *PP_LTB_SELECT (letters, labels, both or none)*
- *PP_CTC_RESRC (resource code)*
- *PP_DATE (date to print on letters)*
- *PP_DATE (contact selection date)*

### Create Acknowledge Letters

*Accesses:* $CARSPATH/modules/common/scripts

*Menuopt File:* $CARSPATH/menuopt/develop/scripts/ltrackno

*Parameters Passed:*
- *PP_LTB_SELECT (letters, labels, both or none)*
- *PP_CTC_RESRC (resource code)*
- *PP_DATE (date to print on letters)*
- *PP_DATE (contact selection date)*
- *PP_GRPNO (donor accounting group number)*

**Note:** Depending on your macro settings, either this option or the next option will appear on the institution's menus.

### Create Acknowledge Receipts

*Accesses:* $CARSPATH/src/utilities/fps

*Menuopt File:* $CARSPATH/menuopt/develop/scripts/ltrackrcpt

*Parameters Passed:*
- *PP_LTB_SELECT (letters, labels, both or none)*
- *PP_CTC_RESRC (resource code)*
- *PP_DATE (date to print on letters)*
- *PP_DATE (contact selection date)*
- *PP_GRPNO (beginning donor accounting group number)*
- *PP_GRPNO (ending donor accounting group number)*

**Note:** Depending on your macro settings, either this option or the previous option will appear on the institution's menus.

### Pledge Acknowledgments

*Accesses:* $CARSPATH/modules/common/scripts/ltbrun

*Menuopt File:* $CARSPATH/develop/scripts/ltrackpldg

*Parameters Passed:*
- *PP_LTB_SELECT (letters, labels, both or none)*
- *PP_CTC_RESRC (resource code)*
- *PP_DATE (date to print on letters)*
- *PP_DATE (contact selection date)*

### Create Pledge Reminders

*Menuopt File:* $CARSPATH/develop/scripts/ltrpldgrem

*Parameters Passed:*
- *PP_SELECT (letters, labels, both or none)*
- *Letter Resource (resource code)*
- *PP_DATE (date to print on letters)*
- *Due Date (date against which payments due are analyzed)*
- *Next Due Date (next due date)*
- *PP_RUNCODE (runcode to use for letters)*
- *Pledge Period (pledge reminder period code)*

### Premium Labels

*Accesses:* $CARSPATH/modules/common/scripts/ltbrun

*Menuopt File:* $CARSPATH/menuopt/develop/scripts/lblprem

*Parameters Passed:*
- *PP_LTB_SELECT (letters, labels, both or none)*
- *PP_CTC_RESRC (resource code)*
- *PP_DATE (date to print on letters)*
- *PP_DATE (contact selection date)*
- *PP_PREM (premium code)*

### Accomplishment Notices

*Accesses:* $CARSPATH/modules/common/scripts/ltbrun

*Menuopt File:* $CARSPATH/menuopt/publicity/scripts/ltraccomp

*Parameters Passed:*
- *PP_LTB_SELECT (letters, labels, both or none)*

- *PP_DATE (date to print on letters)*
- *PP_ACCOMP (accomplishment code)*
- *PP_SESS (session)*
- *PP_YEAR (year)*
- *PP_RUNCODE (runcode to use for letters)*

**Graduates Notices**

*Accesses:* $CARSPATH/modules/common/scripts/ltbrun

*Menuopt File:* $CARSPATH/menuopt/publicity/scripts/ltrgrad

*Parameters Passed:*

- *PP_LTB_SELECT (letters, labels, both or none)*
- *PP_DATE (date to print on letters)*
- *PP_SESS (session)*
- *PP_ACAD_YR (school year)*
- *PP_RUNCODE (runcode to use for letters)*

**Involvements Notices**

*Accesses:* $CARSPATH/modules/common/scripts/ltbrun

*Menuopt File:* $CARSPATH/menuopt/publicity/scripts/ltrinvl

*Parameters Passed:*

- *PP_LTB_SELECT (letters, labels, both or none)*
- *PP_DATE (date to print on letters)*
- *PP_DATE (selection date)*
- *PP_INV (involvement code)*
- *PP_RUNCODE (runcode to use for letters)*

**Create Subscription Labels**

*Accesses:* $CARSPATH/modules/common/scripts/ltbrun

*Menuopt File:* $CARSPATH/menuopt/common/scripts/ltrsbscr

*Parameters Passed:*

- *PP_LTB_SELECT (letters, labels, both or none)*
- *PP_SUBSCRIP (subscription code)*
- *PP_DATE (date to print on letters)*
- *PP_DATE (date for subscription and alternate address verification)*
- *PP_FILECABINET (file cabinet name for optional letter)*

**Print Letters and Labels**

*Accesses:* $CARSPATH/src/util/lps

*Menuopt File:* $CARSPATH/menuopt/utilities/programs/lps

*Parameters Passed:* none

**Contact Batch Entry**

*Accesses:* $CARSPATH/src/common/ctcbatch

*Menuopt File:* $CARSPATH/menuopt/common/programs/ctcbatch

*Parameters Passed:*

- tickler (tickler code)
- resource (letter name)
- status (resource status; default is (E)xpected)
- due date (date contact to complete)
- file (name of file containing IDs for which contacts are to be created)

#### Re-run Contact Batch Entry

*Accesses:* $CARSPATH/src/common/ctcbatch

*Menuopt File:* $CARSPATH/menuopt/common/programs/ctcbatchr

*Parameters Passed:*
- -R Rerun option

#### Contacts by User Parameter

*Accesses:* $CARSPATH/modules/develop/scripts/donctc

*Menuopt File:* $CARSPATH/menuopt/develop/informers/donctc

*Parameters Passed:*
- -DCONSSTAT1 (constituent to process)
- -DCONSSTAT2 (constituent to process)
- -DCONSSTAT3 (constituent to process)
- -DCONSSTAT4 (constituent to process)
- -DLTGIVING1 (minimum lifetime giving)
- -DLTGIVING2 (maximum lifetime giving)
- -DYRGIVING1 (minimum yearly giving)
- -DYRGIVING2 (maximum yearly giving)
- -DTOTTYPE (donor total type for yearly giving)
- -DTOTBEG (donor total beginning date for selecting on yearly giving)
- -DALUM (flag to include alumni)
- -DCLYR1 (minimum alumni class year)
- -DCLYR2 (maximum alumni class year)
- -DGCLUB1 (giving club)
- -DGCLUB2 (giving club)
- -DINVOLVE1 (involvement code)
- -DINVOLVE2 (involvement code)
- -DSTATE (state)
- -DCOUNTY (county)
- -DCOUNTRY (country)
- -DZIP1(beginning zip code)
- -DZIP2 (ending zip code)
- -DCONTACT1 (contact code)
- -DTICKLER (tickler code)
- -DDUEDATE (contact due date)
- -DRESOURCE (contact resource)

### ADR Options

#### ADR Table

*Accesses:* $CARSPATH/modules/common/screens/tadr

*Menuopt File:* $CARSPATH/menuopt/common/screens/tadr

*Parameters Passed:* none

#### Test ADR Output

*Accesses:* $CARSPATH/src/common/adrtest

*Menuopt File:* $CARSPATH/menuopt/common/programs/adrtest

*Parameters Passed:* none

#### ADR Record

*Accesses:* $CARSPATH/modules/common/screens/adr

*Menuopt File:* $CARSPATH/menuopt/common/screens/adr

*Parameters Passed:* none

### Letters/Labels/Reports

#### Create WPVI Letters/Labels

*Accesses:* $CARSPATH/modules/common/scripts/ltbrun

*Menuopt File:* $CARSPATH/menuopt/utilities/scripts/ltbrun.usr

*Parameters Passed:*
- *PP_LTB_SELECT (letters, labels, both or none)*
- *PP_FILECABINET (WPVI file cabinet location of the ACE report)*
- *PP_LETTER (letter name)*
- *PP_DATE (date to print on letters)*
- *PP_DATE (selection date)*

#### Format Labels from a File

*Accesses:* $CARSPATH/src/util/labels

*Menuopt File:* $CARSPATH/menuopt/utilities/programs/lbls.f

*Parameters Passed:*
- *PP_FORM (label form name)*
- *PP_FULLNAME (lps label name)*
- *PP_FULLNAME (input file name)*
- *PP_OUTPUT (row or column output flag)*
- *PP_OUTPUT (compress large labels)*

#### Print Letters and Labels

*Accesses:* $CARSPATH/src/util/lps

*Menuopt File:* $CARSPATH/menuopt/utilities/programs/lps

*Parameters Passed:* none

#### WPVI ACE Report

*Accesses:* $CARSPATH/modules/common/scripts/reprun

*Menuopt File:* $CARSPATH/menuopt/common/scripts/reprun.usr

*Parameters Passed:* *PP_FILECABINET (WPVI file cabinet location of the ACE report)*

## Communications Management PERFORM (Table Maintenance) Screens

### Introduction

Communications Management uses PERFORM screens for displaying maintenance tables. You can access the screen files in the following directory paths:

- $CARSPATH/modules/common/screens
- $CARSPATH/modules/commgmt/screens

### PERFORM Screens

The following lists the PERFORM screens used in Communications Management.

**Note:** In the following list, descriptions of PERFORM screens include:
- Purpose of the screen
- Tables used in the screen

#### Step Table

*Purpose:* Enables you to view, add, and update tickler strategies.

*Menu Access:* Tickler: Tickler System Entry

*Screen File:* $CARSPATH/modules/commgmt/screens/steptbl

*Tables/Records Used:*
- comm_table
- ctc_table
- step_rec
- step_table
- stepctc_rec
- stepobj_rec
- stepreq_rec
- tick_table
- trk_table

#### ADR Record

*Purpose:* Enables you to view, add, and update *adr* information.

*Menu Access:* Table Maintenance: Common menu

*Screen File:* $CARSPATH/modules/common/screens/adr

*Tables/Records Used:*
- aa_table
- adr_rec
- adr_table
- id_rec
- rel_table

#### Alternate Address Table

*Purpose:* Enables you to view, add, and update *adr* information.

*Menu Access:* Table Maintenance: Common menu

*Screen File:* $CARSPATH/modules/common/screens/taa

*Tables/Records Used:*aa_table

**ADR Runcode Table**

*Purpose:* Enables you to view, add, and update *adr* information.

*Menu Access:* Table Maintenance: Common menu

*Screen File:* $CARSPATH/modules/common/screens/tadr

*Tables/Records Used:*adr_table

**Communication Table**

*Purpose:* Enables you to view, add, and update communication type information.

*Menu Access:* Table Maintenance: Common menu

*Screen File:* $CARSPATH/modules/common/screens/tcomm

*Tables/Records Used:*comm_table

**Contact Table**

*Purpose:* Enables you to view, add, and update valid types of contacts.

*Menu Access:* Table Maintenance: Common menu

*Screen File:* $CARSPATH/modules/common/screens/tctc

*Tables/Records Used:*
- ctc_table
- enr_stat_table
- tick_table
- comm_table
- adr_table

**Subscription Table**

*Purpose:* Enables you to view, add, and update types of subscriptions.

*Menu Access:* Table Maintenance: Common menu

*Screen File:* $CARSPATH/modules/common/screens/tsbscr

*Tables/Records Used:*
- comm_table
- ctc_table
- adr_table

**Tickler Table**

*Purpose:* Enables you to view, add, and update valid tickler codes.

*Menu Access:* Table Maintenance: Common menu

*Screen File:* $CARSPATH/modules/common/screens/ttick

*Tables/Records Used:*tick_table

## Communications Management Csh Scripts

### Introduction

Communications Management contains Csh scripts to automate the processing of information. Csh scripts are scripts that contain C shell commands. Such scripts can be very powerful and may include operating system commands. Csh scripts can also execute isql scripts and/or ACE reports. The Communications Management Csh scripts are located in the following directory paths:

- $CARSPATH/modules/common/scripts
- $CARSPATH/modules/commgmt/scripts
- $CARSPATH/modules/util/scripts

### Csh Scripts

The following list provides a description of each Csh script that relates to Communications Management.

**Note:** In the following list, descriptions of Csh scripts include:
- Purpose of the script
- A list of SQL statements used, if applicable
- A list of ACE reports used, if applicable

**lblrun**

Used for nonstandard label production runs.

*File:* $CARSPATH/modules/common/scripts/lblrun

**lpsrun**

Selects and prints scheduled letters and may optionally review a tickler system. May also run Contacts Due report to identify missed contacts.

*File:* $CARSPATH/modules/common/scripts/lpsrun

**lpstat**

Replaces a UNIX command to view the status of a set of printers.

*File:* $CARSPATH/modules/util/scripts/lpstat

**ltbrun**

Creates letters and/or labels, producing an *lps* output file. This is the primary letter writing script.

*File:* $CARSPATH/modules/common/scripts/ltbrun

**ltbwordp**

Creates merge data files for WordPerfect. Used when an institution has a UNIX version of WordPerfect and wants to create a WordPerfect merge file to merge with a letter to create a letter file.

*File:* $CARSPATH/modules/common/scripts/ltbwordp

**ltr_make**

Executes make targets on word processing letters.

*File:* $CARSPATH/modules/util/scripts/ltrmake

**ltrwp_list**

Generates a list of all ltrwp macros found in the ACE reports.  The script reviews all reports under modules/*/reports/l* for the macros, and lists the module, report, ltrwp macro name and value.

*File:* $CARSPATH/modules/util/scripts/ltrwp_list

**unloadtick**

Unloads all necessary Tickler tables for a specified code in ascii format, and creates a cpio file of the unloaded data for simplified table loading, unloading and removal across systems.

*File:* $CARSPATH/modules/commgmt/scripts/unloadtick

## Word Processing ACE Reports

### Introduction

CX uses ACE reports to produce formatted output for various product areas. For example, in the Admissions area, the ACE report *enrstat* produces a recruitment status summary report. In Communications Management, the Tickler Structure report is a commonly used report that shows the interrelationships between steps, objectives and requirements.

However, CX also uses ACE reports to extract the information you need to include in your correspondence. The reports, organized by functional area, reside in the following directories, along with any formatted output reports:

- $CARSPATH/modules/acctsrecv/reports
- $CARSPATH/modules/admit/reports
- $CARSPATH/modules/alumni/reports
- $CARSPATH/modules/commgmt/reports
- $CARSPATH/modules/common/reports
- $CARSPATH/modules/develop/reports
- $CARSPATH/modules/finaid/reports
- $CARSPATH/modules/finaudit/reports
- $CARSPATH/modules/regist/reports
- $CARSPATH/modules/stuserv/reports

### Tickler Reports

The Tickler Structure report displays all steps within your strategies, including their requirements, objectives, and contacts. Use the output from this report to review the contents of the Tickler tables following data entry. The output also provides documentation for Tickler strategies.

Reports on the contents of specific records are available to the menu user under Table Maintenance: Tickler (A-Z) Menu.

### ACE Reports for Data Extraction

The following reports extract information from the CX database for use in letters and reports.

**$CARSPATH/modules/acctsrecv/reports/ltrrecv**
- Selects information for accounts receivable dunning letters.
- Uses ctc_rec, suba_rec, id_rec.

**$CARSPATH/modules/admit/reports/ltrstat**
- Selects and feeds information about documents that have not yet been received in the admissions office. Set up to store each incoming contact made with a person in separate macro for testing in letter.
- Uses ctc_table, ctc_rec, adm_rec, id_rec.

**$CARSPATH/modules/admit/reports/ltradmit**
- Selects information for admissions letters.
- Uses ctc_table, prog_table, major_table, sess_table, title_table, bldg_table, id_rec, prog_enr_rec, fac_rec.

**$CARSPATH/modules/admit/reports/ltrhsg**
- Selects information for admissions housing letters.
- Uses ctc_table, title_table, sess_table, bldg_table, ctc_rec, id_rec, adm_rec, stu_serv_rec.

**$CARSPATH/modules/admit/reports/ltrnoltr**

- Updates Contact records to a status of C, without creating a letter.
- Uses ctc_table, ctc_rec.

**$CARSPATH/modules/admit/reports/ltrparent**

- Selects information for admissions letters sent to an applicant's parent.
- Uses aa_table, ctc_table, major_table, sess_table, title_table, bldg_table, id_rec, prog_enr_rec, fac_rec, relation_rec, adr_rec, ctc_rec, adm_rec.

**$CARSPATH/modules/admit/reports/ltrstatg**

- Produces information about both missing and received admissions documents.
- Uses ctc_table, ctc_rec, id_rec, adm_rec.

**$CARSPATH/modules/admit/reports/ltrtrans**

- Produces information to merge into admissions letters by locating expected ctc_recs with a valid non-zero ctc_corr_id and that are to be produced by this report as determined from the ctc_table. This report expects the letter to be addressed to the id indicated in the corr_id field.
- Uses ctc_table, ctc_rec, id_rec, adm_rec.

**$CARSPATH/modules/admit/reports/ltrstat_pa**

- Selects and feeds information to the LTR_STAT letter. Looks for contacts that are expected and incoming documents that have not been received. If more than ten documents are required, this ACE report and macros/custom/ltrwp must be modified.
- Uses ctc_table, ctc_rec, id_rec, adm_rec.

**$CARSPATH/modules/admit/reports/ltrstati**

- Selects and feeds information into document tracking status letter. Persons with an expected contact resource with a ctc_table communication code of DOCU will be selected in this report.
- Uses ctc_table, ctc_rec, id_rec, ed_rec, adm_rec.

**$CARSPATH/modules/admit/reports/ltrschvst**

- Selects information for letters announcing school visits.
- Uses ctc_table, prog_table, major_table, sess_table, title_table, bldg_table, ctc_rec, ed_rec, id_rec, adm_rec, prog_enr_rec fac_rec, schd_rec.

**$CARSPATH/modules/admit/reports/ltrstat2**

- Selects and feeds information into the STATUS letter. Set up to store each incoming contact made with a person in separate macro for testing in letter.
- Uses ctc_table, ctc_rec, id_rec, adm_rec.

**$CARSPATH/modules/alumni/reports/ltralum**

- Creates input for alumni letters based on expected ALUM contacts.
- Uses ctc_table, title_table, ctc_rec, id_rec, alum_rec.

**$CARSPATH/modules/alumni/reports/ltrzip2**

- Creates input for alumni letters based on expected ALUM contacts.
- Uses id_rec, alum_rec.

**$CARSPATH/modules/common/reports/ltrsbscr**

- Selects information for subscription letters. This is a generalized report for all offices. It runs off the Subscription records after joining to the Contact table.
- Uses ctc_table, sbscr_rec, id_rec.

**$CARSPATH/modules/common/reports/ltrnoltr**

- Selects information for admissions letters.
- Uses ctc_table, ctc_rec.

**$CARSPATH/modules/develop/reports/ltrsol**

- Selects information for development solicitation letters.
- Uses ctc_table, ctc_rec, id_rec, donor_rec, dontot_rec, alum_rec, consstat_rec.

**$CARSPATH/modules/develop/reports/ltrackno**
- Selects information for development acknowledgment letters.
- Uses ctc_table, giftfrm_table, desg_table, prem_table, pdesg_table, title_table, ctc_rec, id_rec, gift_rec, camp_rec, pldg_rec, prem_rec.

**$CARSPATH/modules/develop/reports/ltrackpldg**
- Selects information for letters requesting pledges.
- Uses ctc_table, desg_table, title_table, ctc_rec, id_rec, pldg_rec, camp_rec.

**$CARSPATH/modules/develop/reports/ltrpldgrem**
- Selects information for building the criteria for development pledge reminder letters.
- Uses desg_table, id_rec, pldg_rec, pldgpay_rec, camp_rec.

**$CARSPATH/modules/develop/reports/ltrackrcpt**
- Builds the input file for LPS for Development acknowledgment letters. It creates the acknowledgment letter and receipt as a unit, and is to be run after the *giftpost* process is completed.
- Uses ctc_table, giftfrm_table, desg_table, prem_table, pdesg_table, title_table, dagrp_rec, ctc_rec, id_rec, gift_rec, camp_rec, pldg_rec, prem_rec, ctcdetl_rec.

**$CARSPATH/modules/finaid/reports/ltrfa**
- Produces information needed for general financial aid letters.
- Uses ctc_table, ctc_rec, id_rec.

**$CARSPATH/modules/finaid/reports/ltrfacover**
- Produces information for financial aid award cover letters.
- Uses ctc_table, tltle_table, ctc_rec, id_rec, aid_rec.

**$CARSPATH/modules/finaid/reports/ltrfastat**
- Formats letter information for financial aid status letters and document tracking. Reads in all expected contacts to be received and reads them into macros for insertion of additional sentences/paragraphs. Contacts to be printed must have this ACE in the tctc_ACE field.
- Uses ctc_table, ctc_rec, id_rec.

**$CARSPATH/modules/finaid/reports/ltrfawd**
- Provides labels for financial aid award letters, without updating contacts or Tickler.
- Uses ctc_table, ctc_rec, id_rec.

**$CARSPATH/modules/finaid/reports/ltrstatctc**
- Formats information for financial aid status letters. Reads in all expected contacts to be received and reads them into macros for insertion of additional sentences/paragraphs. Contacts to be printed (e.g., documents to be tracked) must have the ACE hardcoded in the first select statement in the tctc_ACE field in the Contact table.
- Uses ctc_table, ctc_rec, id_rec.

**$CARSPATH/modules/finaid/reports/ltrterm**
- Produces information needed by common financial letters. Looks for one expected contact code, and completes the contact when the letter is created.
- Uses ctc_table, ctc_rec, id_rec.

**$CARSPATH/modules/finaudit/reports/ltrconfpay**
- Selects information for accounts payable confirmation letters.
- Uses suba_rec, id_rec.

**$CARSPATH/modules/finaudit/reports/ltrconfrcv**
- Selects information for accounts receivable confirmation letters.

- Uses suba_rec, id_rec.

**$CARSPATH/modules/publicity/reports/ltraccomp**
- Selects information for accomplishments reporting for newspaper press releases.
- Uses accomp_table, sess_table, title_table, major_table, deg_table, accomp_rec, id_rec, profile_rec, prog_enr_rec, ed_rec, sch_rec.

**$CARSPATH/modules/publicity/reports/ltrgrad**
- Selects information for graduation reporting for newspaper press releases.
- Uses title_table, deg_table, major_table, sess_table, id_rec, profile_rec, prog_enr_rec, stu_acad_rec, ed_rec.

**$CARSPATH/modules/publicity/reports/ltrinvl**
- Selects information for involvements reporting for newspaper press releases.
- Uses title_table, invl_table, major_table, deg_table, involve_rec, id_rec, profile_rec, prog_enr_rec, ed_rec, sch_rec.

**$CARSPATH/modules/regist/reports/ltracadrec**
- Selects information for academic records letters (e.g., dean's list letters or probation letters). Ensure that the academic status codes are updated in the prog_enr_rec first before running this report to create the probation letters.
- Uses ctc_table, title_table, sess_table, major_table, deg_table, ctc_rec, id_rec, stu_acad_rec, prog_enr_rec, profile_rec.

**$CARSPATH/modules/regist/reports/ltrcnclsec**
- Builds the canceled sections letter information.
- Uses ctc_table, sess_table, ctc_rec, id_rec, cw_rec, sec_rec

**$CARSPATH/modules/stuserv/reports/ltrchapel**
- Selects information for chapel attendance letters.
- Uses ctc_table, title_table, sess_table, major_table, deg_table, chap_table, ctc_rec, id_rec, stu_acad_rec, prog_enr_rec, profile_rec, chap_rec.

**$CARSPATH/modules/stuserv/reports/ltrhsgfrm**
- Builds information for housing preference forms.
- Uses id_rec, profile_rec, adm_rec, stu_serv_rec.

**$CARSPATH/modules/stuserv/reports/ltrrmbrd**
- Builds information for room and board letters.
- Uses sess_table, bldg_table, facil_table, id_rec, profile_rec, stu_serv_rec.

**$CARSPATH/modules/stuserv/reports/ltrrmate**
- Builds information for letters about roommates.
- Uses sess_table, bldg_table, facil_table, ctry_table, id_rec, profile_rec, stu_serv_rec.

**$CARSPATH/modules/matric/reports/ltreval**
- Builds information for reports of a student's unsatisfactory performance.
- Uses ctc_table, ctc_rec, id_rec, cweval_rec, cw_rec.

**$CARSPATH/modules/matric/reports/ltrmatric**
- Builds information for general matriculation letters.
- Uses ctc_table, goal_table, goaltime_table, deg_table, acad_stat_table, major_table, title_table, bldg_table, ctc_rec, id_rec, adm_rec, goal_rec, prog_enr_rec, fac_rec.

# SECTION 17 - USING SQL SCRIPTS IN COMMUNICATIONS MANAGEMENT

## Overview

### Introduction

You can use SQL query language to increase your efficiency in Communications Management by both retrieving data and adding new records.  For example, SQL can create Contact records when you want to create correspondence with a large group of individuals who share a common characteristic (e.g., to create contacts for all incoming freshmen).  SQL can also update a large number of records at one time.

### Using SQL in the Recruiting and Admissions Product

To use SQL, you must use the field names associated with the appropriate tables or records. Typically, SQL statements for the CX Recruiting and Admissions product include rows and columns from the following tables:

- accomp_rec (Accomplishment record)
- adm_rec (Admissions record)
- ctc_rec (Contact record)
- ed_rec (Education record)
- exam_rec (Exam record)
- int_rec (Interest record)
- involve_rec (Involvement record)
- profile_rec (Profile record)

## Sample SQL Statements

### Introduction

The SQL statements that follow enable you to perform some Communications Management tasks quickly and efficiently. Modify these statements to meet the needs of your institution, or use them as a model for other statements. For more information about creating and maintaining SQL statements, refer to the documentation for the Informix database.

To create and test SQL scripts, you usually use ISQL, an interactive SQL statement processor. With ISQL, you can create new SQL scripts. You can then run them, modify them as required during testing, and save the corrected versions. When complete, you can print the results, save them in a file, or pipe them to another process.

### How to Select Specific Records

The following SQL script selects the ID number, full name (in the format *lastname*, *firstname*, *mi*,, *suffix*), city, and state for all entities in the id_rec table that match "Smith, Ro" in the first nine character positions of the *fullname* column.

```
select id, fullname, city, st from id_rec where fullname matches "Smith, Ro*";
```

Output from this script might resemble the following, depending on the contents of your database:

```
        id fullname                        city                    st

     22362 Smith, Robert                   Monroe                  MI
      5068 Smith, Robert Daniel            Florence                KY
     20361 Smith, Robert L.                Cincinati               OH
      2892 Smith, Robert Paul              Pleasureville           KY
```

### How to Select Records with a Range of Values

The following SQL script selects the ID number, full name, zip code, and the office-added-by code from the id_rec where the zip code is in the range 51000<= n < 51010. Note the quoted values for the zip codes. These quotes are required because the zip code field is a character field, not numeric.

```
select id, fullname, zip, ofc_add_by from id_rec where
  ((zip >= "51000") and (zip < "51010"));
```

Output from this script might resemble the following, depending on the contents of your database:

```
        id fullname                        zip        ofc_add_by

     56749 Akron-Westfield Community Sch   51001      TAPE
     56758 Alta Junior-Senior High School  51002      TAPE
     56759 Floyd Valley High School        51003      TAPE
     56765 Anthon-Oto Jr-Sr High School    51004      TAPE
     56771 Aurelia High School             51005      TAPE
     56775 Battle Creek Community School   51006      TAPE
```

### How to Use the Count Aggregate Function

The following SQL script demonstrates the use of the count aggregation function. SQL has several aggregation functions built into the language standard. In this case, the count(*) function returns the number of rows in the Select statement which meet the Where clause. In other words, if the count(*) function were removed from the statement and replaced with a column name, then the number returned by the count function will equal the number of rows returned by the modified select statement.

```
select count(*) from ctc_rec where resrc = "ACCEPTED";
```

Output from this script might resemble the following, depending on the contents of your database:

```
 (count(*))

        48
```

### How to Use the Sum and Count Aggregate Functions

The following SQL statement illustrates the use of both the sum() and count(*) aggregation functions along with a more complex Where clause.

```
select count(*), sum(gift_amt) from dontot_rec, id_rec
where ((id_rec.id = dontot_rec.id) and (zip >= "40000")
and (zip < "40500") and (dontot = "LT"));
```

Output from this script might resemble the following, depending on the contents of your database:

```
 (count(*))           (sum)

       814       $590698.45
```

### How to Use the Min, Max and Avg Aggregate Functions

The following SQL statement illustrates the use of the min(), max() and avg() aggregation functions which are present in SQL.

```
select min(cum_gpa), max(cum_gpa), avg(cum_gpa) from stu_stat_rec, int_rec
   where ((int_rec.id = stu_stat_rec.id) and (interest = "FOOT"));
```

Output from this script might resemble the following, depending on the contents of your database:

```
         (min)          (max)            (avg)

          0.00           3.77             1.62
```

### How to Use Temporary Files for Script Output

The following SQL script illustrates the use of a temporary file to retain the output of the Select statement.  This temporary file should normally have the "with no log" clause on it to ensure that the Online engine does not log the transactions that fill this table.  These temporary tables are not normally logged because they do not represent any real data, only intermediate results.

This script also illustrates the use of the Order By clause to sort a result set in a particular order. When in ISQL, the select .. into does not produce any output on the screen, while the select .. without the into ... does show its selected rows on the screen.

```
select int_rec.id, fullname, remark from id_rec, int_rec
  where ((int_rec.id = id_rec.id) and (interest = "FOOT")
    and (id_rec.id < 1000)) order by id into temp a with no log;

select * from a;

select * from a order by fullname into temp b with no log;

select * from b;
```

Output from this script might resemble the following, depending on the contents of your database:

```
         id fullname                        remark

        200 Zeisloft, Mary
        211 Franks, Mary Margaret           Quarterback
        254 Peyton, Ishmael A
        288 Sumrall, Patricia Ann
        313 Neal, David Edward              full back
        318 Tanner, Estilla Rebecca
        364 Boyter, David Carroll
        878 Riffe, Barry Lee


         id fullname                        remark

        364 Boyter, David Carroll
        211 Franks, Mary Margaret           Quarterback
        313 Neal, David Edward              full back
        254 Peyton, Ishmael A
        878 Riffe, Barry Lee
        288 Sumrall, Patricia Ann
        318 Tanner, Estilla Rebecca
        200 Zeisloft, Mary
```

## How to Use the *outer* Keyword in Select Statements

The following SQL script illustrates the use of the *outer* keyword in a Select statement. The normal multiple table selection with *n* tables must have at least *n-1 join* type of Where clauses in order to define the relationships of the tables. You may also use additional Where clauses that apply selection criteria to the rows. In the example, the clause (id_rec.id = alum_rec.id) is a *join* type of Where clause.

In contrast, the clause (fullname matches "Smith*") is a *selection* type of clause that is expected to act as a filter to reduce the number of rows selected.

In the case of the first Select statement, the person must exist in the alum_rec in order for the row to be selected. In the case of the third Select statement, the record will be included even if it does not occur in the alum_rec.

**Note:** Informix performs the outer join by first doing the join. Then, if a null exists in the joined column (in this case alum_rec.id), the row from the Cartesian product will be included in the result set. When you apply a selection type of Where clause to one of the columns from a table which is prefixed by the keyword *outer,* you may receive unexpected results. In this case, Informix again first creates the Cartesian product of the tables in the join. However, the selection type of Where clause may reject some rows from the table joined with the *outer* keyword. In this case, the null value will again appear in the joined column position because of the rejected value. The result may seem inconsistent to the beginning SQL user because the null in the column means the row will be included in the result set. The beginning user expects the row to be omitted because of the selection type of Where clause.

To resolve this selection issue, do not apply the selection Where clause to the Select statement. Instead, route the output of the Select statement *without* the selection type of Where clause into a temporary table. Then, select from this temporary table and apply the selection type of Where clause.

**Example:** Consider the following two tables:

- table A (the id_rec with columns id, and st)
- table B (the ctc_rec with columns id and resrc
  The two columns id_rec.id and ctc_rec.id join together. Assume the tables have the following records:

| | |
|---|---|
| id_rec: | (1,IL) |
| | (2,OH) |
| | (3,IN) |

```
                (4,KY)
      ctc_rec:  (1,APPLIED)
                (1,ACCEPTED)
                (2,APPLIED)
                (3,ACCEPTED)
```

If you use the following Select statement:

```
select id_rec.id, st, resrc from id_rec, ctc_rec where id_rec.id = ctc_rec.id;
```

You obtain the following output:

```
1      IL      APPLIED
1      IL      ACCEPTED
2      OH      APPLIED
3      IN      ACCEPTED
```

If you use an *outer* keyword in the statement, then the Select statement becomes:

```
select id_rec.id, st, resrc from id_rec, outer ctc_rec
  where id_rec.id = ctc_rec.id;
```

The resulting output is:

```
1      IL      APPLIED
1      IL      ACCEPTED
2      OH      APPLIED
3      IN      ACCEPTED
4      KY      -NULL-
```

If you then add a *select* type of Where clause to change the statement to:

```
select id_rec.id, st, resrc from id_rec, outer ctc_rec
  where ((id_rec.id = ctc_rec.id) and (ctc_rec.resrc <> "APPLIED");
```

The result set contains:

```
1      IL      -NULL-
1      IL      ACCEPTED
2      OH      -NULL-
3      IN      ACCEPTED
4      KY      -NULL-
```

No APPLIED resrc value exists in the result set; however, the beginning user may expect a different number of returned rows.

An example of temporary table usage follows:

```
select id_rec.id, fullname, cl_yr from id_rec, alum_rec
   where ((id_rec.id = alum_rec.id) and (fullname matches "Smith*"))
      into temp x with no log;

select count(*) from x;

select id_rec.id, fullname, cl_yr from id_rec, outer alum_rec
   where ((id_rec.id = alum_rec.id) and (fullname matches "Smith*"))
      into temp y with no log;

select count(*) from y;
```

Output from this script might resemble the following, depending on the contents of your database:

```
 (count(*))

          2


 (count(*))

        276
```

### How to Use the *unique* Keyword

The following SQL script illustrates the use of the *unique* keyword. Although *unique* is an Informix usage kept for historical purposes, the ANSI keyword is *distinct*. When you use the keyword *distinct*, then the result set will have all distinct, unique rows.

```
select fullname, resrc from id_rec, ctc_rec
   where ((id_rec.id = ctc_rec.id) and (id_rec.id < 1000))
      into temp a with no log;

select count(*) from a;


select unique fullname, resrc from id_rec, ctc_rec
   where ((id_rec.id = ctc_rec.id) and (id_rec.id < 1000))
      into temp b with no log;

select count(*) from b;
```

Output from this script might resemble the following, depending on the contents of your database:

```
 (count(*))

       1969


 (count(*))

       1097
```

### How to Obtain the *minus* Function in a Set Operation

The following SQL script illustrates a set operation that was called *minus* in previous versions of Informix, after the standard set theory name. If you create two Select statements that produce identical layout rows in the temporary tables *a* and *b*, then the operation *a minus b* produces a result set that contains those values which are in *a* only. Those values which are in both *a* and *b* will not appear in the minus result set.

```
select unique id_rec.id from id_rec, adm_rec, ctc_rec
  where ((id_rec.id = adm_rec.id) and (id_rec.id = ctc_rec.id)
      and (adm_rec.prog = "UNDG") and (resrc = "APPLIED"))
     into temp x with no log;

select count(*) from x;


select unique id_rec.id from id_rec, adm_rec, ctc_rec
  where ((id_rec.id = adm_rec.id) and (id_rec.id = ctc_rec.id)
      and (adm_rec.prog = "UNDG") and (resrc = "ACCEPTED"))
     into temp y with no log;

select count(*) from y;


select id from x i
   where id not in (select id from y)
      into temp z with no log;

select count(*) from z;
```

Output from this script might resemble the following, depending on the contents of your database:

```
 (count(*))

         21


 (count(*))

         33


 (count(*))

         20
```

### How to Create Campus Visit Invitation Contacts

The following SQL statement creates the Contact records to generate a batch of campus visit invitations.

```
select      adm_rec.id

from        adm_rec

where       (adm_rec.enrstat = "INQUIRED" or adm_rec.enrstat matches "APPL*")
  and       plan_enr_yr >= "1997"
  and   adm_rec.add_date >= "09/01/1996"
  into temp tmp_1 with no log;

  select tmp_1.id,
    "INVITE" virtual_01,
    "ADM" virtual_02,
    "E" virtual_03,
    today virtual_04,
    today virtual_05
  from   tmp_1
  into temp tmp_2 with no log;

  insert into ctc_rec
    (id,
    resrc,
    tick,
    stat,
    due_date,
    add_date)
  select * from tmp_2;
```

### How to Add Lead Contact Records

The following SQL statement creates Lead Contact records based on Lead records.  This statement could enable you to schedule and send letters to all leads.

```
select      lead_rec.lead_no
from        lead_rec
where       lead_rec.add_date = today
into temp tmp_1 with no log;

select      tmp_1.lead_no,
   "LEADLET1" virtual_01,
   "LEAD"     virtual_02,
   "E"        virtual_03,
   today      virtual_04,
   today      virtual_05
from        tmp_1
into temp tmp_2 with no log;

insert into ldctc_rec
   (lead_no,
   resrc,
   tick,
   stat,
   add_date,
   due_date)

select * from tmp_2;
```

## How to Update Tickler Records

The following statement performs two functions:

- Upgrades the Tickler level to “A” for students who achieved a score greater than 155 on the LSAT examination.
- Changes the Tickler level to “Z” for students who achieved a score less than 150 on the LSAT examination.

By using this script, applicants with exceptional test scores will receive more recruitment letters, and students with poor scores will receive the minimum number of letters.  To further automate this process, you can add the running of this script to the institution’s *cron* processing.  For more information about processing using *cron*, see your UNIX operating system documentation.

**Note:** The Tickler levels determine the number of contacts to be scheduled for a given recipient.  The higher the level (where A is highest), the more contacts the Tickler strategy creates.

```
select      exam_rec.id
from        exam_rec,
   tick_rec
where       (exam_rec.ctgry = "LSAT"
  and       exam_rec.score1 >= 156)
  and   tick_rec.level != "A"
  and   tick_rec.tick = "ADM"
  and   exam_rec.id = tick_rec.id
into temp lvl_A with no log;

update tick_rec set level = "A"
where tick_rec.id in (select id from lvl_A);



select      exam_rec.id
from        exam_rec,
   tick_rec
where       (exam_rec.ctgry = "LSAT"
  and       exam_rec.score1 <= 149)
  and   tick_rec.level != "Z"
  and   tick_rec.tick = "ADM"
  and   exam_rec.id = tick_rec.id
into temp lvl_Z with no log;

update tick_rec set level = "Z"
where tick_rec.id in (select id from lvl_Z);
```

# SECTION 18 - CUSTOMIZING THE COMMUNICATIONS MANAGEMENT PROCESSES

## Overview

### Introduction

This section provides procedures for setting and installing the features of the Communications Management product. The following procedures are included:

- Assessing institution needs for the product
- Reviewing data in tables and records
- Changing default field values in macros

### Basic Information

This section contains detailed procedures specific to the Communications Management product. For information on performing basic procedures such as using the MAKE processor and reinstalling options, refer to the following resources:

- *Database Tools and Utilities* course notebook
- *Jenzabar CX Technical Manual*

## Assessing the Communications Management Setup

### Introduction

CX provides several ways to implement the options of the Communications Management product. After assessing the needs of your institution, you can change the default settings of Communications Management enable macros and reinstall the product.

This section lists and describes the features that you must assess before you can modify the macros.

**Note:** A variety of word processing macros exist for use with Communications Management. For more information about this type of macros, see *Using ACE and Word Processing Macros* in this section.

### Tickler Processing

The Communications Management standard setup enables individuals in the Admissions office to access Tickler menu options. The following macro that controls this feature is in macros/custom/admissions:

m4_define(‘ENABLE_FEAT_ADM_TICKLER’,’Y’)

### Setting a Communication Code Default

Although you can use as many communication codes as desired, Communications Management requires a default communication code. The first of the following macros defines the default code, and the second macro defines example codes. These macros appear in $CARSPATH/macros/custom/common.

m4_define(`COMM_DEF', `LETT')

m4_define(`COMM_EG', `, eg: (LETT)letters, (LABL)labels, (LTLB)both.')

### Waiving the Contact Span

This macro defines the default value for the Waive Span field in the Contact table. The Waive Span field controls whether *tickler* can override the maximum and minimum span of days between contacts. Jenzabar recommends that you set the value of the macro to N, since most institutions do not need to override this span.

m4_define(`CTC_SPAN_WAIVE_DEF',`Y')

### Setting Reissue Defaults

This macro defines the default value for the Reissue field in the Contact table. The Reissue field controls whether *tickler* can schedule a contact a second time, even if the contact has already been sent. Jenzabar recommends that you set the value of the macro to N, since most institutions do not want to send duplicate correspondence.

m4_define(`CTC_REISSUE_DEF',`Y')

### Use of *fps* (Forms Production System)

This macro enables the *fps* program. The standard CX system enables *fps*. If your institution centralizes the printing process (that is, no printing occurs in end-user offices), then set the macro value to N.

m4_define(`ENABLE_FEAT_FPS',`Y')

## Tickler Types

Tickler processing requires you to define all the Tickler codes in the tick_table in $CARSPATH/macros/custom/common. Typically, institutions do not need to change the values in this file, since they correspond exactly to the Tickler table (tick_table) that is delivered with the standard CX product. However, if the institution adds a new Tickler code to the Tickler table (e.g., ADMG as a strategy for graduate student applicants), then the Tickler type macro must also contain the new code (e.g., m4_define(`TICK_ADMG',`ADMG')). This macro file contains the following definitions:

- m4_define(`TICK_ADM',`ADM')
- m4_define(`TICK_LEAD',`LEAD')
- m4_define(`TICK_ALPR',`ALPR')
- m4_define(`TICK_DEV',`DEV')
- m4_define(`TICK_FA',`FA')
- m4_define(`TICK_ACAD',`ACAD')
- m4_define(`TICK_RECV',`RECV')
- m4_define(`TICK_OTHRECV',`ORCV')
- m4_define(`TICK_REG',`REG')
- m4_define(`TICK_SBSCR',`SBSC')
- m4_define(`TICK_MATRIC',`MAT')
- m4_define(`TICK_TRANS',`TRAN')
- m4_define(`TICK_EOPS',`EOPS')
- m4_define(`TICK_DSPS',`DSPS')

The macro file also contains the following macros that define defaults. These macros do not require modification unless you add one or more Tickler codes.

- m4_define(`TICK_DEF',`TICK_ADM')
- m4_define(`TICK_VALID',`TICK_ADM,TICK_LEAD,TICK_ALPR,TICK_DEV,TICK_MATRIC,TICK_TRANS,TICK_EOPS,TICK_DSPS,TICK_FA')
- m4_define(`TICK_INCL',`include=(TICK_VALID),upshift')

The admissions macro file ($CARSPATH/macros/custom/admissions) also contains a reference to Tickler codes (e.g., ADM_PROG_TICK and ADM_TICK_VALID).

## Automatic Creation of Contacts

This macro causes the system to create required contacts with a status of (E)xpected. If you set this macro value to Y, CX will create records for any contact with a Document Tracking Type of R (as indicated in the Contact table).

m4_define(`AUTO_ADD_DOC_CTCS',`Y')

## Reviewing and Modifying Data in Communications Management Tables and Records

### Introduction

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

### Procedure

Follow these steps to review the values of CX tables and records.

1. For each Communications Management 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 Communications Management system. Determine whether or not the records need to be updated to meet the needs of CX reports. Make updates as appropriate.

### Table and Record Information

For more information about the tables and records in the Communications Management product, see the section *Communications Management Tables and Records* in this manual.

### Order of Implementation

CARS suggests that you implement the Communications Management tables and records in the following order:

**Note:**

- After you complete or verify the contents of these tables and records, the Communications Management product is available for use.
- For detailed instructions for completing the Tickler table and the other supporting tickler tables that define the institution's tickler strategies, see *Customizing The Tickler Processes* in this manual.
- The Program table and the Enrollment Status table contain fields that relate to Communications Management, and therefore appear in the following list. These tables, however, are most extensively used in the CX Recruiting and Admissions product. For more information about these tables, see the *Recruiting and Admissions Technical Manual.*

1. Tickler table (tick_table)

   **Note:** Make sure you reproduce any changes to the Tickler table in the Tickler macros.
2. Program table (prog_table) (if you are implementing Communications Management for use with Admissions)
3. Enrollment Status table (enr_stat_table) (if you are implementing Communications Management for use with Admissions)
4. Alternate Address table (aa_table)
5. Addressing table (adr_table)
6. Addressing record (adr_rec)
7. Contact table (ctc_table)

8. Other tables in any order

### Setting No Nulls During Implementation

When implementing the Communications Management tables, your results are most reliable if you use the nonull command.

After setting up the aa_table and the adr_rec, enter the following commands at the UNIX prompt:

**nonull aa_table**
**nonull adr_rec**

### Order of Table Information in This Section

Information about the setup of these tables appears in the following order in this manual:

- Tables that require modification appear first, in the order of implementation recommended by Jenzabar.
- Tables that do not require modification appear after the required tables, in alphabetical sequence.

### Alternate Address Table

The Alternate Address table (aa_table) defines the types of alternate addresses the institution uses. It contains the following fields:

**Code**
The four-character name for the type of address (e.g., PERM or SUMR).

**Description**
Text describing the address type (e.g., Permanent or Summer).

**Priority**
A number that enables *adr* to prioritize addresses; the lower the number, the higher the priority. Therefore, if two address types have the same date ranges but different priority numbers, *adr* will select the address with the lower priority number.

**Maintenance**
A Y/N flag indicating whether, if you update this type of address, you want the system to save the old address as a previous address. In most cases, the only address code for which this flag is Y is the PERM code.

### Addressing Table

The Addressing table (adr_table) defines valid run codes. Run codes determine the format of labels and salutations, based on criteria specified in the Addressing record. For example, a JOINT run code indicates that two names are included in the address (e.g., Mr. William Smith and Mrs. Mary Smith), while JOINTI indicates the address is informal (e.g., Bill and Mary Smith).

The ADR table contains the following fields:

**Code**
The code for the type of run code (e.g., JOINT or JOINTI and SINGLE or SINGLEI).

**Description**
Text describing the run code (e.g., Joint - Formal or Joint - Informal).

### Addressing Record

The Addressing record (adr_rec) defines the type of salutation and mailing address your correspondence uses. Several Addressing records can exist for each individual or organization, depending on their addressing needs. For example, if letters sent to a student’s parents are addressed to Mr. and Mrs. Jones, but letters sent to the student's father's place of business are addressed to Mr. Jones only, two Addressing records are required.

You can add Addressing records through the ADR Record option on the Utilities: ADR Options menu. Menu users in the Alumni office can also access a detail window where they can add Addressing records.

The Addressing record contains the following fields:

**ID Number**
The ID number of the individual or organization.

**ADR Run Code**
The run code that designates if you want this ID to receive single or joint correspondence, and formal or informal salutations (e.g., JOINT or SINGLEI). The code must be valid in the ADR table.

**Address Code**
Optional - The alternate address code to which the record and runcode applies (e.g., PERM or BUS). If you leave this field blank, the *adr* program will check the priorities of the alternate addresses in the aa_table to determine which address to print.

**Use Primary ID**
Used with the Join Relationship Code, a Y/N flag indicating whether *adr* should use this ID address (Y), or the address on the joined relationship ID (N).

**Join Relationship Code**
Used with the Use Primary ID, the relationship with which you want to join the ID (e.g., HW for husband and wife, or PC for parent and child).

**Example:** Using the Use Primary ID and Join Relationship Code:

- For a student ID, if you set the Use Primary ID to N and the Join Relationship Code to PC, then the correspondence will be sent to the parent.
- For a student ID, if you set the Use Primary ID to Y and the Join Relationship Code to PC (or leave the Join Relationship Code field blank), then the correspondence will be sent to the student.

**Label Style Code**
A code indicating the addressing style you want for a label. To use an alternate addressing style, you must have an Alternate Name record for the individual. Valid values are:

- F (Formal (e.g., Mr. William Jones))
- I (Informal (e.g., Bill Jones))
- blank (none)

**Salutation Style Code**
A code indicating the salutation style you want for a letter. To use an alternate salutation, you must have an Alternate Name record for the individual. Valid values are:

- F (Formal (e.g., Dear Mr. Jones))
- I (Informal (e.g., Dear Bill))
- blank (none)

**Number of Name Lines**
The number of lines on which you want to print names. Valid values are:

- 0 (Print jointly with a single last name on one line (e.g., Mr. and Mrs. Ralph Jones).
- 1 (Print separate last names on one line (e.g., Mr. Ralph Jones and Mrs. Betty Smith).
- 2 (Print separate last names on two lines (e.g., Mr. Ralph Jones on the first line, and Mrs. Betty Smith on the second line).

**Note:** When using 1 or 2, you can reverse the order of the printed names (e.g., Mrs. Betty Smith on the first line, and Mr. Ralph Jones on the second line) by setting the Join Relationship - Primary field in the Relationship record to N.

**Use Suffixes**
A Y/N flag indicating whether you want to include suffixes (e.g., M.D.) on addresses.

**Use Title**
A Y/N flag indicating whether you want to include titles (e.g., Mr. or Dr.) on addresses.

**Allow Duplicates**
A Y/N flag indicating whether you want to create duplicate copies of a contact (e.g., a file copy of a letter). For example, if both John and Mary Smith are alumni of the class of 1990 and you want to produce a single alumni letter, set this flag to N.

**Address Priority**
The priority (where 1 is the highest value) within a group of records for the same ID number and run code. The *adr* program uses the Address Priority to select the correct address. Jenzabar recommends that you leave this field blank, and use the Alternate Address table to define priorities.

## Contact Table

The Contact table defines all valid types of correspondence or interaction with individuals or organizations, and contains information about how to use the contact. It contains the following fields:

**Code**
The eight-character code name of the contact (e.g., APPRECV or ASKDPST).

**Description**
The description of the contact (e.g., Acknowledges the receipt of an application or Request a deposit).

**Tickler**
The code corresponding to the tickler strategy to which the contact relates (e.g., ADM for undergraduate admissions).

**Comm Code**
The four-character code for the type of communication (e.g., LETT for letters and envelopes, PHON for telephone call, LABL for labels only, or LTLB for letters, envelopes, and labels). This code must be valid in the comm_table.

**Routing**
The code (I for incoming or O for outgoing) indicating how the contact is initiated (e.g., communications from the institution to an applicant or donor have a Routing code of “O”, while a document received from an applicant or donor has a Routing code of “I”).

**Span Waived**
A Y/N flag indicating whether you want to waive the normal span of days between contacts for this contact. If the flag contains Y, *tickler* ignores the maximum and minimum span of days between contacts, and schedules the contact with a Due Date of “today”. If the flag contains N, *tickler* calculates the contact’s due date using the maximum span of days.

**Reissued**
A Y/N flag indicating whether you want to be able to send the contact more than once. If the flag contains Y, *tickler* will schedule the contact, even if it already exists in the recipient’s Contact record. If the flag contains N, *tickler* will first check to see if the contact already exists for the recipient; if it does exist, *tickler* will not schedule an additional contact.

**Example:** If the Admissions office elects to send a letter sooner than scheduled and if this flag contains N, then the letter is sent only once (at the earlier date). If this flag contains Y, then the letter is sent twice (at the earlier date, and again at the scheduled date).

**Note:** A contact with a Reissued field set to Y cannot be included more than once within a single tickler step.

**ACE Report**
The name of the ACE report that extracts information for the letter and/or label.

**Run Code**
The *adr* run code associated with the contact (e.g., the SINGLEI run code addresses the applicant as an individual by his/her first name).

**Bulk Mail**
A Y/N flag indicating whether you want to sort the letters or labels generated by this contact in zip code order for bulk mailing.

**Default Type**
The code associated with the method for producing the letter. Valid values are:

- rtf (Creates Rich Text Format for use with Microsoft Word. Merge files will go to the Merge drawer of the admissions FileCabinet for transmission to the c: drive of a PC.)
- stdlps (Creates standard CX letters. The *nroff* program uses the output file with the corresponding letter file in the admissions FileCabinet.)
- wordp (Creates WordPerfect format. Merge files will go to the Merge drawer of the admissions FileCabinet for transmission to the c: drive of a PC.)

**Document Tracking Type**
A one-character code (either R or blank) indicating whether you want to add a required document contact with an (E)xpected status. For example, if the Contact Code is ESSAY (i.e., an admissions essay that is required of every applicant), you can set the Document Tracking Type to “R” to indicate it is required, and CX will automatically create the contact. This code works with the AUTO_ADD_DOC_CTCS macro in macros/custom/admissions.

**Enrollment Status**
The enrollment status that a student achieves with the completion of this contact. When you use this field, the contact performs the following functions:

- Creates letters.
- Updates the student’s enrollment status when you run *admstats.*

**Example of Contact Table**

Below are several examples of entries in the Contact table. Only the required fields appear in this example.

**Code A**
VIEWBOOK

**Description A**
Viewbook information packet

**Comm Code A**
LTLB

**Routing A**
O

**Span Waived A**
N

**Reissued A**
N

**ACE Report A**
ltradmit

**Run Code A**
SINGLEI

**Bulk Mail A**
N

**Enrollment Status A**

INQUIRED

---

**Code B**
ACCEPTED
**Description B**
Acceptance letter
**Comm Code B**
LETT
**Routing B**
O
**Span Waived B**
Y
**Reissued B**
N
**ACE ReportB**
ltradmit
**Run Code B**
SINGLEI
**Bulk Mail B**
N
**Enrollment Status B**
ACCEPTED

---

**Code C**
TESTSCOR
**Description C**
Test scores
**Comm Code C**
DOCU
**Routing C**
I

## Contact Record

Contact records trigger the creation of letters, and provide a record of correspondence with individuals or organizations outside the institution. Although Contact records are not part of the initial implementation of Communications Management, they are the essential records within all correspondence tracking. Correct completion of Contact records can facilitate the institution's communication.

Several Contact records may exist for each individual or organization, whether or not it has a Tickler record. While *tickler* automatically generates some Contact records (e.g., for scheduled outgoing letters), you can add others manually, using an entry program or *tickler* (e.g., for a received incoming application).

The Contact record contains the following fields:

**Contact Number**
The system-generated sequence number that uniquely identifies the record.

**ID**
The ID number of the individual or organization that receives the correspondence.

**Tickler Code**

The tickler strategy to which the contact relates (e.g., an ACCEPTED contact is part of the ADM tickler strategy). You can also consider the Tickler Code as the office that created the contact (e.g., ADM for Admissions or REG for Registrar).

**Note:** Offices can use contacts by the same name (e.g., LABEL for a mailing label), since each will have a unique Tickler Code. The different Tickler Codes prevent offices from being able to view or update other office's Contact records, and also prevent an office's employees from accidentally creating or printing another office's letters or labels with their own letters or labels.

**Correspondent ID**
The ID number of the person who performed or sent the contact, or the person or institution you want to link to the contact.

**Example:** If a student informs you that he/she will attend another institution, you could create a Contact record with the resource DECLINED, and maintain the ID number of the selected institution in the Corresponding ID field. You could then produce an ACE report showing all the DECLINED contacts, and the IDs and names of the institutions to which your school has lost prospects.

Another use for this field is to maintain the name of the interviewing faculty member for a contact resource of INTRVIEW.

**Note:** You can perform a query to retrieve the ID number you want to use. When you enter the ID, the associated name appears in the top left corner of the detail window.

**Imaging Document Number**
The system-generated sequence number that Document Imaging uses to identify the contact.

**Add Date**
The date on which the Contact record was added.

**Due Date**
The date on which you want to make the contact (e.g., mail the letter or make the telephone call).

**Appointment Time**
If the contact is an appointment or call, the time at which it is scheduled, based on 24-hour time (e.g., 1:00 p.m. is 1300).

**Contact Completion Date**
The actual date on which the contact occurred.

**Contact Resource**
The eight-character code that uniquely identifies the contact. This code must be a valid value in the Contact table.

**Contact Status**
The status of the contact. Valid codes are:

- C - Completed contacts (i.e., those that are finished)
- E - Expected contacts (i.e., those scheduled to occur)
- I - Interrupted contact (this status may occur because the Tickler step to which the contact corresponds was interrupted)

  **Note:** Only the *tickler* program uses this code.

- V - Voided contacts (i.e., Expected contacts that are no longer required)

**CGC**

A Y/N flag indicating whether *tickler* can reschedule, void or interrupt the contact. When *tickler* schedules a contact, it sets this flag to Y. The *tickler* program also uses this field to determine if it should recalculate a contact's Due Date if an office is late in completing the previously scheduled contact. Therefore, setting this flag to Y ensures that all the contacts that *tickler* schedules will have a consistently even span of time between them.

If you set this flag to N, *tickler* will not recalculate the next contact's Due Date, even if the previous contact is overdue. Instead, the *tickler* program will maintain the Due Date as it had originally computed it. Therefore, setting this flag to N may cause recipients to receive correspondence at irregular intervals of time (e.g., the specified interval designated in the Maximum Span field).

**Enrollment Status**
An applicant's status in the admissions process, based on the completion of this contact resource. For example, if an incoming application changes the status from LEAD to APPLIED, the Enrollment Status is APPLIED. This field is useful for reporting purposes to determine the admissions status a student had when a document arrived or when a letter was sent.

**Admissions Statistics Status**
A code used in conjunction with the Admissions program *admstats*. Valid values are:

- (blank)
- E (Expected)
- C (Completed)
- V (Voided)
- X (Unsuccessful)

When *admstats* reviews a contact status, it updates this field accordingly. For example, if *admstats* reviews a contact with a status of C (and a valid Due Date), and successfully updates the student's admissions status, it also updates the Admissions Statistics Status field to C.

**Note:** If *admstats* is unsuccessful in updating the Enrollment Status field in the Admissions record, it updates the Admissions Statistics Status field in the Contact record to X. The *admstats* program will continue to review and attempt to act on this contact each time it is run. When admstats succeeds in acting on this contact, it updates this field to Completed status.

**Comment**
Any descriptive information about the contact.

**Example of Contact Record**

Below are examples of entries in the ctc_rec. The two examples indicate a completed lead, and an expected acceptance, due on 12/5/96.

**ID A**
20152
**Tickler A**
ADM
**Contact A**
LEAD
**Status A**
C
**Add Date A**
06/19/96
**Due Date A**
12/05/96
**Completed Date A**

12/05/96

**CGC Flag A**

N

---

**ID B**

20152

**Tickler B**

ADM

**Contact B**

ACCEPTED

**Status B**

E

**Add Date B**

06/21/96

**Due Date B**

12/05/96

**Completed Date B**

00/00/00

**CGC Flag B**

Y

## Using WP Macros in Letters

### Introduction

Communications Management uses Word Processing macros in a variety of ways. This section lists and describes the purpose of each of the following types of WP macros:

- Data definition
- Conditional
- Formatting
- Font
- Laser Printer

### Data Definition Macros

Over 200 WP data definition macros exist in CX. Definitions and descriptions of these macros are located in $CARSPATH/macros/custom/ltrwp.

Data definition macros have the format of the following example:

WP_DEF('WP_ACAD_STAT', 'TA')

In this example, *nroff* uses the two-character code TA when it expands the macros during the letter creation process. Therefore, every two-character code must be unique within the *ltrwp* macro file. This code can consist of any combination of upper-case letters and numbers.

If the institution adds additional macros, they should also be defined in this file and the file should be searched to ensure that the two-character code is unique.

### Conditional Macros

WP macro values can also incorporate conditional statements in letters. The conditional macros enable you to print certain information in a letter if your conditions are met. For example, if an applicant has not submitted the application fee, you can insert a line in a letter that asks for the fee to be remitted. For other applicants, you can omit the reference to the fee.

Conditional macros test macro values. Depending on the test results, your letters can have varying contents. Three macros can help you produce this type of customized letter.

#### WP_IF macro

**Example:** You can test for the current status of the student.

For the students with a status of APPLNOFE (those who have applied but not paid an application fee), you can insert a reminder line in a letter. The syntax for the conditional macro is:

```
WP_IF(WP_CUR_STAT,APPLNOFE)\{Your application fee was not received.  Please remit your payment as
soon as possible so the Admissions office can begin processing your application.\}
```

**Example:** You can test for a student's ethnic background, and send copies of correspondence to the Minority Affairs Office for all non-white students. The syntax for the conditional macro is:

```
WP_IF(NOT,WP_ETHNIC,WH)\{cc:  Minority Affairs Office\}
```

#### WP_IFELSE macros

**Example:** You can test for a student's eligibility for in-state tuition. The syntax for the conditional macro is:

```
WP_IFELSE(WP_RESIDENT,OH)\{You have been accepted for WP_PLAN_ENR_SESS WP_PLAN_ENR_YEAR, and your
in-state tuition is $5,000 per semester.\}

WP_ELSE\{You have been accepted for WP_PLAN_ENR_SESS WP_PLAN_ENR_YEAR, and your out-of-state
tuition is $10,000 per semester.\}
```

### WP_IFOR macros

**Example:** You can test for incomplete applications, and create a customized letter for each applicant for whom additional application information is required. The syntax for the conditional macro is:

```
Dear WP_SALUT
Your application to <institution name> is incomplete.  Please ensure that our Admissions office
receives the following information so that we can begin to process your application.
WP_IFOR(WP_REC1,E,WP_REC2,E,WP_REC3,E)\{Reference(s) from:
.sp
.in 5
.nf
WP_IF(WP_REC1,E)\{WP_REC1_NAME\}
WP_IF(WP_REC2,E)\{WP_REC1_NAME\}
WP_IF(WP_REC3,E)\{WP_REC1_NAME\}
.sp\}
.in -5
WP_IFOR(NOT,WP_TRANS1,C,NOT,WP_TRANS2,C)\{Transcripts(s) from:
.sp
.in 5
.nf
WP_IF(NOT,WP_TRANS1,C)\{WP_TRANS1\}
WP_IF(NOT,WP_TRANS2,C)\{WP_TRANS2\}
.sp\}
.in -5
WP_IFOR(WP_ACT,E,WP_GRE,E,WP_TOEFL,E)\{Test Score(s) from:
.sp
in 5
.nf
WP_IF(WP_ACT,E)\{American College Test\}
WP_IF(WP_GRE,E)\{Graduate Record Exam\}
WP_IF(WP_TOEFL,E)\{Test of English as a Foreign :Language\}
.sp\}
WP_CLOSE
Director of Admissions
WP_END
```

## Formatting Macros

The formatting macros enable letter writers to minimize the number of characters they type when creating letters. These macros combine several formatting commands into a single character string. Formatting macros include the following:

**WP_FILL**
Fills lines.

**WP_NOFILL**
Stacks lines.

**WP_INDENT(+5)**
Indents five spaces; may use any integer for number of spaces.

**WP_HYPHENATE**
Hyphenates words at the end of a line.

**WP_NOHYPHENATE**
Turns off the hyphenation feature.

**WP_PAR**
Indents a paragraph five spaces.

**WP_END**

Indicates the end of a letter; no text, spaces or blank lines can follow this code.

### Font Macros

Font macros enable you to change the standard fonts the *vi* editor uses within WPVI. The font macro definitions are located in $CARSPATH/macros/user/fonts. The basic format of the font macros follows:

WP_FONT_SET(font_name,strike_weight,line_length)

**Note:** You must end a fonted block with either WP_FONT_STD or WP_FONT_DEFAULT.

The arguments that you pass with the font macros are:

**font_name**
The font_name argument is required when using WP_FONT_SET. Define font_name as one of the following:

- lp8f (Line Printer 8 point with fixed spacing)
- tr8p (Times-Roman 8 point with proportional spacing)
- cr10f (Courier 10 point with fixed spacing)
- cr10if (Courier 10 point with fixed spacing and italics)
- cr12f (Courier 12 point with fixed spacing)
- cr12if (Courier 12 point with fixed spacing and italics)
- tr12p (Times-Roman 12 point with proportional spacing)
- tr12ip (Times-Roman 12 point with proportional spacing and italics)
- hv14p (Helvetica 14 point with proportional spacing)

**strike_weight**
The strike_weight argument is required. The argument is one of the following:

- B or b (Bold type)
- M or m (Medium type)
- L or l (Light type)

**line_length**
The line_length argument is not required. The default value for line_length is seven inches, but any length is valid. If a line is too long on a page, decrease the value of line_length. The line_length argument (using the .ll *nroff* command) allows you to compensate for the size of a particular point-sized font or place it on a nonstandard (8.5" by 11") paper size.

### Examples of Valid Font Macros

The following is a list of valid WP_FONT_SET usages. The examples assume that the fonts only affect blocks of text and not a word or words within a line.

**Note:** Add a blank line after the WP_FONT_SET command to keep the printer sequence from distorting the size of the current line.

**WP_FONT_SET(hv14p,B)**
Helvetica bolded with line length 7 inches

**WP_FONT_SET(hv14p,M,10)**
Helvetica medium with line length 10 inches

**WP_FONT_SET(lp8f,l,5)**
Line printer light with line length 5 inches

### Examples of Invalid Font Macros

The following are two invalid WP_FONT_SET usages. The examples contain an explanation of the error associated with the macro usage.

**WP_FONT_SET(tr14f,B)**
No font named 'tr14f'.

**WP_FONT_SET(hv14p,t)**
No strike weight for 't'.

### Setting Fonts Without Macros

Occasionally, you may want to change a font by making it bold or italicized without using macros. For example, you cannot use a macro to create a single italicized word on a line with other non-italicized words. To attain this level of control, you must use escape sequences. When your printer detects an escape sequence, it changes the appearance of the printed words according to the instructions that follow the escape sequence.

To produce a bold word, the escape sequence is as follows:

<Ctrl-v><Esc>(s1B

To turn off bolding, the escape sequence is as follows:

<Ctrl-v><Esc>(s0B

In each case, the number that precedes the upper case B (e.g., 2) is the number of words that follow that you want to show in bold. Note also that the <Ctrl-v><Esc> characters display on your screen as ^[.

**Example:** Assume you want to bold the words **Jenzabar CX product** in the following sentence:

The CX product provides administrative software solutions.

Enter the following:

The <Ctrl-v><Esc>(s2BCX product<Ctrl-v><Esc>(s0B provides administrative software solutions.

The following appears on your screen:

The ^[(s2BCX product^[(s0B provides administrative software solutions.

The following appears on the printed paper:

The **Jenzabar CX product** provides administrative software solutions.

To produce selective italics in a letter, you use a similar escape sequence.

To produce an italicized word, the escape sequence is as follows:

<Ctrl-v><Esc>(s1S

To turn off italics, the escape sequence is as follows:

<Ctrl-v><Esc>(s0S

### Laser Printer Macros

Additional macros help you use the laser printer to control the appearance of your letters. These macros are defined in the file macros/user/fonts. The following is a list of these macros and their purposes.

**Note:** These macros require no arguments.

**WP_FONT_DEFAULT**
Use the default laser font.

**WP_FONT_LANDSCAPE**

Set the laser to landscape mode.

**WP_FONT_PORTRAIT**
Set the laser to portrait mode.

**WP_FONT_RESET**
Reset the laser.

**WP_FONT_STD**
Set to standard laser font(same as default).

### Headers and Footers

If you use a header on the next page (or a footer at the bottom of the current page), use the following procedure:

1. Determine the last line on the page for a block that extends over the end of a page and uses fonts.
2. Edit the formatted document.
3. Go to the last line before the page break and add one line before it.
4. On this blank line add WP_FONT_STD.
5. Add two blank lines after this line and place the WP_FONT_SET command on the second blank line.

## Defining Additional WP Macros

### Introduction

If your review of the test_in file in WPVI indicates that the data element that you want in a letter is not being extracted, you may want to define an additional WP macro. The example in this section assumes you want to be able to extract an ethnic code so, if the code has a specific value, the Minority Affairs office on a campus can receive a copy of the letter.

### How to Produce a List of All Macros Created by ACE Reports

Jenzabar CX includes a script you can run from the UNIX shell to produce a list of all the macros created by ACE reports. Enter the following commands to obtain the list, substituting any valid file name for *filename*.out.

```
cd $SCPPATH/util
./ltrwp_list.scp>filename.out
```

### How to Define Additional Macros

Complete the following steps to define an additional WP macro

1. In $CARSPATH/macros/custom/ltrwp, define the macro name that you want to add (e.g., WP_ETHNIC). Use the other macros in the file as examples, and ensure the two-character code you designate is unique.
2. Perform a *tinstall* of the ltrwp file.
3. Modify the ACE report as follows:
   - In the Select statement of the ACE report file, add the desired field (e.g., profile_rec.ethnic_code).
   - Verify the From and Where clauses of the ACE report have references to the required record (i.e., profile_rec). If they do not, add the references.
   - Ensure any field selected is passed through any subsequent Select statements or temporary files.
   - Associate the database field and macro name in the Format section (e.g., WP_MAC(WP_ETHNIC, ethnic_code). The association occurs because WP_ETHNIC is in ltrwp, and ethnic_code is in the database.
4. Perform a *tinstall* of the ACE report.
5. Add WP_ETHNIC to the test_in file in WPVI, along with an example code. Use the other entries in the file as examples.
6. Test the macro in a letter by processing it against the test_in file.
7. Test the ACE report by creating the letter from a contact.

## Using Macros in School Letters

### Introduction

School letters, a type of letter available exclusively to Admissions offices, provide a way for institutions to send letters to key contact personnel at high schools (e.g., counselors or coaches). These letters are dependent on the *ltrschlabl* ACE report to extract the necessary information for letters.

### Use of Relationships in School Letters

You must define a set of Relationship codes in the Relation table to indicate the employees' relationship with their school (e.g., HSGC for guidance counselors or HSFC for football coaches).

You must enter the name of the person as the First Relationship in the First Relationship detail window. The system links this name to the WP_REL_NAME and WP_TITLE_SALUT word processing macros (used for address labels and letter salutations, respectively). If a title is not entered for a school employee, the WP_REL_NAME macro will print the employee's full name (e.g., John Smith), and the WP_TITLE_SALUT macro will print the employee's first name (e.g., Dear John). If a title is entered, the WP_REL_NAME macro will print the employee's title and full name (e.g., Mr. John Smith), and the WP_TITLE_SALUT macro will print the title and the last name (e.g., Dear Mr. Smith).

### Word Processing Macros Created by the *ltrschlabl* ACE Report

The word processing macros created by the *ltrschlabl* report are:

**WP_FIRSTNAME**
The name of the school.

**WP_LASTNAME**
The name of the school.

**WP_FULLNAME**
The name of the school.

**WP_SALUT**
The name of the school

**WP_TODAY**
The date entered as the Date parameter on the Create School Letters screen is expanded before being linked to this macro. For example, a Date parameter value of "01/01/2000" will be expanded to print January 1, 2000.

**WP_LABEL**
The name and address of the school.

**WP_LTR_NAME**
The CX WPVI letter file directory path and letter file name.

**WP_ID**
The ID number of the school, as entered in sch_rec.id..

**WP_DENOM_CODE**
The denomination code of the school as specified in sch_rec.denom. This macro might be used with an If/Then/Else statement to print additional text in letters for certain schools.

**WP_SCH_CTGRY**

The school category code indicating if this institution is a high school, two year college, four year college, or professional school.  This information is stored in sch_rec.ctgry.  This macro might be used with an If/Then/Else statement to print additional text in letters for certain schools.

**WP_SCH_TYPE**
The school type code indicating if the school is a private or public institution, as specified in sch_rec.sch.  This macro might be used with an If/Then/Else statement to print additional text in letters for certain schools.

**WP_COUNTY**
The code indicating the county in which the school resides, according to sch_rec.res_cty.  This macro might be used with an If/Then/Else statement to print additional text in letters for certain schools.

**WP_CEEB**
The CEEB (College Entrance Examination Board) number assigned to the school.  This information comes from sch_rec.ceeb.

**WP_COUNSELOR_ID**
The ID number of the admissions counselor responsible for recruiting students from the school, as maintained in sch_rec.cnslr_id.

**WP_COUNSELOR_NAME**
The name of the admissions counselor responsible for recruiting students from the school.

**WP_CORR_ID**
The ID number entered in the Correspondent ID field (ctc_rec.corr_id) of the school's Contact record used to create the merge file.

**WP_CORR_NAME**
The name associated with the Correspondent ID number.

**WP_SCHEDULE_DATE**
The date entered in the College Day field of the school form is expanded before being linked to this macro.  This date is stored in sch_rec.col_day_date.  For example, a College day date of "01/01/2000" will be expanded to print January 1, 2000.

**WP_APPT_TIME**
The time entered in the Appointment Time field of the school's Contact record (ctc_rec.appt_tm) used to create the merge field.  Although a contact appointment time is entered using military time format, the *ltrschlabl* Ace report converts this value to civilian time before being linked the this macro, (i.e., 1500 = 3:00 p.m.).

**WP_REL_NAME**
The full name of the person who has a specified Relationship record linked to the school receiving a letter.

**WP_TITLE_SALUT**
The title and/or name of the person who has a specified Relationship record linked to the school receiving a letter.

**WP_DEF_ADDRESSEE**
The value entered in the Default Addressee parameter is linked to this macro.

**WP_ADDRLINE1**
The first address line of the school's address is linked to this macro.

**WP_ADDRLINE2**
The second address line of the school's address is linked to this macro.

**WP_ADDRLINE3**

The third address line of the school's address (when an alternate address record is used), is linked to this macro.

**WP_CITY**
The city of the school's address is linked to this macro.

**WP_STATE**
The state code of the school's address is linked to this macro.

**WP_STATE_TEXT**
The full state name of the school's address is linked to this macro.

**WP_ZIP**
The zip code of the school's address is linked to this macro.

**WP_COUNTRY**
The country code of the school's address is lined to this macro.

**WP_COUNTRY_TEXT**
The full country name of the school's address is linked to this macro.

**WP_PHONE**
The phone number of the school is linked to this macro.

**WP_PHONE_EXT**
The phone number extension of the school is linked to this macro.

### Sample School Letter

The following letter uses some of the WP macros to personalize a school letter.

{MERGEFIELD TODAY}

{IF {MERGEFIELD REL_NAME} = "" "{MERGEFIELD DEF_ADDRESSEE}" "{MERGEFIELD REL_NAME}"}
{MERGEFIELD LABEL}

Dear {IF {MERGEFIELD TITLE_SALUT} = "" "{MERGEFIELD DEF_ADDRESSEE}" "{MERGEFIELD TITLE_SALUT}"},

Enclosed with this letter is our latest college catalog.

Sincerely,

{MERGEFIELD COUNSELOR_NAME}
Admissions Counselor

## Using *nroff* Commands to Create User-Defined Macros

### Introduction

A *macro* is a collection of commands that can be executed with one command. If you frequently repeat a series of commands, consider grouping these commands into a macro.

To create a macro, complete the following process:

1. Name the macro. Names consist of a period (.) and two letters, usually capital letters.
2. Define the macro, to ensure the *nroff* process knows what commands are to be executed. Use the .de command to define the macro.
3. After you complete your macro definition, use the .. command to close the macro.

After the commands have been defined and given a name, the macro can be executed any time in the text by inserting the name at the appropriate place within the text file.

### Examples of User Created Macros

In the following example, the macro name is AA. When you insert .AA in the text, *nroff* will insert a blank line (.sp) and temporarily indent five spaces (.ti +5). This configuration is common for indented paragraphs in letters.

```
.de AA
.sp
.ti +5
..
```

### Executing a Macro With the .wh Command

You can automatically execute a macro by using the .wh command. The .wh command requires parameters that indicate to *nroff* which macro to execute, and where to place the results from executing the macro. In the following example, *nroff* will execute the macro HD at the top of the page (i.e., line 0)

```
.wh 0 HD
```

You can assign a negative number for the location parameter. A negative number causes the macro to be executed that many lines from the bottom of the page. In the following example, nroff will execute the FO macro five lines from the bottom of the page. This is a typical structure for a footer.

```
.wh -5 FO
```

When creating header macros and footer macros, Jenzabar suggests you use the single quote (') in place of the period. The use of the single quote will prevent a single word from being left at the bottom of a page. In the following example, *nroff* will automatically insert one blank line, left justify the title, right justify the page number, and insert two blank lines before resuming the printing of text. The % sign causes the program to automatically number the pages of the document, and to insert the page number.

```
.de AA
'sp
'tl 'Appendix A' 'Page %'
'sp 2
..
.wh 0 AA
```

before resuming the printing of text. The "%" sign allows the computer to automatically number the pages of the document and insert the page number in the ‘tl command.

## Skeleton File for Producing Formal Reports

By combining the above examples, the following skeleton file can be created for use in producing formal reports. Explanations of the commands in the example file appear in the next topic.

```
.ll 6.5i
.lt 6.5i
.po 0.75i
.pl 11i
.na
.nf
.nh
.sp 3
.de aa
'sp 2
'tl 'Annual Report''Page %'
'sp 2
..
.wh 0 aa
.de bb
'sp 2
.tl ''March 26, 1997''
'bp
..
.wh -6 bb
.sp 3
.nf
.ce
.ul
Annual Report to Board of Trustees
.sp 2
.ti +5
```

## Explanation of Commands in the Example Skeleton File

The following commands appear in the skeleton file.

| Command | Purpose |
|---|---|
| .ll 6.5i | Set line length to 6.5 inches |
| .lt 6.5i | Set title length to 6.5 inches |
| .po 0.75i | Offset text to .75 inch |
| .pl 11i | Set page length to 11 inches<br>**Note:**<br>You can also set the page to a specified number of lines with the .pl command. For example, .pl 60 defines a page length of 60 lines. |
| .na | Do not adjust margins |
| .nf | No filling or adjusting of output lines |
| .nh | Turn hyphenation off |
| .sp 3 | Leave 3 blank lines |
| .de aa | Define macro aa |
| 'sp 2 | Leave 2 blank lines |
| 'tl 'Annual Report''Page %' | Specify left and right title |
| 'sp 2 | Leave 2 blank lines |
| .. | End macro definition |
| .wh 0 aa | When position n is reached, execute macro aa |

| Command | Purpose |
|---|---|
| .de bb | Define macro bb |
| 'sp 2 | Leave 2 blank lines |
| .tl "March 26, 1997" | Specify centered text |
| 'bp | Begin new page |
| .. | End macro definition |
| .wh -6 bb | When position -6 is reached, execute macro bb; negative value is with respect to page bottom |
| .sp 3 | Leave 3 blank lines |
| .nf | No filling or adjusting of output lines |
| .ce | Center next line |
| .ul | Underline the words on the next line<br>**Note:**<br>To produce a continuous underline (i.e., both words and spaces) use the *nroff* command .cu. |
| .sp 2 | Leave 2 blank lines |
| .ti +5 | Indent next output line 5 spaces, or increment the current indent by 5 spaces for the next output line |

## Using Macros in ACE Reports

### Introduction

ACE report macros have been created to simplify the use of ACE as it extracts the data you need to produce letters.  The macros used in the typical ACE report are as follows:

- LTB_DEFINE
- LTB_FORMAT
- LTB_FORMAT_END
- LTB_GROUP_CTC
- LTB_GROUP_RUN_CODE
- LTB_LAST_REC
- SRT_DEFINE
- SRT_SORT_BY
- WP_MAC
- WP_OUTPUT

### Purpose of ACE Reports

The output from ACE reports drives the rest of the letters and labels production.  The typical Letter/Label ACE report performs these three functions.

1. Selects a group for which to generate letters and/or labels.
2. Retrieves personal data for the selected group.
3. Outputs embedded commands for *adr*, *sortpage*, *splitpage*, *ltrformat*, *nroff* and *lps.*

### Selecting the Group to Process

Contact records (ctc_rec) frequently determine the group to select for letter creation.  ACE reports looking for expected contacts use the parameters passed to the program to selectively load records from the ctc_table and then the ctc_rec.  Other specialized reports may use hard coded criteria to select individuals.  At minimum, the result of the final Select statement must consist of an ID number and a name.  The final Select statement must include additional fields if the institution does not use *adr*, or if the letters being produced include additional pieces of customized personal information.

### Retrieval and Output of Personal Data

Personal data items of interest must appear in the final Select statement of the ACE report. These data items are associated to *nroff* macros by use of the WP_MAC macro.

### Embedded Commands

The macros automatically handle embedded commands.  As long as multiple records are not required within the ACE letter and if standard macros are used by the ACE, then commands for the subsequent processes will be generated as needed.

### ACE Macros

ACE reports recognize and use the following macros:

**LTB_DEFINE**

This macro must appear as the first line of the define section in all letter/label ACE reports. It defines the following seven standard parameters that all letter/label ACE reports have in addition to other definitions:

- param[1] tickler (type character length 4)
- param[2] enterace (type character length 10)
- param[3] resource (type character length 8)
- param[4] commcode (type character length 4)
- param[5] passdate (type character length 10)
- param[6] duedate (type character length 10)
- param[7] printit (type character length 1)

**Note:** Although all of these parameters are passed, the ACE report may not need to use all of them. In particular, the ACE reports that do not depend on contacts do not require all of these parameters.

You can define additional parameters to the ACE report by passing parameters to this macro.

**Example:** param[8] sess (char(4))

param[9] passid (integer)

LTB_DEFINE(sess type character length 4, passid type integer)

This example defines *sess* as the eighth parameter (e.g., a four-character field for tracking the session, FALL, SPNG, or SUMR), and *passid* as the ninth parameter of the ACE report (e.g., an ID for a school or business). However, if you use additional parameters, your letters produced by this type of ACE report cannot be set up to run automatically overnight with the current *lpsrun* script, because you must enter the parameters at each run.

**LTB_FORMAT**

The LTB_FORMAT macro appears as the first line of the format section under the following conditions:

1. You are using *adr.*
2. You require only one record per letter to be processed.

You can use the LTB_FORMAT macro only when you meet both of these conditions.

**Example:** The production of letters to leads (using the ldctc_rec) does not use *adr*, because *adr* requires an ID record, and leads have only Lead records. The ACE report for such a letter is in $CARSPATH/admit/reports/ltrlead and cannot use LTB_FORMAT.

The production of admission status letters requires you to process more than one record per letter. The ACE report for such a letter is in $CARSPATH/admit/reports/ltrstat and cannot use LTB_FORMAT.

The LTB_FORMAT macro performs a variety of functions, including the following:

- Outputs the necessary *lps* header command lines.
- Sets up *sortpage* criteria.
- Produces the output necessary for labels.
- Defines the following *nroff* macros: WP_TODAY, WP_SALUT, WP_LABEL, WP_FIRSTNAME, WP_LASTNAME, and WP_FULLNAME.
- Expands to an *on first page* and an *on every row* clause.

The syntax for the macro follows:

LTB_FORMAT(runcode, wp_drawer, id_num, name, resrc)

In this syntax, the parameters are as follows:

- runcode
  - A string (enclosed in quotation marks (“”)) containing the runcode for *adr* to use. Typically, you do not specify the runcode if the ACE is using the Contact table as input, or if you are generating different types of letters with different run codes in a single ACE report run. In this case, you can specify the runcode to *adr* by using the LTB_GROUP_RUN_CODE macro. If you do not use this macro, enter a comma in the parameter string. An example of the runcode parameter usage is:

  LTB_FORMAT(,”admissions,”id_no, name, resrc).
- wp_drawer
  - A string (enclosed in quotation marks (“”)) specifying the wp FileCabinet location of the form letter.
- id_num
  - An ACE variable containing the ID number for the group to be printed. This ID number passes to *adr* in an embedded *adr* command.
- name
  - The ACE variable containing the name. It determines the value of the WP_FIRSTNAME, WP_LASTNAME, and WP_FULLNAME *nroff* macros.
- resrc
  - Either a string (enclosed in quotation marks (“”)) or an ACE variable that specifies the name of the letter/label to be produced. The name of this variable must exactly match the name of the form letter located in the Word Processing File Cabinet (e.g., admissions).

**LTB_FORMAT_END**

The LTB_FORMAT_END macro closes the *nroff* macro definition section. It appears after the LTB_FORMAT macro and all WP_MAC macros or the equivalents of the WP_MAC. The LTB_FORMAT_END macro signals the end of the individual’s personal data to *nroff*. It also generates embedded *informer* commands to update Contact records and Tickler records, depending upon the parameters passed.

The syntax for the macro follows:

LTB_FORMAT_END(update_ctc, update_tick)

In this syntax, the parameters are as follows:

- update_ctc
  - A Y/N flag that, if Y, causes an ISQL statement to be generated to complete the status and completion date of the Contact record.
- update_tick
  - A Y/N flag that, if Y, causes an ISQL statement to be generated to set the Next Review Date in the Tickler record to be the current date. This process causes *tickler* to review the Tickler record the next time it is run.

**LTB_GROUP_CTC**

The LTB_GROUP_CTC macro is optional. It has two parameters, as follows:

- The ACE variable that holds the name of the letter/label to be generated (i.e., the resource code or other designation).
- The number of labels produced; if this parameter is missing, the macro uses the group count.

If the macro exists, it is expanded into an *after group of* clause to produce a total label. Use this macro if different types of letters/labels might be produced in one ACE run. If you use the SRT_ macros, the total labels that are produced could appear anywhere in the lps output.

**LTB_GROUP_RUN_CODE**
The LTB_GROUP_RUN_CODE macro is useful if different types of letters/labels with different runcodes could be produced in one ACE run. It has one parameter which is the name of the ACE variable that holds the runcode value. This macro should not be used if the runcode is specified in the LTB_FORMAT macro.

**LTB_LAST_REC**
The LTB_LAST_REC macro appears as the last line in the Format section of the ACE report. In terms of ACE, it expands to an *on last row* clause and produces exit or end commands for *sortpage*, *adr*, *nroff*, *informer* and *lps*. It has no parameters.

**SRT_DEFINE**
The SRT_DEFINE macro appears in the define section of the ACE report, following the LTB_DEFINE macro. You must use this macro if you use the SRT_SORT_BY macro.

**SRT_SORT_BY**
The SRT_SORT_BY macro specifies the sort criteria desired so the LTB_FORMAT macro automatically generates embedded *sortpage* commands.

The syntax for the macro follows:

SRT_SORT_BY(value1 [descending], value2 [descending], ...)

**Example:** For bulk mailings that use ltradmbulk to extract data, the correct syntax is as follows: SRT_SORT_BY(ADR_ZIP_VALUE)

In this syntax, the parameters are:

- value
  - An ACE variable from the last Select statement or one of the following ADR_VALUE macros:
    - ADR_ID_VALUE
    - ADR_NAME_VALUE
    - ADR_LINE1_VALUE
    - ADR_LINE2_VALUE
    - ADR_LINE3_VALUE
    - ADR_CITY_VALUE
    - ADR_ST_VALUE
    - ADR_ZIP_VALUE
    - ADR_CTRY_VALUE
    - ADR_PHONE_VALUE
    - ADR_PHONE_EXT_VALUE
    - ADR_COUNT_VALUE
- descending
  - An optional keyword that will cause the sort on the associated value to be descending instead of ascending.

**WP_MAC**

The WP_MAC macro associates personal data fields with WP macros. It appears after the LTB_FORMAT macro and before the LTB_FORMAT_END macro.

The syntax for the macro is:

WP_MAC(WP_MACRO, value, flag, keep)

In this syntax, the parameters are:

- WP_MACRO
  - Any of the WP_macros defined in the macros/users/ltrwp file. The *nroff* process can handle only two character macro names. The ltrwp file defines longer (and therefore clearer) WP_macros that are expanded in the ACE report and the form letter to the defined two character sequence, if the desired format is stdlps. Otherwise the WP_ portion of the name is stripped off and the rest of the name is used.
- value
  - Usually an ACE report variable from the last read statement. However it may also be an ADR_VALUE macro, a double-quoted string or anything else that expanded will produce an ACE acceptable syntax. The ADR_VALUE macro expands to an ACE print statement. It expects any one of the ADR_VALUE parameters that were listed in the SRT_SORT_BY macro section.
- flag
  - Defines the type of expansion, as follows:
    - <not specified>, CLIP (value expands in ACE to print value clipped)
    - NP, NO PRINT (value expands in ACE to print a blank space)
    - NC, NO CLIP (value expands in ACE to print without clipping)
    - <alt_value> (if value is not blank, then print value clipped; otherwise, print alt_value)
- keep
  - If you specify a value of KEEP, the command is generated to instruct *nroff* to keep the value together across multiple lines when the value is merged into the form letter.

**WP_OUTPUT**

The WP_OUTPUT macro must appear as the only line in the output section in all letter/label ACE reports. It defines the page margins and offsets.

**Note:** ACE ignores a right margin command, even if it is included with this macro.

## Creating User-Defined ACE Reports

### Introduction

Selecting a group of individuals from the database and merging the name list with a letter format, requires a specific ACE report with specific text and selection criteria. This section describes the process of creating a user-defined ACE report.

The setup of these reports is an important task in Jenzabar CX implementation, since users must run ACE reports to extract data whenever they want to produce letters that require database information. The standard implementation does not usually satisfy all the correspondence needs of every institution. Additional needs can be met by modifying standard CX ACE reports during implementation, or by users at the institution creating their own ACE reports. User-defined ACE reports are created in the *wpreports* drawer using WPVI.

**Note:** Creation of custom ACE reports requires advanced knowledge of ISQL and the Jenzabar CX database. It frequently includes collaboration between an institution's computer center and the end-users who need to vary the information they include in reports.

### Sample ACE Reports

Sample ACE reports that contain the basic format appear in the WPVI *wpreports* drawer. Typically, users must only modify the Select section, but may also change the Format section.

**Note:** For most efficient letter production, Jenzabar recommends you use Contact records. If Contact records are used, the standard ACE reports perform the selections based upon the scheduled contacts. In these cases, little or no modification to ACE reports is required to determine the selection of individuals. Modification may, however, be required to determine the data extracted for each individual.

### How to Run User-Created ACE Reports

The following procedure enables users to create and run ACE reports:

1. Identify which ACE report corresponds to the letter you wish to send.

   **Note:** Menu options assume the report appears in the *wpreports* drawer; the letter appears in the *letters* drawer.

2. Modify the Select statement to extract the desired information.
3. Modify the Where clause to assure the proper selection criteria for your merge data.
4. Review and modify the Format section as needed.
5. Translate the ACE file to prepare it for the extraction process.
6. From the Utilities: Letters/Labels and Reports menu, select Create WPVI Letters/Labels, then respond to the screen prompts with the name of your ACE report and the other parameters required. Since the program assumes the report resides in the *wpreports* drawer, enter the name of the report as in the following example:

   admissions/*reportname*

7. Execute the report, which extracts the required data, merges it with the letter, and holds it in the spooler file for use by LPS.
8. Run LPS to print your letters or labels.

## Modifying the Select Section

The Select section contains the selection statement for the extraction program. It must include all fields that are required by the letter and associated with the macros included in the letter text.

You can use the same ACE report as long as the fields to extract are the same; however, you may need to change the Where clause to update selections.

### The Format Section

You must include the LTB_FORMAT macro in the FORMAT section of the ACE report. The macro may appear as follows:

```
format - LTB_FORMAT("JOINT","development",id_no,name,)
```

Since the LTB_FORMAT macro statement is already in the sample ACE report, review it to determine if any of the parameters within the parentheses need to be changed.

Parameters for the LTB_FORMAT macro are as follows:

- JOINT
  - The runcode for the *adr* program.
- "development"
  - The name of the FileCabinet in WPVI containing the letter to be merged with this ACE report.
- id_no
  - The field in the selection containing the ID number of the individual; although the value is usually id_no, it can also be any of the other ID fields (e.g., alum_id or donor_id).
- name
  - The field in the selection containing the name of the individual selected (e.g., *stu_name*.

If you need to change any of these parameters, modify them before running the merge program.

**Note:** If you use a certain ACE report frequently and modify it in similar ways each time, you can rename and save it in under various names. This would save you from having to modify the report for successive runs.

## Setting Up Letters That Use Sourcing

### Introduction

*Sourcing* is a technique of reading one letter into another. This procedure is helpful if, for example, the Admissions office wants to schedule and send a departmental letter to all applicants, but the originating department will vary depending on the applicants' intended majors. Sourcing is particularly helpful if you use *tickler* to schedule contacts, because *tickler* does not access information about a student's intended major and therefore cannot schedule a specific letter contact from the correct department. The *tickler* program can schedule a generic departmental contact or letter, and sourcing then determines the correct department information to customize the letter for each recipient.

### Procedure

Follow these steps to set up and use a simple sourcing letter:

1. Use *tickler* to schedule a generic contact (e.g., DEPTLETT for a departmental letter).
2. Create a generic letter in the WPVI letters drawer by the same name (e.g., DEPTLETT). The letter contains only the following information:
   - Date
   - Inside Address
   - Salutation
   - WP_IF statements to define which departmental letter is to be sourced into the generic letter
3. Create specific letters for each department that contain the following information, and ensure that the departmental letters have unique names (e.g., WP_ART_DEPT_LTR):
   - Body of letter
   - Closing
   - Name of sender (e.g., department chairperson)

   **Note:** Create the letters in the FileCabinet of the group creating the letters (e.g., admissions).
4. Modify the ACE Report that extracts the information for the letters to link the Major code to the departmental letter names.
5. Add the WP_ART_DEPT_LTR macro to the $CARSPATH/macros/custom/ltrwp file and assign it a unique two-character value.

### Example of Generic Letter for Sourcing

The following example shows a generic letter you can use for sourcing. In this example, the line .so WP_DEPT_LTR\} expands to .soARTDEPT\}. The ACE report expands to find the value of the letter name.

```
WP_HEAD(10),WP_SALUT)

WP_IF(WP_MAJOR1,GRAP)\{
.so WP_DEPT_LTR\}
WP_IF(WP_MAJOR1,BIOL)\{
.so WP_DEPT_LTR\}
WP_IF(WP_MAJOR1,CHEM)\{
.so WP_DEPT_LTR\}
WP_END
```

**CAUTION:** Observe the following conditions to ensure the letters process correctly:

- The dot command (i.e., .so) must be flush left against the left margin for each WP_IF statement.
- The WP_END macro must also reside within the main DEPTLETT letter, and *not* each individual letter.
- The major codes (e.g., BIOL) *must* exactly match the major code in the Major table.

### Example of Departmental Letters for Sourcing

The following examples show sample departmental letters you can use for sourcing:

**Note:** You cannot use WP macros (e.g., WP_PLAN_ENR_YEAR or WP_CLOSE) within departmental letters. If you want to use these macros, they must appear in an opening or closing paragraph within the DEPTLETT letter.

**ARTDEPT**

```
Along with the rest of the faculty within the Art Department, I would like to congratulate you on
your acceptance into CARS College.  The department offices are located in Sims Hall.  We look
forward to seeing you on campus.

Sincerely,


Dr. Ellen Williams
Department Chairperson
```

**BIODEPT**

```
Along with the rest of the faculty within the Biology Department, I would like to congratulate
you on your acceptance into CARS College.  The department offices are located in Weston Hall.  We
look forward to seeing you on campus.

Sincerely,


Dr. David Bachman
Department Chairperson
```

**CHEMDEPT**

```
Along with the rest of the faculty within the Chemistry Department, I would like to congratulate
you on your acceptance into CARS College.  The departmental offices are located in Davis Hall.
We look forward to seeing you on campus.

Sincerely,


Dr. Alice Jordan
Department Chairperson
```

### Example of ACE Report for Sourcing

The following example shows required changes to the *ltradmit* ACE report, which extracts information for the example departmental letters:

**Note:** Remember the following when modifying ACE reports:

- The major codes (e.g., BIOL) *must* exactly match the major codes in the Major table.
- The major code field (e.g., adm_rec.major) must be passed through the ACE report to the final Select statement.

```
Define section:

variable      var_dept       char(8)

format section:

if (major = “GRAP” or major = “INTR” or major = “ART”) then let var_dept = “ARTDEPT”

if (major = “BIOL” or major = “IMMU”) then let var_dept = “BIODEPT”

if (major = “CHEM” or major = “CENG”) then let var_dept = “CHEMDEPT”

WP_MAC(WP_DEPT_LTR,’WP_MAKEPATH(“admissions”,”letters”,var_dept clipped)’,NP)
```

**Note:** The following list contains the codes and descriptions in the order they appear in the above ACE report Define section:

- GRAP (Graphic Design)
- INTR (Interior Design)
- ART (Art)
- BIOL (Biology)
- IMMU (Immunology)
- CHEM (Chemistry)
- CENG (Chemical Engineering)

## Setting Up the Name/Address Management Program

### Introduction

CARS uses the term *adr* to describe both the program and the process used to create addresses for mailings based on your criteria. This process includes the mailing address of the individual as well as how the individual's name appears on an envelope or salutation of a letter.

The *adr* program allows you to:

- Address mail to an individual at different locations based on:
  - Time of year or date
  - Process (e.g., billing)
  - Preference of the individual
- Use longer addresses in the individual's record
- Use various labels and salutations for an individual, including:
  - Formal or informal (e.g., nicknames)
  - Joined individual records
  - Preferences of the individual
- Omit undesired mail because of:
  - An incorrect address
  - A deceased individual
  - An individual with item holds
  - Duplicate items attributed to the same individual

### Setting Up Multiple Mailing Addresses

An institution can use the Alternate Address record (*aa_rec*) to define an address which may replace the permanent address in an ID record (*id_rec*). There are various ways to determine if an alternate address is appropriate in a given situation.

One way of determining when to use an alternate address is the date. If the current date falls between the dates given in an aa_rec, an institution can use the alternate address at that given time to reach that certain student. If more than one address is possible during those dates for a student, the Alternate Address table (aa_table) determines address has the highest priority from the addresses listed in the aa_rec and the permanent address given in the id_rec. The following shows a sample situation:

```
 File    AA Code  Begin Date    End Date    Priority   Yearly
------   -------  ----------    --------    --------    ------
id_rec    PERM                                 20
aa_rec    BUS     00/00/0000   00/00/0000     30         N
aa_rec    SUMR    06/01/1998   08/22/1998     11         Y
```

For this example, an ID has three addresses. The permanent address (PERM) from the id_rec has no date limitations and has a priority of 20 (defined in the aa_table). The business address (BUS) from the aa_rec has no date limitations since the dates in the aa_rec are zero-filled, and has a priority of 30. The summer address (SUMR) from the aa_rec is only valid between June 1 and August 22 and has a priority of 11. Since the value for Yearly is set to "Y", this summer address is valid in 1998 and every year thereafter. If the mailing date is between June 1 and August 22, the system will choose the summer address since it has the highest priority (lowest number). Otherwise, the system will choose the permanent address because its priority is higher than the business address. Other alternate addresses could include a campus mailbox number, a local off-campus address, special holiday addresses, etc.

An institution can also use the Alternate Address record (aa_rec) to determine which mailing (e.g., registration information, gift receipts, or student statements) goes to which address. There is an *adr* record (adr_rec) for overriding the type of address record that the institution will use for a given process. For example, the institution can use the business address in place of the permanent address in the id_rec and the summer address in the aa_rec if the process being run is gift receipts. So, an institution can send a bill to the student's permanent address, an on-campus activity notice to the student's campus box, and a fall registration packet to his or her summer address. To accomplish this, the institution would add an adr_rec with the following values:

| | Run | AA | | Use | Use | Label | Salut | Join | Name | |
|---|---|---|---|---|---|---|---|---|---|---|
| ID | Code | Code | Prior | Rel | Prim | Style | Style | Rel | Line | Dup |
| 0 | GIFTRCPT | BUS | 0 | | | | | | 1 | Y |

The zero ID number means that this record is valid for any ID. The code "GIFTRCPT" identifies the gift receipt process, and "BUS" identifies the business address. The other values will be explained later.

Another way that an institution can use the Alternate Address record (aa_rec) to address mail to varying locations for an individual is by the preference of that individual. This takes place when the individual requests that the institution send mail to a specified address instead of any other address. For example, ID number 17 can have all mail sent to his or her business address by adding an adr_rec with these values:

| | Run | AA | | Use | Use | Label | Salut | Join | Name | |
|---|---|---|---|---|---|---|---|---|---|---|
| ID | Code | Code | Prior | Rel | Prim | Style | Style | Rel | Line | Dup |
| 17 | | BUS | 0 | | | | | | | |

The "17" denotes the ID for which this record is in effect, the blank run code means that this record is valid for any process, and "BUS" identifies the business address. If the individual still uses the summer address when the business address is in effect, two records must exist as follows:

| | Run | AA | | Use | Use | Label | Salut | Join | Name | |
|---|---|---|---|---|---|---|---|---|---|---|
| ID | Code | Code | Prior | Rel | Prim | Style | Style | Rel | Line | Dup |
| 17 | | SUMR | 0 | | | | | | | |
| 17 | | BUS | 1 | | | | | | | |

You must specify a priority to determine which of the two records *adr* will evaluate first. The summer address is now set for *adr* to use when it is in effect. When it is not, then *adr* will use the business address. If the individual only prefers to receive student statements at the business address with all other mailings delivered to their normal locations, then the previous example would be modified as follows:

| | Run | AA | | Use | Use | Label | Salut | Join | Name | |
|---|---|---|---|---|---|---|---|---|---|---|
| ID | Code | Code | Prior | Rel | Prim | Style | Style | Rel | Line | Dup |
| 17 | STMT | BUS | 0 | | | | | | | |

If more than one of the conditions described above exists at the same time, your institution must establish the order in which it will evaluate *adr* records. The *adr* program considers:

1. The individual's preference for a specific process
2. The individual's preference for mailings in general
3. The process that *adr* is running
4. The type of address available

The following example summarizes this order:

| Files | Conditions |
|---|---|
| adr_rec | ID matches and Run Code matches |
| adr_rec | ID matches and Run Code is blank |
| adr_rec | ID is zero and Run Code matches |
| aa_rec and | Highest priority address type satisfying |
| id_rec | Date limitations |

## Setting Up Long Addresses

If some of your IDs have addresses that are too long for the standard id_rec, the aa_rec can be used. The aa_rec has more space available for address lines which reduces the need to use non-standard abbreviations. Enter the abbreviated address into the id_rec with a code of ABBR and add an aa_rec with the address in full. This id_rec would contain the following:

| Files | AA Code | Begin Date | End Date | Priority | Yearly |
|---|---|---|---|---|---|
| id_rec | ABBR | | | | |

## Setting Up Multiple Labels and Salutations

Whenever you print the name of an individual, your institution must consider the format or style as well as whether it will appear in a label or a salutation. An institution must also decide whether to include titles or professional suffixes. There are formal and informal styles which the institution can determine from the name field in the id_rec according to the following format:

| | Style | Type | Use Title | Use Suffix | Format |
|---|---|---|---|---|---|
| | Formal | Label | N | N | Fullname |
| | Formal | Label | Y | N | Title Fullname |
| | Formal | Label | N | Y | Fullname, Suffix |
| * | Formal | Label | Y | Y | Fullname, Suffix |
| ** | Formal | Label | Y | Y | Title Fullname |
| | Informal | Label | N/A | N | Fullname |
| | Informal | Label | N/A | Y | Fullname, Suffix |
| | Formal | Salutation | N/A | N/A | Title Lastname |
| | Informal | Salutation | N/A | N/A | Firstname |
| | | | | | Middlename |
| | | | | | (if First is an initial) |
| | | | | | (Initial Middle-Initial) |
| | | | | | (if both are initials) |

*The id_rec has a suffix and may or may not have a title.
**The id_rec does not have a suffix but has a title.

If your institution does not desire any of these formats for a particular individual, you can add an Addressee record (addree_rec) to override the normal cases. For example, if the name “Jack Martin” in the id_rec (ID number 100) is always to appear as “Mr. Jack Martin” on a label, you would add the following addree_rec:

| Primary | Secondary | Style | Type | Title | Line |
|---|---|---|---|---|---|
| 100 | 0 | Formal | Label | MR | Jack Martin |

Notice that *adr* enters addressee lines in the same type of format as names in the id_rec. Many times the institution needs to address two individuals on the same label or salutation. For example, if “Smith, John R.” and “Smith, Mary S.” are married, then they should receive one mailing addressed to both of them. The *adr* program will join the two names according to the following rules:

|  | Style | Type | Use Title | Use Suffix | Format |
|---|---|---|---|---|---|
|  | F | L | N | N | First1 and First2 Last1 |
|  | F | L | Y | N | Title1 and Title2 Full1 |
|  | F | L | N | Y | First1 and First2 Last1, SuffixJ |
| * | F | L | Y | Y | First1 and First2 Last1, Suffix** |
|  | F | L | Y | Y | Title1 and Title2 Full1 |
|  | I | L | N/A | N | First1 and First2 Last1 |
| * | I | L | N/A | Y | First1 and First2 Last1, SuffixJ |
|  | F | S | N/A | N/A | Title1 and Title2 Last1 |
|  | I | S | N/A | N/A | First1 and First2 |

*Both id_recs have the same suffix and may or may not have titles.
** One of the id_recs does not have a suffix but both have titles.

**Note:** SuffixJ refers to the case where both id_recs have the same suffix. If each id_rec has a different suffix, *adr* will use the 2-line format described in the paragraph following the next example.

These rules only apply to individuals whose last names are the same. If you wish to join "Smith, John R." and "Doe, Jane L.", the following applies:

|  | Style | Type | Use Title | Use Suffix | Format |
|---|---|---|---|---|---|
|  | F | L | N | N | Full1<br>Full2 |
|  | F | L | Y | N | Title1 Full1<br>Title2 Full2 |
|  | F | L | N | Y | Full1, Suffix1<br>Full2, Suffix2 |
| * | F | L | Y | Y | Full1, Suffix1<br>Full2, Suffix2 |
| ** | F | L | Y | Y | Title1 Full1<br>Title2 Full2 |
|  | I | L | N/A | N | Full1<br>Full2 |
|  | I | L | N/A | Y | Full1, Suffix1<br>Full2, Suffix2 |
|  | F | S | N/A | N/A | Title1 Last1 and Title2 Last2 |
|  | I | S | N/A | N/A | First1 and First2 |

*Both id_recs have suffixes and may or may not have titles.
**One of the id_recs does not have a suffix but both have titles.

If the last names are the same and your institution desires two name lines, you can set the name lines value in the *adr* record (adr_rec) to "2" instead of "1". The *adr* program will then format the names as if the last names were different. It is also necessary to identify which ID *adr* will print first. The join_prim field in the Relation record determines which ID *adr* will print first. If the value is "Y", *adr* will print the prim_id first. If the value is "N", *adr* will print the sec_id first. Therefore, in this example, if John R. Smith's ID is the prim_id, then the join_prim field in the Relation record would be "Y". If you are not joining a particular ID, you can add an *adr* record with adr_rec.id set to the ID number and the adr_rec.join_rel field set to "NONE".

If your institution uses the ID record as a "joint" record, it should use a title code that has a joint value of "Y". You could use a "joint" ID record for parents of students, constituents, or any other persons who receive joint mailings. If the parents of a student were "Mr. and Mrs. John Doe", you could add an ID record with the name "Doe, John" in the name field and a title code of "MRMS" for "Mr. and Mrs." (title_table.joint would be set to "Y"). In this case, *adr* would address every mailing sent to that ID jointly and there is no need to have an ID record for the spouse. However, since the rules for formatting names and addresses state that *adr* only uses the title code for formal mailings, *adr* would address an informal mailing to “John Doe” on the label and “John” on the salutation line. This is why you should set the title_table.joint field to “Y”. It does not allow the informal type of formatting for records that use it. It is not necessary, then, to add *adr* records to force formal mailings to this ID. It is possible to get an informal style using Addressee records. The Addressee records would have “John and Jane Doe” for the label and “John and Jane” for the salutation.

**Note:** The secondary spouse’s first name will never appear unless you add an Addressee record for the informal style.

It is important to remember that *all* mailings to this ID will be joint. If there are any cases where your institution would send mail to only one of the persons in the joint ID record, you must add two separate ID records.

An additional option for labels is the possibility of adding professional suffixes (e.g., MD, Ph.D., Esquire). If an ID has a code in its suffix field, *adr* will use the corresponding suffix text. Otherwise, processing is the same as described earlier. To include suffixes, set the adr_rec.use_suffix field to “Y”. The *adr* program also provides the adr_rec.use_title field to control the processing of titles. Usually this field is “Y”; however, an “N” setting will prevent titles in formal labels. Salutations are not influenced by adr_rec.use_suffix or adr_rec.use_title.

You may add Addressee records to override any of the *adr* name joining rules. If John was known as Jack and wished to be informally addressed as Jack when joined with his wife, you would add the following addree_recs:

| Primary | Secondary | Style | Type | Line |
|---|---|---|---|---|
| 10 | 11 | I | L | Smith, Jack and Mary |
| 10 | 11 | I | S | Jack and Mary |

You can also use the adr_rec to indicate that *adr* should join all the selected ID records having a specific type of relationship. In the following example, an institution is sending a newsletter to all donors:

| ID | Run Code | AA Code | Prior | Use Rel | Use Prim | Label Style | Salut Style | Join Rel | Name Lines | Dup |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | NEWSLETT | | 0 | | | | | HW | 1 | N |

If either John or Mary Smith was a donor, they would receive this mailing addressed to both of them since the “Join Rel” value specifies the husband-wife relationship.

Situations may also exist where an ID has a preference to be addressed formally or informally in mailings. You can accommodate by adding an adr_rec as follows:

| ID | Run Code | AA Code | Prior | Use Rel | Use Prim | Label Style | Salut Style | Join Rel | Name Lines | Dup |
|---|---|---|---|---|---|---|---|---|---|---|
| 17 | | | 0 | | | I | I | | | |

The significant values here indicate the label and salutation style both should be informal whenever ID number 17 receives mailings.

### Setting Up Mail Deliveries to Relations

In some situations you may want to send mail to a relation of an ID instead of the ID that was selected. An example might be sending a letter to the parents of a student. The following adr_recs would accomplish this:

| ID | Run Code | AA Code | Prior | Use Rel | Use Prim | Label Style | Salut Style | Join Rel | Name Lines | Dup |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | PARNTLET | | 0 | PC | | | | HW | 1 | Y |

The “Use Rel” code indicates that the desired ID to whom you are sending mail has a parent-child relationship with the selected ID. The “Join Rel” code indicates that the address should be a joint address using the husband-wife relationship, if *adr* can find it. Once *adr* has found the relation you are sending mail to, *adr* will apply the normal rules to determine which address it should use for that ID since the AA code was left blank.

## Eliminating Mail for Specific Individuals

### id_rec flags

You can indicate situations where a mailing might be inappropriate through two flags in the id_rec.

- When you set the Correct Address flag to “N”, the address in the id_rec is not valid and should not be used.
- When you set the Deceased flag to “Y”, the program will not send mailings to an individual or to an individual’s estate.

### Hold record

You can put “*holds*” on an individual for specific purposes. You can add a Hold record (hold_rec) for an ID indicating the action you are trying to prevent. This may include the sending of certain types of letters, as well as student statements or other mailings. The *adr* program will check for a hold on an ID before sending the mail item and will omit the mailing if the hold is absolute.

### Duplicate field

Another case where an institution should not send a piece of mail is when the same ID would receive a duplicate. This could happen in the selection process itself, or more likely where *adr* has joined an ID with another ID from a previous piece of mail in the same run. You can eliminate these cases if you set the “Dup” field to “N” in the adr_rec, as in the newsletter example that you sent to all donors.

| ID | Run Code | AA Code | Prior | Use Rel | Use Prim | Label Style | Salut Style | Join Rel | Name Lines | Dup |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | NEWSLETT | | 0 | | | | | HW | 1 | N |

If John and Mary Smith were both donors, *adr* could have selected both of them to receive this mailing. Therefore, when *adr* selected John Smith, a newsletter would be sent to them jointly; and when Mary Smith was selected, another newsletter would be sent. However, since the duplicate flag is set to “N”, the system will not send the second newsletter because Mary Smith already received the newsletter.

**Note:** The *adr* program will also send e-mail to the operator listing every ID for whom mail was not sent for any of these three reasons.

## Maintaining Previous Addresses

With Alternate Address records, you can maintain an unlimited number of previous addresses for any ID. Add the previous address with the range of dates for which it was valid (or change the dates of a current aa_rec).

**Note:** Make sure the yearly flag is set to “N” for all previous addresses.

The first example could be modified if two previous permanent addresses were known:

| File | AA Code | Begin Date | End Date | Priority | Yearly |
|---|---|---|---|---|---|
| id_rec | PERM | | | 20 | |
| aa_rec | BUS | 00/00/0000 | 00/00/0000 | 30 | N |
| aa_rec | PERM | 00/00/0000 | 03/10/1998 | 20 | N |
| aa_rec | PERM | 03/11/1998 | 07/17/1998 | 20 | N |
| aa_rec | SUMR | 06/01/1998 | 08/22/1998 | 11 | Y |

The current permanent address is located in the id_rec, so that processes not using *adr* will have a current address.

## Setting Up Zero Priority Addresses

You may want to maintain addresses that *adr* should not use under normal circumstances.  You can accomplish this by creating an Alternate Address code with a priority of 0 in the Alternate Address table (aa_table).  When you assign this code to an Alternate Address record, *adr* will ignore it.  If you desire the Alternate Address record, an *adr* record (adr_rec) specifying the aa_code will produce it.

## Using *adr* Runcodes with Relationship Codes

### Introduction

Although *adr* runcodes usually simply designate whether names and addresses on correspondence should be single or joint, or formal or informal, you can also use runcodes with relationship codes to further automate your contacts/letters.

### Example Use for Runcodes

You can use runcodes to automate the following type of communication.

A medical school admissions office wants to send a letter to the Health Careers advisor at an undergraduate institution, thanking the advisor for sending a recommendation for an applicant.  In addition, the medical school admissions office wants to be able to notify the applicant's Health Career advisor of the applicant's acceptance into medical school, and ultimately of the applicant's graduation from medical school.

You create a new relationship code and a new runcode, so the addition of a Contact record for the applicant will create a letter for the advisor.

### Procedure to Set Up the Example

Complete the following steps to create the correspondence described in the example:

1. Make the following single entry in the Relationship table:
   - Code is STHA
   - Description is Student/Health Advisor
   - Maintenance is N
2. Make the following single entry in the ADR table:
   - Code is ADVISOR
   - Description is Health Careers Advisor
3. Make the following single entry in the ADR record:
   - ADR Run Code is ADVISOR
   - Address Code is PERM
   - Relationship Usage Code is STHA
   - Use Primary ID is N
   - Join Relationship Code is left blank
   - Label Style Code is F
   - Salutation Style Code is F
   - Number of Name Lines is 1
   - Use Suffixes is Y
   - Use Title is Y
   - Allow Duplicates is N
   - Address Priority is 40
4. Make the following single entry in the Contact table:
   - Code is RECTHANK
   - Description is Thanks for Recommendation
   - Tickler is ADM
   - Comm Code is LTLB
   - Routing is O
   - Span Waived is N

- Reissued is N
- ACE Report is *ltradmit*
- Run Code is ADVISOR
- Bulk Mail is N
- Document Tracking Type is N
- Enrollment Status is left blank

5. If the advisor does not already have an ID record on the institution’s database, make a single entry in the ID record for the advisor, using the advisor’s work/school address in the address field. You can add this record from the Relationship detail window when creating the Relationship record.
6. Make the following single entry in a Relationship record that is linked to the student’s ID:
   - Rel ID is the advisor’s ID
   - Name is the advisor’s name (the system inserts this name when you enter the Rel ID)
   - Code is STHA
   - Begin Date is current date
   - End Date is blank
   - Print flag is N
   - Maintain flag is N
7. Make the following single entry in a Contact record that is linked to the student’s ID:
   - Contact is RECTHANK
   - Status is E
   - Expected Date is the date you want to send the letter
   - Remaining fields are blank or contain default values
8. Create a letter in WPVI letters drawer that is similar to the following:

   **Note:** When this letter is printed, it contains the inside address and salutation for the advisor, but the macros WP_FULLNAME and WP_FIRSTNAME assume the name of the applicant.

```
WP_HEAD(10,(WP_SALUT)

Thank you for your recommendation for WP_FULLNAME.  Both our office and WP_FIRSTNAME appreciate
the time you have spent completing the recommendation form.  I will be sure to notify you if
WP_FIRSTNAME is accepted.  If you would find it helpful, I can also send you a full list of
students from your institution who have applied to our school.  If you know of other well-
qualified students from your institution, we would appreciate your referring them to CARS Medical
College.

WP_CLOSE
WP_COUNSELOR_NAME
WP_END
```

## Disabling the Letter Format Program in a Menu Option

### Introduction

Most letters require the use of the Letter Format program (*ltrformat*) to serve as the front end processor for Jenzabar CX letter writing. However, if your institution uses explicit reads in any letters (i.e., if any letters use the *nroff* command .rd), you must create a menu option for processing these letters that bypasses the *ltrformat* program.

### How to Disable the Letter Format Program in a Menu Option

To disable *ltrformat* in a menu option, add the lines that appear in bold type to the menuopt file.

**Note:** The example below is from $CARSPATH/menuopt/admit/scripts/ltrrun.adm

PR=!SCP_PATH/common/ltbrun.scp
OPT_LTB_SELECT(BOTH,Y)
**PP=**
**PA=-FORMATTER**
**PP=**
**PA=nroff**
**PP=**
**PA=-STDLPS**
**PP=**
**PA=stdlps_old**
PP=
PA=TICK_ADM
PP=
PA=admit/LEAD_ACE_DEF
PP=PP_CTC_RESOURCE eg: LEAD_CTC_DEF, blank for all.
PA=length 8,upshift,default=LEAD_CTC_DEF
PP=PP_PRINT_LETTER_DATE
PA=PA_DATE
PP=PP_SELECT_CTC_DATE
PA=PA_DATE
OPT_LABELS(,,,N)
SP

### How to Enable the Letter Format Program in a Menu Option

If you want to reinstate the *ltrformat* program as part of a menuopt, add the lines that appear in bold type to the menuopt file.

**Note:** The example below is from $CARSPATH/menuopt/admit/scripts/ltrrun.adm

```
PR=!SCP_PATH/common/ltbrun.scp
OPT_LTB_SELECT(BOTH,Y)
PP=
PA=-FORMATTER
PP=
PA=BINPATH/ltrformat
PP=
PA=-STDLPS
PP=
PA=stdlps
PP=
PA=TICK_ADM
PP=
PA=admit/LEAD_ACE_DEF
PP=PP_CTC_RESOURCE eg: LEAD_CTC_DEF, blank for all.
PA=length 8,upshift,default=LEAD_CTC_DEF
PP=PP_PRINT_LETTER_DATE
PA=PA_DATE
PP=PP_SELECT_CTC_DATE
PA=PA_DATE
OPT_LABELS(,,,N)
SP
```

### How to Globally Disable the Letter Format Program

If all your institution's letters use .rd *nroff* commands, you can disable the *ltrformat* program for all letters. If you want to disable the *ltrformat* program for all letters, modify the lines that appear in bold type in this excerpt of $CARSPATH/modules/common/scripts/ltbrun.

```
set M4parms = ""
set Adrparms = ""
set Lblparms = ""
set Select = LTLB
set Instpath = ""
set Addlevel = ""
set Stdlps = stdlps_old
set Formatter = nroff
set ext = ".arc"
```

### How to Globally Enable the Letter Format Program

If your institution has previously disabled the *ltrformat* program and you determine later that you want to enable the program again for some or all letters, modify the lines that appear in bold type in this excerpt of $CARSPATH/modules/common/scripts/ltbrun.

```
set M4parms = ""
set Adrparms = ""
set Lblparms = ""
set Select = LTLB
set Instpath = ""
set Addlevel = ""
set Stdlps = stdlps
set Formatter = BINPATH/ltrformat
set ext = ".arc"
```

## Defining Alternate Editors for Word Processing

### Introduction

The *vi* editor is the default editor for use with Jenzabar CX. However, users at your institution may prefer to use an alternate editor. To implement another editor that exists on your system, you can set one or more environment variables. For more information on setting environment variables, see the *Jenzabar CX Technical Manual*.

### Environment Variables for Defining Editors

Three environment variables exist to define the editors to use with CX, as follows:

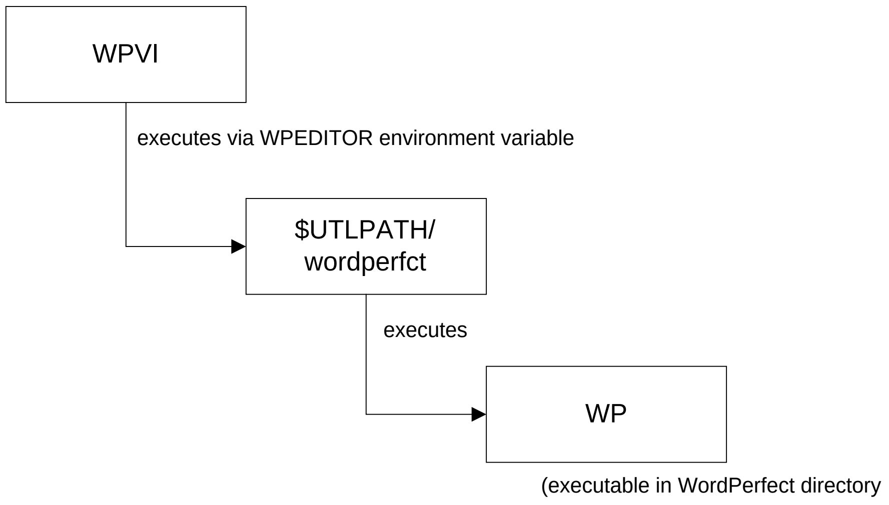
**DBEDIT**
Defines the editor used in BLOB fields.

**EDITOR**
Defines the editor when users perform file operations from the Utility menu.

**WPEDITOR**
Defines the editor for use within WPVI.

### Defining WordPerfect as the Alternate Editor

To make WordPerfect available as an alternate editor to VI, you must make several modifications to the user's environment and CX. WordPerfect must be installed on the system before trying to integrate it into CX. The location of WordPerfect will not affect CX. The first step in WordPerfect installation places a link to the WordPerfect program in a specific location. The script *wordperfct* performs some setup for WordPerfect and then executes the WordPerfect executable. The following illustration shows the WordPerfect relationship.


WPVI
executes via WPEDITOR environment variable
$UTLPATH/ wordperfct
executes
WP
(executable in WordPerfect directory

Perform the following steps to set up the WordPerfect spooler.

1. Log on as *root*.
2. Run the *wpport* program.
3. Set up a port for each printer as follows.
    - Select option 1 (Create) from the displayed menu, and enter a description of the port.

- Select **Other Spooler** for the output type.
- Select option 3 (Spooler Command) from the displayed menu, and enter the printer name, followed by a space, and the filename.

4. Exit the program.
5. Enter **wordperfct** to bring up WordPerfect to attach a printer driver to each printer port.

   **Note:** If you are in *root* when bringing up WordPerfect, you attach printers for the entire system. If you are in your own login, you can attach printers only for yourself.
6. Press **Shift-F7** to select the print option.
7. Select the **Select Printer** option.
8. Select Option 2 (Additional Printers).
9. Select **Printer Driver**. If this print driver has already been installed, select option 4 (List Printer Files).
10. Select **Print Driver**.
11. Enter the following information:
    - Name
    - Port (as previously defined)
    - Sheet Feeder
    - Initial Base Font
    - Path for downloadable fonts and printer command files
12. Exit the Print option.

**System Setup for UNIX WordPerfect Use**

After WordPerfect is installed, perform these steps to configure the system to work with it.

1. In the directory $CARSPATH/modules/util/commands, change the script to point to the WordPerfect executable.

   **Example:** /usr/wp51/wpbin/wp
2. Edit $CARSPATH/modules/util/commands/wordperfct as follows:
   - Modify WPTERM to WPTERM51 if using WordPerfect 5.1.
   - Modify $CARSPATH/install/utl/wordp to be the location of WordPerfect's executable (e.g., /usr/wp51/wpbin/wp). Change all three occurrences that appear in the file.
3. Check in and install the changes.
4. If you want to use WordPerfect instead of VI, modify the environment variable WPEDITOR.

   **Example:** setenv WPEDITOR wordperfct (edit $HOME/.cshrc)

Whenever you require WordPerfect, use the command *wordperfct*. This command is located in $CARSPATH/install/utl/wordperfct.

## Setting Up the Transmit Command

### Introduction

Before you can use the Transmit command in the WPVI Merge drawer, you must complete the following steps:

- Modify a letter/label/creation menu option to include a Format parameter.
- Set macros used to transmit files
- Reinstall the xfer file
- Modify the menu users' .login files

This section provides instructions for completing these steps.

### Modifying a Letter/Label Creation Menu Option

If the menu option your institution's menu users are using to create letters or labels does not have the Format parameter on the menu option screen, you must add it. In this way, the menu users can select "rtf" (Rich Text Format) if using MS Word, "wordp" if using WordPerfect, or "stdlps" if using Standard Letter Production System (*nroff*) to create their letters. Both the screen section and the attribute section of the menuopt file in question must be modified by adding the following lines.

```
screen section:
.
.
.
PP_FORMAT[PA9]


attribute section:
.
.
.
PA8: optional,
    default = "-STDLPS";

PA9: optional,
    comments = "COMMENT_FILE_FORMAT COMMENT_TBL",
    default = "FILE_FORMAT_DEF",
    FILE_FORMAT_INCL,
    length = 6;
```

**Note:** Temporarily install the menuopt file until you are sure it works properly, by entering the following at the UNIX prompt:

**make tinstall F=<filename>**

### Setting Macros Used for Transmitting Files When Using QuickMate

You must set four macros before files can be transmitted. These macros reside in the $CARSPATH/macros/custom/common file. The macro names are listed below along with the values needed if you are using QuickMate.

```
{>>COMMON }
{*** Defines the file transfer protocol to be used when transferring          ***}
{*** files from the UNIX host system to a PC.                          ***}
{*** ZMODEM for zmodem; XMODEM for xmodem; KERMIT for kermit;  ***}
{*** FTP for ftp                                                               ***}
m4_define(`XFER_PROTOCOL', `CSERV')

{>>COMMON }
{*** Defines the receive directory for files that are downloaded using         ***}
{*** ftp as protocol with the xfer command from wpvi or from utility           ***}
{*** menu.
   ***}
m4_define(`XFER_REMOTE_DIR', `c:')

{>>COMMON }
{*** Defines the remote host and user for fpt used for downloading files       ***}
{*** in wpvi or via the utility menu option.
   ***}
m4_define(`XFER_REMOTE_HOST', `$CSERVHOST')
m4_define(`XFER_REMOTE_USER', `$LOGNAME')
```

### Setting Macros Used for Transmitting Files When Not Using QuickMate

If you are not using QuickMate, you must set the XFER_PROTOCOL to FTP, and you must have an FTP server running on the PC side of the transmittal. You must then create a .xferrc file in the home directory of all menu users who need to transmit files from the CX system to their PCs.

Complete the following steps to set the needed environment variables for file transfer in the .xferrc file.

1. Enter the following at the UNIX prompt:
   **cd ~<end-user's login name>**
   **vi .xferrc**
2. Add the following lines to the .xferrc file.
   **RemoteHost=<end-user's PC name or IP address>**
   **RemoteUser=<end-user's login name>**
   **RemoteDir=c:**
   **Module=FTP**
3. Save and close the file.

### Reinstalling the xfer File

After you set and reinstall the XFER_ macros in the $CARSPATH/macros/custom/common file, you must reinstall the $CARSPATH/modules/util/commands/xfer file to recognize your macro changes by entering the following commands at the UNIX prompt:

**cd $CARSPATH/modules/util/commands**
**make reinstall F=xfer**

### Modifying .login Files for Menu Users

If you are using QuickMate, you must set the environment variable "UserSource" before menu users have the proper permissions to transfer a file to their PC.

Complete the following steps to set the environment variable.

1. Enter the following at the UNIX prompt:
   **cd $CARSPATH/skel**
   **make co F=login.s**
   **vi login.s**
2. At the bottom of the login.s file add the following line:

**setenv UserSource true**
**make cii F=login.s**

3. Save and close the file.

**Note:** If you are using an IBM computer, you also must make this modification to the $CARSPATH/skel/cshrc.s file.

### Modifying .login Files for Selected Menu Users

If you do not want all menu users to have permissions to transmit files (and if you use QuickMate), you can selectively set the environment variable “UserSource” in the desired menu user’s .login file.

Complete the following steps to set the environment variable for selected menu users.

1. Enter the following at the UNIX prompt:
   **cd ~<login name>**
   **vi .login**
2. At the bottom of the .login file, add the following line:
   **setenv UserSource true**
3. Save and close the file.

**Note:** If you are using an IBM computer, you also must make this modification in each end-user’s .cshrc file.

### Directing Transmitted Files to a Different Location

Ordinarily, the macro XFER_REMOTE_DIR is set to send all transmitted files to the c: drive of a menu user’s PC. If your menu users would prefer to have transmitted files go to another drive or another folder within their c: drives, you must modify the .xferrc files in the menu users’ home directories.

Complete the following steps to modify the .xferrc files. In this example procedure, the transmitted files will be sent to the msoffice/winword/letters folder on the c: drive of the menu user’s PC.

1. Enter the following at the UNIX prompt:
   **cd ~<end-user’s home directory>**
   **vi .xferrc**
2. Add the following line to the end-user’s .xferrc file:
   **RemoteDir=c:/msoffice/winword/letters**
3. Save and close the file.

# SECTION 19 - SETTING UP MERGE FILES FOR WORD PROCESSING

## Overview

### Introduction

To take advantage of the power of word processing programs (e.g., WordPerfect and Microsoft Word), you can generate a merge file in place of an LPS file that is created through *nroff*. This merge file can then be merged with a word processor primary document to produce letters with variable information.

Users can, when running a CX letters option, specify what letter generation format to use. CX supports three main formats:

- *nroff*, which results in letters in LPS
- WordPerfect format, which results in a merge file (or files) in a Merge subdirectory under $CARSPATH/wp/<module>/FileCabinet/Merge
- Word for Windows format, which results in a merge file (or files) in a Merge subdirectory under $CARSPATH/wp/<module>/FileCabinet/Merge

### When to Use this Feature

This feature may be used with any CX letter option, whether it is associated with a standard CX ACE report or with *wpreports* that users create.

### WP Macros

The term *wp macros* refers to the macros found in the $CARSPATH/macros/custom/ltrwp file. These macros are the link between the CX database and the letters that are eventually produced.

# Jenzabar CX Letter Production System

## Introduction

To following explanation of the Letter Production System provides a foundation for an understanding of the merge file process.

### ACE Reports in Letter Production

Typically, Contact records that exist for specific IDs control the selection of those IDs for letter creation. The CX letter/label production system then produces letters and labels selected through an Informix SQL database ACE report.

The ACE report:

- Selects the IDs
- Pulls in variable information to be available for the letter
- Creates an ISQL script to update contacts

### The ltbrun Script in Letter Production

The script that is called to run the ACE report (*ltbrun*) also performs the following other functions:

- Calls the *adr* process to get appropriate mailing information.
- Sorts the output.
- Splits the output into letter text and ISQL text and other miscellaneous information needed by LPS.
- Processes the report output based on whether it is for use with nroff, WordPerfect, or Word for Windows; *ltrformat* is the program that performs this function.
- For *nroff* letters, pipes report output to nroff to generate letters and stores letters in LPS directory.
- For wp merge files, generates properly formatted merge data file and stores in Merge directory under *wpvi.*
- Runs ISQL script to update contacts.

### Passing an Argument to ltbrun

You pass an argument to the *ltbrun* script (which in turn is passed to the *ltrformat* program) which indicates which letter format to use.

The parameter name for this argument is "-STDLPS". For generation of the standard *nroff* letter, the letter production options use the *nroff* file, $CARSPATH/modules/common/letters/stdlps as the argument to this parameter.

## Merge File Creation

### Field Names and WPMacros

Word processors recognize names that are attached to fields within records in the merge file. When the user composes a letter and wants to refer to one of the fields in the merge file, he/she can use either the name or a number denoting the relative position of the field within a record. Many users find names easier to use. These names are merge file field names.

The wp macros contain information that a letter may include. These macros are used in many letter ACE reports so that letters produced in CX can include the value they represent.

The Letter Production System uses the wp macro names found in letter ACE reports as a basis for forming the merge file field names. The names are the same as the macro name, except they do not have the WP_ prefix.

**Example:** For example, the wp macro WP-GIFT-CAMPAIGN is used in the ACE report that produces letters for development. If the merge feature is used, the merge file field name for this element would be GIFT_CAMPAIGN.

**Note:** Any two-character macro name referenced directly in an ACE report is not changed for merge files. The same two-character name is used as the word perfect merge field name.

### The Merge File

The Letter Production System (and more precisely, the *ltrformat* program that is called from within it) can generate a merge file or *nroffed* letters. A created merge file is either compatible with WordPerfect or with Microsoft Word for Windows. The user selects the desired letter generation menu option that will generate letters in the desired format: *nroff*; a merge data file in Word Perfect format; or a merge data file in Word for Windows format. The CX menu provides menu options for each type of output format.

If more than one resource (i.e., letter) is processed in a single menu option or single ACE report, the system creates a separate merge file or letter file for each resource.

The name of the merge file consists of the value of the WP-LTR-NAME macro in the ACE report, followed by a .mrg or .doc extension, based on whether the merge format is WordPerfect or Word for Windows. The value of the extension for the merge file names is determined by the macros MERGE-WORDP-FILE-EXT and MERGE-WORDW-FILE-EXT in $CARSPATH/macros/custom/common.

### WPVI

The creation of the word processor merge file is integrated into the WPVI module. A drawer called *Merge* exists for all FileCabinets in WPVI, and these Merge drawers exist specifically for this feature. The *ltrformat* process places the merge file in the Merge drawer in the FileCabinet associated with the ACE report being run. In the ACE report, the WP-LTR-NAME macro always defines the FileCabinet location of the letter. The WP-LTR-NAME macro is either directly defined in the ACE report, or it may be defined as part of the LTB_FORMAT macro in the report.

If the process creates merge files, there may be no letter in WPVI associated with the letter name macro. However, the letter name macro also defines the FileCabinet and *ltrformat* uses that value to determine in what Merge directory to store the merge data file. Also, that value actually becomes the name of the merge file itself.

**Example:** For example if you run the menu option to product admission letters, you pass the resource ADM1, and your menu option was set up to create merge files instead of LPS letters, your merge data file would be located in $WPPATH/admissions/FileCabinet/Merge under the name ADM1.mrg.

After the process creates the merge file, it sends mail to the user giving the location of the Merge drawer where it is stored.

## Merge File Processing

### File Transfer

All WPVI Merge drawers offer a transmit option that allows users to transmit a merge file to a PC.

In order for the transmit option in WPVI to work properly, you must set up the file transfer method. For more information on setting up the file transfer method, see *Setup Procedures* in this section.

To perform the transfer, the user performs the following steps:

1. Access the WPVI Merge drawer containing the file to be transferred.
2. Select the transmit option by entering **t** and the name or number of the file to be transmitted; the program displays the message "Transmitting file...".

**Note:** From the UNIX Merge drawer, once a file has been transmitted, it is renamed with an ".xmt" extension to identify it as a transmitted file.

### Transmitting Files

The Utilities menu option Download File enables a user to transmit a file anywhere on the UNIX host system to a PC by specifying the absolute path name of the file to be transferred. Should merge data files be produced via some function other than the CX Letters Production System (e.g., IQ), the output files can be transmitted to the PC with this option.

### Role of Word Processing

The actual merging process and letter printing occurs within the word processing software. Note that the size of the merge file that can be processed on the PC is a function of available PC memory.

The SRT_SORT_BY macro in the ACE report that gathers information needed for the merge file controls data sorting. If users want to perform additional sorting of the merge file, they can use through the sort features available in the word processor.

### UNIX-Based WordPerfect

In addition to using the merge data file with a PC-based word processor, you also have the option of using the merge data file with the UNIX-based version of WordPerfect. The primary difference between running WordPerfect from a PC and from the UNIX system for purposes of merge file generation is that file transfer is unnecessary.

If you use UNIX-based WordPerfect, the entire letter generation process, from selection of contacts to creating the merge data file and merging and printing, can be accomplished in one step. To do so, you can use the CX script in $SCPPATH/common/ltbwordp from a locally created menu option, or from a *cron* file to execute all processes for letter generation for a given FileCabinet. The parameters required by this script are documented in the header of the script itself.

This script calls WordPerfect with a previously user-created macro file to actually perform the merge and print. To work correctly, the script requires the following:

- The WordPerfect macro file must be created in WordPerfect and must contain all commands needed to merge the documents, print them, and exit from WordPerfect.
- The macro file must be placed under the $TXTPATH/wordp directory.
- The name of the macro file must match the resource name associated with the merge data file. For example, if a merge data file called INQUIRE.mrg is created in

$WPPATH/admissions/FileCabinet/Merge, an associated macro file called INQUIRE must exist in $TXTPATH/wordp.

Once merge processing is complete, the script renames the *.mrg file in the Merge drawer to a *.fin file.

### Process for Generating a Merge Data File From the UNIX Host

The steps associated with generating a merge data file from the UNIX host system and generating and printing a merged output file from a PC are:

1. The user executes the letters and labels menu option appropriate for the letter (it is assumed that the option is set up to produce merge data files).
2. The ACE report selects the data.
3. The ADR processor sorts the data.
4. The program creates the word processing merge file and stores it in the WPVI Merge directory associated with the ACE report run.
5. The user accesses the Merge directory, selects the transmit option, and enters the file to transmit.
6. The user, as a PC user, loads the word processing software and merges the PC letter (primary document) with the file downloaded from the UNIX host (secondary document).
7. The user prints the merged output from the PC.

## Setup Procedures

### Introduction

The setup procedures include both macro customization and your implementation of a selected file transfer method.

### Macros

The following macros must be customized to set up merge files:

**Macros for Format Selection**

Macros control which formatter to in generating merge files or *nroff* files in the menu options that execute the letter production system. All of these macros are defined in $CARSPATH/macros/custom/common.

Menu options used to generate letters pass a parameter for -STDLPS to the *ltbrun* script. As delivered, the value is *stdlps*, indicating *ltbrun* should generate *nroff* files using the *nroff* format file, $CARSPATH/common/letters/stdlps.

Menuopts use the following macros:

| Macro name | Value |
|---|---|
| FILE-FORMAT-DEF | stdlps |
| FILE-FORMAT-VALID | rtf, stdlps, wordp |

Currently, all letter generating menuopts (with the exception of *ltrackrcpt*, which uses stdlps-enc) pass the value of FILE_FORMAT_DEF (stdlps) with the -STDLPS parameter.

If you want to have all or most of the letter generation menu options create merge data files rather than *nroff* LPS letters, change the value of the macro FILE-FORMAT-DEF from stdlps to either *wordp* or *rtf*. The wordp argument to the -STDLPS parameter will result in a data file compatible with WordPerfect. The *rtf* argument to the -STDLPS parameter will result in a data file compatible with Word for Windows. For those options in which you want to continue using *nroff*, you must edit the menuopt so the argument to the -STDLPS parameter is *stdlps*.

If you have only a few letter generation options you want to use with a word processor, you can modify the option to pass *wordp* or *rtf* as the argument to the -STDLPS parameter in the menuopt.

If the users at your institution understand the significance of the _STDLPS parameter, you can modify the menu options so the user can designate the desired format method.

**Macros for Protocol Selection**

The macro XFER-PROTOCOL specifies which protocol to use to transfer files from the UNIX host system to the PC system. The protocols given as options are: ZMODEM, XMODEM, KERMIT, FTP.

If you are using Procomm Plus, this value should be consistent with the protocol you have chosen to use with Procomm Plus.

**Macros for File Extension Names**

Two macros in $CARSPATH/macros/custom/common specify the extensions that are added to the filenames in the Merge drawer for merge data files:

| Macro name | Value | Description |
|---|---|---|
| MERGE_WORDP_FILE_EXT | .mrg | Filename extension for WordPerfect |

| Macro name | Value | Description |
|---|---|---|
| | | merge files |
| MERGE_WORDW_FILE_EXT | .doc | Filename extension for Word for Windows merge files |

## File Transfer Setup

The *xfer* script, located in $CARSPATH/modules/util/commands/xfer, performs the file transfer function. The following two modes of file transfer are supported with this feature: Procomm Plus for Windows and FTP.

### User Permission for File Transfer

A permission requirement ensures that only users whose PCs can support file transfer can execute the file transfer option from the CX system; this requirement must be set up on an individual user basis. For each user to whom you want to give access to this file transfer utility, you must set the environment variable UserSource in the user's .cshrc file as follows:

```
setenv UserSource true
```

### Procomm Plus

Each PC should handle the terminal emulation necessary to function as a user on the UNIX platform.

Procomm must be fully installed (i.e., specifying protocol, receive file directory, terminal emulation, etc.) prior to executing a file transfer from within WPVI or from the Utility menu. The protocol specified in Procomm should be the same protocol specified in the macro XFER-PROTOCOL. Should you need to have some users run with a different protocol, you may set up a .xferrc file for the users in their home directories which specifies transfer parameters that differ from the defaults. This .xferrc file, if it exists, is used in the execution of the transmit command to provide information about how the transfer should be performed.

For example, if your macro is set up to use Kermit as the protocol, but a particular user must use zmodem, this user might have a .xferrc file consisting solely of the line:

```
Module=ZMODEM
```

The values in a .xferrc file for a user override the macro-defined defaults.

The Module specification in the .xferrc file is the only information pertinent to the Procomm file transfer method that can be given in the .xferrc file. There are additional values that may be defined in the .xferrc file for the user with the ftp file transfer method. For more information, see *FTP* in this section.

**CAUTION:** CARS considers use of a .xferrc for a user as a security risk since the user has access to this file and it is sourced in the transmit process.

To perform file transfer with Procomm, users must have a login session from the PC to the host from within Procomm. The CX software is run from this login session and the transfer is executed also from within it.

### FTP

To transfer files using FTP, you must have an FTP server running on the PC that is to receive the file. You must specify several values for each user who will be using FTP for file transfer.

The defaults for these necessary values are in the $CARSPATH/macros/custom/common file. These macros are:

| Macro name | Value | Description |
|---|---|---|

| Macro name | Value | Description |
|---|---|---|
| XFER-REMOTE-DIR | /wpwin | The receive directory on the PC |
| XFER-REMOTE-HOST | $LOGNAME | The remote server; the PC to link to |
| XFER-REMOTE-USER | $LOGNAME | The login name of the user on the PC |

If the defaults given by these macro values are inappropriate for a particular user, an .xferrc file in that user's home directory with the proper settings is required. The values in a .xferrc file for a user will override the macro defined defaults.

Below is a table showing the variable names that can be used in the .xferrc file and the macro which defines the associated default value. All are applicable to an FTP installation; only the Module variable is applicable to a Procomm installation.

| Name | Value | Description |
|---|---|---|
| RemoteDir | XFER-REMOTE-DIR | The receive directory on the PC |
| RemoteHost | XFER-REMOTE-HOST | The remote server; the PC to link to |
| RemoteUser | XFER-REMOTE-USER | The login name of the user on the PC |
| Password | | Password for the user on the PC system for FTP |
| Module | XFER-PROTOCOL | Protocol to be used for file transfer. This value should be set to FTP if you use networked PCs with FTP. Valid values are ZMODEM, XMODEM, KERMIT, and FTP. |

**Syntax**

Note that the syntax used for specifying the remote directory is not the same as that used to actually change directories in DOS on the PC. For FTP, you must specify the path beginning with a "/" relative to your current drive. For example, if your current drive on the PC is C:, and you want the receive directory to be C:\wpwin, the RemoteDir value (and the XFER-REMOTE-DIR macro, if appropriate for all users) should be set to "/wpwin".

A sample .xferrc script to transmit via FTP to a networked PC called *maxwell* for a user with a login of *maxwell*, receiving files into the directory C:\wpwin\ follows:

```
        RemoteHost=maxwell
        RemoteUser=maxwell
        RemoteDir=/wpwin
        Module=FTP
```

**Note:** The values in the above example must be included in the xferrc file only if they differ from the macro values for XFER-REMOTE-DIR, XFER-REMOTE-HOST, XFER-REMOTE-USER, and XFER-PROTOCOL.

**CAUTION:** CARS considers use of a .xferrc for a user as a security risk since the user has access to this file and it is sourced in the transmit process.

**Use of a .netrc file**

You may also optionally have a .netrc file for each FTP user. This file specifies values used by FTP when FTP is invoked by this user. The names for each value and short description follows. For more information on this file, see the manual pages for "netrc".

| Name | Description |
|---|---|
| machine | The value that allows autologin with FTP. The machine value should match the value of the RemoteHost variable specified in the .xferrc file. |
| login | The PC login name of the user. This value should match the value of the RemoteUser variable specified in the .xferrc file. The value can be *anonymous*. |

| Name | Description |
|---|---|
| password | The password for the remote login. If this is not specified, but you require passwords, then when the transmit option is executed, the user will be prompted to enter a password. If *anonymous* is the login value, no password is required. |

**Example:** A sample .netrc file that is consistent with the sample .xferrc file is:

```
machine maxwell
login maxwell
password wilbur
```

# SECTION 20 - CUSTOMIZING THE TICKLER PROCESSES

## Overview

### Introduction

This section provides information about the Tickler feature of the Communications Management product, and procedures for setting up a Tickler strategy. The following procedures are included:

- Assessing institution needs for module
- Reviewing data in tables and records

This section also includes the following:

- A checklist for establishing a Tickler strategy
- A comprehensive example of Tickler usage in the Admissions area that you can use as a basis for Tickler strategies at your institution

### Basic Information

This section contains detailed procedures specific to the Tickler feature in the Communications Management product. For information on performing basic procedures such as using the MAKE processor and reinstalling options, refer to the following resources:

- *Database Tools and Utilities* course notebook
- *CX Implementation and Maintenance Technical Manual*

### Tickler Strategies

The basic concept within the Tickler feature is the *strategy*. A Tickler strategy is an institution's policy for corresponding with individuals or organizations (e.g., recruits, students, financial aid recipients, donors or alumni), entered in tables so the Tickler programs can interpret that policy. An institution can have as many strategies as desired (e.g., different correspondence policies for transfer and first-time students), and can customize strategies to support overall goals (e.g., providing more recruitment information to the most desirable applicants to encourage their enrollment).

### Using Levels to Support Communications Goals

To support its communications goals, the institution must define a strategy that includes levels. Tickler recognizes the level (tick_rec.level) as the means to differentiate individual correspondence recipients and the number of communications each will receive. The level is an alphabetic code that designates a subcategory of a strategy. Level A defines the most desirable correspondence recipients (e.g., the most active alumni or most desirable recruit prospect). Level M defines the average recipient (e.g., the average alumni or the applicant with average grades or test scores). Level Z defines the least desirable recipient (e.g., the alumni who has not responded to previous correspondence, or the applicant with low grades or test scores). If desired, you can define up to 26 correspondence levels (A-Z), although the complexity of a many-level strategy may make it difficult to maintain.

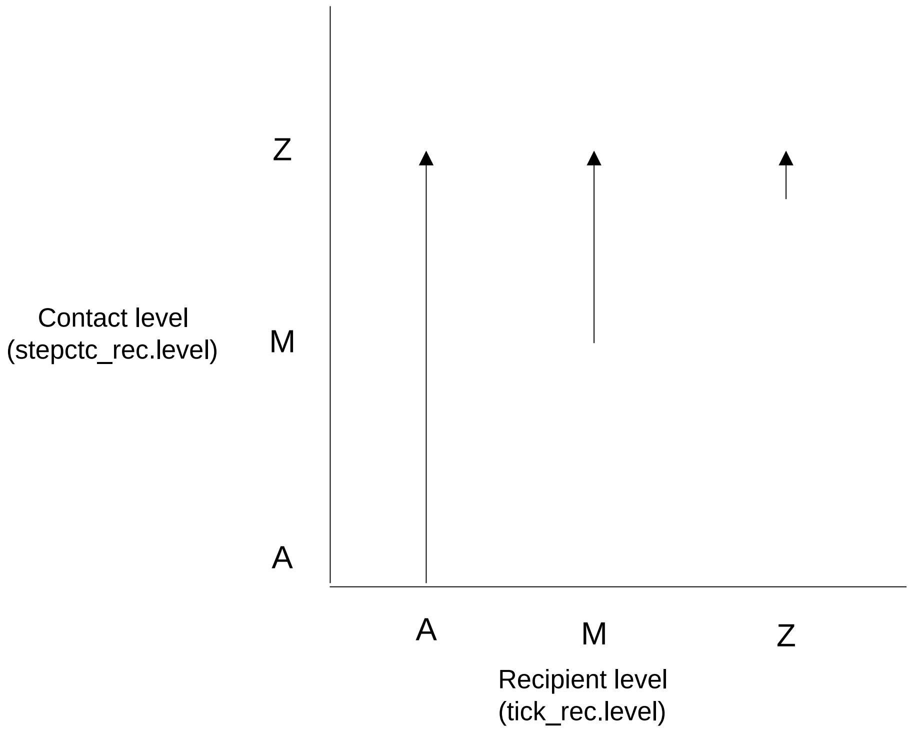
You define the Tickler level for each correspondence recipient in the Tickler record (tick_rec), and the Tickler level for the correspondence itself in the Step Contact record (stepctc_rec). For example, if you want to send a promotional video to only the most desirable applicants, you would designate the VIDEO contact code with a level of A. A general letter that you want to send to every prospect would have a level of Z.

Tickler creates contacts according to the following logic:

- Tickler will schedule a contact with a level of Z for every individual or organization in the strategy.

- Tickler will schedule a contact with a level of M for every individual or organization in the strategy whose Tickler record has a level of M or higher (e.g., M-A).
- Tickler will schedule a contact with a level of A only for those individuals whose Tickler record has a level of A.

The following diagram shows the relationship between the correspondence levels in the stepctc_rec and recipient levels in the tick_rec:



## Completing the Tickler Records and Tables

### Introduction

The Tickler tables and records interact to provide a means for creating Contact records according to your institution's policy. For example, if an applicant has not completed an application within a prescribed number of days, you can automatically create a Contact record that sends a letter to the applicant requesting missing documents. You can also automatically send a letter acknowledging the receipt of the documents at some subsequent date.

You must set up the records and tables described in this section to create the Tickler strategies that conform to your institution's policies and needs.

### Tickler Table

The Tickler table (tick_table) defines all Tickler codes used within an institution. For example, you might define an Admissions tickler with the following values:

**Code**
A four-character code name for the Tickler (e.g., ADM).

**Description**
A descriptive title for the Tickler (e.g., Undergraduate Admissions tickler).

**Minimum Contact Span**
The minimum number of days allowed between contacts (e.g., 10 days).

**Note:** The value in this field assumes your institution has a letter or information to send. In the Admissions area, a likely Minimum Contact Span is 5 to 10 days; within Development, a more probable span is 30 to 60 days.

**Maximum Contact Span**
The maximum number of days allowed between contacts (e.g., 30 days).

**Note:** The value in this field assumes your institution has a letter or information to send. In the Admissions area, a likely Maximum Contact Span is 30 days; within Development, a more probable span is 180 to 360 days.

**Maximum Review Span**
The maximum number of days between reviews of a person on this Tickler (e.g., 30 days). The value in this field sets the next review date in Tickler records.

**Note:** Ensure the Maximum Review Span value is not greater than the Maximum Contact Span.

**Mail User**
The login name of the user who receives mail from the *tickler* program (e.g., jdoe).

### How the Tickler Process Uses Maximum and Minimum Spans

When *tickler* schedules contacts, it attempts to schedule all the contacts using the number of days in the Maximum Contact Span between each scheduled contacts. However, it also reviews the recipient's Completion Date. If the Completion Date would pass before the Maximum Contact Span could be accommodated for every contact, then *tickler* uses the remaining number of days until the Completion Date, the number of contacts, and the Maximum and Minimum Spans to recompute the number of days to schedule between contacts. The *tickler* program will continue to maintain even spacing between contacts, even if it needs to adjust the span of time between contacts. The *tickler* program will not, however, schedule a contact closer than the Minimum Span. This ensures that no recipient will receive too many letters in a short period of time.

**Example:** Your strategy calls for interested students to receive recruitment information every three weeks (21 days). However, if students express interest late in the recruiting cycle, you do not want them to receive letters more frequently than once a week. In this situation, the Maximum Contact Span is 21 and the Minimum Contact Span is 7.

## Track Table

The Track table (trk_table) defines the tracks used with each Tickler code. Within any Tickler system, you can define an unlimited number of tracks (e.g., ADLT for adults, HS for high school students, or TRN for transfer students). For example, an Admissions tickler could define the high school track with the following values:

**Code**
A four-character code name for the track (e.g., HS).

**Description**
A descriptive title for the track (e.g., High School student).

**Tickler**
The Tickler system as specified in the tick_table (e.g., ADM).

## Step Table

The Step table defines each step for each Tickler code. Within every track, however, you may not need each of the steps you have established. For example, if you have a non-traditional admissions track, it may not require the same steps you use for traditional applicants.

Steps usually have some type of logical objective. For example, when a student has inquired about your institution, the objective is to receive an application. After you receive an application, the objective is to receive all the supporting documentation you need to evaluate the student for admission (e.g., test scores, essays, and recommendations). After you accept the student for admission, the objective is to obtain a deposit.

Occasionally, however, a step will not have an easily defined objective. For example, sometimes the objective is for the institution to send a series of letters. This type of objective is common for the last step of a strategy. When, for example, the step is to confirm enrollment, the objective is to complete sending all necessary information to the student.

## Example of Step Table

Below are several sample steps (designated A, B and C) for the ADM Tickler code.

**Step A**
APPLIED
**Text A**
Letters for applicants
**Tickler A**
ADM
**Priority A**
25
**Interrupt Step A**
N

---

**Step B**
REJECTED
**Text B**
Denied admission
**Tickler B**

ADM

**Priority B**

85

**Interrupt Step B**

Y

---

**Step C**

ACCEPTED

**Text C**

Letters to accepted students

**Tickler C**

ADM

**Priority C**

90

**Interrupt Step C**

N

**Note:** In each case, the Priority is a unique ranking for the step within its Tickler code. Zero (0) is the highest priority. Higher priorities (lower numbers) indicate that the step will occur earlier within the Tickler strategy than lower priorities (higher numbers). If an *interrupt* step becomes active, the *tickler* program stops progression of all subsequent steps with a lower priority (higher number). Using the above examples, if an applicant is rejected, all steps with a Priority number greater than 85 are interrupted or prevented. For example, as your institution plans steps and their priorities, note that a *request for deposit* step should have a priority number greater than a *rejected* step (as in the example) to avoid requesting a deposit from a rejected applicant.

### Tickler Record

The Tickler record (tick_rec) defines the characteristics of each individual within a Tickler system. Each record is unique, based on the Tickler code and ID number. This record key allows an individual to be part of more than one Tickler system at a time. For example, a prospective student may have one record for the Admissions Tickler, and another for the Financial Aid Tickler. However, an individual can be part of only one track within each Tickler system.

Fields in the tick_rec are as follows:

**ID**

The ID number of the individual or organization.

**Tickler code**

The code that identifies the Tickler system, as defined in the tick_table (e.g., ADM).

**Track code**

The code that identifies the track with which the individual or organization is associated (e.g., HS).

**Completion Date**

The projected completion date (e.g., the date of enrollment for the ADM Tickler, or the final date of a capital campaign for an ALUM Tickler). This date determines the deadline for which all letters to a person or organization must be sent.

**Last Review Date**

The date on which *tickler* last reviewed an individual's or organization's Contact record. The *tickler* program updates this field automatically.

**Note:** Review dates set the interval of time for *tickler's* review of contacts. If you specify 30 days between dates, then 30 days must elapse after the last completed contact for *tickler* to select the ID for review. Admissions Entry and Interactive Tickler can modify the Last Review Date.

**Next Review Date**

The next date on which *tickler* will review the individual or organization for Contact record creation. The *tickler* program automatically updates the value in this field, based on the Maximum Review Span value in the Tickler table. If the Next Review Date is the current date, *tickler* reviews the individual's Contact records, triggering any required action. After completing the review, *tickler* updates the value in this field by adding the number of days in the Tickler Review field (in *tickent*) to the current date. For example, if the Tickler Review field in *tickent* contains 7, then *tickler* will not review the individual's Contact record for seven days, even if the Tickler Review process runs every night.

Every time you add or update a Contact record (either from the Contact detail window of an entry program or during the letter creation process), the Next Review Date is also updated to the current date. Therefore, *tickler* reviews all of an individual's Contact records for any necessary actions, even if seven days have not elapsed since the last review.

**Level**

An alphabetic code that defines the number of contacts the individual or organization receives. Define the level where the higher the value (A higher than Z), the more contacts the individual will receive. For example, if an applicant should receive five contacts, you could set the Level field to M; for the applicant who should receive the same five contacts as well as five others, you could set the Level field to A.

**Manual Flag**

A Y/N flag indicating whether the record is to be maintained manually or by *tickler*. This flag serves as an on/off switch that can immediately stop the *tickler* program from scheduling any more contacts for a recipient. If *tickler* is to automatically schedule contacts, set the flag to N. If an applicant or alumnus asks you to stop sending them letters, set this flag to Y.

### Example of Tickler Record

Below are three examples of entries in the tick_rec.

| | |
|---|---|
| **ID A** | 20128 |
| **Tickler A** | ADM |
| **Track A** | HS |
| **Completion Date A** | 09/01/1997 |
| **Last Review A** | 00/00/0000 |
| **Next Review A** | 00/00/0000 |
| **Level A** | Z |
| **Manual Update A** | N |
| **ID B** | 20152 |
| **Tickler B** | ADM |
| **Track B** | |

HS

**Completion Date B**
09/01/1997

**Last Review B**
06/25/1996

**Next Review B**
07/25/1996

**Level B**
M

**Manual Update B**
N

---

**ID C**
20315

**Tickler C**
ADM

**Track C**
HS

**Completion Date C**
09/01/1997

**Last Review C**
00/00/0000

**Next Review C**
00/00/0000

**Level C**
A

**Manual Update C**
N

## Step Record

Step records define the valid stages for each Tickler track. The record contains the following:

**Beginning Day**
The number of days before the Tickler completion date that the step becomes active.

**Expiration Day**
The number of days before the Tickler completion date the step should stop being active.

**Step Type**
A code indicating whether the step is required, optional, or selective, based on the following criteria:

- (R)equired **-** A step that must be completed for the Tickler strategy. In Admissions, APPLIED is a required step.
- (O)ptional **-** A step that anyone in the Tickler/Track may complete, depending on time intervals and other contacts. Optional steps appear in the output of steps and contacts for an individual. In Admissions, FRESHMAN, SOPHMORE, and JUNIOR are optional steps.
- (S)elective **-** A special step that must be scheduled manually only for those individuals to whom it applies. Selective steps appear only if they have been activated for an individual.

**Note:** For more information about selective and optional steps, see *Using Selective and Optional Steps in Tickler Strategies* in this section.

## Example of Step Record

Below are three examples of Step records.

**Step A**
APPLIED
**Tickler A**
ADM
**Track A**
HS
**Beginning Days A**
365
**Expected Days A**
30
**Type A**
R

---

**Step B**
CANCELED
**Tickler B**
ADM
**Track B**
HS
**Beginning Days B**
730
**Expected Days B**
366
**Type B**
O

---

**Step C**
CONGRATS
**Tickler C**
ADM
**Track C**
HS
**Beginning Days C**
365
**Expected Days C**
-30
**Type C**
S

## Step Contact Record

The Step Contact record (stepctc_rec) defines the contacts to be sent when a step becomes active.

The Step Contact record contains the following fields:

**Step**
An eight-character code name for the step (e.g., INQUIRED)

**Tickler**
The four-character code name for the tickler. The code must be valid in the tick_table.

**Track**
The four-character code name for the track. The code must be valid in the trk_table.

**Contact**

The eight-character code name for the contact to be scheduled by *tickler*. The code must be valid in the ctc_table, or you can create it while using Tickler Entry (*tickent*).

**Priority**
Numeric indicator specifying the order in which the contacts will be scheduled. If you define a contact with a Priority of 2 within a step, only a contact with a Priority of 1 could occur before it. Priorities are set within a given Step, Tickler, Track combination.

**Level**
Alphabetic indicator that specifies which individuals within the Tickler/Track should receive the particular contact. Individuals having a level of Z will not receive contacts coded with a level between A and Y. For more information about the use of level, see *Using Levels to Support Communications Goals* in this section.

## Example of Step Contact Record

The following examples, since they are set at a Level of Z, may be scheduled for anyone in the HS track of the ADM Tickler.

**Note:** In the following examples, both records have a Priority of 1. This Priority indicates this is the first contact to be scheduled with this step.

**Step A**
INQUIRED
**Tickler A**
ADM
**Track A**
HS
**Contact A**
VIEWBOOK
**Priority A**
1
**Level A**
Z

---

**Step B**
ACCEPTED
**Tickler B**
ADM
**Track B**
HS
**Contact B**
ACPTLTR
**Priority B**
1
**Level B**
Z

## Step Requirement Record

A step requirement, like a prerequisite, is a step or contact that must be completed before a Tickler step can become active. If you specify more than one requirement for a particular step, each one must be met before the step can begin. The Step Requirement record contains the following fields:

**Step**
The eight-character code name associated with the step.

**Tickler**

The four-character code name associated with the tickler.

**Requirement Type**
The requirement category. Valid values are:

- C (The requirement is the name of a contact)
- S (The requirement is the name of a previous step)

**Step Requirement**
The eight-character code name of the contact or previous step which is the requirement.

### Example of Step Requirement Record

In the following example, the institution must receive an application (APPRECV) before the COMPLETE step may begin, and the APPLIED step must be complete before the applicant can begin in the ACCEPTED step.

**Step A**
APPLIED
**Tickler A**
ADM
**Type A**
(C)ontact
**Code A**
APPRECV

---

**Step B**
COMPLETE
**Tickler B**
ADM
**Type B**
(C)ontact
**Code B**
APPRECV

---

**Step C**
ACCEPTED
**Tickler C**
ADM
**Type C**
(S)tep
**Code C**
APPLIED

---

**Step D**
REGIST
**Tickler D**
ADM
**Type D**
(C)ontact
**Code D**
R_REG

### Step Objective Record

The Step Objective record usually defines an incoming contact that completes a step. It is the purpose for which the step was designed. If you specify more than one objective for a particular step, any one of them may be met to complete the step.

The Step Objective record contains the following fields:

**Step**
The eight-character step in the Tickler with which the objective is associated.

**Tickler**
The four-character Tickler code with which the objective is associated.

**Contact**
The eight-character contact resources that must be sent or received to complete this Tickler step.

**Add Contact Flag**
A Y/N flag indicating whether *tickler* should add this contact code with an (E)xpected status when the step becomes active. A Y value indicates that *tickler* should automatically add an expected contact for step objective. This feature works on both incoming and outgoing contacts.

**Note:** If an outgoing contact is used for an objective and if the contact is also part of the step contacts, set the Add Contact flag to N.

## Example of Step Objective Record

Below are several examples of Step Objective records.

**Step A**
TRANS
**Tickler A**
ADM
**Contact A**
R_FULLTR
**Add Contact Flag A**
Y

---

**Step B**
SENIOR
**Tickler B**
ADM
**Contact: B**
APPRECV
**Add Contact Flag B**
Y

---

**Step C**
SCORES
**Tickler C**
ADM
**Contact C**
COMPLETE
**Add Contact Flag C**
N

## Implementation Process for Tickler

### Introduction

After you have defined and set up the appropriate entries in the Tickler tables and records, the *tickler* program creates Contact records that can be used to produce letters in CX.

This section defines the process for designing and implementing communication strategies. The process includes planning, table/data entry, and testing.

**Note:** Within this section, examples relate to the Admissions area, but you have similar setup issues in other functional areas.

### Planning

The planning phase of implementation includes reviewing your communications policies and determining that other features in Communications Management are set up correctly for use with *tickler*.

**Review Admissions Strategy**
A strategy specifies the letters to be sent to each person in the intended audience, along with the timing and circumstances for every correspondence. Strategies are highly flexible; and the simplest type is a fixed sequence of letters. Strategies usually depend on incoming communications received (or not received) from a person. For example, the receipt of an application from a prospective student usually triggers some type of response from your institution.

1. Create a list of the kinds of contacts, and with whom, in the approximate order you want them to be sent. Keep the earliest expected contacts at the top of your list. Concentrate on what letters you send or would like to send. Keep in mind that your contacts may also be telephone calls or visits.
2. Identify the process that prospects typically follow, from inquiry through enrollment or registration.
3. Group all identified outgoing and incoming contacts within the most appropriate step. Order each step so that the earliest expected contacts remain at the top of your list.

**Perform Data Entry Prerequisites**
You must complete the following tasks before using the Tickler programs (*tickent* and *tickler*).

1. Verify the Enrollment Status and the Enrollment Sequence tables are set up if you are creating an Admissions Tickler.
2. Verify the Communications table is set up. The LTLB, LETT, LABL, PHON, MEMO and DOCU codes are standard.

   **Note:** Do not change LTLB, LETT, LABL, PHON, and DOCU; these are hard-coded in various ACE reports.
3. Compose letters in the *letters* drawer of the appropriate FileCabinet for all contacts identified through the process described above.
4. Analyze letters to verify spelling.
5. Test letters for format.
6. Translate letters for printing.
7. Enter a contact resource for each outgoing letter into the Contact table.

**Note:** Within the Admissions area, if the Contact table and the Enrollment Status table are set up properly, a single contact can create a letter, update the admissions status, and instruct the *tickler* program about what to process next. This setup can eliminate the institution’s excessive use of contacts, and a Contact detail window with more information than is usable and useful. For example, in Admissions, the ACCEPTED contact can produce an acceptance letter, update a student's admissions status to ACCEPTED, and cause *tickler* to schedule a follow-up letter to the acceptance letter at a later date. For more information about setting up the Contact table and the Enrollment Status table, see *Customizing the Communications Management Processes* in this manual, and *Implementing Admissions*.

## Table/Data Entry

Depending on the results of your review of existing Tickler tables, you may need to add tracks, set the time spans between contacts, define steps, and then verify your data entry.

### Identify the Tracks

Using Tickler Entry, enter the Tickler/Tracks appropriate for your Tickler. A *track* is a specific path within a Tickler system by which your prospects are categorized, based on their circumstances or background. It specifies the personal characteristics for the group of prospects and may be considered as the target audience for a particular correspondence strategy.

### Set the Scheduling Parameters

Using Tickler Entry, specify the minimum and maximum spans. *Minimum span* is the minimum number of days between scheduled contacts. *Maximum span* is the maximum number of elapsed days between scheduled contacts, assuming that there are contacts appropriate to schedule. The spans ensure an even distribution of correspondence with your prospects.

1. Identify the frequency of *tickler* reviews. When *tickler* is automatically run each night, it reviews those individuals who either have today's date as their Next Review Date or those who have had contacts added or completed since their last review.
2. Identify the user to whom *tickler* will send e-mail. The *tickler* program uses the user's login name when sending e-mail messages.

### Identify Each Step

1. Within *tickent*, define each step along with the possible responses. Steps, the building blocks of a Tickler strategy, are actions specific to a stage in an overall process. In admissions, they might correspond to the progression through your enrollment procedures.
2. Decide which steps are optional, selective, required, or interrupt.
3. Establish the target date as day 0, then work backwards in establishing the timing of each step. Dates must be realistic for all individuals who would be assigned to the same Tickler and track. Consider, for example, differences in Fall and Spring prospects. Such prospects may belong in the same track, but not only the correspondence dates but also the timing for the correspondence may be different. If you place Fall and Spring prospects within the same track, maintain enough flexibility in your timing to accommodate both groups.

### Identify Step Details

1. Set up the requirements for each step. Requirements are a previous step or contact code. They may be considered as prerequisites to enter a new step. All requirements must be met for a step to be active.
2. Set up the contacts for each step. Contacts are the correspondences, visits or calls within a given step.
3. Set up the objectives for each step. Objectives are what the step is intending to accomplish. An objective may be one or more contacts. If there are multiple objectives, completion of any one objective will complete the step.

### Verify Data Entry

Use the Tickler Structure Report option to run a report that illustrates all the components of the Tickler strategy. This report will help you verify your data entry.

**Note:** Access this report from the Communications Management: Tickler menu.

The following is an excerpt from the Tickler Structure report. The legend for the numbered sections follows the excerpt.

```
Tickler: ADM   Min/Max Contact Span: 7/14  Max Review Span: 7         1
-------------------------------------------------------------

                                              Required
                                              --------
                    |------------------------INQUIRED (Contact)       2
                    |
                    |                          Object
  Step   Type Intr  |                         --------
-------- ---- ----  |                      |--APPLFEWV           3
FRESHMAN   O    N --|----------------------|--APPLIED
                    |                      |--APPLNOFE
                6   |
   5                |      Track  Beg  Exp      Contact  Level
                    |      ----- ---- ----     -------- -----
                    |----- HS   1460 1096-----FRESHLTR   Z        4
                        5
```

**Legend for Tickler Structure Report**

1. Tickler name, contact spans, and review spans - from the tick_table.
2. Requirement name(s) - from the stepreq_rec.
3. Object name(s) - from the stepobj_rec.
4. Contact name(s) and level(s) - from the stepctc_rec.
5. Step, Type, Track, Beg, and Exp - from the step_rec.
6. Interrupt code - from the step_table.

## Testing

Testing the Tickler strategy that you have created requires you to run the *tickler* program using a test ID.

1. Using the appropriate entry program, add a test ID.
2. Using Interactive Tickler, create a tick_rec for the test ID.
3. Using Interactive Tickler, test your strategy with the test ID.

**Note:** For more information about using Interactive Tickler for testing see *Testing with Interactive Tickler* in this section.

## Example Tickler for Admissions Processing

### Introduction

This section contains an example of a Tickler system developed for use in the Admissions office. It appears here as a guideline as you create Tickler systems for your institution.

### An Admissions Strategy

The sample Admissions Tickler consists of contacts designed to move prospective students through the various stages of enrollment. The *tickler* program reacts only to (C)ompleted Contact records, since a completion indicates a new phase of communications processing is ready to begin.

- Inquired
- Applied
- Application Complete
- Accepted
- Deposit Paid
- Registered.

In the example Admissions Tickler system, each stage of enrollment becomes a step with specific requirements and objectives to be met. Steps are always completed in a specified order, although more than one step can be active at the same time. Both the step requirements and step objectives are completed contacts. Sometimes, however, the completion of one step becomes the requirement for the next step. In all cases, *tickler* will add expected outgoing contacts until one of the three following situations occurs:

1. All outgoing contacts are sent.
2. The step objective is met (This is usually a contact, e.g., APPRECV).
3. An interrupt step is activated (This is usually a contact, e.g., CANCEL).

Therefore, if the step objective has not been met, *tickler* schedules the next contact, as prioritized in the tables. If the step objective is met, then *tickler* activates the next step in the priority listing. If an interrupt step is activated, *tickler* voids all uncompleted contacts that are scheduled.

### Assumptions in the Example Tickler

In the example Admissions Tickler, the following assumptions exist:

- Several contacts have been established to send to someone who makes an inquiry.
- If the prospect submits an application, the objective of the Inquired step (APPLIED) is met and unsent letters are voided.
- When the institution receives an application, the sender is eligible for additional contacts.
- When an applicant is accepted for admission, the institution can send any of several contacts to encourage the sending of an enrollment deposit and subsequent enrollment. Of these contacts, some may already have been sent. If so, the Reissue flag in the Contact table for each contact prevents the accepted applicant from receiving the same contact twice.

**Note:** These assumptions and the following supporting diagram are an example only. In actual practice, you may have as many or as few contacts in a step as you need to meet your objectives. This illustration demonstrates this point by dividing the Inquired step into several different steps for more specific audiences. For example, inquiries from high school sophomores require different strategies from their high school senior counterparts.

## Example Contact Table Entries

The example Admissions Tickler strategy requires the following Contact table entries:

```
                                        CARS College                        Page   1
                                     CONTACT TABLE REPORT                        tctc
                                    Tickler: ADM
                                                              Spn
Code           Text               Comm   Report   Run Code Rt Wv Rs Dc Enr_Stat
-------- ------------------------ ---- ---------- -------- -- -- -- -- --------
ACCEPTED Accepted for Admission                            I  N  N     ACCEPTED
ACPTLTR  Acceptance Letter        LTLB ltradmit   SINGLEI  O  Y  N
ACT_SAT  Take ACT/SAT Exam letter LTLB ltradmit   SINGLEI  O  N  N
APPCOMP  Application Complete     MEMO ltradmit            I  N  N     APPCOMP
APPLFEWV Applied - fee waived                              I           APPLFEWV
APPLIED  Applied for admission                             I           APPLIED
APPLNOFE Applied - no fee                                  I           APPLNOFE
CAMPLIFE Campus Life letter       LTLB ltradmit   SINGLEI  O  N  N
CATALOG  Catalog labels           LABL ltrschlabl SINGLEI  O  Y  N
CONFIRM  Enrolled student         MEMO                     I  Y  Y     CONFIRM
CONFNOFE Confirmed - no fee paid                           I           CONFNOFE
COUNSELR Letter from Counselor    LTLB ltradmit   SINGLEI  O  Y  N
DEANSTU  Dean of Students letter  LTLB ltradmit   SINGLEI  O  N  N
DEFERRED Deferred admission       LETT ltradmit   SINGLEI  O  Y  N
DENIED   Denied Admission                                  I  Y  N     DENIED
DUMMY1   1st Dummy Concact        NONE ltrnoltr   SINGLEI  O  N  N
ENROLLED Enrolled                 MEMO ltradmit            I  N  N     ENROLLED
EXPECTAP Expect Application       LTLB ltrexptapp SINGLEI  O  Y  N
FACULTY  Faculty Letter           LTLB ltradmit   SINGLEI  O  N  N
FINAID   Send FAF forms           LETT ltradmit   SINGLEI  O  N  N
FRESHLTR H.S. Freshman letter     LTLB ltradmit   SINGLEI  O  Y  N
HEARSOON Decision to be made soon LTLB ltradmit   SINGLEI  O  Y  N
HONORS   letter to Honors Student LTLB ltradmit   SINGLEI  O  N  N
HOUSING  Housing Options letter   LTLB ltradmit   SINGLEI  O  N  N
INCAPP1  Incomplete Application   LTLB ltrstat    SINGLEI  O  N  N
INCAPP2  2nd Incomplete appl ltr  LTLB ltrstat    SINGLEI  O  N  N
INQUIRED Inquired about school                             I           INQUIRED
JUNIORLT Letter to Juniors        LTLB ltradmit   SINGLEI  O  N  N  N
NOSHOW   Did not show             LETT            SINGLEI  I  N  N     NOSHOW
ORIENT   Orientation              LETT ltradmit   SINGLEI  O  N  N
PARENT   Letter to Parents        LTLB ltradmit   PARENT   O  Y  N
REC1     1st Recommendation       DOCU                     I
RECEIVED Received Application ltr LTLB ltradmit   SINGLEI  O  Y  N
RECSURV  Received survey                                   I  N  N
RESLIFE  Residence Hall living    LTLB ltradmit   SINGLEI  O  N  N
SOPHLTR  Sophomore letter         LTLB ltradmit   SINGLEI  O  Y  N
STUGOV   Student Government ltr.  LTLB ltradmit   SINGLEI  O  N  N
SURVRECV Survey Recieved          DOCU                     I
SURVSENT Survey Sent to Student   LTLB ltradmit   SINGLEI  O  N  N
TRANSFER Transfer Student Orient.                          I  Y  N
TSHIRT   College T-shirt          LABL ltradmit   SINGLEI  O  N  N
VIEWBOOK College Viewbook         LABL ltradmit   SINGLEI  O  N  N
VISIT    Campus Visit Listing     LTLB ltradmit   SINGLEI  O  N  Y
WAITLIST Placed on waiting list   LETT ltradmit   SINGLEI  O  N  N
WDLETTER Withdraw notification    LTLB ltradmit   SINGLEI  O  N  N
WELCOME  Wecome to school         LETT ltradmit   SINGLEI  O  N  N
WITHDREW Student withdrew appl                             I
```

## Tickler Structure Report Excerpts for Example Tickler

The following excerpts from the Tickler Structure Report shows the relationship of steps, requirements, contacts, and objectives.

**Note:** This Tickler Structure Report is not complete; it includes only those steps that require additional explanation, or that illustrate a solution to a particular correspondence need.

### JUNIOR and JUNIOR2 steps

The JUNIOR and JUNIOR2 steps illustrate the use of the contact DUMMY to extend the length of time between contacts, and also how to alter the time between contacts without using dummy contacts.

In the JUNIOR step, only two letters are being sent, and the time between the two letters is increased by using a dummy contact. A dummy contact does not create a letter or label, but simply increases the time between contacts. By using a dummy contact in this example, and assuming a Maximum Contact Span of 14 days, you are causing 28 days between the time the JUNIORLT contact and the ACT_SAT contact.

An alternative to the use of dummy contacts is to define two steps. For example, assume an Admissions Office wants to send two letters to juniors in high school shortly after they inquire, and then send them a third letter 100 days after the second letter. To accomplish this, you create two steps to send all three letters, as follows:

**JUNIOR step**

- The JUNIOR step sends the first two letters.
- The beginning and ending number of days for the JUNIOR step are 730 and 366 respectively. This means that students who inquire 366 to 730 days before their potential day of enrollment will receive the JUNIORLT and ACT_SAT contacts/letters.
- The JUNIORLT contact could, for example, create an introductory cover letter to send along with a prospectus or viewbook. The ACT_SAT contact could create a letter telling a student your institution's ACT and SAT identification numbers, and/or sending the test registration materials. The requirement for the JUNIOR step is that the student have a contact named INQUIRED, (which the *admstats* program also uses to update the student's admissions status to INQUIRED). The objective is for the student to apply in any of the following ways:
  - APPLIED (Applied - fee paid)
  - APPLNOFE (Applied - no fee paid)
  - APPLFEWV Applied - fee waived)

Since the student is only a high school junior and may not be prepared to apply yet, then another potential objective is to simply have sent the student the final correspondence in the step, in this case ACT_SAT.

**JUNIOR2 step**

- The JUNIOR2 step sends the last one of the three letters you want to send to high school juniors.
- The step has a beginning date that is 100 days after the JUNIOR step became active.
- The requirements for this step are:
  - The student must have the contact INQUIRED
  - The previous JUNIOR step must have been completed
- The objective of the JUNIOR2 step is that the student has the contact APPLIED, APPLNOFE, or APPLFEWV. These objectives insure the student will not receive the recruitment letter named COUNSELOR if he/she has applied for admission.
- Because a large number of days exists between the contacts you want to send to high school juniors, creating the JUNIOR2 step is much simpler than adding multiple dummy contacts between the ACT_SAT contact and the COUNSELOR contact all within the JUNIOR step. In this case, you would need seven dummy contacts to attain the desired date spans. Jenzabar recommends you minimize your use of dummy contacts whenever possible, as they create unneeded Contact records for the students, and can make the Contact detail window difficult to read. If, however, you cannot avoid the use of dummy contacts, be sure to give them unique names throughout your Tickler strategy, (e.g., DUMMY1, DUMMY2, DUMMY3).

```
                    |                          Object
                    |                         --------
  Step   Type Intr  |                      |--ACT_SAT
-------- ---- ----  |----------------------|--APPLFEWV
JUNIOR     O    N --|                      |--APPLIED
                    |                      |--APPLNOFE
                    |
                    |     Track  Beg  Exp     Contact  Level
                    |     ----- ---- ----     -------- -----
                    |                      |--JUNIORLT   Z
                    |----- HS    730  366--|--DUMMY1     Z
                                           |--ACT_SAT    Z

                                              Required
                                              --------
                    |----------------------|--INQUIRED (Contact)
                    |                      |--JUNIOR   (Step)
                    |
  Step   Type Intr  |                          Object
-------- ---- ----  |                         --------
JUNIOR2    O    N --|                      |--APPLFEWV
                    |----------------------|--APPLIED
                    |                      |--APPLNOFE
                    |
                    |     Track  Beg  Exp     Contact  Level
                    |     ----- ---- ----     -------- -----
                    |----- HS    670  366-----COUNSELR   Z
```

**WITHDREW step**

The WITHDREW step illustrates the use of an interrupt step. After an interrupt step is completed, the *tickler* program updates the student's next review date (tick_rec.next_rvw_date) to a null date value (12/31/1899), eliminating the possibility that *tickler* will schedule contacts for any of the subsequent steps.

```
                                              Required
                                              --------
                    |-------------------------WITHDREW (Contact)
                    |
  Step   Type Intr  |                          Object
-------- ---- ----  |                         --------
WITHDREW   R    Y --|-------------------------SURVRECV (Auto add)
                    |
                    |     Track  Beg  Exp     Contact  Level
                    |     ----- ---- ----     -------- -----
                    |----- HS    365    1--|--WDLETTER   Z
                                           |--SURVSENT   Z
```

**ACCEPTED step**

The ACCEPTED step illustrates the use of assigning different levels to contacts. In this strategy, the HONORS contact has a level of A, designating it as a contact for potential honor students (the students identified as level A). In addition, the other four contacts have a level of Z, designating them as appropriate correspondence for all students (e.g., those students with a level of A -Z). Therefore, in this strategy, honor students will receive five contacts, and all other students will receive four contacts.

```
                                              Required
                                              --------
                    |-------------------------ACCEPTED (Contact)
                    |
                    |                          Object
  Step   Type Intr  |                         --------
-------- ---- ----  |----------------------|--CONFIRM
ACCEPTED   R    N --|                      |--CONFNOFE
                    |
                    |     Track  Beg  Exp     Contact  Level
                    |     ----- ---- ----     -------- -----
                    |                      |--ACPTLTR    Z
                    |                      |--PARENT     Z
                    |----- HS    365    1--|--HONORS     A
                                           |--TSHIRT     Z
                                           |--FACULTY    Z
```

**CONFIRM step**

Occasionally, an Admissions office may want to send a contact or letter if the step's objective is met before *tickler* can schedule the contact. The ACCEPTED and CONFIRM steps illustrate this situation.

Both the ACCEPTED and the CONFIRM steps are set up to schedule the FACULTY contact. In the Contact table, the FACULTY contact is defined with the Reissued flag set to N. Therefore, if the objective is met in the ACCEPTED step before *tickler* has an opportunity to schedule the FACULTY contact, then the FACULTY contact will be scheduled when the CONFIRM step becomes active. However, if the FACULTY contact was sent when the ACCEPTED step was active, it will not be sent again when the CONFIRM step becomes active.

```
                                              Required
                                              --------
                    |-------------------------CONFIRM  (Contact)
                    |
                    |                          Object
                    |                         --------
  Step   Type Intr  |-------------------------DEANSTU
-------- ---- ----  |
CONFIRM    R    N --|     Track  Beg  Exp     Contact  Level
                    |     ----- ---- ----     -------- -----
                    |                      |--ORIENT     Z
                    |                      |--FACULTY    Z
                    |  |-- HS    365    1--|--STUGOV     Z
                    |  |                   |--RESLIFE    Z
                    |--|                   |--DEANSTU    Z
                       |
                       |-- XFER  365    1--|--TRANSFER   Z
                                           |--DEANSTU    Z
```

**ENROLLED step**

The ENROLLED step illustrates two *tickler* features. In this step, the intent is to send a survey to the newly enrolled freshman class, asking the students about their experiences with the Admissions office. In this situation, negative numbers are used to schedule the contact after a student's Tickler completion date. This step will schedule the contact SURVSENT, if the student has the contact ENROLLED (used to update their admissions status to ENROLLED), anytime between 1-14 days after their Tickler target completion date (i.e., the first day of classes). The objective is to receive the completed survey back from the student. In this example, this incoming document (SURVRECV) contact will be added automatically as an (E)xpected contact for the student. When the survey arrives, the Admissions office only needs to update the SURVRECV contact to (C)ompleted, and add the date the survey arrived at the office.

```
                                              Required
                                              --------
                    |-------------------------ENROLLED (Contact)
                    |
  Step   Type Intr  |                          Object
-------- ---- ----  |                         --------
ENROLLED   R    N --|-------------------------SURVRECV (Auto add)
                    |
                    |     Track  Beg  Exp     Contact  Level
                    |     ----- ---- ----     -------- -----
                    |----- HS     -1  -14-----SURVSENT   Z
```

## Tickler Customization Hints

### New Tickler Creation

When creating a new Tickler for other administrative areas, use the example Admissions Tickler strategy as a guide.

### Fine Tuning a Tickler Strategy

Consider the following points when designing a new Tickler strategy:

1. The *tickler* program determines the order of contacts within multiple concurrent steps based upon the maximum span in the Tickler table. Consider adding dummy contacts to increase the maximum span, if appropriate. For example, if the maximum span is 30 days, and in a particular situation you want to force 60 elapsed days between contacts, insert a dummy contact (that creates no letter) in between the actual contacts.

   **Note:** Dummy contacts must have unique names (e.g., DUMMY1, DUMMY2). Set the comm field in the ctc_table to NONE, and the ACE field to *ltrnoltr*.
2. Check the priorities to be sure that the contacts have a unique priority number within the step. Eliminate conflicting priorities.
3. Consider the number of contacts to complete the Tickler objective. Use the most contacts for only your best prospects, using the Level field.
4. Consider the frequency of the Tickler review. The review span should generally be less than or equal to the maximum contact span.

## Using Selective and Optional Steps in Tickler Strategies

### Introduction

The Step record contains a Step Type field that contains one of the following codes:

- O (Optional)
- R (Required)
- S (Selective)

These pages define the differences between optional and selective steps, and provide examples.

### Selective Steps

A common example of using selective steps is when an Admissions office receives a recommendation, test score or other document for an applicant before a formal application arrives. The following table setup causes *tickler* to schedule a letter, RECLTR, to acknowledge the arrival of the document and to encourage the applicant to submit an application.

**Step record**

**Step A**
RECLTR
**Tickler A**
ADM
**Track A**
HS
**Beginning Days A**
730
**Expected Days A**
-30
**Type A**
S

**Step Requirement record**

**Step A**
RECLTR
**Tickler A**
ADM
**Type A**
(C)ontact
**Code A**
INQUIRED

**Step B**
RECLTR
**Tickler B**
ADM
**Type B**
(C)ontact
**Code B**
REC1

**Step Contact record**

**Step A**
RECLTR
**Tickler A**

ADM

**Track A**

HS

**Contact A**

EXPECTAP

**Priority A**

1

**Level A**

Z

**Step Objective record**

**Step A**

RECLTR

**Tickler A**

ADM

**Contact A**

APPLIED

**Add Contact Flag A**

Y

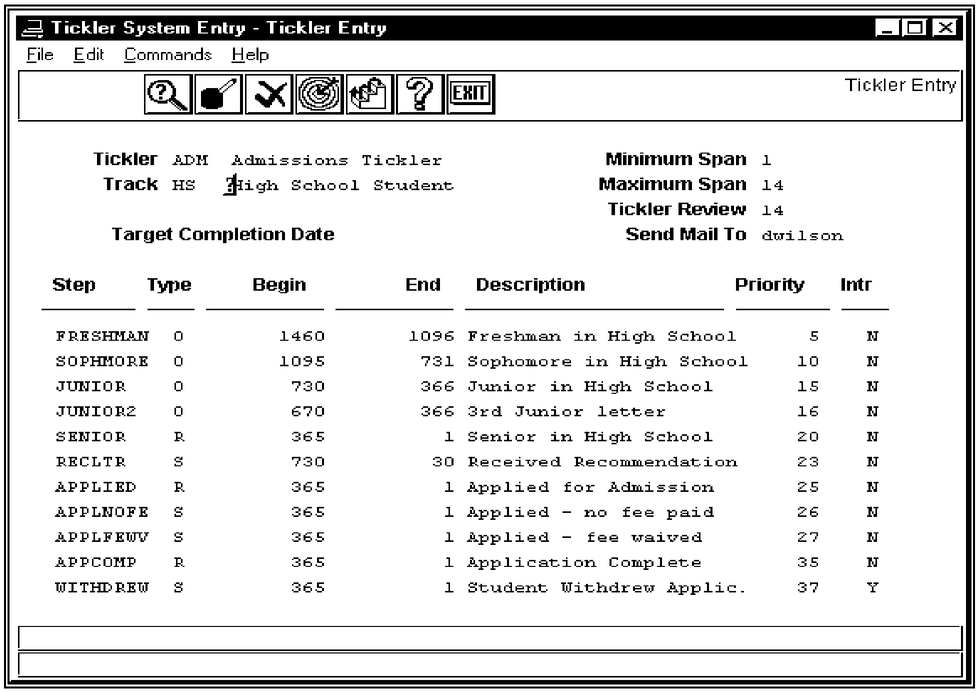
## Optional Steps

A common example of using optional steps is when an Admissions office receives a request for information from a student who is still more than one year from high school graduation. Under these conditions, the office might develop a strategy that sends the information, follows up with a letter each year until the senior year, and then sends several letters during the senior year.

If the current session is Fall 1997, and a high school sophomore requests information, then that student's expected graduation would be Spring 2000. The Admissions office would enter the student in the Admissions Inquiry screen form, designating the expected enrollment session as Fall 2000, and specifying a Target Completion Date of 09/01/2000.

The table entries in the following screen set up two optional steps, SOPHMORE and JUNIOR. Note that the table setup for the optional steps include the following considerations:

1. The Target Completion Date is blank
2. The Type is Optional.
3. The dates must be calculated as follows:
   - A student is considered a high school senior for 1-365 days before the Target Completion Date.
   - A student is considered a high school junior for 366-730 days before the Target Completion Date.
   - A student is considered a high school sophomore for 731-1096 days before the Target Completion Date.
4. INQUIRED is the requirement for the SOPHMORE, JUNIOR and SENIOR steps.
5. APPLIED is the objective for the SOPHMORE, JUNIOR and SENIOR steps.



Tickler ADM Admissions Tickler
Track HS High School Student
Target Completion Date

Minimum Span 1
Maximum Span 14
Tickler Review 14
Send Mail To dwilson

| Step | Type | Begin | End | Description | Priority | Intr |
|---|---|---|---|---|---|---|
| FRESHMAN | O | 1460 | 1096 | Freshman in High School | 5 | N |
| SOPHMORE | O | 1095 | 731 | Sophomore in High School | 10 | N |
| JUNIOR | O | 730 | 366 | Junior in High School | 15 | N |
| JUNIOR2 | O | 670 | 366 | 3rd Junior letter | 16 | N |
| SENIOR | R | 365 | 1 | Senior in High School | 20 | N |
| RECLTR | S | 730 | 30 | Received Recommendation | 23 | N |
| APPLIED | R | 365 | 1 | Applied for Admission | 25 | N |
| APPLNOFE | S | 365 | 1 | Applied - no fee paid | 26 | N |
| APPLFEWV | S | 365 | 1 | Applied - fee waived | 27 | N |
| APPCOMP | R | 365 | 1 | Application Complete | 35 | N |
| WITHDREW | S | 365 | 1 | Student Withdrew Applic. | 37 | Y |

## Testing With Interactive Tickler

### Introduction

Interactive Tickler is a specialized use of the *tickler* program. Its purpose is to test the values in the Tickler tables to ensure they produce the desired outcome. You can test your strategies after you enter all the table values described in this section.

### Using a Test ID With Interactive Tickler

Since Interactive Tickler is designed for testing, Jenzabar recommends you establish a test ID to use with the process, and set up the test ID with Tickler records. Using Interactive Tickler creates actual Contact records that should not be associated with any active ID records on your database.

### Types of Tests With Interactive Tickler

With Interactive Tickler, you can test a student's Tickler Track, Level, Target Completion Date, and Manual flag. You can also change the current date to determine how *tickler* will react to a past or future date compared to the student's Target Completion Date.

### How to Use Interactive Tickler

Complete the following steps to use Interactive Tickler:

1. From the Communications Management: Tickler menu, select Interactive Tickler. The Interactive Tickler parameter window appears.
2. Enter the code for the Tickler track you want to test, then select **Finish**. The initial screen appears.
3. Perform a query to locate the ID and Tickler records for your test ID. The basic information about the test ID appears on the initial screen.
4. Select **Contact-edit**. The Contact Edit screen appears, showing all the contacts scheduled and/or completed for the test ID.
5. Make the changes you want to test, then select **Done**. The initial screen appears.
6. Select **Step**. The *tickler* program begins to review the changed records, and displays the changes on the screen.

   **Note:**
   - The *tickler* program labels the status of each step as Active or Pending.
   - Selective steps that are not active will not appear on the screen.
7. Move your cursor to a step and select the **Detail** command. Detail information will appear, including the status of all required contacts, letter contacts, and objective contacts.
8. If more than two requirement or objective contacts exist for a step, use the **Forward** command to view additional contacts.

### Using Interactive Tickler With Actual Student IDs

You can also use Interactive Tickler to determine when the *tickler* program will schedule future contacts for an actual student. Follow these steps to use Interactive Tickler with an actual student's records:

1. From the Communications Management: Tickler menu, select Interactive Tickler. The

Interactive Tickler parameter window appears.

2. Enter the code for the Tickler track you want to test, then select **Finish**. The initial screen appears.
3. Perform a query to locate the ID and Tickler records for the student. If the student does not already have a Tickler record, create the required record, then use the **Step** command to run *tickler*. You can also use the **Output** command to obtain a detailed report of the selected student's Tickler strategy.

   **Note:** You can also use Interactive Tickler to create Tickler records in offices that do not have Tickler record fields on any of their entry program screens.

# SECTION 21 - PROGRAM AND REPORT ERRORS

## Overview

### Introduction

This section provides the following:

- A list of serious and fatal errors
- Problem resolution procedures

**Note:** Refer to *Using Communications Management* for a list of the more common status, field error, and warning messages that can occur when menu users execute the programs in Communications Management.

### Programs That Can Have Fatal Errors

Fatal errors can occur in the following programs in Communications Management:

- *ltrformat*
- *labels*
- *m4*
- *adr*
- *sacego*
- *isql*
- *splitpage*
- *sortpage*

## Error and Crash Recovery Procedures

### Introduction

The procedures to recover from a crash are organized by the seriousness of the error.

### Core Dump Recovery

The following procedure describes the steps to recover from a core dump of a program.

1. Access the program screens directory for the program.

   **Example:** % cd/$CARSPATH/modules/commgmt/progscr/tickent

2. Reinstall each program screen file.

   **Example:** % make reinstall F=<filename>

   **Note:** You can also reinstall all of the screens by entering the following:
   - **% make reinstall F=all**

3. Attempt to execute the program. Did the reinstall of the program screens fix the error?
   - If <u>yes</u>, you are done.
   - If <u>no</u>, go to step 4.

4. Access the source code directory of the program.

   **Example:** % cd/$CARSPATH/ src/commgmt/tickent

5. Reinstall the source code for the program.

   **Example:** % make reinstall

6. Attempt to execute the program. Did the reinstall of the program source code fix the error?
   - If <u>yes</u>, you are done.
   - If <u>no</u>, go to step 7.

7. In the source code for the program, delete the old compiled code for the program.

   **Example:** % make cleanup

8. Reinstall the program source code.

   **Example:** % make reinstall

9. Attempt to execute the program. Did the deletion of the old code and reinstallation of the program source code fix the error?
   - If <u>yes</u>, you are done.
   - If <u>no</u>, go to step 10.

10. Review the libraries for the program. In the source code for the 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 program and reinstall the source for the program.

    **Example:** % cd <to appropriate library>

    % make reinstall

```
% cd/$CARSPATH/src/commgmt/tickent

% make reinstall
```

**Note:** You must reinstall the source program to include any library changes.

12. Attempt to execute the program. Did the reinstallation of the libraries for the program fix the error?
    - If yes, you are done.
    - If no, call Jenzabar Support Services.

## Program Errors

### Errors from the *ctcbatch* Program

The following is an example of an error message from *ctcbatch*:

**Message Group:  Error Messages**

**ctcupdate1 LOCK failed on idctc_rec errno=-289.**

**Unable to complete batch update. Select rerun batch to try again.**

**/usr/carsi/install/bin/ctcbatch  -t  DEV  -r  ACK  -s  E  -d 04/21/1994  -f**
You attempted to run *ctcbatch* when some records were locked.  If you receive the above error message, see *Contact Batch Entry* in this manual for more information on how to re-run *ctcbatch*.

### Errors from the *labels* Program

The following messages could appear in the running of the *labels* program:

**LPS file error occurred on %name**
This message relates to an error in creating the *lps* output file.

**LPS error occurred in lbl_print on %name**
This message relates to an error in printing to the *lps* output file.

**%number label(s) were written to the LPS form: %name**
This is a standard mail message which tells the number of labels printed for a form.  There should be one of these lines for every form used.

**%number label(s) were written to the error file: %name**
This is the mail message generated if any labels did not fit on the selected forms.

**Bad parameter: %option**
The *labels* program did not recognize the option.

**labels: %inputname %sys_err_mesg**
This message appears if the user does not have permission to read the input file.

**Error loading '%scrname' screen**
This message indicates a problem with the screen file.  Verify permissions, determine that the screen file is not corrupt and does not contain errors, and check that the screen has been correctly installed.

**Screen bind error in %name**
The screen file fields have not been correctly defined.

**Usage: labels -l scr1 [scr2...scrN][-f input_file] [-t type_text] [-c][-r][-s][-n][-C]**
The user attempted to execute the *labels* program without required parameters or with parameters which are not recognized by the program.

### Errors from the *sortpage* Program

The following error messages may occur while running *sortpage* .  The program routes error messages through user e-mail.

**Cannot open temporary file.**

The temporary file being used for intermediate storage of all the lines of input could not be opened (created). This error usually occurs because of permissions issues.

**Cannot open sort file.**
The temporary file being used for sorting (i.e., the file with the fields and values in it) could not be opened (created). This error usually occurs because of permissions issues.

**Could not sort input.**
The UNIX sort command failed. This error can occur because the field lengths were incorrect or the wrong data type was specified.

**Fields exceed Names.**
**-or-**
**Names exceed Fields.**
The number of fields in the Fields record is different from the Names record.

**Note:** If a system crash occurs when sortpage is processing, you can restart the process from the beginning.

## Errors from the *tickler* Program

The following list describes the error messages that may occur while running *tickler*, along with their probable causes. Many of the *tickler* error messages occur if you perform procedures in the incorrect sequence.

**You must select a Tickler system before choosing Edit.**
You attempted to select the Edit command before choosing a valid Tickler system code. Select a valid Tickler code.

**You must Query for an ID number before choosing Edit.**
You attempted to select the Edit command before selecting the ID of an individual whose records require editing. Select an ID.

**You must select a Tickler system before choosing Output.**
You attempted to select the Output command (in batch review mode) before choosing a valid Tickler system code. Select a valid Tickler code.

**You must Query for an ID number before choosing Output.**
You attempted to select the Output command (in interactive mode) before selecting the ID of an individual whose records you wish to print or view. Select an ID.

**You must select a Tickler system before choosing Query.**
You attempted to select the Query command before choosing a valid Tickler system code. Select an ID.

**Not a valid ID number.**
You attempted to enter an invalid ID number (within Query Mode). The ID probably does not exist on your database. Select a valid ID.

**You must first find an ID to UPDATE.**
You attempted to enter Tickler Update Mode before selecting the ID record you wish to update. Select an ID.

**This ID is locked by another user.**
The ID record you selected is locked and currently being used by another user. Try selecting the locked ID record at a later time.

**Invalid contact resource for this Tickler.**

Within Edit Mode, you are attempting to add a contact resource code that is not defined for the Tickler system selected.  If the contact is new, add its definition to the Contact table before using the code.

**You must select a Tickler system before choosing Step.**
You attempted to select the Step command before choosing a valid Tickler system code. Select a valid Tickler code.

**You must Query for an ID before choosing Step.**
You attempted to select the Step command before choosing the ID of the individual with whom you wish to work.  Select an ID.

**Not a valid Tickler code.**
You attempted to use a Tickler code that is not in the Tickler table.  If the code is new, add its definition to the Tickler table before using the code.

**Invalid track code.**
You attempted to use a Track code that is not in the Track table.  If the code is new, add its definition to the Track table before using the code.

**No Tickler system parameter.**
One of the background versions of Tickler, (Output or Review), was attempted without passing a Tickler code.

**ID#NNNNNN: no current step.**
The *tickler* program is unable to schedule any contacts for this individual because there are no active steps.

**ID#NNNNNN: no contacts to schedule.**
The *tickler* program could not find any contacts to schedule within the currently active steps for this individual.

**XXXXXXXXX already has a Tickler record.**
You attempted to add a Tickler record for an individual who already has a Tickler record. Duplicate records are not permitted.  Duplicate means a record with the same Tickler code and ID.

**Cannot open file 'XXXXXXX'**
You have a file in your working directory for which there are no "write" permissions, and *tickler* is attempting to write to the file.

**Example:** This error occurs if the Tickler.out file had no write permissions for you, and you attempted to execute the Output command that produces the "Tickler.out" file.

# SECTION 22 - LETTER WRITING ERRORS

## Overview

### Introduction

This section provides information about resolving errors that might occur in the letter writing or label generation processes, and includes the following:

- Parameters for running the *ltbrun* script
- Setting up *ltbrun*
- The *ltbrun* debugging process
- Resolving problems
- Debugging examples

Errors in letter writing can occur from a variety of sources. For example, macros may be incorrect or invalid file names may be used. Debugging letters is an important aspect of ensuring accurate correspondence. Because letter production involves a number of steps, the debugging process can also require a number of steps.

The letter and/or label generation process normally uses a script, $CARSPATH/modules/common/scripts/ltbrun, to run a number of different processes to accomplish its goals. These processes include *sacego*, *adr*, *sortpage*, *splitpage*, *ltrformat*, *labels* and *isql*. The *ltbrun* script uses the information passed to it and parses it to determine some of the input parameters to be used for the above mentioned processes.

For example, the tick code will determine which module the ACE is located in, and the File Cabinet containing the letter. It uses the following variables to do this:

| | | | | | |
|---|---|---|---|---|---|
| tickcodes | TICK_ADM | TICK_ADMG | TICK_LEAD | TICK_DEV | " " |
| tickmod | admit | admit | admit | develop | common |
| usrmod | admissions | admissions | admissions | development | 'groups' |

At this time, the standard product only handles admissions, development, etc. If your institution wants to run letterwriting in other areas, then you must modify the previous three variables.

**Note:** If an entry is added, a corresponding entry <u>must</u> be added to the others also.

The arrays, tickcodes, tickmod, and usrmod, determine much of the basic operation of *ltbrun*. If the process is going to run an ACE report to collect data which will be merged with a letter, the process must determine the locations of the ACE report and the letter. The tickmod array gives ltbrun information regarding the location of the ACE report. The usrmod array gives the name of the WPVI File Cabinet to use. The letter will be in the letters drawer of this File Cabinet. Upon evaluating the input parameters, *ltbrun* sets up input parameters for the processes it is going to start.

### Prerequisites

Before you begin debugging letters, determine the following:

- All parameters you used to run the process from the menu (see menu history file)
- The name and location of the ACE report you used
- The parameters required to run the ACE report
- The name and location of the letter you used for the process
- The technique you used to format the letter (either *nroff*, *ltrformat*, or word processor)
- The Contact table entry that relates to the letter

## Parameters for Running the *ltbrun* Script

The *ltbrun* script uses required and optional parameters.

### Required *ltbrun* Parameters

You must enter the required parameters in the sequence shown below. They may be interspersed in any fashion with the optional parameters. These parameters are passed to the letter ACE report in addition to any use made of them by *ltbrun*.

- $1 - Tickler Code for labels (e.g., ADM)
- $2 - Ace report path from ARC__PATH (e.g., admit/ltradmit/lbladmit)
- $3 - Contact Resource name for labels (e.g., CATALOG)
- $4 - Date for printing on letters
- $5 - Date for checking due date
- $(6+n) - Additional optional parameters to be passed to the ACE report

### Optional *ltbrun* Parameters

You can use these optional parameters in any sequence when you use the *ltbrun* script:

**-DEBUG**
Optional - Outputs intermediate commands/files to the user's home directory. This parameter is minimally useful.

**-USERACE**
Optional - Specifies that the ACE report is in WPPATH.

**-SELECT**
Optional - Determines the type of output. Specify BOTH, LETTERS, LABELS or NONE.

**-ADR**
Optional - Controls several selection options, depending on the values of the following flags you specify with the parameter. If you use -ADR, four values should follow it with a space between each value (e.g. N N N O).

- -b
  Indicates that you want to join related IDs for addresses and salutations only if both of the IDs satisfy the selection criteria.
- -i
  Indicates that you want to include incorrect addresses in the output.
- -e
  Indicates that you want to include deceased individuals in the output.
- <id>
  Indicates that you want to select only the specified ID.

**FORMATTER**
Optional - The letter formatting program. Default is $BINPATH/ltrformat.

**-STDLPS**
Optional - Indicates the formatter control filename (without the .ltr extension) in ${CARSPATH}/install/ltr/common directory. Currently, only *stdlps*, *wordp*, and *rtf* are supported.

**-LABELS**
Optional - Controls several label printing options, depending on which of the following flags or values you specify with the parameter:

- Y/N
  Indicates that you want to print the ID number and date on first label line.

- <string>
  Specifies the form or blank-separated forms you will use for printing.
- -c
  Indicates that you want to print the labels in column format. If you do not use this flag, the labels print in row format.
- -s
  Indicates that you want to truncate the labels to fit on the last form. You cannot use this option with 1-up labels.
- -C
  Produces labels in all capital letters.

**-BULK**
Optional - Controls whether or not the new bulk mail processing will be used. Specify either Y or N.

**-MSGS**
Optional - If specified, the next parameter should be the name of the message file in which mail messages will accumulate during the run of the script. This is primarily useful if ltbrun will run under the control of another script.

### Example to Run ***ltradmit*** Using the ***ltbrun*** Script

```
$SCPPATH/common/ltbrun.scp -SELECT BOTH ADM ltradmit "ACCEPTED" "09/01/1997"
"09/01/1997" -ADR N N N 0 -LABELS N 1up5x35 R N N
```

## Setting Up the Debugging Feature

### Considerations

Although the *ltbrun* script offers the debugging capability, setting up *ltbrun* to use the feature in a practical way requires some consideration.

If you modify menuopts to pass the debugging parameter, all users selecting the menuopt will obtain the debugging version of *ltbrun*. An alternative is to edit the installed version of ltbrun, but if you do that, all users will have debugging enabled.

If you want menu users to be able to run letter writing in debug mode and not affect any user not wanting debug mode, you must create a new menuopt. To do this, copy the normal menuopt and add the line as shown below:

```
PAn:
    optional, default = “-DEBUG”;
```

where n is the next number to be used.

Then the user can turn debugging on when developing a new letter or user ACE report. Normally, only advanced menu users will be interested in this option. However, shell users will frequently need this capability. This feature makes it easy for the shell user to individually activate the debugging capability while running letter writing (whether from the shell or menu). To activate this feature, the environment variable LTBDEBUG must be defined.

The following lines:

```
#
#     Process Parameters looking for Optional - Parameters
#
set Debug = 0
set Aceparms = “ “
set Addparms = “ “
```

were changed to:

```
#
#     Process Parameters looking for Optional - Parameters
#
set Debug = 0
set Debug = $?LTBDEBUG
set Aceparms = “ “
set Addparms = “ “
```

When you want to debug the letter writing process, set the environment variable using the following command:

```
setenv LTBDEBUG 1
```

The debugging feature captures some of the intermediate files and copies them to the user’s home directory along with information on parameters passed to the individual processes. However, you should use discretion; for large letter runs (e.g., 15,000), debugging should probably not be enabled. When debugging is enabled, the following files are created in the user’s home directory in the order shown:

- ltbrun1.ltr
- ltbrunm4.ltr
- ltbrun2.adr
- ltbrun2.ltr
- ltbrun2.lbl

- ltbrun2.sql
- ltbrun2.blk
- ltbrun.debug

The file ltbrun.debug is appended to instead of being opened for output so it continually grows. It does not grow very fast, however. It contains the actual command lines used by *ltbrun* to call the various subprocesses, such as *runreports*, *labels*, etc. The other files are copies of intermediate files and are useful to track down specific problems. To determine exactly what they are, refer to the letterwriting process flow diagram.

- The file ltbrun1.ltr is the actual output of the ACE report with no other processing.
- The file ltbrunm4.ltr is the output of m4 which processes the file in ltbrun1.ltr, replacing the WP_ macros with their correct values. m4 is called at this time to allow the letterwriting process to create word processor merge files. The ltbrun1.ltr file will normally contain, for example, the WP_ macro, WP_TODAY. If the value passed for -STDLPS was stdlps, then WP_TODAY is expanded to TD. If the value passed for -STDLPS was either *wordp* or *rtf*, then WP_TODAY expands to TODAY.
- The file ltbrun2.adr is the output of the *adr* program. The input file to *adr* was the contents of the file ltbrunm4.ltr. The *adr* program will process all lines which are prefixed with the string “&&&”. The lines with &&&run_code and &&&prin_id are input to *adr*, while lines such as &&&label: and &&&value:name are requests for data from *adr*.
- The file ltbrun2.ltr is output from *splitpage*, and has been run through *adr* to get correct address information, and *sortpage* for sorting in correct order. It should only contain merge file information.
- The file ltbrun2.lbl is output from the *splitpage* program and contains data for the *labels* program.
- The file ltbrun2.sql is output from *splitpage* and contains SQL statements to be sent to the program *isql*. These statements will affect the contact record and tickler records for the primary id in the record. The statements will set the completion date, compl_date, to today’s date and set the status, stat, to “E”. The statements will set the next review date of the tickler record, next_rvw_date, to today so that the next time tickler runs, the individual specified by the primary id will be reviewed.
- The file ltbrun2.blk contains information for the CX Bulk Mail System. If bulk mail processing was requested in the menuopt, then this file is copied to the Merge drawer of the appropriate File Cabinet.
- The ltbrun.debug file contains a set of lines for each run of *ltbrun* during which debugging was enabled. It contains a date-time stamp followed by the parameters that were actually passed to *ltbrun*, followed by the command line used by ltbrun to call subsidiary processes.

## Debugging Process Used by *ltbrun*

### *ltbrun* Process

The following process is used by *ltbrun* in Debug mode.

1. The *ltbrun* script creates the following six temporary files when run in Debug mode:
   - inffile = /tmp/label$$.sql
   - ltrfile = /tmp/label$$.ltr
   - lblfile = /tmp/label$$.lbl
   - tmpfile = /tmp/label$$.tmp
   - blkfile = /tmp/label$$.blk
   - adrfile + /tmp/label$$.adr
2. The script then determines the correct ACE report to run, and after selecting the ACE report, ltbrun runs the script *runreports*. Every call to another script or program is recorded in complete detail in the file ~/ltrbrun.debug. Your review of this file confirms that the menuopt passed the correct parameters, and that *ltbrun* passed the correct parameters to its subsidiary processes.
3. The tmpfile contains the output of the ACE report. In Debug mode, *ltbrun* copies this file into ~/ltbrun1.ltr. ltbrun1.ltr contains the unaltered output of the ACE report. Perusal of this file will frequently uncover problems due to the function of the ACE report, or data entry errors. This ACE report output contains macro names that must be put into the correct format.
   - If nroff letters are desired, the macros must be replaced with correct 2 character codes.
   - If word processor merge files are desired, the prefix WP_ must be removed.

   **Note:** The Debug mode may create large output files with detailed information about the process. Therefore, Jenzabar recommends that you execute only small letter runs in Debug mode to minimize the debugging file sizes.
4. The m4 processor then operates on the file $tmpfile to create the output file $ltrfile. In Debug mode, this file is copied into ~/ltbrunm4.ltr.
5. The output from m4 then serves as input to *adr* which is copied to ~/ltbrunm4.ltr
6. Output from *adr* becomes input to *sortpage*. . In debug mode, adr sends its output to ~/ltbrun2.adr.
7. Output from *sortpage* becomes input to *splitpage*.
8. Letter output from *splitpage* is stored in $ltrfile, while label output is stored in $lblfile. SQL output is stored in $inffile and bulk mail output is stored in $blkfile. In Debug mode, each of these files is copied into a respective *ltbrun* file, designated as follows:
   - ~/ltbrun2.ltr
   - ~/ltbrun2.ltr
   - ~/ltbrun2.lbl
   - ~/ltbrun2.sql
   - ~/ltbrun2.blk
9. If the original value of the parameter -SELECT was “LETTERS” or “BOTH”, then the script *runletters* runs next. Input for *runletters* comes from the file $ltrfile, and its output is sent directly to $CARSPATH/spool/lps. The value of the desired formatter and format type are also sent to *runletters* along with the resource code. Normally, the formatter used will be ltrformat, but if the institution has written letters that use the nroff explicit .rd command, then nroff must be used. The value of the parameter -STDLPS determines the correct letter template file to use. A value of *stdlps* means the file stdlps.ltr will be used and *ltrformat* will

produce *nroff* letters in the *lps* directory.

10. A value of *rtf* or *wordp* means that *ltbformat* will create a word processor merge file and place it in the merge drawer of the user's FileCabinet. The user then needs to download the file to the PC and perform the merge manually.
11. If you designated that you wanted labels, then the *labels* program executes next. Input for *labels* comes from $lblfile, and its output is written directly into the ${CARSPATH}Spool/lps directory.
12. As the final step, if no errors have occurred, then the *runinfs* script executes to update the Contact and Tickler records, using the $inffile as input.

    **Note:** If any errors occur in any of the steps 1 - 7, then this step does not execute. You can therefore run the process as many times as required without impacting the records that track correspondence and scheduling. If there were no runtime errors, but you are developing a complex letter and want to keep rerunning the process, you can use the environment variable LTBRERUN. If defined, LTBRERUN will disable the isql process, thus allowing the process to rerun as many times as necessary during testing using the same set of contact records.

## Resolving Letter Writing Problems

### Debugging Process

Since the *ltbrun* script uses several different processes, you must isolate which process is failing. If this is not obvious from the E-mail message, do the following.

1. Begin by turning on debugging and producing the debug file. This file will contain the actual output for the ACE report. Frequently, the problem with a letter writing run is due to incorrect data in the database. This problem can often be uncovered by looking at ltbrun2.ltr. The user may expect some letters (and the letters due report may indicate this is true), but they do not appear in *lps*. If there are no records in ltbrun1.ltr for the individual, then no letters will result. In this case, examine the Select conditions to determine if the absence of a record in the database may result in a record being skipped.

   See the example output from ACE report (ltbrun1ltr) in the *Debugging Examples* section of this guide.

2. Verify that the ACE report has selected the individuals expected and the macros have been set up with correct values. This step is done by examining the file ltbrunm4.ltr. Make sure the macros have expanded to the correct format for the desired merge file type. Make sure the value of the macro WP_LETTER_NAME is correct. Examine the values of all macros. Verify that there are as many record in lbrunm4.ltr as there were in ltbrun1.ltr.

   See the example output from the macro preprocessor m4 (ltbrunm4.ltr) in the *Debugging Examples* section of this guide.

3. Even if the ACE report has selected data for an individual, it is still possible no letters are produced. The output from the ACE report is passed through adr. If *adr* rejects the individual (for example, because of a deceased flag set to Y), then the corresponding merge record will be missing.

   Run the output of the ACE through *adr*. The *adr* process can take a -b (join records), -i (include incorrect addresses), -e (include deceased individuals), -u (user id to be used for address selection) and -d (date to be used to evaluation of data). The -d parameter is always passed to *adr*. All others are optional. The runcode being used by *adr* is part of the output of the ACE.

   ```
   adr -d "08/30/1993" < ltbrun1.ltr > adr.out &
   ```

   See the example output from the *adr* program (ltbrun2.adr) in the *Debugging Examples* section of this guide.

4. Review the output of *adr* and determine that the appropriate *adr* values are present.

5. If the ACE uses the sort macros (these macros that begin with SRT) the output of *adr* is then processed by *sortpage*.

   ```
   sortpage < adr.out > sortpage.out &
   ```

6. Review the output of *sortpage* and verify that the output is sorted in the order specified.

7. The next process splits the output of *sortpage* into four parts. One to be used with the *labels* program, one to be used with the *isql* program, one to be used by *ltrformat,* and the other used with the new bulk mail process.

8. The *splitpage* program splits entries as follows:
   - All entries between "%%%" (marked at the beginning and end of a block) are split into a labelfile to be used by the *labels* program.
   - All entries between "+++" (marked at the beginning and end of a block) are split into a isql file to be used by the *isql* program.

- All lines between "---" (marked at the beginning and end of a block) are split into a bulk file to be placed in the Merge drawer.

9. All remaining parts of the output are saved into a file to be used with the *ltrformat* program.
   ```
   splitpage +"%%%" labelfile +"+++" isqlfile +"---" bulkfile < sortpage.out > ltrfile &
   ```
10. Review the four output files and verify their validity.
11. The letters can now be created and will be stored in $CARSPATH/spool/lps/resource.l??. The *runletters* script will search the *lps* directory looking for a file name to use. Normally, the output name for a letter will be the value of the resource code followed by a period, followed by l, and then a number in the range of 1-99. The *runletters* script will determine a number that is currently unused. The *runletters* script queues message to mail queue. The *ltrformat* program uses the stdlps.ltr file to determine how to format the output. For example, you could manually run the script.
   ```
   $SCPPATH/common/runletters.scp -MSGS message.out resource $BINPATH/ltrformat
   $LTRPATH/common/stdlps.ltr < ltbrun2.ltr >
   ```
12. If the *ltbrun* process was run with the labels option (i.e., SELECT is LABELS or BOTH), it will then execute the label creation process. The *labels* program accepts the following parameters from *ltbrun*:
   - -f [label file name to be processed] (required)
   - -t [the label output file name to be used in LPS] (required)
   - -l [the label format screen name] (required)
   - -c arrange by column - optional
   - -s squeeze option - optional
   - -C capitalize label - optional
   ```
   labels -f labelfile -t Lresource -l 1up5x35
   ```
13. Review the output in $CARSPATH/spool/lps.
14. The final step is to update the contacts or any other record in the database.
   ```
   $SCPPATH/common/runinfs.scp -MSGS ltbrun.msg isqlfile
   ```
15. Verify that the appropriate records have been updated.

## Testing Letters

You can also test letters you have written by entering the following command at the UNIX prompt:

```
setenv LTBRERUN1
```

You must enter this command before you run the *ltrformat* process. The command causes the system to format your letters but does not update the Status of Contact records to (C)ompleted. You can therefore rerun the letter process as many times as desired to ensure the desired output.

## Debugging Process Example

The following example illustrates the debugging process:

Assume a contact was added, ACCPTEST, to the individual with id 3. This contact was set up as a LTLB (BOTH) type with report set to ltradmit. Debugging was enabled and the debug files were collected. These are shown in the examples section.

Since no deliberate error existed at the time, there were no failures, but due to a bug in ltradmit, multiple records were produced and shown in ltbrun1.ltr. However, duplicates were rejected so only one record occurs in ltbrun2.adr and later files.

**Note:** In the interests of brevity, all macros and their values not used in the letter were deleted from these files. The ACE report, ltradmit, produces many more lines in the merge file.

## Common Letter Writing Problems

Errors can occur under the following conditions:

**A problem develops even though none of the components of the letter writing process are thought to have changed.**
The most probable explanation of this condition is the installation of a recent product update, or a change in the data (e.g., the Contact table, or the records used by *adr*).

One of the supporting tables used in the ACE report which has a non-optional join may have been changed. This type of problem can usually be resolved by examining the contents of the file ltbrun1.ltr in conjunction with the ACE report and the actual data.

**The Jenzabar coordinator or other shell user added new fields or macros to an ACE report.**
Examine the ACE report output to determine that it is selecting the correct information.

**Note:** A change to an ACE report can also uncover errors in the data that had gone previously undiscovered.

**The Jenzabar coordinator or other shell user created a new ACE report.**
Check the output against a standard CX report (e.g., ltradmit) that you have used successfully.

**The Jenzabar coordinator or other shell user added a new menuopt to a user's menu.**
In this case, review the parameter string used with the scripts. The most likely script to experience errors is *ltbrun*. To debug ltbrun, use the ltbrun debugging process. Examine the menu history file.

## Error Messages

Common errors include the following: **{** XE "errors:letter writing" **}{** XE "letter writing:errors" **}**

**Invalid macro name <filename.exp>**
The test file used with the letter must contain the macro WP_LTR_NAME on the last line. Verify the last line of the test file.

**Missing letter file in qpvi**
A letter with the same name as the resource code must be in the letter drawer of the user's File Cabinet for a common code of LTLB or LETT. A letter must be present even if it is not going to be used, as in the case of a WordPerfect Merge file.

**Invalid value given for Comm code** (e.g., LETTERS or BOTH)
The user-entered value must match the type in the contact table for the resource code used, or most standard CX letter ACE reports will reject the resource code.

**No run code found**
This error can occur when nothing has changed. It is one of the most common. It occurs because the ACE report did not find any appropriate data. It may either output nothing or only lps header information. This is not actually an error , but adr handles it as such.

**Unexpected End of File**
This error is usually from m4 or nroff. This error can occur if:

- A user entered an id record row with a name containing a back quote `. During the macro conversion stage when m4 encounters this character, it enters a different mode and scans for the next occurrence. Since there usually is not one, it hits the end of file.  If it is absolutely necessary to use the back quote, you can modify the ltbrun script to use an alternate quote.

- The second problem occurs in the use of the WP_IF macros if the user does not properly terminate the if section on a rarely used option.

## Debugging Examples

You can use the following examples in your debugging process.  Examples are provided for:

- Source letter
- Compiled letter
- Output from ACE report
- Output from the macro preprocessor m4
- Output from adr program
- Bulk mail file
- Label file
- File for ltr format
- File to be sent to isql
- File of commands used in process

**Example Source Letter**

```
WP_HEAD(10,WP_SALUT,:)
WP_PAR
WP_IF(WP_DECISION,FULL) \{
WP_PAR
It is my pleasure to inform you that you have been admitted to CARS
College for the WP_PLAN_ENR_YEAR WP_PLAN_ENR_SESS session.\}
WP_IF(WP_DECISION,COND) \{
WP_PAR
It is my pleasure to inform you that you have been conditionally
admitted to CARS College for the WP_PLAN_ENR_YEAR WP_PLAN_ENR_SESS
session.  Prior to your enrollment at CARS you will need to
successfully complete two college level courses consisting of six
semester hours or eight quarter hours.  We would consider a 'C'
in each course to be successful.  You may take the courses at any
local community college, or other college or university.
WP_PAR
Your previous work will be to enable you to carry a lighter load
(12-13 credit hours) in the WP_PLAN_ENR_SESS.  This will permit a
greater portion of study time on the courses you do take; and you
will not be at any disadvantage in taking a lighter load.
WP_PAR
Please do not hesitate to contact me if you need anything clarified.
The faculty, students, and administration join me in welcoming you
to CARS College!\}
WP_IF(WP_DECISION,TRAN) \{
WP_PAR
Your acceptance is conditional until we receive a final transcript of
your college work.  Upon completion of all coursework, request that
the Registrar's Office send an official transcript to the Office of
Admissions at CARS College.  The Registrar's Office here will evaluate
your transcript for the purpose of granting transfer credit.\}
WP_IF(WP_DECISION,TEST) \{
WP_PAR
Your acceptance is conditional until we receive your SAT or ACT results.
One of the two tests must be taken before registering for WP_PLAN_ENR_SESS
classes.  The ACT test is given on campus.  Please call if you are
interested in more details.\}
WP_IF(WP_DECISION,PROB) \{
WP_PAR
Due to your past academic record, you have been admitted on probation
for your first semester.  Please refer to pages 21-22 of your college
catalog as to the restrictions this imposes.  At the end of the first
semester, your progress will be reviewed and if satisfactory progress
has been made you will be removed from probation.\}
WP_PAR
The faculty, students, and administration join me in welcoming you to
CARS College!  Please do not hesitate to contact our Admissions Office
if you need other matters clarified.
WP_PAR
We do ask that you confirm your intention to attend CARS by submitting
your $150.00 general student deposit.  If you plan to live in a college
dormitory, please note that HOUSING PREFERENCE IS GIVEN AS ADVANCE
DEPOSITS ARE RECEIVED.
WP_CLOSE
Jack Evans, Ed.D.
Director of Admissions


JE/pam
WP_END
```

### Example Compiled Letter

```
.ll (115*7i/100)
.sp 10
\*(TD
.sp 2
\*(LB
.sp
.fi
Dear \*(SD:

.ti +5
.if '\*(DC'FULL'  \{

.ti +5
It is my pleasure to inform you that you have been admitted to CARS
College for the \*(PY \*(PS session.\}
.if '\*(DC'COND'  \{

.ti +5
It is my pleasure to inform you that you have been conditionally
admitted to CARS College for the \*(PY \*(PS
session.  Prior to your enrollment at CARS you will need to
successfully complete two college level courses consisting of six
semester hours or eight quarter hours.  We would consider a 'C'
in each course to be successful.  You may take the courses at any
local community college, or other college or university.

.ti +5
Your previous work will be to enable you to carry a lighter load
(12-13 credit hours) in the \*(PS.  This will permit a
greater portion of study time on the courses you do take; and you
will not be at any disadvantage in taking a lighter load.

.ti +5
Please do not hesitate to contact me if you need anything clarified.
The faculty, students, and administration join me in welcoming you
to CARS College!\}
.if '\*(DC'TRAN'  \{

.ti +5
Your acceptance is conditional until we receive a final transcript of
your college work.  Upon completion of all coursework, request that
the Registrar's Office send an official transcript to the Office of
Admissions at CARS College.  The Registrar's Office here will evaluate
your transcript for the purpose of granting transfer credit.\}
.if '\*(DC'TEST'  \{

.ti +5
Your acceptance is conditional until we receive your SAT or ACT results.
One of the two tests must be taken before registering for \*(PS
classes.  The ACT test is given on campus.  Please call if you are
interested in more details.\}
.if '\*(DC'PROB'  \{

.ti +5
Due to your past academic record, you have been admitted on probation
for your first semester.  Please refer to pages 21-22 of your college
catalog as to the restrictions this imposes.  At the end of the first
semester, your progress will be reviewed and if satisfactory progress
has been made you will be removed from probation.\}

.ti +5
The faculty, students, and administration join me in welcoming you to
CARS College!  Please do not hesitate to contact our Admissions Office
if you need other matters clarified.

.ti +5
```

```
We do ask that you confirm your intention to attend CARS by submitting
your $150.00 general student deposit.  If you plan to live in a college
dormitory, please note that HOUSING PREFERENCE IS GIVEN AS ADVANCE
DEPOSITS ARE RECEIVED.

.nf
Sincerely,




Jack Evans, Ed.D.
Director of Admissions


JE/pam
.ch tt
.ch bt
.po 0
.in 0
```

**Example Output from ACE Report (ltbrun1.ltr)**

```
>>>date created:02/11/97
>>>time created: 9:45 pm
>>>letter type:ACCPTEST letters
>>>subtype:envelope
>>>subtype:letter
>>>identifier:id_no
&&&run_code:SINGLEI
&&&prim_id:3                                                info to adr
>>id:00000003

%%%
&&&label:                                          for labels program via splitpage

%%%
---
>>1
&&&value:name
>>2
&&&value:line1
>>3
&&&value:line2                                     for Merge drawer - bulk mail file via
splitpage
>>4                                        adr will replace value requests with data
&&&value:city
>>5
&&&value:st
>>6
&&&value:zip
>>END

---
>>type:envelope

&&&label:                                          adr will replace with value

>>type:letter

\!WP_TODAY

February 11, 1997

\!WP_SALUT

&&&salut:                                          adr

\!WP_LABEL

&&&label:

\!WP_LTR_NAME

/usr/carsdevi/wp/admissions/FileCabinet/letters/ACCPTEST.ltr          for ltr format

\!WP_PLAN_ENR_SESS

\!Transcript Foot Session

\!WP_PLAN_ENR_YEAR

\!1992

\!WP_DECISION

\!COND

##

+++
update ctc_rec
  set cmpl_date = "02/11/1997",
      stat = "C"
```

```
where ctc_no =     1181591;
+++
+++
update tick_rec                                             for isql via splitpage
  set next_rvw_date = "02/11/1997"
where tick = "ADM "
  and id =           3;
+++
&&&end:
&&&prim_id:3
>>id:00000003
```

***Note:*** *Due to the way ltrformat processes, it can produce multiple letters.*

```
%%%
&&&label:

%%%
---
>>1
&&&value:name
>>2
&&&value:line1
>>3
&&&value:line2
>>4
&&&value:city
>>5
&&&value:st
>>6
&&&value:zip
>>END

---
>>type:envelope

&&&label:

>>type:letter

\!WP_TODAY

February 11, 1997

\!WP_SALUT

&&&salut:

\!WP_LABEL

&&&label:

\!WP_LTR_NAME

/usr/carsdevi/wp/admissions/FileCabinet/letters/ACCPTEST.ltr

\!WP_PLAN_ENR_SESS

\!Transcript Foot Session

\!WP_PLAN_ENR_YEAR

\!1992

\!WP_DECISION

\!COND

##

+++
update ctc_rec
  set cmpl_date = "02/11/1997",
      stat = "C"
where ctc_no =     1181591;
+++
+++
update tick_rec
  set next_rvw_date = "02/11/1997"
where tick = "ADM "
  and id =           3;
+++
&&&end:
%%%
#       3  ACCPTEST labels  #
#    Some may be skipped   #
#######################
```

```
%%%
&&&value:count:>>>end:

.fl
.ex
```

**Example Output from Macro Preprocessor m4 (ltbrunm4.ltr)**

```
>>>date created:02/11/97
>>>time created: 9:45 pm
>>>letter type:ACCPTEST letters
>>>subtype:envelope
>>>subtype:letter
>>>identifier:id_no
&&&run_code:SINGLEI
&&&prim_id:3
>>id:00000003

%%%
&&&label:

%%%
---
>>1
&&&value:name
>>2
&&&value:line1
>>3
&&&value:line2
>>4
&&&value:city
>>5
&&&value:st
>>6
&&&value:zip
>>END

---
>>type:envelope

&&&label:

>>type:letter

\!TD

February 11, 1997

\!SD

&&&salut:

\!LB

&&&label:

\!LT

/usr/carsdevi/wp/admissions/FileCabinet/letters/ACCPTEST.ltr
```

```
\!PS

\!Transcript Foot Session

\!PY

\!1992

\!DC

\!COND

##

+++
update ctc_rec
  set cmpl_date = "02/11/1997",
      stat = "C"
where ctc_no =     1181591;
+++
+++
update tick_rec
  set next_rvw_date = "02/11/1997"
where tick = "ADM "
  and id =           3;
+++                                                    To save space, the second record
&&&end:                                                (duplicate record) has been deleted.

%%%
#     3 ACCPTEST labels #
#  Some may be skipped  #
#########################

%%%
&&&value:count:>>>end:

.fl
.ex
```

**Example Output from *adr* Program (ltbrun2.adr)**

```
>>>date created:02/11/97
>>>time created: 9:45 pm
>>>letter type:ACCPTEST letters
>>>subtype:envelope                                    Note:  All adr substitution is now done.
>>>subtype:letter                                      Since the runcode was SINGLEI which
>>>identifier:id_no                                    does not allow duplicates, the duplicate
>>id:00000003                                                  record has been discarded.

%%%
Frank H Collins, Esq
333 Bayshore Dr.
Lot # 3
Fort Meyers, FL  22344

%%%
---
>>1
Collins, Frank H
>>2
333 Bayshore Dr.
>>3
Lot # 3
>>4
Fort Meyers
>>5
FL
>>6
22344
>>END

---
>>type:envelope

Frank H Collins, Esq
333 Bayshore Dr.
Lot # 3
Fort Meyers, FL  22344

>>type:letter

\!TD

February 11, 1997

\!SD

Frank

\!LB

Frank H Collins, Esq
333 Bayshore Dr.
Lot # 3
Fort Meyers, FL  22344

\!LT

/usr/carsdevi/wp/admissions/FileCabinet/letters/ACCPTEST.ltr

\!PS
```

```
\!Transcript Foot Session

\!PY

\!1992

\!DC

\!COND

##

+++
update ctc_rec
  set cmpl_date = "02/11/1997",
      stat = "C"
where ctc_no =     1181591;
+++
+++
update tick_rec
  set next_rvw_date = "02/11/1997"
where tick = "ADM "
  and id =           3;
+++
%%%
#     2 ACCPTEST labels #
#  Some may be skipped  #
#########################

%%%
>>>end:1

.fl
.ex
```

**Example Bulk Mail File (ltbrun2.blk)**

```
>>1
Collins, Frank H
>>2
333 Bayshore Dr.
>>3
Lot # 3
>>4
Fort Meyers
>>5
FL
>>6
22344
>>END
```

**Example Label File (ltbrun2.lbl)**

```
Frank H Collins, Esq
333 Bayshore Dr.
Lot # 3
Fort Meyers, FL  22344

#     2 ACCPTEST labels #
#  Some may be skipped  #
#########################
```

**Example File to be Sent to *ltrformat* (ltbrun2.ltr)**

```
>>>date created:02/11/97
>>>time created: 9:45 pm
>>>letter type:ACCPTEST letters
>>>subtype:envelope
>>>subtype:letter
>>>identifier:id_no
>>id:00000003

>>type:envelope

Frank H Collins, Esq
333 Bayshore Dr.
Lot # 3
Fort Meyers, FL  22344

>>type:letter

\!TD

February 11, 1997

\!SD

Frank

\!LB

Frank H Collins, Esq
333 Bayshore Dr.
Lot # 3
Fort Meyers, FL  22344

\!LT

/usr/carsdevi/wp/admissions/FileCabinet/letters/ACCPTEST.ltr

\!PS

\!Transcript Foot Session

\!PY

\!1992

\!DC

\!COND

##

>>>end:1

.fl
.ex
```

### Example File to be Sent to ISQL (ltbrun2.sql)

```
update ctc_rec
  set cmpl_date = "02/11/1997",
      stat = "C"
where ctc_no =     1181591;
update tick_rec
  set next_rvw_date = "02/11/1997"
where tick = "ADM "
  and id =           3;
```

### Example Letter File (accptest.l1)

```
>>>date created:02/12/97
>>>time created: 8:51 am
>>>letter type:ACCPTEST letters
>>>subtype:envelope                                    lps header
>>>subtype:letter
>>>identifier:id_no
>>id:00000003
>>type:envelope                                       lps separator














                                        Frank H Collins, Esq
                                        333 Bayshore Dr.
                                        Lot # 3
                                        Fort Meyers, FL  22344



>>type:letter                                                  lps separator
```

```
     February 12, 1997


     Frank H Collins, Esq
     333 Bayshore Dr.
     Lot # 3
     Fort Meyers, FL  22344


     Dear Frank:



          It is my pleasure to inform you that you have been conditionally
     admitted to CARS College for the 1992 Transcript Foot Session session.
     Prior to your enrollment at CARS you will need to successfully
     complete two college level courses consisting of six semester hours or
     eight quarter hours.  We would consider a 'C' in each course to be
     successful.  You may take the courses at any local community college,
     or other college or university.

          Your previous work will be to enable you to carry a lighter load
     (12-13 credit hours) in the Transcript Foot Session.  This will permit
     a greater portion of study time on the courses you do take; and you
     will not be at any disadvantage in taking a lighter load.

          Please do not hesitate to contact me if you need anything
     clarified. The faculty, students, and administration join me in
     welcoming you to CARS College!

          The faculty, students, and administration join me in welcoming
     you to CARS College!  Please do not hesitate to contact our Admissions
     Office if you need other matters clarified.

          We do ask that you confirm your intention to attend CARS by
     submitting your $150.00 general student deposit.  If you plan to live
     in a college dormitory, please note that HOUSING PREFERENCE IS GIVEN
     AS ADVANCE DEPOSITS ARE RECEIVED.

     Sincerely,




     Jack Evans, Ed.D.
     Director of Admissions


     JE/pam
                                              To accept the test letter, compare to
                                              ensure the correct paragraphs are included.


>>>end:1
```

**Example Letter File (laccptest.l1)**

```
>>>date created:02/11/97
>>>time created: 9:45 pm
>>>letter type:LACCPTEST
>>>subtype:1up5x35
>>>identifier:row_id
>>id:Frank H Collins, Esq
>>type:1up5x35
 Frank H Collins, Esq
 333 Bayshore Dr.
 Lot # 3
 Fort Meyers, FL  22344


>>id:#     2 ACCPTEST labels #
>>type:1up5x35
 #     2 ACCPTEST labels #
 #  Some may be skipped  #
 #########################



>>>end:2
```

### Example File with Commands Used in Debugging Process (ltbrun.debug)

```
--------------------------------
Tue Feb 11 21:45:04 EST 1997
/usr/carsdevi/install/scp/common/ltbrun.scp -SELECT BOTH ADM ltradmit ACCPTEST 02/11/1997
02/11/1997 -STDLPS stdlps -BULK N -ADR N N N 0 -LABELS N 1up5x35 R N N
--------------------------------
/usr/carsdevi/install/scp/common/runreports.scp -MSGS ltbrun.msg
/usr/carsdevi/install/arc/admit//ltradmit   ADM ltradmit ACCPTEST LTLB 02/11/1997 02/11/1997 N
stdout >& ltbrun1.ltr
--------------------------------
m4 -DM4PATH=. -DWP_MODE=define ./util/start.m4 ./user/ltrwp.m4 ./util/end.m4 ltbrun1.ltr >&
ltbrunm4.ltr
--------------------------------
/usr/carsdevi/install/scp/common/runpipe.scp -MSGS ltbrun.msg /usr/carsdevi/install/bin/adr -d
02/11/1997 < ltbrun1.ltr -PIPE /usr/carsdevi/install/bin/sortpage -PIPE
/usr/carsdevi/install/utl/splitpage +--- /tmp/label19067.blk +%%% ltbrun2.lbl ++++ ltbrun2.sql >&
ltbrun2.ltr
--------------------------------
/usr/carsdevi/install/scp/common/runletters.scp -MSGS ltbrun.msg ACCPTEST ltrformat
/usr/carsdevi/install/ltr/common/stdlps.ltr < ltbrun2.ltr
--------------------------------
/usr/carsdevi/install/utl/labels -f ltbrun2.lbl -t LACCPTEST -l 1up5x35
--------------------------------
/usr/carsdevi/install/scp/common/runinfs.scp -MSGS ltbrun.msg ltbrun2.sql
```

# INDEX