CA 2E

Toolkit Concepts Guide

Release 8.7

Documentation Changes

The following documentation updates have been made since the last release of this documentation:

- Use of external source members to hold preprocessor directives (see page 126)—Updated code examples to include the block form.
- Examples of the Work With Panel Design Displays (see page 93)
- Exit Work with Panel display (see page 96)

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Introduction

CA 2E Toolkit is an integrated package comprised of software utilities for IBM i. This chapter provides an overview of Toolkit documentation, including help text and help menus. The arrangement of the *Toolkit Concepts Guide* is summarized and related publications are listed.

This section contains the following topics:

Documentation (see page 11)

Documentation

The documentation for the Toolkit utilities is divided into two guides:

- The CA 2E *Concepts Guide* gives a functional overview of the utilities, and how they link together.
- The *CA 2E Reference Guide* contains detailed explanations of each of the Toolkit commands that run the utilities.

This guide provides the basic concepts behind the product utilities, and how you can efficiently use these tools. If you have purchased only specific CA 2E modules, some of the utilities described in this guide are not available. Appendix E of this guide provides the contents of each CA 2E module.

For details on how to invoke the individual utilities, see the *CA 2E Reference Guide*.

Online Help

Further documentation is available in the form of online help. All of the interactive CA 2E utility programs have full user procedures. You can access context-sensitive help by pressing **Help** in the application.

Help Menu

Help menus are also supplied for CA 2E .They display all of the commands in alternate groupings by subject and name. To display the help menus you can use the command Go to Menu **(YGO *Y1)**.

Summary of Capabilities

Each chapter in this guide begins with an overview that summarizes the specific capabilities covered. The overview is followed by a more detailed consideration of the topics.

The following is a list of related publications for this guide:

- *Toolkit Reference Guide*
- *Standards Guide*
- *IBM i Programming: Control Language Programmer's guide*
- *IBM i Programming: Control Language Reference* Volume 1, Volume 2 SC21-9776-0, Volume 3, Volume 4, Volume 5.
- *IBM i Programming: Data Description Specifications*
- *IBM i Text Management/38 User's guide and Reference Manual*

Assumptions

It is assumed that the reader of this guide has a working knowledge of the Toolkit and, in particular, familiar with the following i OS concepts:

- Libraries
- Library lists
- Objects
- Object authorities
- Commands
- Source files

Data description specifications (DDS, i OS definition DS) are as follows:

- Subfiles
- Messages

Refer to the appropriate i OS manuals if you need more information on these topics.

CA 2E Toolkit Concept Summary

Toolkit is intended to provide the following benefits:

- Increase your productivity
- Improve the quality of what you produce
- Provide insight about using the Toolkit to the best advantage

Toolkit embodies fourth generation design principles as follows:

- The computer is used to assist and to organize both the developer and the user wherever possible.
- A consistent user interface is used throughout development.
- Help text, help menus, and select functions are available throughout the interface.
- A high-level object-orientated design approach is incorporated.

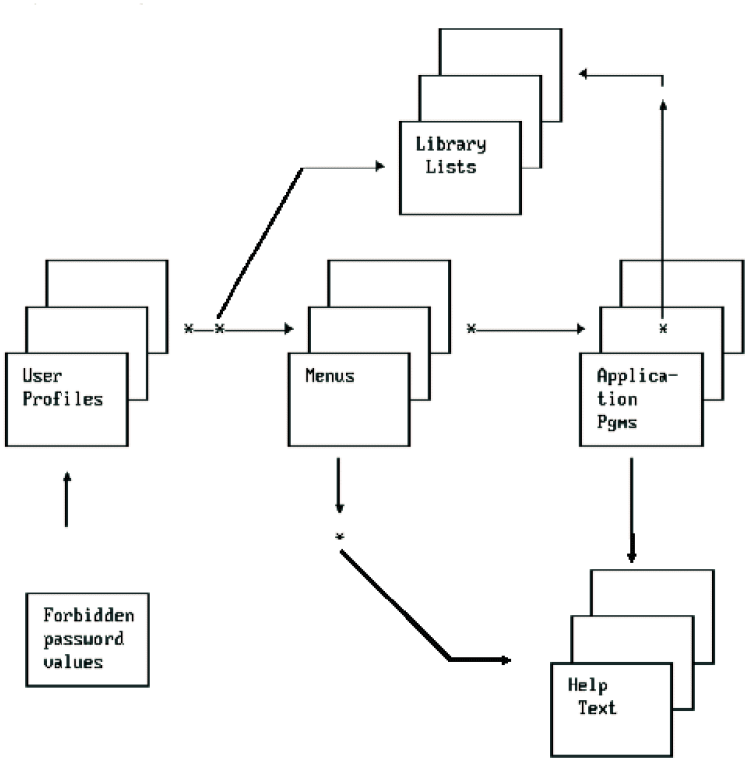
Chapter 2: User Access Utilities

This chapter provides an overview of user access aids including library lists, menus, help text, and user profiles. Linking user access components is discussed and examples are provided.

This section contains the following topics:

Introduction

The CA 2E Toolkit user access aids provide an easy way to set up and maintain operating environments for your Toolkit users. With the Toolkit user access aids, you can create and maintain a highly supportive environment for your users, without the need for programming changes.



Overview

There are four main areas covered by the user access aids:

- Library lists
- Menus
- Help text
- User profiles

Each of the components can be used separately or all of the components can be linked together to form an integrated system.

The four areas are discussed individually in detail in the pages that follow.

Type	Edit command	Print command	Display command	Other Toolkit commands
LIBL	YWRKLIBLST	YDOCLIBLST	YDSPLIBLST	YCHGLIBL YCHGJOBDLL YBLDLIBLST YDLTLIBLST YCPYLIBLST YEDTLIBLST YCHGLIBLST YWRKLIBLST YRNMLIBLST YRNMLIB
MNU	YWRKMNU	YDOCMNU	YGO	YDLTMNU YCPYMNU YRNMMNU YCRTDSNF YADDDSNFM
TXT	(STRSEU)	YDOCSRC	YDSPHLP	YBLDDOC YADDHLPTBL
USR	YWRKUSRPRF	YDOCUSRPRF	YDSPUSRPRF	YCPYUSRPRF YRTVUSRPRF YCRTUSRPRF YCHGUSRPRF YDLTUSRPRF YRNMUSRPRF YCVTUSRPRF
PWD	YEDTPWDVAL			YCHKPWDVAL YCHGPWD

Example

The menus and help text provided as part of the Toolkit documentation have been prepared, and are displayed, using the Toolkit utilities. They provide examples of what they describe, and illustrate some of the ways that you yourself can use the utilities.

Library List Facilities

Allows a list of libraries to be stored under a given name. The stored list may be retrieved, and used to replace the library list of the same or another job. The stored list may be edited. Job description library lists may also be retrieved or replaced.

CA 2E library list features include:

- An interactive list editor
- Documentation facilities
- Retrieval and replace facilities
- Selection facilities

Menu Facilities

CA 2E provides you with *virtual* menus: menus that have all the function and more of program type menus, but which require no compilation, and which can be interactively created and maintained.

Menu features include:

- SAA design
- Fully interpretive, that is, no compilation is required
- Message handling, including logging
- Option and menu level Help text
- Full recursion
- Integral system security
- Trapping of Cancel requests: the menu program is a requestor
- Command request entry (optional)
- Multi-page menus (optional)
- Option confirm prompt (optional)
- Direct transfer to a named menu (optional)
- Central logging of exception messages (optional)
- Immediate exit facility (optional)
- Optional debug and syntax checking facilities
- Automatic numbering of options (optional)
- Variable text display attributes, including color
- Full screen menu editor, similar to SEU
- Menu selection functions
- Full documentation facilities, including cross-referencing
- Option authority checking

Help Text Facilities

The Toolkit help text facilities enable you to provide interactive help text for all of your menus and application programs.

Toolkit help text features include:

- Fully interpretive, that is, no compilation
- Full screen entry
- Automatic numbering
- Support of display attributes
- Support for embedding subdocuments

Support for help compliant with common user access (CUA) standards:

- Help text index
- Contextual (cursor-sensitive) Help text
- Keys help
- Extended help
- Help for help

User Profile Facilities

CA 2E provides facilities both to help manage your existing user profiles, and to set up and maintain "extension attributes" for your profiles. The user profile extension attributes are mainly concerned with the control of access onto the system, and include an initial library list, an initial menu, and an initial break message severity.

Toolkit user profile features include:

- Full integration with system commands
- Interactive edit display
- Generic documentation facilities
- Extension attribute retrieval facilities
- Password validation

Linking User Access Components

Each of the user access aids can be used separately. CA 2E Toolkit also includes an initial program that can be used to connect user profiles, library lists, help text and menus into an integrated system.

User Access Example

To add a user to the system using the CA 2E user access aids, you might take the following steps:

1. Decide which initial library list you want the new user to have, for example, a library list called LIBEL containing the following libraries:

   QTEMP QGPL Y1SY ORDENT

2. Use the command Build Library List (YBLDLIBLST) to create the list:

   YBLDLIBLST LIBLST(LIBEL) LIBL(QTEMP QGPL Y1SY ORDENT) TEXT('Library list for FRED')

3. Decide which initial menu or program you wish the user to have. For example, menu ORDENT in file YDSNMNU in library ORDENT.

4. Use the Work with Menus (YWRKMNU) command to create the menu, if it does not already exist.

   YWRKMNU MENU(ORDENT)

5. Decide what name you wish to give the new user profile; for example, FRED.

6. Use the command Create User Profile (YCRTUSRPRF) to create the profile, specifying LIBEL as the initial library list, and ORDENT as the initial menu:

   YCRTUSRPRF USRPRF(FRED) TEXT('Fred's user profile') LIBLST(LIBEL) MENU(ORDE

7. Teach FRED how to sign on

Library List Contents

Toolkit library lists contain:

- User portion of library list (*USRLIBL) including current library. Special value 'no change' (*NOCHG) for current library makes storage of current library optional.
- Optionally, the name of a job description with an initial library list maintained by Toolkit commands to match the Toolkit library list.
- Library list type to enable subsetting Toolkit library lists.
- Text associated with the library list.

Comparing i OS and Toolkit menus

The following two diagrams illustrate the difference between the conventional i OS approach to menus and the Toolkit approach.

Note that a simple routine change, like moving an option from one menu to another, may involve as many as four recompilations, as well as a hiatus for the user while the new versions are installed.

Changing any menu can be done instantaneously.

Integrating Design and Development

The menu utilities are used not only to store application menu sets, but also as both design and development tools.

As a design tool:

User functions and menus are rapidly designed in outline, and presented to the user for comment. Help text and panel designs can be set up for each option, and the associated documentation prepared. See the chapter "Toolkit Design Aids" for further details.

As a development tool:

The design menus are carried straight into development, with actual programs replacing panel and report designs when they are completed. Menus are also used:

- To provide a framework to hold large documents for presentation, and for maintenance
- To provide a training aid: users can move around the menus in a system, looking at the help text for options, rather than executing them. Extra explanatory text and formatting can make menus very easy to understand.
- To group commonly used commands with different default parameters, without the need to write CL programs or alternative commands.
- As a presentation tool: functions are combined with explanatory text to provide a structure for computer-based presentations.

Importance of Menus

Menus are especially important for ensuring user satisfaction because they are the part of the system that is used most often. The requirements of your users as to how their menus are arranged are almost bound to change as the users become more familiar with a system, and as their work patterns evolve. It is important that you can respond quickly (and economically) to their demands.

The Toolkit menus also allow you to cater to different levels of user sophistication.

Styles of Help Text

Toolkit provides two styles of help text: Full Panel (User Option *HLPFULL) and CUA/Text (User Option *HLPWDW). Help index, help keys, and extended help are only available when *CUATEXT and the YMHPOPA data area are set to *HLPWDW.

Following is an example of a help text window:

General Considerations for Help Text

To manipulate Toolkit help text, see the following Toolkit command diagrams in the *CA 2E Toolkit Reference Guide*:

Storing Help Text

Toolkit help text is stored as normal text source. Each member in the source file constitutes a document, and can be referenced and displayed separately.

Displaying Help Text

You can display help text using the Toolkit command Display Help (YDSPHLP), or by calling the Help display program directly from another HLL program. For details, see the command diagram in the *CA 2E Toolkit Reference Guide*.

Each call to the help text display program names a single text member. From this member, you can display:

- All the text, starting at the beginning
- An index, which allows you to jump to the part of the text you are interested in
- Text relevant to the position of the cursor on the panel from which the help text was called
- Text relevant to the conditions under which the help text was called (for example, different text depending on whether you are adding or changing records)

Associating Help Text with a Menu Option

To associate help text with a menu option, use the Toolkit command Work with Menu (YWRKMNU) to specify the name of a help text document for the option. When the menu is displayed using the Toolkit Go to Menu utility (YGO), contextual help text is available for the option if the user moves the cursor to the option line, and presses F1=HELP. Note that users can also access this information by choosing Help Index from the Help action bar pull-down, then selecting the desired topic.

Associating Help Text with an HLL Program

To have help text displayed by an HLL program, you must enable a help function key in the DDS of the display file used by the program, and also encode a call to the Toolkit program Display Help (YDDSHPR) in the program source. This can be done very simply using a standardized subroutine: see the Toolkit command Display Help (YDSPHLP) diagram for details and an example.

Displaying Help Text Using the YDSPHLP Command

Help text can be displayed interactively by using the Toolkit command Display Help (YDSPHLP). See the command diagram for the Display Help command for details. The Display Help command can be used to check help text, or to present a document to a user as a menu option.

Naming Help Text Documents

Give careful consideration to the names for help text documents. Factors to consider include:

- The default document name used by the Toolkit program Go to Menu (YGO). If no help text member is specified for an option, the default help text document name used by the menu program is the name of the program or command specified for the option. It will save you coding effort if you give a help text member the same name as that of the program or command that it explains.
- How to associate programs with their help text. The standard help text call included in application programs can use the program name (retrieved from the Program Status Data structure) as the help text document name, to save coding effort.
- How the documents will be grouped when they appear in indices. It helps to have documents grouped in a logical order. If you already give your programs names that cause them to be grouped in a logical order on indices, then it would be sensible to give each help text document the same name as the program that it explains.

Note: You should give a help text member the same name as the command or program for which it gives an explanation.

Help Text Default Source Files

Also give careful consideration to what name you give to the file or files containing your help text.

When you specify a help text document for a menu option or program, you may specify the name of a file and library containing the document. If you leave the file and library blank, the default help text file will be used. The default help text file is set by the data areas YMHPFLA and YMHPLBA:

- YMHPFLA - specifies the name of the default help text file.
- YMHPLBA - specifies the name of the default help text file library.

Default versions of the data areas are provided in the Toolkit library. You are free to change the values of these, or to provide additional copies in other libraries.

If the YMHPFLA data area contains a value of QTXTSRC, and YMHPLBA contains a value of *LIBL, then the following text member names will be used:

PGM/cmd	Specified on MENU or CALL			Used by YDSPHLP		
	Mbr	File	Library	Mbr	File	Library
BILL	--	--	--	BILL	QTXTSRC	*LIBL
BILL	--	FREDTXT	--	BILL	FREDTXT	*LIBL

PGM/cmd	Specified on MENU or CALL			Used by YDSPHLP		
	Mbr	File	Library	Mbr	File	Library
BILL	--	--	HUBERT	BILL	QTXTSRCT	HUBERT
BILL	--	FREDTXT		BILL	FREDTXT	
BILL	HUBERT			HUBERT		
BILL	FRED	--	--	FRED	QTXTSRC	*LIBL
BILL	FRED	FREDTXT	--	FRED	FREDTXT	*LIBL
BILL	FRED	--		FRED	QTXTSRC	
	HUBERT			HUBERT		
	FRED	FREDTXT		FRED	FREDTXT	
	HUBERT			HUBERT		

Note: A member name must be specified when calling the help text display program from within an application program.

Library List Resolution

If a value of *LIBL is specified for the help text file library, the help display program uses the invoking job's library list to find the text member: the first file containing a source member of the specified name is used.

For example:

If you use the following request to display a help text member FRED:

```
YDSPHLP MBR(FRED) FILE(*LIBL/UHLPTXT)
```

Your library list contains the following files:

Library	File	Member
QGPL	UHLPTXT	FRED
EVERYMAN	UHLPTXT	FRED
BIBLIO	UHLPTXT	FRED

Then the YDSPHLP command does the following:

- Examines UHLPTXT in QGPL for FRED and does not find a candidate member.
- Examines UHLPTXT in EVERYMAN for member FRED, finds a candidate, and uses the candidate.

Types of Help Information

Toolkit supports the common user access (CUA) help text model which provides the following types of help:

- Contextual - Specific to cursor position gives the purpose and use of the item for which the user requested help.
- Extended - Describes the application panel from which the user requested Help.
- Help Index - Sequenced list of help topics for the application from which the user can select a topic for display.
- Keys help - List of application function keys, including brief descriptions.
- Help for Help - Information on how to use help facilities, including how to access contextual, extended, keys, and index help.

Internal Help Functions

Toolkit supports several user-directed options, including:

- Enlarged help enlarges the help window to full window size (CUA/Text style only).
- Reposition moves the help window to the cursor position (CUA/Text style only).
- Print help prints extended help for the application.
- Edit help invokes the edit function on the extended help source member.

Requesting Help

Users can request help in two general ways:

- Select one of the Help action bar pull-down choices, which results in display of a help pop-up window containing the requested topic.
- Press function keys assigned to common help actions, which results in one of the following:
  - For F1= Help from an application panel, access to contextual or extended help in a help panel, depending on cursor location
  - For function keys from help panels, access to the type of help specific to the function key

Users can access specific types of help as follows:

- To request Contextual help, press F1= Help from an entry field, a selection field, or other defined contextual help area on the application panel. A help panel appears containing information about the specific field or item.
- To request Extended help, press F1= Help from anywhere on the application panel other than a defined contextual help area, or press F2= Extended from any help panel. A full help panel appears containing information on how to use the application panel.
- To request Keys help, press F9= Keys help from any help panel. A help window displays a list of application function keys and the purpose of each.
- To request Help Index, press F11= Index from any help panel. A help window presents a scrollable list of help topics from which to select one for display.
- To request Help for Help, press F1= Help from any help panel. A help pop-up displays information on using the help facilities.

Users navigating through various help panels can either end the help session directly by pressing F3= Exit or back out one panel at a time by pressing F12=Cancel.

Once a help panel appears, users can size and position the panel to their preference as follows:

- To enlarge the help window, simply press F20= Enlarge from any help window.
- To reposition the help window, position the cursor at the desired location and press F19= Reposition. (An enlarged help window will return to its original size when repositioned.)

  Users can also have the option to print and edit help as follows:
- To print help, press F14= Print from any help panel (except Help Index).
- To edit help, press F15= Edit from any help panel (except Help for Help and Help Index). You can disable the edit capability by removing the F15 function key definition from the shipped window definition for help panels.

Help Function Keys

Function keys allow users to navigate through help text, control window size and position and, if appropriate, edit help text:

Key	Function
F1	Help for help
F2	Extended
F3	Exit
F7	Backward
F8	Forward
F9	Keys help
F11	Index
F12	Cancel
F14	Print
F15	Edit
F19	Reposition
F20	Enlarge
F24	More keys

Library Lists

This section highlights CA 2E Toolkit library list facilities including naming, creating, changing, deleting, copying, editing, retrieving, displaying, and selecting library lists. The contents of the library lists are outlined and the commands that link the library lists, user profiles, and job descriptions are listed.

Introduction

Library lists provide an easy way of controlling what objects a job can access. The i OS library list is a very simple, yet powerful concept, but it does have some limitations, most notably that a library list is transient. The library list lasts only for the duration of a job or until replaced by another list, and all of the libraries in a list must be specified every time the list is required. For batch jobs, only the complete specification of all the libraries in a list, or the addition or removal of a single library from a list is possible, although the libraries in a list are edited interactively. Toolkit provides the capability to edit library lists, store them away permanently under a given name, and then retrieve and use the lists.

General Considerations for Library Lists

To manipulate Toolkit library lists, see the following Toolkit command diagrams in the *CA 2E Toolkit Reference Guide*. The following diagram shows the commands that links libraries, library lists, user profiles and job descriptions together.

Naming Library Lists

You can refer directly to a stored library list in a given library by a qualified name, as if it were an object. For example, the following command would replace your library list with the contents of library list FRED in QGPL:

```
YCHGLIBL LIBLST(QGPL/FRED)
```

Storing Library Lists

Library lists are stored in a database file YLIBLST in the specified library. The file name is transparent in all commands that use library lists.

A default version of the library list file is included in the shipped version of Toolkit. Additional copies of the library list file can be created using the i OS Create Duplicate Object command (CRTDUPOBJ). For instance:

```
CRTDUPOBJ OBJ(YLIBLSTX) FROMLIB (IBM i Toolkit-Product-library) OBJTYPE(*FILE) TOLIB(FRED)
NEWOBJ(YLIBLST)
```

Creating Library Lists

You can build library lists from either a specified list of libraries, from the current job's library list, or from the initial library list of a specified job description. Create library lists using the command Build Library List (YBLDLIBLST).

For instance, the following command would build a library list called FRED containing five library names: QTEMP, QGPL, Y1SY, QRPG, and current library FREDLIB.

```
YBLDLIBLST LIBLST(FRED) LIBL(QTEMP QGPL Y1SY FREDLIB QRPG) CURLIB(FREDLIB) TEXT('Library list for FRED')
```

You can synchronize maintenance of an initial library list with a library list you are building. For example, using YBLDLIBLST to build a library list FRED in QGPL from the current job's library list and to ensure that the initial library list (INLLIBL) of job description QBATCH in FREDLIB is synchronized, you would use the following command:

```
YBLDLIBLST LIBLST(QGPL/FRED) LIBL(*JOB) LSTTYPE(*INLL)  CURLIB(*JOB)
LSTJOBD(FREDLIB/QBATCH) TEXT('Library list for FRED')
```

Changing Library Lists

You can change library lists using the command Change Library List (YCHGLIBLange Library List (YCHGLIBLST).

```
YCHGLIBLST LIBLST(QGPL/FRED) TEXT('Library list for George')
```

Renaming Library Lists

You can rename library lists using the command Rename Library List (YRNMLIBLST).

```
YRNMLIBLST LIBLST(QGPL/FRED) NEWLST(GEORGE)
```

Deleting Library Lists

You can delete library lists using the command Delete Library List (YDLTLIBLST).

```
YDLTLIBLST LIBLST(QGPL/FRED)
```

Copying Library Lists

You can copy library lists using the command Copy Library List (YCPYLIBLST).

For instance, the following command would copy the contents of a list called FRED to a list called GEORGE:

```
YCPYLIBLST FROMLST(FRED) TOLST(GEORGE)
```

Displaying Library Lists

You can display library lists using the command Display Library List (YDSPLIBLST).

For instance, the following command would display the contents of a list called FRED:

```
YDSPLIBLST LIBLST(FRED)
```

Checking Library Lists

You can determine the existence of a library list using the Check Library List command (YCHKLIBLST).

For instance, the following command would check whether library list FRED in QGPL exists:

```
YCHKLIBLST LIBLST(QGPL/FRED)
```

You can also use this command to retrieve the contents of a library list into a program. Note that the command first verifies your authority to use and maintain the list.

Working With/Editing Library Lists

Library lists can be changed interactively using the Toolkit Work with Library List (YWRKLIBLST) or Edit Library List (YEDTLIBLST) utilities. You can reorder list libraries and add or remove libraries from the list at any position. The Work with Library Lists utility provides full subsetting. You can also use the commands to edit the current library list, that is invoke the job's library list, as well as edit the initial library list of an associated job description. For instance:

```
YWRKLIBLST LIBLST(FR*)LSTTYPE(*MDL) YEDTLIBLST LIBLST(FRED)
```

Retrieving Library Lists

An edited library list can be saved under a given name and retrieved to set the library list of a job. You may want to make use of a stored library list of a job, job description, and user profile.

Changing the Library List of a Job

You can reset the library list for a job by using the command Change Library List (YCHGLIBL). The command replaces the current library list of a job with a stored named library list. For example, if "FRED" is the name of an existing library list containing ten libraries, then the following command would replace the library list of the invoking job to consist of the ten libraries:

```
YCHGLIBL LIBLST(FRED)
```

If a Toolkit library list contains a Current Library entry, then the command Change Library List (YCHGLIBL) will, by default, also change the current library of the invoking job:

```
YCHGLIBL LIBLST(FRED) CURLIB(*LST)
```

The command Change Library List is available in an abbreviated form R, which enables you to refresh or replace your library list very conveniently. For example:

```
R FRED
```

You can use a library list on the command Create Objects (YCRTOBJ). The list is used for the Initial Library List (INLLIBL) and Current Library (CURLIB) parameters of the i OS command Submit Job (SBMJOB) used to submit the creates. For instance:

```
YCRTOBJ OBJLIB(FRED) LIBLST(FRED) MBRLST(TEMPLST)
```

Changing the Library List of a Job Description

You can reset the library list for a job description by using the command Change Job Description Library List (YCHGJOBDLL). The command changes the initial library list (INLLIBL) on a specified job description with the contents of a specified library list. For example:

```
YCHGJOBDLL JOBD(QBATCH) LIBLST(FRED)
```

You can set the Job Description (JOBD) parameter to the special value List Job Description (*LSTJOBD). This enables you to rematch the initial library list (INLLIBL) for the job descriptions referenced within library lists with their corresponding library lists.

```
YCHGJOBDLL JOBD(*LSTJOBD) LJBLST(*ALL)
```

The library list for a job description referenced on a library list is synchronized with the library list whenever the library list is updated using a library list command. You can turn off this action using the Update Job Description (UPDJOBD) parameter. For example:

```
YRMVLLE LIB(QRPG) LIBLST(FRED) UPDJOBD(*NO)
```

Changing the Library List of a User Profile

With the user profile utilities, it is possible to associate a library list with a user profile. When an interactive job is started by a user signing on to that profile, the library list of the job is set to the contents of the list.

The initial library list for a user profile can be specified with the CA 2E Toolkit commands Create User Profile (YCRTUSRPRF) or Change User Profile (YCHGUSRPRF). The CA 2E Toolkit initial program, YINLPGM, then uses the specified stored library list to set the initial library list for a user at sign-on. For example:

```
YCHGUSRPRF USRPRF(FRED) LIBLST(FRED)
```

Changing Individual Library List Entries

Library List Entry Commands

Individual list entries can be added, removed or renamed in one or more library lists in a given library. The library lists can be named generically.

The command Add Library List Entry (YADDLLE) adds a named library to a list or lists at a given position or relative to another named library:

```
YADDLLE LIB(QPLI) POSITION(*LAST) LIBLST(FRED*)
YADDLLE LIB(Y1SYVENG) POSITION(*BEFORE Y1SY) LIBLST(*ALL)
```

The command Remove Library List Entry (YRMVLLE) removes a named library from a list or lists:

```
YRMVLLE LIB(QPLI) LIBLST(FRED*)
```

The Toolkit command Rename Library List Entry (YRNMLLE) renames a named library list entry in a library list or lists:

```
YRNMLLE FROMLIB(QPLI) TOLIB(YPLI) LIBLST(FRED*)
```

Renaming Library List Entries

When renaming a library you should update all references to that library in all library lists. This can be done using the Library command (YRNMLIB). For instance, the following would rename library BEAM to MOTE, and update all library lists which reference library BEAM. A message would be sent to the job log to indicate each library list updated.

```
YRNMLIB FROMLIB(BEAM) TOLIB(MOTE) LIBLST(*ALL)
```

In addition, you can specify that all references to the library by user profiles are also updated. For example:

```
YRNMLIB FROMLIB(BEAM) TOLIB(MOTE) LIBLST(*ALL)  UPDUSRPRF(*YES)
```

Selecting a Library List

A selection function is available to facilitate the use of library lists. This enables you to select/add to a list of existing library lists, or to perform some of the library list manipulation functions upon a selected library list. The select function is invoked by specifying a value of LIBLST(*SELECT), when using the various library list commands, or LIBLST(*ALL) when using the command Work with Library List (YWRKLIBLST). For example:

```
YWRKLIBLST LIBLST(*ALL)YCHGLIBL LIBLST(*SELECT)
```

A value of *SELECT allows you to choose one library list, while a value of *ALL allows you to choose many items.

Menus

This section highlights menus, indicating how the menu utilities can be used as both design and development tools. An illustration of a CA 2E Toolkit menu is provided and menu features are described. Other topics covered include:

- Editing menus
- Documenting menus
- Manipulating menus
- Displaying menus

Introduction

Dynamic menus system removes the need to have an individual i OS display file (and accompanying HLL program), for each application menu. All menus are instead stored in a special form with the help of a sophisticated interactive utility, the CA 2E Toolkit Work with Menus (YWRKMNU) program. A single menu display utility Go to menu (YGO), is used to present any menu to the user. This dynamic menu approach has many advantages:

- Menus can easily be changed; in fact the process of creating or altering menus becomes so easy, that menus can be used for many other purposes.
- Menu changes are effective immediately; no compilation is involved.
- The menu display program conforms to SAA common user interface standards. Menus created with the System/38 version of Toolkit can automatically be displayed in Toolkit format.
- Changing or adding a menu does not disrupt live systems.
- Different versions of menus can easily be kept.
- The same menus can appear in different environments with different command keys.
- Advanced functions can be provided in all menus, including the ability to jump to any other menu, and to provide help text.
- Toolkit menus allow both a "fast path" and a "slow path", so menus are more efficient to use: your user’s time is saved, especially when response times are slow.
- Less CL programming is required, as the menu display program can provide commonly required functions, such as obtaining confirmation, and job submission.
- Greater standardization is achieved.
- Less storage is required.

General Considerations for Menus

To manipulate Toolkit help text, see the following Toolkit command diagrams in the CA 2E Toolkit Reference Guide:

Toolkit Menu Description

A Toolkit menu consists of title information and a list of options. Each option calls an executable function. Only one menu option is specified per menu line. Menus have a maximum size of 100 lines (about six pages) including the title line. Menus larger than a page are shown as a subfile, which can be scrolled through. Each menu page may contain up to 14 menu options.

The Toolkit menu display program is designed to be consistent with IBM's SAA standards, and in particular with IBM's menu display program (see the i OS command Go to Menu (GO)). Particular features include:

- Request line entry: i OS CL commands may be entered on the bottom lines of the display.
- F09 - Duplicate previous request
- F10 - Invoke i OS QCMD program (optional feature)
- F14 - Display submitted jobs
- F16 - Return to major menu
- F23 - Reset current menu to be major menu

The menus you create with Toolkit are efficient to use, whatever the level of the user's experience: beginners can take a step by step slow path, supported by explanatory text, while experts can take a fast path to specify their requirements quickly and concisely.

Naming Menus

Menus are referred to by a menu name. Menu names must be valid system names, that is, up to ten characters long and begin with a letter. For instance, to display a menu called FRED:

```
YGO MENU(FRED)
```

Storing Menus

Toolkit menus are stored in a data base file member. Many different menus can be stored in a single member, and a menu file can contain many members. When you refer to a menu, you may optionally also specify a name of the file and member containing the menu. For example:

```
YGO MENU(FRED) FILE(FREDSMENU) MBR(FREDERICK)
```

You can keep different sets of menus for different users in different files, and control access to the menus by granting or revoking authorization rights to the files.

A default menu file is supplied with the Toolkit shipped system; it is named YDSNMNU. You can create your own copies of the menu file and give them any name you like, but each copy must have the same format as the shipped YDSNMNU file. The Toolkit command Create Design File (YCRTDSNF) is provided to give you an easy method of creating new menu files. For example, to create a menu file in library QGPL:

```
YCRTDSNF TYPE(*MNU) FILE(QGPL/YDSNMNU)
TEXT('Menus for my system')
```

The recommended name for Toolkit menu files is YDSNMNU because it is the default name for the menu file on all the Toolkit commands that use menus. It is suggested that you call all your menu files YDSNMNU, and control which version you use using the library list. Menus within a menu file member can be regarded as analogous to source members within an i OS source file, such as QRPGSRC.

You may have different sets of menus in different members in the design file. The Toolkit command Add Design File Member (YADDDSNFM) is provided to give you an easy method of adding additional members to design files. For example, to add a new member to menu file in library QGPL:

```
YADDDSNFM TYPE(*MNU) FILE(QGPL/YDSNMNU) MBR(EXTRA) TEXT
```

Menu Features

Two types of Toolkit menus are allowed:

- Single option menus allow selection of only one option at a time: options are requested by specifying the option number.
- Multiple option menus allow the selection of several options at a time, by typing a **1** against each option that is to be executed. When you select more than one option, the selected options are executed in the order in which they appear on the menu. If an error occurs among any of the selected options, the selections after the erroneous option will not be executed: un-executed options will be highlighted. Multiple selection menus are useful for occasions when you are likely to require a variable selection of several items, such as selecting reports to print.

The menu type is specified on the detailed edit display for the menu title line: see the section, Editing Menus.

Menu Initial Programs

You can specify an initial program for each menu. The initial program will be called whenever the menu is displayed; the initial program can be used to carry out initialization common to all the options on the menu, for instance, opening shared files, or beginning commitment control.

Menu Option Types

Three different types of menu option are allowed, or one of three actions can be specified to happen when you select a menu option. These are as follows:

- Execute a given command, with or without parameters. The command request can either be in i OS syntax or in S/38E syntax.
- Execute a given program, with or without parameters (PGM).
- Call another menu (MNU).

Menu Option Functions

To help reduce the amount of CL programming required to set up an application system, the menu display program has three standard functions available, which can be specified for any option:

- Prompt the option: when the option is taken, the command specified as the option request will be prompted for some or all parameter values. This facility is mainly for use with commands. Note that selective prompting can also be used to protect particular parameter values. See the *i OS Programmers Guide* for details on the use of selective prompting.
- Submit the option: when the option is taken, the request associated with the option is not to be executed immediately, but rather is to be submitted to a batch queue. This feature can assist in development, performance tuning, and problem finding, since the function that determines whether a program runs in batch or interactively can easily be changed at any time.

  The job description used by the menu program to submit jobs can be specified as a parameter (JOBD) on the Toolkit command Go to Menu (YGO).

  You may specify for the option either:

  — Y: The values on the specified job description are used to route the job.

  — J: The job description values are overridden with values from the current interactive job. This can be used, for instance, to ensure that the submitted job had the same library list as the submitting job. The following values are overridden: INLLIBL, SWS, OUTQ, OUTQLIB, DATE, INQMSGRPY, LOG, Color
- Confirm the option: when the option is taken, an answer to an additional confirmation prompt will be requested before the option request is executed. The confirmation prompt can require positive action either to confirm or to cancel execution. Thus:

  — **CONFIRM Y**: Positive action to cancel.

  Pressing ENTER will confirm execution.

  Typing **N** will cancel.

  — **CONFIRM N**: Positive action to confirm.

  Pressing ENTER will cancel execution.

  Typing **Y** will confirm.

Automatic Numbering of Menu Options

The numbering of options can either be absolute, for example, **5,** or relative, **.N**; the latter will cause the display program to generate an option number at display time. Using relative numbering, you can insert options on a menu at any time, and have the menu option numbering automatically re-sequenced.

Display Attributes for Menu Option Text

Several features are available that make menus clearer to the user: text lines can be highlighted or underlined, or displayed in a different color, marginal headings can be specified on the left, and blank lines can be included to space out options. The color options will only be affected at terminals that support color attributes.

Menus can continue onto several pages.

Specifying Help Text for Menu Options

You can specify help text for each menu option. This will be displayed if, when displaying the menu with the Toolkit utility Go to Menu (YGO), you move the cursor to the line showing the option you want to read the help text, and press the HELP key.

If no help text document name is specified for an option, the menu display program assumes a default document name - which will be the same as the name of the program, command or menu invoked by the option. The name of the file in which it will look for the document name defaults to QTXTSRC: you may override this value yourself. Specification of a default help file name is done using two data areas:

- YMHPFLA, containing the name of the default help file
- YMHPLBA, containing the name of the default help library

You can also keep different copies of the data areas, containing different default values, and control which is used using the library list. See the section in this manual on Help Text for further details.

Editing Menus

Menus are created or changed using the Toolkit utility Work with Menus that is invoked by the Toolkit command Work with Menus (YWRKMNU).

The menu to be added/edited can either be named directly, or else chosen from a selection list:

- **YWRKMNU MENU(FRED)** /* Work with Menus FRED */
- **YWRKMNU MENU(*SELECT)** /* display existing menus for seln*/

Menu editing can be done at two levels:

- An outline edit allows rapid full-panel design of menus, including text and numbering. The outline edit display allows you to enter online edit commands similar to those of the i OS SEU source editor Start SEU (STRSEU).
- Detailed editing allows you to specify what each option does. The detailed menu edit display has i OS syntax checking and prompting support available.

The menu outline edit display looks like this:

Menu Line Edit Commands

The following menu line edit commands are available:

Each option on the menu can be edited in detail using the detail display, which is obtained by placing a **Z** against an individual menu line on the outline edit display. The detail display appears as follows:

Syntax Checking on Menu Option Details

For each option, you can specify the name of the menu, command, or program to be executed when the option is taken. Execution parameters may be specified for the command or program if appropriate.

The command prompter Command prompter can be invoked (by pressing F04) to prompt the parameters for an option, and to check that a valid request has been coded. The syntax checker will parse the specified request string and diagnose syntax errors, without actually executing the option. If the option is of type CMD the request string will be checked with the i OS QCMDCHK program. If the option is of type C38 the request string will be checked with the S/38E program QCACHECK.

You can display or edit which user profiles are authorized to use the program or command specified for the menu option by pressing F05. The Display/Edit object authority (DSPOBJAUT) display will be shown.

You can select the menu title line for detailed editing by placing a **Z** in the selection column on the title line; the menu title display allows you to specify menu level details, such as the menu type, and the initial program for the menu (if any).

The display attributes or colors specified for a menu title control the display attributes or colors used for the title line and the command line of Toolkit menus. You may use this facility to override the shipped defaults (green, reverse image).

Exiting the Work with Menus Program

The menu that you are creating or updating is not saved until you leave the Work with Menus program; at exit you will be prompted with an exit display that allows you to confirm or cancel the changes that you have just made.

The following diagram shows the main interconnections between the displays of the Toolkit program Work with Menu.

Documenting Menus

You can print menus using the Toolkit command Document Menu (YDOCMNU). This command prints a picture of each menu as it appears when used. The action associated with each option on the menu can optionally be listed below the picture of the menu.

An additional Toolkit command Document Menu References (YDOCMNUREF) enables you to list where all commands, menus or programs are referenced by a menu or menus. The Toolkit command Document Execution References (YDOCEXCREF) also shows where Toolkit menus are used and which options are called from menus. See the section on documentation later in this manual.

Menu Manipulation Commands

Service functions are available to do the following:

- YCRTDSNF: Create a new menu file.
- YADDDSNFM: Add a new member to a menu file.
- YCPYMNU: Copy a menu.
- YDLTMNU: Delete a menu.
- YRNMMNU: Rename a menu.

The Toolkit command Rename Menu (YRNMMNU) allows you not only to rename a Toolkit menu, but also to update all references to the menu on other menus within the same menu file so that they reflect the change.

Selecting a menu

Using a select function facilitates the use of menus. You can select or add to a list of existing menus, or perform any of the menu manipulation functions for a selected menu. The select function is invoked by specifying a value of MENU(*SELECT) when using the command Work with Menu (YWRKMNU). For example:

```
YWRKMNU MENU(*SELECT) FILE(YDSNMNU)
```

Displaying Menus

To display a menu use the Toolkit menu display utility which is invoked using the Toolkit command Go to Menu (YGO). The command requires you to specify a menu name: the named menu will appear as an initial menu. The YGO command can be entered either from the i OS command entry display (QCMD), or from any menu with a command request line, such as the i OS programmer's menu (QPGMMENU), or be called from your own HLL programs. The command is also called by the Toolkit initial program (YINLPGM).

Users may branch to other menus from the initial menu. The branching may either be restricted to a hierarchical tree, or may be unrestricted, that is, branching is allowed to any existing named menu. When using the menu program they may at any point return to the previous menu (F12), or the initial menu (F03), or sign off (SO) altogether.

As well as allowing users to branch to named menus, you may allow them to enter commands directly from the menu display.

Direct Menu Calling

A parameter Menu entry allowed (MNUENT) on the Toolkit command Go to Menu (YGO), lets you specify whether direct menu calling is to be allowed during the menu display session: direct menu calling is the ability to call any menu in the current menu file member by name using the *GO facility, rather than by following a strict menu tree.

A user who is not allowed to call menus directly may only follow a hierarchical tree, that is, call menus that are explicitly set up as menu options on the menus. With hierarchical calling of menus, in the following diagram menus D and E can only be reached and left using menu B, just as menus F and G can only be reached using menu C. Menu H cannot be reached at all.

A user who is permitted direct menu calling may either follow a hierarchical tree, or at any time transfer to another named menu:

Only menus within the same menu file member as that of the menu currently being displayed may be called directly. To call a menu directly, you type in *GO, followed by the name of the menu which you want to display.

You will probably want to set up a separate menu file member for each user profile, user profile group, or both. This would permit users or user groups to move around their individuals freely, but only allow access to other menus belonging to other users on a strictly controlled basis.

You may combine restricted and unrestricted calls in the same menu tree.

Grouping Menus By User

The menu display program can switch between menus from different files. To change the menu file being used by the menu program, specify the Toolkit command Go to Menu (YGO) as the menu option action, with the name of the desired file in the FILE parameter.

The menu display program will display menus from different files transparently: that is, although a menu may call another menu from a totally different menu file, when displaying the menu you will not be aware that you have changed menu files.

Command Entry from Menus

A parameter Allow command entry' (CMDENT), on the Toolkit command Go to Menu (YGO), lets you specify whether the entry of commands is to be allowed during the menu display session.

Commands are entered on the request line at the bottom of the display. Parameters may be specified for entered commands. The command prompter may be invoked to help with command entry by pressing F04.

Commands can also be submitted: to submit a command, *SBM (or *S for short) must be entered immediately before the request. The job will be submitted using the job description specified on the JOBD parameter of the YGO command. Alternatively you may enter a value of *JOBSBM (or *J for short) which will submit the request, overriding the job description with the current job's attributes, for example, library list, and output queue. To examine submitted jobs use F14 to invoke the i OS command Work with submitted jobs (WRKSBMJOB).

If command entry is allowed, F10 will call the i OS Command entry program (QCMD).

Menu Debug Functions

Four interactive functions are provided to help you to resolve problems arising from the incorrect specification of menu options. The functions can be invoked when using the Toolkit program Go to menu by entering special values into the request line at the bottom of the display:

If *DISPLAY (or *D for short) is specified on the request line prior to an option number, for example, *D 3, the request string specified for any menu option taken will be displayed as an information message on the job's external message queue. The request string is built from the name of the program or command specified as the option action, plus any parameters that may also have been specified for the option. This enables you to examine the menu option request without executing it.

If *CHECK (or *C for short) is specified prior to the option number (for example, *CHECK 3), the request string for any menu option taken will be syntax checked by the i OS command checkers (QCMDCHK or QCACHECK according to the option type), but not actually executed.

If *MNUWRK (or *M for short) is specified on the request line, the Toolkit command Work with Menus (YWRKMNU) will be invoked for the current menu. On returning from the edit program, the menu display will be refreshed with the newly edited menu. This gives a convenient way of correcting menus in error.

If *AUT (or *A for short) is specified prior to the option number, the command Display Object Authority (DSPOBJAUT) will be invoked for the program or command specified as the option action.

Note that to be allowed to use the debug aids while displaying menus, you must specify CMDENT(*YES) on the command Go to Menu (YGO) when you enter the menu program.

Menu Request Functions

The following table summarizes the menu request functions that are available.

Function	Keyword	Value in request line	MNUENT rqd	CMDENT rqd
Submit a request - JOBD	*SBM	Command request	-	Yes
Submit a request - *JOB	*JOBSBM	Command request	-	Yes
Go to a menu	*GO	Menu name	Yes	-
Work with current menu	*MNUWRK	-	-	Yes

Function	Keyword	Value in request line	MNUENT rqd	CMDENT rqd
Display option request	*DISPLAY	Option number	-	Yes
Check option syntax	*CHECK	Option number	-	Yes
Check option authority	*AUT	Option number		Yes

Obtaining Help Text for a Menu Option

You can display help text by pressing Help. The text displayed depends upon the location of the cursor when Help is pressed:

- When the cursor is on one of the menu option lines, the Help text for that option is displayed.
- When the cursor is on the menu title line, the Help text for the menu is displayed.
- When the cursor is anywhere else, the Help text for the menu display program is displayed.

For information on other types of help available, see the section on Help Text later in this chapter.

Messages and the Menu Program

i OS has sophisticated message handling facilities that enable you both to provide a continuous dialogue between user and machine, and to obtain an audit log. The Toolkit display menu program helps you to make full use of i OS messages, and includes the following features:

- The menu program has a program message queue subfile: all messages appear on line 24 as a subfile. You can examine the messages by moving the cursor to line 24 of the menu display, and scrolling up or down and/or pressing Help.
- If your programs crash, the escape messages will automatically be trapped and displayed on the message line.
- Completion messages and diagnostic messages will automatically be displayed on the message line, where they can be scrolled.
- Your own messages can be made to appear on the message line of the menu display, complete with second level Help text and substitution variables. To make a message appear on the display you may ei Simply send the message to the previous program message queue in the invocation stack.

  For example, the following impromptu message will cause HELLO SAILOR to appear on line 24 of the menu display:

  SNDPGMMSG MSG('HELLO SAILOR')

  While this predefined message:

  ADDMSGD MSGID(USR0001) MSGF(QUSRMSG) MSG('&1 Pints today &2') SECLVL('Must be gold top.') FMT((*DEC 3) (*CHAR 10))

  will cause the message "002 Pints today please" to appear on line 24 if it is sent to the menu program by your own CL program:

  SNDPGMMSG MSGID(USR0001) MSGF(QUSRMSG) MSGSend the message directly to a named program message queue. If you have nested invocations of application programs - program A calling program B calling program C - you may wish to send messages directly to the Toolkit menu program by name. Such messages should be sent to the Toolkit menu message handling program YDMNGOC. For example:

  SNDPGMMSG MSGID(USR0001) MSGF(QUSRMSG) MSGDTA('002please') TOPGMQ(*SAME YDMNGOC)

  The menu program provides a completion message for each option executed. The message (YMN0027 in the Toolkit message file, YYYYMSG) has the form:

  Last option was "N." "option text"

  Use of this feature can reduce the amount of CL programming that you have to do.
- The execution messages of programs called by menu options will be logged according to the job's logging level: refer to chapter 13 in the *i OS Programmers Handbook* for details on message logging levels.

- Request messages are also logged: it is possible to use the display low level messages facility (F10) of the command entry display to examine request messages that were initially sent as the result of taking menu options on a Toolkit menu.

  The message logging level of your jobs, which can be changed by, for instance, the i OS command Change Job (CHGJOB) - affects which messages are displayed in your log.

Security Considerations

To be able to execute a menu option, users must be authorized to the program or command that the option calls. The appropriate i OS error message will be displayed if the user is not authorized, namely:

CPD0032 : Not authorized to command CMDNAME in LIB

CPF9802 : Not authorized to object OBJNAME in LIB type PGM.

Authorization to menus - the right to display a menu or menus (though not necessarily to invoke the menu options) - can be implemented by dividing menus into separate files, and granting or revoking users' authorizations to those files. For example:

```
GRTOBJAUT OBJ(YDSNMNU) OBJTYPE(*FILE) USER(FRED)       AUT(*READ)
 RVKOBJAUT OBJ(YDSNMNU) OBJTYPE(*FILE) USER(FRED)     AUT(*READ)
```

Use of an Exception Message Queue

The EXCPMSGQ parameter on the command Go to Menu (YGO) allows you to specify an "exception message queue". If an exception message queue is specified, copies of any unmonitored exception messages detected by the menu display program will be resent to the nominated message queue. A copy of the preceding diagnostic messages (if any) will also be sent; each set of messages will be preceded by a Toolkit message (YMN0044).

The EXCPMSGQ facility enables you to alert your system support staff when a problem occurs; for instance when an application program has crashed.

You may want to do the following:

```
CRTMSGQ MSGQ(QGPL/EXCPMSG) TEXT('Exception message queue')
 YGO MENU(MASTER) EXCPMSGQ(QGPL/EXCPMSG)
```

Concurrency Considerations

A user is given the version of the menu that is current when he first displays the menu, that is, a menu is not reloaded between the selection of options on the same menu. Therefore, if a menu that is being displayed at one workstation is changed at another workstation, the change does not alter the displayed version until the displayed version is refreshed. Refreshing will occur if a second menu is called, and then the original menu called again.

Recursion Considerations

The Toolkit program Display Menu can appear more than once in the job invocation stack. A new invocation of the program will be made every time the command YGO is invoked, either as a menu option or through the request line. Typically this will be done to change menu file, or menu file member, or when the menu display program is called recursively using another program.

New Invocation: If recursive calling of the menu display program using another program:

New Invocation: If recursive calling of the menu display program by means of YGO, so as to change file or member:

Same Invocation: If new menu is called by a menu name (for example, '*GO Fred') or a menu type option:

Signing off from Toolkit menus

Users do not necessarily have to return up the menu invocation stack to sign off from a job that uses Toolkit menus; a value other than *NONE for the SIGNOFF parameter on the Toolkit command Go to Menu (YGO) will cause the i OS command Sign Off (SIGNOFF) to be executed directly when the user specifies SO as a menu option. For example, for the SO option to sign the user off with LOG(*LIST):

```
YGO MENU(FRED) SIGNOFF(*LIST)
```

You may specify a value for the Go to Menu SIGNOFF option as part of the Toolkit user profile extension attributes:

```
YCHGUSRPRF USRPRF(FRED) LOGOFF(*LIST)
```

Other Uses for Menus

Here are some other ways menus can be used:

- As a means of providing selection options from your own programs.
- As a framework to hold documents. Say that you have a large document such as a system specification, that requires repeated editing. You could set up a menu with a separate option to edit each section. A final option could print the complete document.
- As a training tool. Users may go to Menus and examine help text without actually executing options. Additional explanatory text can be added to the menu displays. Problem determination 'trees' can be created to guide users through areas requiring operator intervention.
- As a framework for CL commands, that is, to present commonly required CL commands in a specified order, with or without override parameter values. This can be especially useful for functions where a lot of operator intervention may be required, or where frequent procedural changes are required. For instance, you could set up a menu for your backup.

```
1. DSPDKT ??DEV(DKT01)??DATA(*SAVRST) See what's on diskette
2. ?DSPLIB See how big library is
10. SAVLIB FRED DEV(DKT01) CLEAR(*ALL)/* Mondays */
11. SAVLIB ALFONSO DEV(DKT01) CLEAR(*ALL)/*Fridays/
12. SAVLIB GREGORY DEV(DKT01) CLEAR(*ALL)/*  Both days */
13. SAVOBJ CYRIL/*ALL DEV(DKT01) OBJTYPE(*JRNRCV) CLEAR(*ALL)
```

- As a presentation tool. If you need to present information using the computer, you can use Toolkit menus to provide a framework. The headings and text can be placed on menus, interspersed with calls to the appropriate programs or functions that illustrate your argument or explanation. Course handouts can be prepared directly from the printed menu images.

Help Text

This section tells you how to write, index, display, and document help text. Contextual and conditioned help text is described, and commands to manipulate help text are listed. Help text directives are outlined indicating how to control the display of help text.

Introduction

The Toolkit help text facilities enable you to provide interactive help text for all of your menus and application programs.

- You can specify an index for the help text.
- You can specify which part of the text is to be displayed for a given cursor location (contextual or cursor sensitive help text).
- You can condition the relevant part of a help topic to be displayed, based on the circumstances.

Using Toolkit you can create online help text in a fast, flexible, way using the command Start SEU (STRSEU) of the i OS Source Entry Utility or S/38 Edit Text command (QSYS38/EDTTXT) of IBM's Text Management/38. Both utilities have full screen editors that allow you to lay out text as it is to appear ("What-you-see-is-what-you-get"), and to specify an index for the help text.

Help text for a program is stored as a text document. When the user presses Help, either while displaying a menu with the Toolkit Go to Menu program (YGO), or while using an interactive application program, the program is temporarily suspended, and a specified document is displayed as help text. True context- sensitive help text can easily be provided in your own application, such as the right words in the right place.

You can write help text as it is to appear. Several special facilities provide useful 'short cuts', such as numbering sections automatically, and imbedding standard paragraphs. Most of the print control facilities of 'Text Management/38', such as underlining, are implemented, so you can format readable help text.

Writing Help Text

You can use SEU/EDTTXT to edit help text. Help text is entered either using the S/38 Text Management/38 or the i OS Source Entry Utility facility; refer to the appropriate manuals for further details.

Text Management/38 Control Characters

Text Management/38 control characters that have a special meaning when included in the text source; they direct how the text will be printed; for instance ‘. pa’ causes a new page to be thrown. All Text Management/38 control characters begin with a ‘.‘.

The Toolkit help text program interprets most Text Management/38 control characters in the same way as the Text Management/38 print function: for instance, ‘.date’ causes the current date to be displayed at the indicated point. Certain control characters, however, are ignored because they are inappropriate for an interactive display, for instance ‘. sk’ (skip).

Text Management/38 text control characters can be used to format your help text documents, even if you do not have the Text Management/38 utility. For the benefit of those who do not have Text Management/38, the control characters are summarized in the following table.

EDTTXT FACILITY	Character	Notes
Automatic numbering	.N m	Ascending numbers (N is reset to m if m is specified)
Section numbering	.H0 to .H6	Various effects, for example, underline, capitals
Embedding of sub-documents	.IM (doc)	Document reference has syntax: (Document File Library)
Current date	.date	
Document’s identity	.docid	Document/File/Library will appear where specified
Comment line	.*	Line will not be displayed

Text Directives

Toolkit also introduces a number of extra control characters (directives) to provide additional functions. All Toolkit control characters begin with '.*': this means they do not appear when help text is printed.

The following table includes Toolkit text control characters executed by YDSPHLP

Toolkit FACILITY	Character	Notes
Title line	. *T: 'text'	Text appears as title line on Help text display
Display comment	. *Y*: 'text'	Text appears on display, but not in printed document.
Vector table entry	. *YV: data LBL	Not displayed, but used to control which part of the text is displayed. LBL points to a 'Label entry' line: it can be up to 10 characters long.
Index entry	*YI: LBL 'text'	Text appears as entry in Help display index. LBL points to a 'Label entry' line: it can be up to 10 characters long.
Label entry	. *YH: LBL	LBL corresponds to an 'Index entry' line, or to a 'Vector table entry' line.

In each case the directive must begin in column one with a '.' (full stop) followed by the directive name ***T:** in columns 2-4 or ***Y*:, *YV:, *YI:** or ***YH:** in columns 2-5 . The directive name, labels, and so on, may be entered in upper or lower case ('abc' and 'ABC' are equivalent).

CUA/Text Specific Directives

The Toolkit help text display utility processes the same help text source in both a window and an extended help full panel. You can design help text documents for a full help panel and place appropriate window format directives before text to be word-wrapped in the window. You can use window directives alone or with Text Management/38 paragraph format definitions. Text Management/38 paragraph format definitions are sufficient to ensure word-wrap in help windows.

Toolkit FACILITY	Character	Notes
Size	. *YW: width depth vary-depth	Dimensions of window to display help text. If .*YW: not present, default is from data area YMHPYWA.

Toolkit FACILITY	Character	Notes
Format definition	. *YD: start size blank before	Tab and margin formats of help text. (Not required if TM/38 paragraph definitions are used.)
Text entry	. *YF: ID	Identifier of format definition (. *YD:) for text entry
Keys help label	. *YK: LBL	Label corresponds to help text about function keys. If . *YK not present default is taken from data area YMHPYKA.

Purpose of Toolkit Directives

For details on the directives, see Specifying Help Text Directives later in this chapter. The directives allow you:

- To specify points in the document to which the help display program may jump:
  - At entry (vectored entry).
  - Under user control once within the document (index choices).
- To specify an index to be displayed on entry or at the request of the user.
- To specify a title to appear on the top of all help displays.
- To specify help window attributes.

Example

The following diagram illustrates how a help text document might be structured to provide both an index and to support vectored entry, that is, display of the help text starting at a specified point.

There are three ways of displaying the document:

- Vectored entry by cursor location

YDSPHLP MBR(FRED) LABEL(*CSRLOC) CSRLOC(rrrccc)

1. A cursor coordinate is passed into the help display program.
2. The help display program uses the vector table at the beginning of the text to translate this into a label.
3. The help display program displays the help text indicated by the specified label.

- Vectored entry by label

YDSPHLP MBR(FRED) LABEL(label-name)

1. A label name is passed into the help display program.
2. The help display program displays the help text indicated by the specified label.

- Entry by indexDisplay Help Text command (YDSPHLP)

YDSPHLP MBR(FRED)

1. The help display program displays extended help.
2. The user specifies an option from the help index.
3. The help display program translates this into a label.
4. The help display program displays the help text indicated by the specified label.

Help Text Attributes: Underline/High Intensity

To make your help text clearer to read, you may use formatting attributes such as underline and high intensity. The formatting attributes are only conveniently available through the Edit Text program (QSYS38/EDTTXT). Refer to the on-line help text for Text Management/38 to see how they are specified, or to the Text Management/38 User's Guide.

Invoking EDTTXT

While displaying a help text document, you can invoke the S/38 Text Management/38 Edit Text command (QSYS38/EDTTXT) to edit the same document, simply by pressing F15= Edit. If desired, this facility can be used to allow users to write or amend their own help text. You may not update a help text member unless you are authorized to update the file containing the member.

Displaying Help Text

Typically you will want to arrange matters so that whenever users get stuck, they can do one of the following:

- Press Help to get the help text appropriate to the program being used.
- Select a choice from the help action bar pull-down.

The following section describes how to specify which help text document is displayed, and when.

Indexing Help Text

The Toolkit help text display has an index facility that can automatically provide an index to a help text document. An index will be provided if 'index entry' lines, that is, lines containing the Toolkit index control characters ('. *YI:') are present at the beginning of a help text document. Each Toolkit '. *YI:' line must contain a label; the same label must also be present on a Toolkit label entry line ('. *YH:') later in the document.

The following figure illustrates the use of indexing. It shows the actual text source lines necessary to produce an index.

Contextual (Cursor-Sensitive) Help Text

The Toolkit help text display has a facility which provides help text relevant to the position of the cursor on the panel when the user presses Help. The cursor position is translated into the label, which is used to position the text with a 'vector table' at the start of the help text document. The vector table can be added manually, or can be generated from the panel DDS using the Toolkit Add Help Vector Table (YADDHLPTBL) command. See the YADDHLPTBL command diagram for details and an example.

Each entry in the vector table consists of label groups (see the following section on Conditioned Help Text), row and column coordinates defining the extent of the field, and the label which identifies the corresponding section of the text which is to be displayed. For details of specifying Toolkit text directives, see Specifying Help Text Directives, later in this chapter.

Specification Order of Vector Table Entries

The help display program selects the first vector table entry which satisfies its search criteria; that is, for which the vector table label groups match those on the LBLGRP parameter (see section on Conditioned Help Text) and for which the cursor coordinates as specified by the CSRLOC parameter lie within the specified area.

If you have overlapping text areas you should specify the most specific areas first.

For example, in the following display there are three areas of text. The smallest areas (customer code, command key explanations) are specified before the larger area (panel details), so that if the CSRLOC passed to the help display program lies within one of the smaller areas, the more specific text will be used in preference. If no specific label is found, the default action displays the help text from the beginning.

Conditioned Help Text

The Toolkit help text display has a facility to provide help text dependent on circumstances. Examples of where you might use this facility are:

- Where a panel can operate in more than one mode (for example, adding or changing records) and different help text is required for each mode.
- Where there is a choice of fields to display in one place on a panel (for example, fields conditioned by an indicator).
- Where you wish to provide more than one level of help text (for example, novices or experts).

The Toolkit help display program makes use of label groups to determine which section of help text will be displayed. Label groups are specified as part of the 'vector table' entries described previously.

Up to three label groups may be specified for each vector table entry, each preceded by a space or an 'N'. The 'N' indicates 'not', that is, not belonging to that label group. If more than one label group is specified on a vector table entry, the conditions are ANDed together.

The YDSPHLP program compares these label groups with those supplied at run time by the LBLGRP parameter to decide whether a particular vector table entry is eligible for use or is to be ignored. For a vector table entry to be eligible, all the label group conditions specified on the entry must be satisfied by the list of label groups supplied by the LBLGRP parameter. Thus, both the following conditions must be satisfied for an entry to be eligible:

- If an entry has a label group specified for it (not preceded by an 'N'), that label group must be in the list supplied by the LBLGRP parameter.
- If an entry has a label group preceded by 'N' specified for it, it must definitely not be in the list supplied by the LBLGRP parameter.

If LBLGRP(*NOCHK) is specified, then no label group checking will take place.

If LBLGRP(*NONE) is specified, then only those entries with no label group at all, or with all label groups preceded by an 'N', will be eligible.

For example:

LBLGRP	Label Group	Result
A		Eligible
A	A	Eligible
A	NA	Not eligible
A	B	Not eligible
A	A B	Not eligible
A	ANB	Eligible

A B	A	Eligible
A B	A B	Eligible
A B	NA B	Not eligible
A B	ANB	Not eligible
*NONE		Eligible
*NONE	A	Not eligible
*NONE	NA	Eligible
*NONE	B	Not eligible
*NOCHK	A	Eligible
*NOCHK	NA	Eligible
*NOCHK	B	Eligible
*NOCHK		Eligible

If LBLGRP is specified and no vector table entry is found:

- Index is displayed
- First page of help text will be displayed if there are no index entries

Example of the Use of Label Groups

This example illustrates how you could produce different help text for the same area of a panel under different circumstances. For instance, you might have a customer code which, depending on what the user has specified on preceding displays, could either be the code of a customer to whom delivery is to be made, or the code of the customer to invoice. The help text could have two entries in the vector table, each with a different label group, for example, 'A' or 'B', and a different label, for example, CUS_INV, or CUS_DELIV.

```
        Label  LHS      RHS
       group rowcol  rowcol  Label
YV:     A      004003 006037  CUS_INV
YV:     B      004003 006037  CUS_DELIV
 ....
YH: CUS_INV Customer code to invoice
    This is the code of the customer to whom the invoice
     will be sent.
YH:  CUS_DELIV Customer code for delivery
     This is the code of the customer to whom delivery will be
     made not less than two weeks after purchase.
```

The actual text displayed for a cursor location within the specified area would then depend on the label group specified:

- If LBLGRP(A) is specified, the display of help text starts at label CUS_INV.
- If LBLGRP(B) is specified, the display of help text starts at label CUS_DELIV.

YDSPHLP Program Display

The initial display shown by the help text display program depends on both the contents of the document and the run time values passed to the help text display program.

The following diagram shows the possible variations:

Commands To Manipulate Help Text

Help text members can be manipulated by using the normal i OS commands:

- Create Source Physical File CRTSRCPF: Create a file for Help documents.
- Copy Source File CPYSRCF: Copy a Help document or documents.
- Remove Member RMVM: Delete a Help document.
- Rename Member RNMM: Rename a Help document.

Documenting Help Text

You can print help text with the Toolkit command Document Source Members (YDOCSRC), with SRCPRTOPT(*TXT). For use with Text Management/38, you can print help text using the command Print Document (QSYS38/PRTDOC) of Text Management/38. See the *Text Management/38* guide for details.

When using the Toolkit program Display Help (YDSPHLP), you can obtain a print of the document currently being displayed by pressing F14.

Converting a Member List to a Master Document

If you wish to print all of your help text in one print run, then the Toolkit command Build Document (YBLDDOC) can be very useful. This is a special Toolkit utility that converts a list of source file members into a master document. The master document created will contain Text Management/38 ‘imbed’ control commands for all the documents in the member list, interspersed by ‘new page’ control characters. For further details, see the section on List Processing in the Programmer Aids chapter of this manual.

Specifying Help Text Directives

The Toolkit help text display program (YDSPHLP) uses text directives within the help text source to control the initial display of help text. The directives must be included in the text in a specific order and in a specific format for the help display program to function correctly.

Order of Directives

The directives must appear in the following order:

1. *T: (Title) must occur before any *YH: entries.
2. *YW: (Window size) must occur before any other entries except *T:.
3. *YD: (Window format definition) must occur before any *YF: , *YV: entries.
4. *YK: (Keys help vector) must occur before any *YV: entries.
5. *YV: (Vector table) must occur before any *YI:, *Y*: or *YH: entries.
6. *YI: (Index entry) must occur before any *YH: entries.
7. *Y*: (Index display-only entry) must occur before any *YH: entries.
8. *YH: (Label entry) must occur after all *T:, *YV:, *YI:, *Y*: entries.
9. F: (Window format) must occur after all *T:, *YD:, *YK:, *YV:, *YI:, and *Y*: entries.

The overall structure of a help text document is thus:

*T: Title Directive

The title directive specifies a title that is shown at the top of every panel of help text.

Title text: The text specified in columns 6-80 will be used as the title.

*YW: Window Size Directives (CUA/Text)

The window size directive specifies dimensions of the window in which help text is to be displayed. If this directive is not present, YDSPHLP takes the size from data area YMHPWDA.

- Window width: 2-digit number (32 through 74) for the width of the help window.
- Window depth: 2-digit number (09 through 21) for the depth of the help window.
- Vary depth: 1-character indicator:

  Y - adjust initial help window depth lower than the given depth either to fit displayed help text or to size window on the application panel

  N - or blank - no adjustment

*YD: Window Format Definition Directives (CUA/Text)

Window format definition directives specify the formats of help text to be displayed in a window, including the margins and indentation, so text will fit window width. Note that you can use these directives along with, or instead of, Text Management/38 paragraph format definitions. If your help document has Text Management/38 paragraphs, you may not need to use this directive. Window format definition directives are not processed when help is displayed in a full panel.

- Definition ID: 2-digit identifier (01 through 99) for the definition. You can reference this identifier in subsequent window format (*YF:) directives.
- Start Tab position: 2-digit number (00 through 09) for number of tabs from the left margin at which to start text. Window width is divided into ten tabs. YDSPHLP calculates tab size based on window width.
- Start Text size: 3-digit number (-99 through +99) for the absolute size of starting text for the format. You can use starting text for headings and definition terms (to which you can attach word-wrapped descriptive text). Starting text is formatted exactly as entered including blanks. A negative number indicates how many spaces to indent the first line. The length of starting text and the start tab position determine the left margin to which the formatted text wraps.
- Blank before: 1-character indicator; Y - precede format with a blank line, N or blank - do not precede with a blank line.

*YK: Keys Help Vector Directives (CUA/Text)

The Keys help vector directive labels text about the application function keys within the help document. YDSPHLP displays this text when the user presses F9= Keys help.

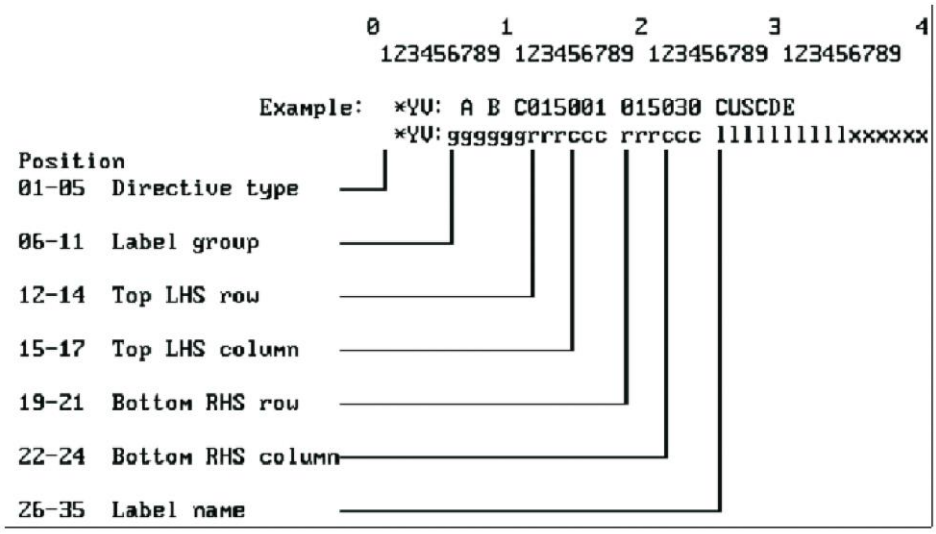
Enter keys help vector directives exactly as shown:

Keys help label: Marks the location of Keys help within the help document.

*YV: Vector Table Directives

Vector table directives specify how the YDSPHLP program should translate an area of the display into a label; if the cursor location is within the area, the specified label will be used to position the display. They are only required if cursor-sensitive help text is to be used. At run time you will need to supply values for the CSRLOC and optionally for the LBLGRP parameter on the YDSPHLP command.

Enter vector table entries exactly as shown:



Label group: up to three separate label groups (a label-group is one character long) may be specified, each preceded by a space or an 'N'. Label groups allow you to display different parts of the same text member under different circumstances. Any alphabetic or numeric character may be used to identify a label group. The YDSPHLP program will compare these label groups with those supplied at run time by the LBLGRP parameter, if any, to decide whether a particular vector table entry is eligible for use or is to be ignored (see the following section on conditioned help text).

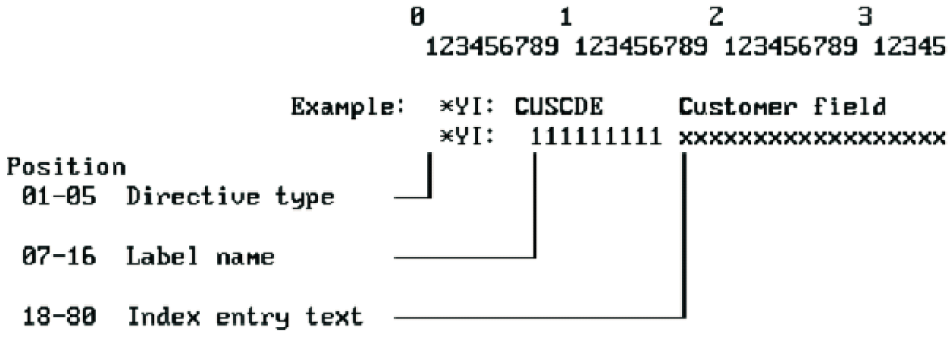
Row coordinates: must be between 1 and 24; leading zeroes may be entered as blanks.

Column coordinate: must be between 1 and 132; leading zeroes may be entered as blanks.

Label name: must refer to the label specified on a *YH: directive placed in the body of the help text. You may also specify the same label on an index entry directive (*YI:), though this is optional. Any text in columns 36-80 following a label name will be ignored.

*YI: Index Directives

An index directive specifies an entry in the help display index; if any index entries are present the help display program will automatically build an index from them. Each index entry points to a label in the text body.



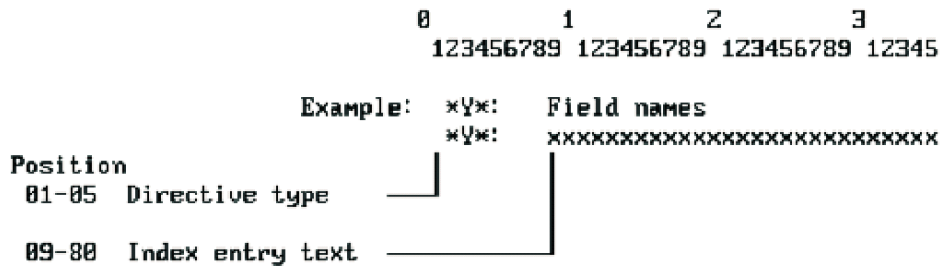
Label name: must be a unique identifier, up to ten characters long, of an index entry. For each *YI: entry a corresponding *YH: label entry is required.

Index entry text: The text specified in columns 18-80 will be displayed in the help text as the index entry.

Note: If the index entry text has a display attribute, that is, is highlighted or underlined, then it should not start before column 19 (to allow for the display attribute byte).

Y: Index Comment Directives

Index comments appear in the displayed help text as part of the index: they are used to make the index more readable for the user.

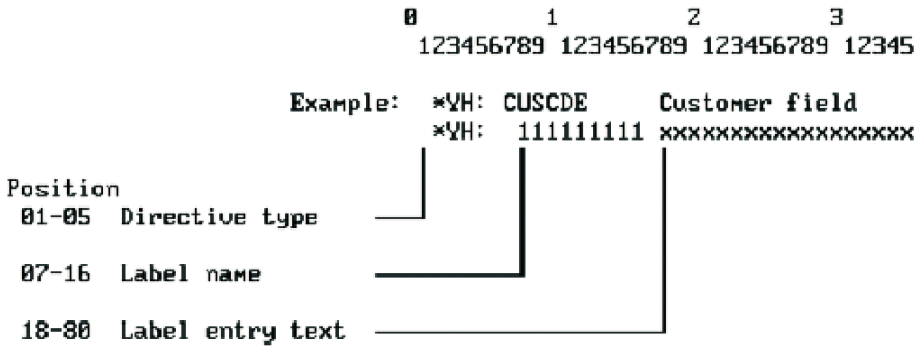


Index entry text: The text specified in columns 9-80 will be displayed as part of the help text. If the text has a display attribute, that is, is highlighted or underlined, then it should not start before column 10 (to allow for the display attribute byte).

*YH: Label directives

Each label directive identifies a point in the help text at which the help text display is to start or to which it may branch under the user's direction. Each *YH: entry should have a label corresponding to that of a *YV: or *YI: entry (unmatched entries will be ignored). The Help text display program starts a new page for each label found.

When the help text for the label is displayed in a window, the specified label entry text is used for the title of the window. If the label is referenced in an index directive (*YI:), the text on the index directive is used in place of the text on the label. The text ' - Help' is appended to the label entry text to form the window title.



Label name: must be a unique identifier up to ten characters long corresponding to an index or vector table label entry.

Label entry text: The text specified in columns 18-80 will be displayed as part of the help text. If the text has a display attribute, that is, is highlighted or underlined, then it should not start before column 19 (to allow for the display attribute byte).

*YF: Window Format Directives (CUA/Text)

Window format directives specify that the format of text to follow is to be based on a given window format definition (*YD:). Note that window format directives do not apply to help displayed in a full panel, such as Extended help.

Definition ID: 2-digit identifier for the format of text to follow. Enter the same identifier as that of the window format definition to be used. If the Definition ID is zero, the text to follow remains unformatted and is not word-wrapped.

Text/38 Paragraph Formats (CUA/Text)

To allow for window expansion and to allow both Extended Help and Windowed Help to originate from the same source document (which may have been created before Help windows became available), Text Management/38 paragraph definitions are automatically interpreted into Window Format definitions as follows:

Text Management/38 Paragraph Option	Display Help Derived Format Option
Left margin	Divide by tabsize and round to nearest whole tab. Set Start Tab Position to ths number.
Right margin	Right margin is always the current window width.
Align Right	Text is not aligned right.
Paragraph Indentation	Negate this to give Start Text size. Set Blank Before option to this.
Blank Line before paragraph	Text is not hyphenated.
Automatic Hyphenation	

Positioning the Help Window (CUA/Text)

When the help display program is called with a cursor location (in ‘cursor-sensitive’ mode), the help window positions with an offset from that location, choosing a corner to locate relative to that position. The corner to locate is chosen to give a "best-fit" on the underlying panel.

When the help display program is called with a start label, the current cursor location is retrieved and used for positioning as described previously. If further help windows are displayed from the initial help panel, these are positioned to overwrite the previously displayed help window. By pressing a function key (F19), any help window can be re-positioned at or near a location pointed to by the cursor.

Examples of text source for help and the resulting formats are shown in the following diagram:

For this example of text source, the Keys Help would be displayed as follows:

And the Index Help would be displayed as follows:

User Profiles

This section tells you how to create, change, delete, rename, document, and copy user profiles. Other topics covered include retrieving user profile details and using user profile extension attributes.

Introduction

CA 2E Toolkit provides facilities both to help manage your existing user profiles, and to set up and maintain Toolkit "user profile extension attributes" for your profiles.

A user profile is the i OS object used to control user access to Toolkit, and to define what objects a user may use on the Toolkit.

There are good reasons for making full use of i OS user profiles:

- The i OS security that is obtained through user profiles is implemented at a microcode level and so is very secure.
- User profiles give you a convenient means of specifying exactly what a given user may or may not do.
- User profiles give you a convenient means of finding out exactly who did what, when; that is, they provide a means for system auditing.
- User profiles give you a convenient means of costing system usage.

A Toolkit user profile corresponds to the i OS definition; that is, it is a description of what a particular user may do on the computer, accessed by a single password, and guarded by security implemented at the microcode level. Toolkit allows extensions to the attributes that can be given to a user profile.

The Toolkit user profile extension attributes are mainly concerned with the initiating of interactive jobs; that is with controlling user access onto the system, and include an initial library list, an initial menu, and a suspension status. Users may be suspended so that they cannot sign on.

The Toolkit Initial Program

The Toolkit initial program provides a special program for user profiles, YINLPGM, which is in the Toolkit library. The Toolkit initial program makes use of the Toolkit user profile extension attributes to set up interactive job sessions, for instance:

- The library list for the job is set to the specified stored library list.
- Password expiry is checked.
- The specified initial menu is displayed, or initial menu option executed.

The source of YINLPGM is available, so that you may add in any requirements that are peculiar to your environment.

With the Toolkit user profile extensions, you can create and maintain a highly supportive environment for your users, without the need for programming changes.

General Considerations for User Profiles

To manipulate user profiles, see the following command diagrams in the *Toolkit Reference Guide*.

- YCHGUSRPRF
- YCPYUSRPRF
- YCRTUSRPRF
- YCVTUSRPRF
- YDLTURSPRF
- YDOCUSRPRF
- YDSPUSRPRF
- YRNMUSRPRF
- YRTVUSRPRF
- YWRKUSRPRF

The following diagram shows how the Toolkit user profile commands interconnect.

Storing User Profiles

User profile extension information is stored in a database file, YUSRPRF. This file is transparent on commands: you do not need to name it. Typically you will want to keep only one copy of this file. A default copy is shipped in the Toolkit library.

User profile extension information is used by the Toolkit initial program, YINLPGM, to initiate interactive jobs. This program calls a separate program, YINLPGMPWD to check the password expiry date.

Extension Attributes

Toolkit allows you to specify the following extension attributes for user profiles:

- A password expiry date or change period (see below)
- An initial library list to be used at sign-on
- An initial menu, (and menu file + library) to be used at sign-on
- Whether the user is restricted to direct menu calling (see section on menus in this manual)
- Whether the user is allowed to enter commands from his menu (see section on menus in this manual)
- Whether the user is suspended: this option can be used to prevent users signing on, for example, while new application programs are being implemented
- The name of a job description to be used when the user submits jobs from his menu
- Whether to sign off with LOG(*YES)

Working with User Profiles

The Toolkit Work with User Profile command (YWRKUSRPRF) provides a convenient way of reviewing and editing your user profiles. It provides an interactive display of all the existing user profiles that have previously been either created or amended using the Toolkit Create User Profile (YCRTUSRPRF), or Change User Profile (YCHGUSRPRF) commands. From the display you can select user profiles for detailed examination or changing.

If **5** is specified as a select value against a line on the previous display, the Display User Profile details display is obtained, which allows you to examine and change the Toolkit attributes of existing user profiles. From the display you may invoke commands to both to change the profile and to edit the contents of the initial library list and initial menu associated with the profile.

The following diagram shows the interconnections between the displays of the Toolkit Work with User Profile program.

Creating User Profiles

User profiles are created using the Toolkit command Create User Profile (YCRTUSRPRF). The command allows you to specify both the system and Toolkit user profile attributes.

```
YCRTUSRPRF USRPRF(HERRICK) TEXT('Rev. Herrick"s profile') LIBLST(JULIA) MENU(CHLOE) PTYLMT(2)
LMTCPB(*PARTIAL)
```

Any new profiles created with the Toolkit Create User Profile utility will have the Toolkit initial program (YINLPGM) as their initial program, unless you specify an override value. LMTCPB(*PARTIAL) is specified to prevent a user from overriding the initial program from the Toolkit sign-on screen or the i OS command Change Profile (CHGPRFF).

Changing User Profiles

You can change user profiles using the Toolkit utility Change User Profile (YCHGUSRPRF). The command allows you to change both the system and Toolkit attributes.

```
YCHGUSRPRF USRPRF(HERRICK) MENU(AMARYLLIS) PTYLMT(3)
```

Existing user profiles can be brought within the Toolkit system by using the Toolkit change user profile command upon them. YINLPGM should be specified as the initial program. If you wish to bring all of your existing user profiles into the Toolkit system, the Toolkit object list utilities can be used. See the section on lists later in this manual for further details. For example:

```
YBLDOBJLST OBJ (QSYS/*ALL) OBJTYPE (*USRPRF) /* Build list */
YFLTOBJLST FILTER(*OMIT) OBJ(Q*) /* Omit all sys prfs */
YEXCOBJLST RQSDTA('YCHGUSRPRF USRPRF(@O)
INLPGM(Y1SY/YINLPGM) INLMNU(*SIGNOFF)')       rfs */
```

Deleting User Profiles

User profiles can be deleted using the Toolkit Delete User Profile utility (YDLTUSRPRF). The command deletes both the system and Toolkit entries.

```
YDLTUSRPRF USRPRF(HERRICK)
```

Renaming User Profiles

User profiles can be renamed using the Toolkit Rename User Profile utility (YRNMUSRPRF). The YRNMUSRPRF command does not handle the renaming of profiles which are enrolled in Office/400: any such enrollment should be deleted before a profile is renamed.

```
YRNMUSRPRF FROMUSRPRF(BRAY) TOUSRPRF(FRED)
```

Documenting User Profiles

Both the system and Toolkit user profile attributes can be printed out for some or all profiles, using the Toolkit command Document User Profile (YDOCUSRPRF).

```
YDOCUSRPRF USRPRF(HERRICK)
```

Retrieving User Profile Details

Details about the Toolkit user profile attributes can be retrieved in your own CL programs by using the Toolkit command Retrieve User Profile (YRTVUSRPRF).

Copying User Profiles

User profiles can be copied using the Toolkit Copy User Profile utility (YCPYUSRPRF). The command copies both the system and Toolkit user profile information, and provides an option to have the copied user profile's authorities granted to the new user profile by invoking the Grant User Authorities command (GRTUSRAUT).

```
YCPYUSRPRF USRPRF(HERRICK) TOUSRPRF (DUNNE) TEXT('John Donne''s user profile')
```

Security Considerations

To be able to use the Toolkit user profile manipulation commands, you must be authorized to the system's create, change and delete user profile commands: typically (and in the i OS shipped system) only the security officer will have such authorization.

User Profile Extension Attributes

The main use for the Toolkit user profile extension attributes is to control the initiation of interactive jobs, that is, "signing on". Any user profile having the Toolkit initial program, YINLPGM, as its initial program can make use of the extension attributes.

The Toolkit initial program - which you may alter or enhance - will do the following:

1. Retrieve the user profile name.
2. Get the user profile extension attributes for the user.

   If the user profile has no extension attributes, send a diagnostic message (YUS0001), and terminate the job.
3. Check that the user is allowed to sign on.

   If the user profile is suspended, send a diagnostic message (YUS0014), and terminate the job.

   If the user's profile has a password expiry date call the password checking program, YINLPGMPWD. This program, and the password expiry control values, are described below in the section on Password validation.

   If the password expiry check fails, sign off using the i OS SIGNOFF command.

   If the password expiry check succeeds, proceed.
4. Set the initial library list for the user.

   If the initial library is not found, send a diagnostic message and terminate the job.

   If the initial library list contains libraries that either do not exist, or that the user is not authorized to use, send a diagnostic message and terminate the job.
5. Execute the specified action for the user:

   Either: display the specified initial menu for the user.

   Or: execute the program specified by the initial menu option for the user. See Using Different Control Programs below.
6. Return. Subsequent processing will depend on the value specified for the INLMNU parameter on the profile:

   If INLMNU(*SIGNOFF) is specified, the job will terminate.

   If any other value is specified for INLMNU, the nominated i OS menu will be displayed.

Modifying the Initial Programs

The source for the Toolkit initial programs is available so that if necessary you can:

- Satisfy audit requirements.
- Develop your own versions.

Note: If you alter the Toolkit initial programs we suggest that you keep the modified versions in a different library from the Toolkit shipped library. This is because modified versions of the initial program may be shipped in subsequent releases of Toolkit.

Using Different Control Programs

In the simplest case, the Toolkit initial program invokes the Toolkit program Go to menu (YGO) in order to display the appropriate menu. Subsequent processing is then under the control of the menu display program. You may, however, use the Toolkit user access system to invoke a control program other than the Toolkit menu display program. This gives you the capability:

- To use Toolkit user profile extension facilities without having to code your own versions of the Toolkit initial programs
- To invoke additional initialization processing - for instance to set overrides which are to be in effect globally
- To supply parameters to an initial control program
- To use Toolkit user profile extension facilities in conjunction with i OS menu objects

The Toolkit initial program (YINLPGM) will still be used as the initial program for the profile.

The overall sequence of control is shown in the following diagram:

Summary of INLPGM Control

The following diagram shows the factors controlling which menus, menu options, or both are displayed for a job:

Using Different Control Programs

You can incorporate other control programs into the Toolkit user profile system as follows:

1. Set up a Toolkit dynamic menu with a menu option specifying the requisite initial program as the action for the menu option. Parameters may be specified.
2. On the user profile extension attributes, specify not only the initial Toolkit dynamic menu, but also the number of the initial menu option.
   - To provide QCMD as the initial program you would specify:

YCHGUSRPRF USRPRF(FRED) MENU(FRED) OPTION('1')

   - To provide QPGMMENU as the initial program, with the restricted changes allowed, you would specify:

YCHGUSRPRF USRPRF(FRED) MENU(FRED) OPTION('2')

This method gives you an easy and instant way of specifying and changing the parameters for the i OS Programmer's Menu (QPGMMENU): you could specify a menu option consisting of the i OS Start Programmer Menu (STRPGMMNU) command with the appropriate override values.

Note: You should not use relative numbering to number menu options used by the initial program. For a definition of relative numbering, see the section in this manual on menus.

Using i OS menus

You may also use i OS menus in conjunction with the Toolkit user access system: on normal completion, the initial program YINLPGM returns without invoking the i OS command Sign off (SIGNOFF). This causes control to be returned to the system, which will then display any menu specified by the INLMNU parameter on the user profile.

1. If you want to display a Toolkit menu (or menu option) but not an i OS menu, specify the name of the Toolkit menu for the MENU parameter, the name of the Toolkit menu file for the MENUFILE parameter, and a value of *SIGNOFF for the INLMNU parameter. For this to be effective, you must specify LMTCPB(*YES) on the user profile (otherwise INLMNU can be overridden from the Toolkit sign-on screen or the i OS command Change User Profile (CHGPRF) .
2. If you want to display an i OS menu but not a Toolkit menu, specify the name of the i OS menu for the INLMNU parameter and a value of *NONE for the MENU and MENUFILE parameters.

Normally you will not wish to display both a Toolkit menu and an i OS menu, but you may do so. You might for example use a Toolkit menu option to run a job initialization step prior to displaying an i OS menu.

Password Validation

Toolkit provides two optional features to assist you in enforcing the effective use of passwords by your users:

- Password expiry checking
- Password value checking

Password Expiry Checking

Toolkit will provide password expiry checking to ensure that the passwords for specified profiles have been changed within a given period. Password expiry checking is controlled by four extension attributes on the user profile. The options allow you to specify:

- A password expiry date: this specifies an absolute value for the expiry date; for example, that the password will expire on 01/04/05.
- A password expiry period in days: this specifies a value for password expiry relative to the profile's password last change date - for example, that users must change their passwords at least every 30 days
- A password expiry option: this specifies what is to happen if the password has expired. User may either be prevented from signing on, or be allowed to change their passwords, provided that they are within a grace period: see the following section.
- A password expiry grace period: this specifies the number of days after the notional password expiry date, during which users may still sign on. During this period they will be automatically prompted to change their passwords every time they sign on.

The password control values are used to calculate two dates - a base expiry date and a grace expiry date. The next section, on the Initial Password Expiry Program, indicates how the dates are calculated. The following two diagrams show how these two dates are used to determine whether the user may sign on or not.

Initial Password Expiry Program

If a password expiry value is set for the profile (for example, if either an absolute password expiry date, or a relative expiry period in days is specified for the profile), the Toolkit initial program, YINLPGM, calls a separate program YINLPGMPWD to carry out the actual checking.

The YINLPGMPWD program will do the following:

1. Retrieve the password last change date for the user profile (PWDCHGDAT on i OS user profile), and the current system date (i OS QDATE system value).
2. Determine a base password expiry date, as follows:

   If the profile has a password expiry date which is earlier than the current system date, it will be used as the base expiry date.

   Otherwise, the password expiry period (if any) will be added to the password last change date to give the base password expiry date.
3. Determine the grace password expiry date by adding the password expiry grace period for the user profile to the base expiry date computed previously.
4. Compare the current system date with the calculated expiry dates.
   - If the current system date is earlier than the base password expiry date, the password is still current, so return without further checking.
   - If the current system date is later than the grace password expiry date, the password has expired, so send a diagnostic message (YUS0027) and sign off. User must have their user profiles changed by a security officer before they may sign on again.
   - If the current system date is later than the base password expiry date but earlier than the grace password expiry date, the password is within the grace period.
   - If the password expiry option is *NOSIGNON, the user will not be allowed to specify a new password, so send a diagnostic message (YUS0027) and sign off.
   - If the password expiry option is *PMTCHG, send message (YUS0048) and prompt the user with the Toolkit display Change Password (YCHGPWD) (see the section Checking New Password Values) to change the password (that is, users are still allowed to sign on, but should change their passwords before the grace period expires). When the user exits from the display, whether the password has been changed or not, proceed with the sign-on process. Note that when the password is changed, if there is an absolute password expiry date and it is earlier than the grace period expiry date, then the password expiry date will be reset to zero.

Password Value Checking

i OS provides some validation for password values. Toolkit will optionally provide additional password value checking to ensure that users do not use obvious values for passwords. You may specify:

- That user profile names are not used as password names.
- That new passwords are not values on a list of forbidden values. You may add to this list of forbidden values yourself. You may forbid a value for a particular profile (for example, QSECOFR) or for all profiles.
- That a new password has not already been used. You may specify that Toolkit is to log old passwords to ensure that they are not reused: the passwords are added to the list of forbidden values.

Editing Password Control Values

Password value checking is controlled by the values which you specify using the Toolkit command Edit Password Values (YEDTPWDVAL).

Storing Forbidden Password Values

Forbidden password values are stored in a database file, YPWDVAL. This file is transparent on commands: you do not need to name it. Typically you will want to keep only one copy of this file, in the same library as the YUSRPRF file. A default copy is shipped in the Toolkit library: the installation process activates this.

Checking New Password Values

Toolkit password value checking is implemented by a Toolkit command Check New Password (YCHKPWDVAL). The command is provided so that you can check a candidate password against the validation options specified with the YEDTPWDVAL command.

Changing a Password

Users may change their own passwords using either the i OS command Change Password (CHGPWD) or the Toolkit command Change Password (YCHGPWD). The Toolkit command YCHGPWD will be prompted automatically by the Toolkit initial program if the user's password expiry date has been exceeded, see the previous section.

If password logging is specified, then on a successful change of password, the old password value will automatically be added to the 'forbidden passwords' file.

At release V1R2 (or higher) of i OS, Toolkit password validation and logging is carried out by the Password validation program (YPWDVLDPGM) (see following) called from the i OS CHGPWD command. The i OS CHGPWD command can be called either directly or from the Toolkit display Change Password YCHGPWD.

Password Validation Program PWDVLDPGM)

The Password Validation Program is supplied to carry out Toolkit password value checking and logging where i OS Release V1R2 (or above) is installed. If you choose to implement Toolkit password checking (specify this using an option on the Edit Password Values (YEDTPWDVAL) display) the 'Password validation program' system value (QPWDVLDPGM) is set to Y1SY/YPWDVLDPGM in the product library; this program will then be called from the i OS Change Password (CHGPWD) facility.

The source for the YPWDVLDPGM program is shipped with the Toolkit product, so that you can include any additional password validation that you require. If you amend the Toolkit YPWDVLDPGM program, we suggest that you keep the modified version in a library different from the Toolkit product library; this is because new versions of the program may be shipped in future releases.

Password Validation Restrictions

Toolkit password validation and logging is implemented only using the Change Password commands as described previously. It is not invoked from Toolkit or i OS commands, which allow you to create or change user profiles.

Chapter 3: Design Utilities

This chapter provides an introduction to the design aids including the panel designs and the report designs. An example of how to use the design aids is provided.

This section contains the following topics:

Introduction

The CA 2E Toolkit design aids provide an easy and rapid method of designing application systems. Design aids covering menus, panels, and reports are provided, as well as the means to connect the component parts into an integrated whole.

Design is done on-line using interactive utilities.

All parts of the design can be printed to a high standard, or presented to the user interactively.

- The printed version can form the basis of a system specification, as it is straightforward enough for your user to understand, yet specific enough for a programmer to work from.
- The interactive prototyping facility enables a complete model of the system to be created, with realistic data and program connection. You and your user can test this model and work out any problems before programming is begun. The advantages of greater user involvement and feedback are self-evident.

Full use of the information contained in the designs can be made in programming the system.

Overview

There are two main components to the Toolkit design utilities:

- Panel designs
- Report designs

There are other components of Toolkit that may be useful when designing systems. In particular the facilities provided by the Toolkit user access aids may be of interest:

- Menus
- Help text
- User profiles
- Library lists

Each of the components can be used separately; all of the components can be linked together to form an integrated system.

Commands

See the command diagrams for the following Toolkit commands in the *CA 2E Reference Guide* for operational details:

Type	Edit	Print	Display	Delete	Copy	Generate
PNL	YWRKPNL	YDOCPNL	YDSPPNL	YDLTPNL	YCPYPNL	YCRTPNLDDS
RPT	YWRKRPT	YDOCRPT	YGO	YDLTRPT	YCPYRPT	YCRTRPTDDS. . .
MNU	YWRKMNU	YDOCMNU		YDLTMNU	YCPYMNU	

The following is a list of other relevant commands:

- YADDDSNFM
- YCRTDSNF
- YDFNPNLDSN
- YEDTDSNDFT
- YRTVPNLDSN
- YRTVRPTDSN

Panel Design

The Toolkit panel design aids include facilities both for creating and presenting panel designs.

Panel designs can quickly be created using an interactive editor:

- Panel images, narrative and branching logic to describe how the panel designs connect with other panel designs can all be specified and documented.
- The panel design utility has many features to accelerate design, including line move, block move, save, and copy functions.
- Existing display file source can be converted back to a design.

Panel designs can be displayed as simulated programs using a special prototyping utility:

- All fields have their appropriate display attributes for example, only input capable fields are input capable). No compilation is required.
- Realistic data values can be set up in both input and output fields to help the user understand what goes where.
- The specified branching logic will be followed, so that panel conversations can be fully illustrated.

Panel designs can be documented to a high standard.

- Indexing and print sequencing of the designs is allowed.

Display file DDS can be generated directly from the designs.

- Standards will automatically be applied to the generated code.

Report Design

Report designs can quickly be created using an interactive editor:

- Report images and narrative can both be specified.
- The report design utility has many features to accelerate design, including windowing, line move, block move, save, and copy functions.
- Existing print file source can be converted back to a design.

Report designs can be documented to a high standard.

- Indexing and sequencing of the designs is allowed.

Print file DDS can be generated directly from the designs.

- Standards will automatically be applied to the generated code.

Menus in Design

The Toolkit menu utilities provide a rapid way of designing and presenting menus.

Menus are the ideal top down framework within which to design a system. They represent how the system will appear to the user, and so provide a checklist of available functions. Since every task is allocated a menu, and every menu is associated with a user, menus help you to highlight an often neglected aspect of system design: who will do what, when.

The Toolkit Menu concepts are discussed in detail in the section on menus in this manual. From the point of view of design, the following features of Toolkit menus should be stressed:

- User functions and menus can rapidly be designed in outline and presented to the user for comment.
- Help text, panels and reports can be associated with each menu option so that when a user takes the option, the appropriate design is displayed.
- The design menus can be carried straight into development, with actual programs replacing designs when they are completed

Help Text in Design

Help text can be written in advance of programming and connected into the system design.

The Toolkit help text concepts are discussed in detail in the "Toolkit User Access Aids" chapter.

User Profiles in Design

User profiles can be set up as part of the design, each with its own menu sets. The user can than realistically sign on to a system, complete with menus and panels, without any programming being done.

For a detailed discussion of Toolkit user profiles see the earlier chapter "User Access Aids".

Using the Design Aids Example

The following are steps and an outline example of how to use the design aids to design a system.

1. Design a single menu, using the Toolkit utility Work with Menu (YWRKMNU).

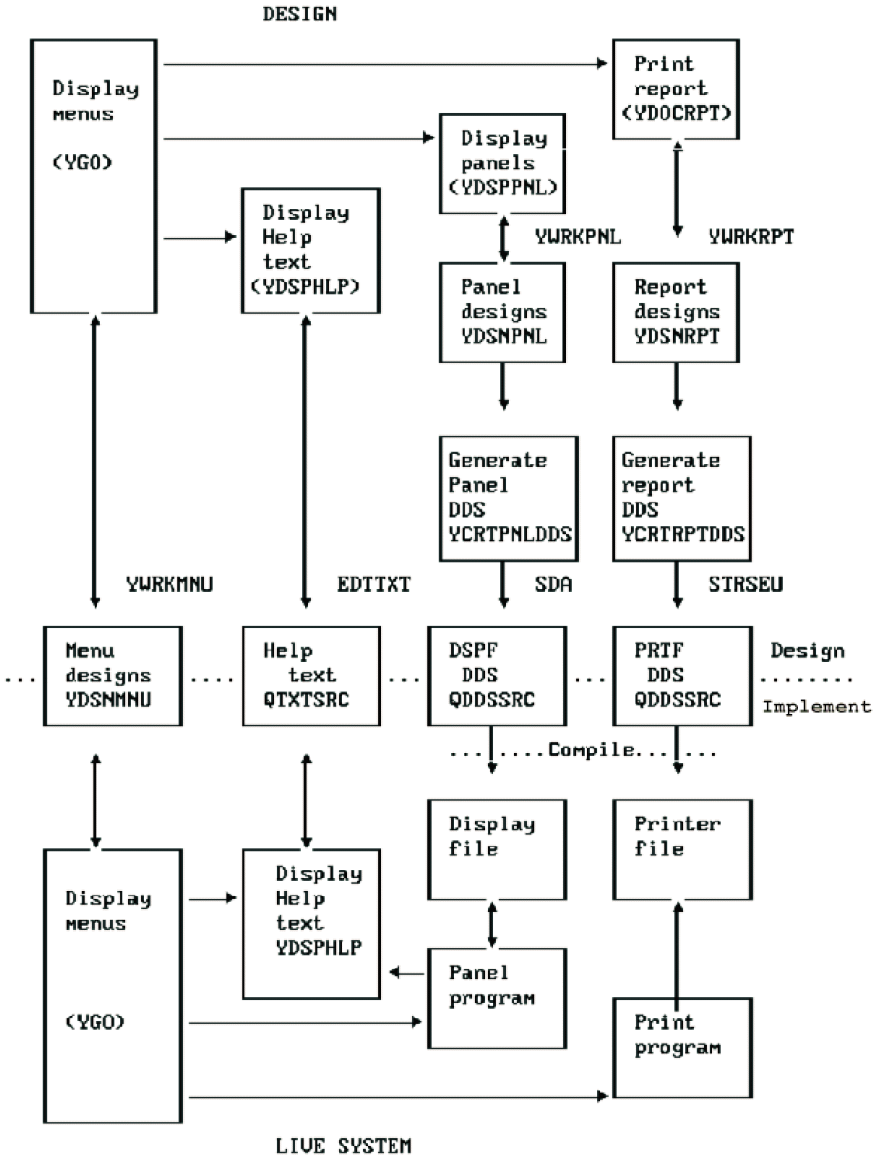
2. Refine the menu into three sub-menus. For example:

3. If necessary, add a help text to explain the scope of each function.

4. For each option, prepare panel designs or report designs using the Toolkit Work with Design utilities YWRKPNL and YWRKRPT.

5. Link the panel and report designs into the appropriate menus; for example the Toolkit command Display Panel Design (YDSPPNL) could be specified as a menu option to display a panel design when an option is taken.

After only a few minutes work, you can show a user a realistic simulation of what the system will look like and print a description of that system.


DESIGN
Display menus (YGO)
Print report (YDOCRPT)
Display panels (YDSPPNL)
Display Help text (YDSPHLP)
YWRKPNL
YWRKRPT
Panel designs YDSMPNL
Report designs YDSNRPT
Generate Panel DDS YCRTPNLDDS
Generate report DDS YCRTRPTDDS
YWRKMNU
EDTTXT
SDA
SIRSEU
Menu designs YDSNMNU
Help text QTXTSRC
DSPF DDS QDDSSRC
PRTF DDS QDDSSRC
Design
Implement
Compile
Display file
Printer file
Display menus (YGO)
Display Help text YDSPHLP
Panel program
Print program
LIVE SYSTEM

Panel Designs

This section describes CA 2E Toolkit panel design concepts. Toolkit provides facilities for:

- The rapid design of panels
- The specification of how the panel designs connect to menus and to other panel designs
- The presentation of the panel designs to the user

Once finalized, the panel designs can be converted into i OS Data Description Specifications (DDS), ready for programming.

Panel presentation standards - standard layouts, display attributes, message presentation and so on, can be specified centrally. The standards will then be automatically implemented on all the displays that are generated, reducing the amount of work required to specify a panel design, and also improving the standard of presentation throughout the application system.

General Considerations for Panel Designs

Before you create the panel designs, consider the following: commands you need to manipulate the panel designs, presentation standards, naming and storing conventions, panel design components, and panel design defaults.

Panel Design Commands

The following commands manipulate the Toolkit panel designs. See the following command diagrams in the *CA 2E Toolkit Reference Guide*.

- YCRTDSNF
- YADDDSNFM
- YWRKPNL
- YDSPPNL
- YDOCPNL
- YCPYPNL
- YDLTPNL
- YRTVPNLDSN
- YDFNPNLDSN

Presentation Standards

Panel presentation standards may be specified centrally using the Toolkit command Edit Design Defaults (YEDTDSNDFT). The presentation standards are used both when entering and when displaying panel designs. Among the items that can be specified are:

Display attributes (high intensity, column separators, and so on) for each type of field (input, output, input/update, and so on).

Representation characters for each type of field on panel designs (I, O, B, 3, 6, 9). These are shipped as being the same as those used by the Toolkit utility Screen Design Aid (SDA).

Standard values to be used when generating source, including the name of the field reference file, the print key file, and the message file. Standard indicators can be assigned to condition certain DDS keywords, for example SFLEND, SFLCLR.

Naming Conventions

Each Toolkit panel design is given a name and the designs are referred to in Toolkit panel design commands by this name. Names must be valid system names (that is, begin with a letter and be up to ten characters long). For example:

```
YWRKPNL PANEL(FRED)
```

Storing Conventions

Toolkit panel designs are stored in database files. When using Toolkit panel design commands you may also specify the name of the data base file containing the panel design. For example:

```
YWRKPNL PANEL(FRED) FILE(CHARLOTTE)
```

Assigning Default Names:

If you do not specify a name for a panel design file, the default name is used: *LIBL/YDSNPNL.

Since the default name for a panel design file on all the Toolkit panel design commands is YDSNPNL, it is recommended that you call all of your panel design files YDSNPNL, and make use of your library list to distinguish between different versions of designs (in the same way as you would use the i OS default source file names) this will save you having to specify the file name.

Creating Panel Design Files:

Toolkit panel designs are stored in a special database file format. Additional files of the correct type to contain panel designs can be created using the Toolkit command Create Design File (YCRTDSNF), for example:

```
YCRTDSNF TYPE(*PNL) FILE(QGPL/YDSNPNL)
TEXT('Panel designs for my system.')
```

You may have different sets of panel designs in different members in the design file. The Toolkit command Add Design File Member (YADDDSNFM) is provided to give you an easy method of adding additional members to design files. For example, to add a new member to panel design file YDSNPNL in library QGPL:

```
YADDDSNFM TYPE(*PNL) FILE(QGPL/YDSNPNL) MBR(EXTRA) TEXT('More panels')
```

Panel Design Components

There are four parts to a Toolkit panel design:

- Title information
- The layout of the panel design
- Narrative describing the panel design
- Command key usage, including command key and data dependent branching

Title Information:

For each panel design you may specify title information: this includes a title, the order the panel is to be printed, and whether the design uses a fixed standard layout for the top and bottom of the panel.

Layout Information:

The panel design layout information consists of the actual panel image: special characters, defined with the Toolkit command Edit Design Defaults (YEDTDSNDFT), are used to indicate field positions and attributes, just as for the Screen Design Aid (SDA).

Narrative Information:

For each panel design, write up to three pages of twenty-four lines of narrative information. This narrative text can be made to appear under the panel image when the panel design is printed, or appear as the Help text for the panel if the HELP key is pressed when using the Toolkit Panel Design Prototyping Program Display Panel Designs (YDSPPNL).

Command Key Usage:

Specify how the panel design connects with other panel designs. This can be done in two different ways:

1. For each command key on a panel design, specify an action to be taken if the command key is pressed; the action may be expressed either as the name of another panel design to be displayed, or as a keyword denoting a standard function:
   - *PRV: display previous panel.
   - *SAME: re-display panel.
   - *EXIT: exit program.
   - *EXEC: execute a command string, specified in the text field.
2. Specify an action to be taken conditional on a value being entered for a particular line and column on the panel design. For instance, on a subfile display, specify that a **5** entered against a selection field on a line will cause a separate display of details for that line to be displayed.

The branching information has a number of uses:

- It can appear as part of the documentation for the panel design.
- It will be used by the interactive Display Panel Design utility (YDSPPNL) to simulate the program's action if a command key is pressed, or an option selected.
- It provides programmers with an exact description of a designer's intentions regarding program flow.

Using a Default Panel Design

Make most of your application panels conform to a standard layout; that is, have items like the panel title, command keys and messages in the same place on all panels. This not only looks better, but also makes application systems easier to understand. Toolkit allows you to set up a default panel design, which will be provided as a starting position on any new panel design that you add.

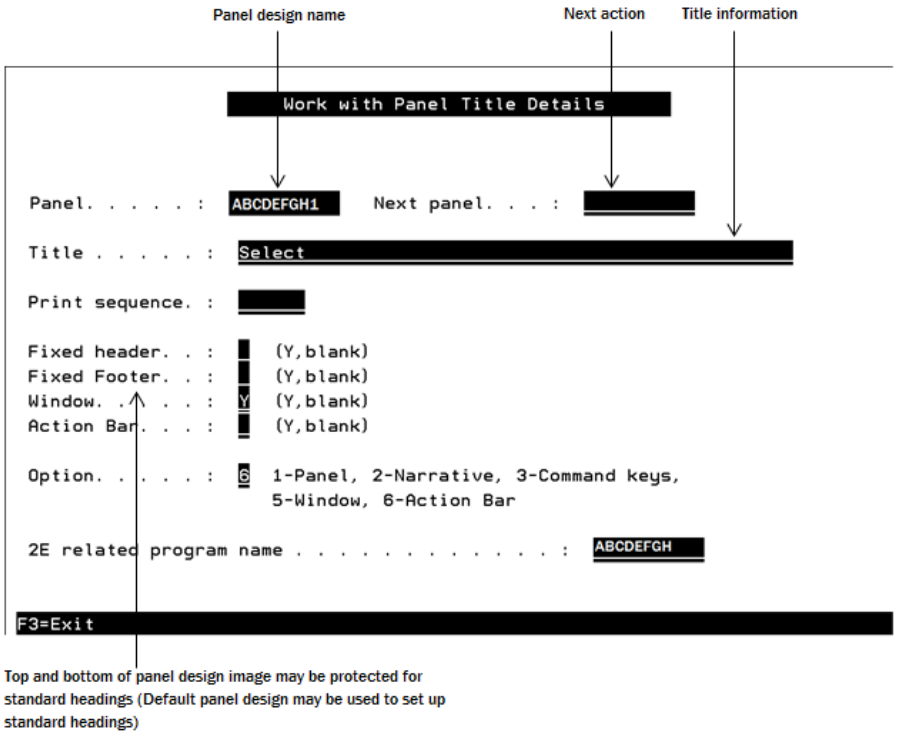
Specify one default panel design for each member in a panel design file. The default panel design must be called #DFTPNL. The default panel design can include narrative, data, and command key usage information.

Working with Panel Design Utility

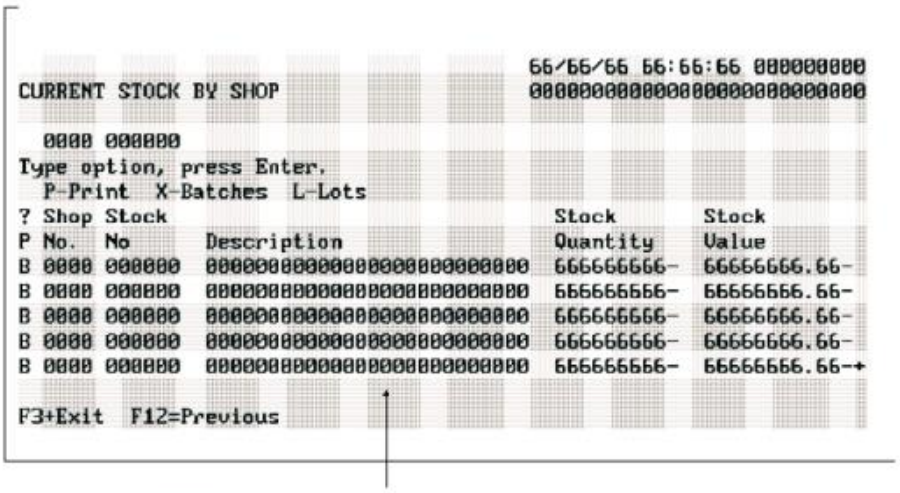
Toolkit panel designs are entered and maintained with the Toolkit utility Work with Panel Design (YWRKPNL). The Work with Panel Design utility is an interactive program that allows rapid, full-screen, editing of panel designs. It has many functions to assist in panel layout, including line copy, line duplication, block copy, block copy, block move, and block save facilities. Refer to the program's help text for details on these facilities.

Examples of the Work With Panel Design Displays

The examples shown on the following pages are the different Work with Panel Design displays including the Work with Panel Title Details display, the Work with Panel Design Image display, Work with Panel Narrative display, Work with Panel Command Key Usage display, Exit Work with Panel display, and the Display Connection map.

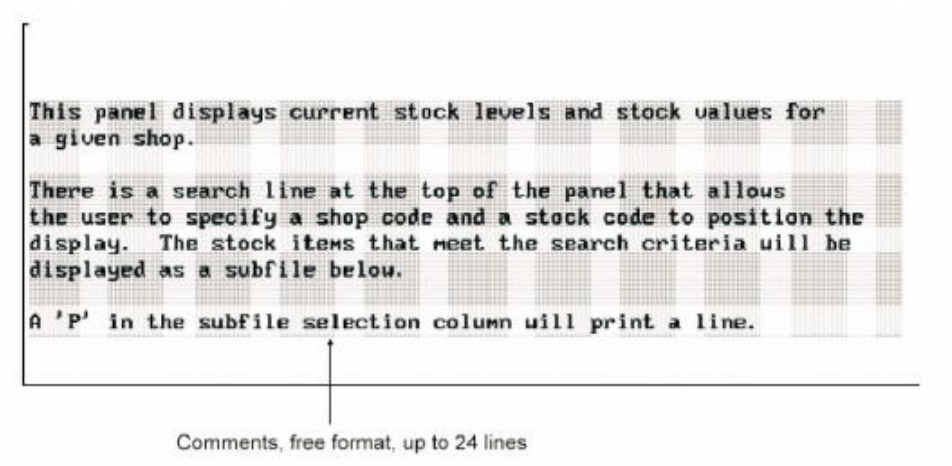


The YWRKPNL title display enables you both to specify title details, and to indicate what you wish to do next, for example edit the image.



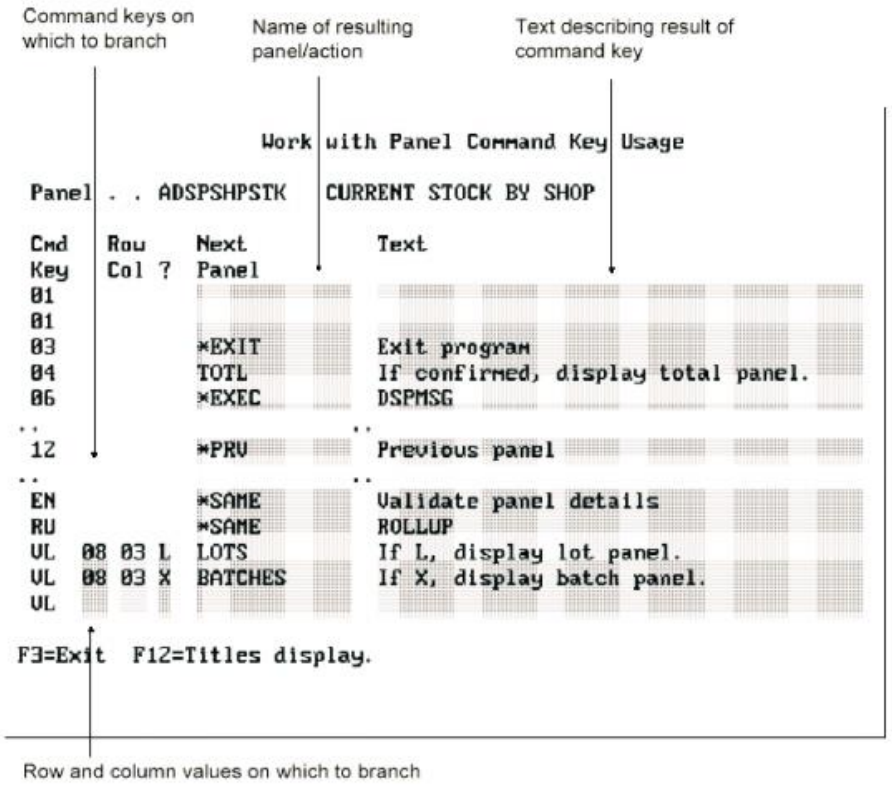
The panel image is laid out as it is to appear, using standard symbols to represent fields e.g., 0 = Output, B = Both Input/Output. (The symbols used may be changed using **YEDTDSNDFT**.)

Option 1 from the panel design title display causes the panel design image display to be shown. The image of the panel design is laid out using a free format display. The top and bottom may optionally be protected.

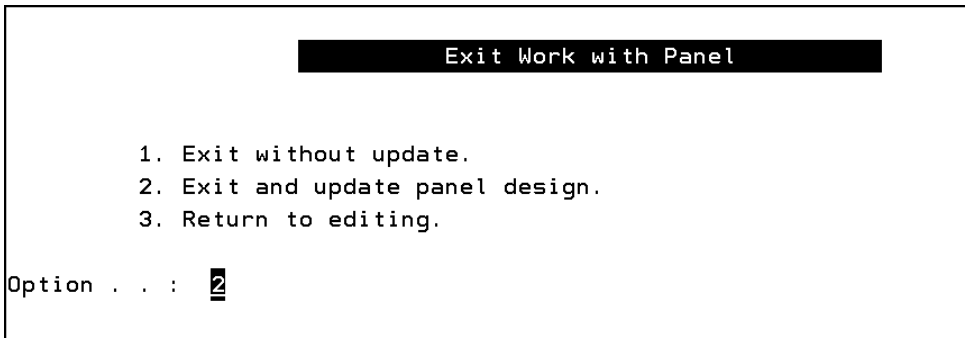


Comments, free format, up to 24 lines

Option 2 from the title panel causes the panel design narrative display to be shown. The explanatory narrative for the panel design is laid out using a free format display of 24 lines. You may use the ROLL keys to move between pages, up to three pages are supported.

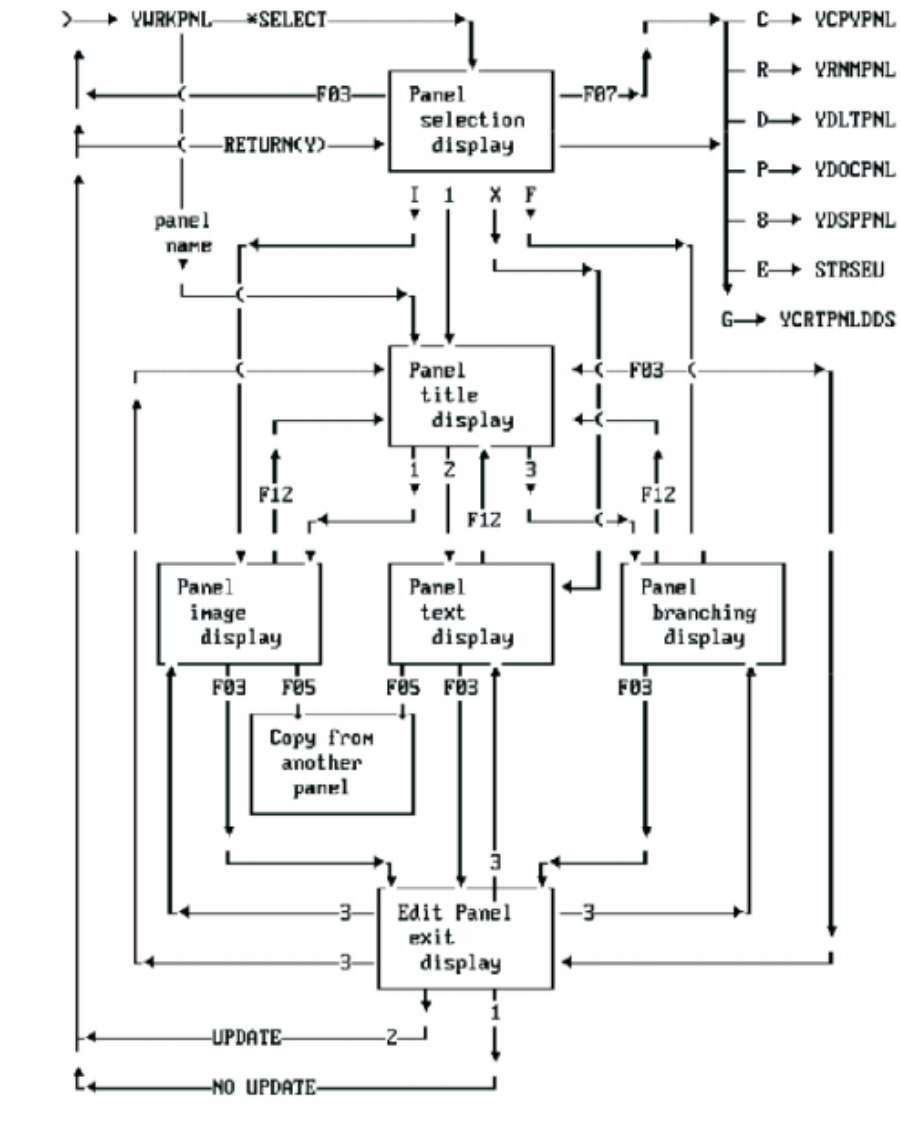


Option 3 from the panel design title display causes the panel design branching display to be shown. The display for specifying branching looks like this:



Exit Work with Panel display

Pressing F03 from any of the panel design displays causes an exit display to be shown. The exit display allows you to make a controlled exit from the Work with Panel Design program. The design is not updated to the panel design database file until this point.



This diagram shows the main interconnections between the displays of the Toolkit program Work with Panel Design.

Presenting Panel Designs

To present panel designs, use the print command or interactive display utility.

Printing Panel Designs

The Toolkit command Document Panel Design (YDOCPNL), will print a panel design, or range of panel designs, in any order. The designs are printed to a high quality standard, suitable for inclusion in a system specification.

The prints of panel designs can include sample data in the fields on the panel design image. The sample data is set up with the Toolkit command Display Panel (YDSPPNL).

Interactive Display

Toolkit provides you with a sophisticated means of presenting panel designs interactively to your users, thus giving them a realistic view of what the finished system will look like - a form of prototyping. Panel design presentation, is done using the Toolkit Display Panel Design utility (YDSPPNL).

Panel designs can contain sample data in input and output fields: sample data can help users to understand a design, as they can see what goes where.

The panel design display utility can be used in one of three modes, as specified by the OPTION parameter on the Toolkit command Display Panel (YDSPPNL):

- CHGDTA mode allows the designer to set up realistic data values for the fields on a panel design, regardless of attribute, so that your user can understand what goes where.
- DSPDTA mode displays a panel design with the realistic data values in the appropriate places, and with all fields having their correct attributes, that is, input and update fields are input capable, output fields and constants are not.
- DSPATR mode displays a panel design with all fields having their correct display attributes, leaving out any example data.

The Display Panel Design utility initially displays a specified panel design. It will branch to other panel designs if command keys are pressed, or if field values are input: the exact nature of the branching is specified as part of the panel design edit process.

The Toolkit utility Go to Menu (YGO) provides a framework for design prototyping. Simply create a menu, specifying as the menu option action the command Display Panel Design and giving as parameters on the command the names of panel designs which you want prototyped when the option is taken.

No compilation is required. A panel design may be displayed as soon as it has been edited.

Using the panel design example shown earlier, we could add sample data using the YDSPPNL command with OPTION (CHGDTA):

And then display the panel design using YDSPPNL with OPTION (DSPDTA).

Design Cycle

Using the Display Panel Design facilities, the typical design cycle consists of the following steps:

- Use YWRKPNL to enter a panel design and narrative text.
- Use YDSPPNL with OPTION(*CHGDTA) to set up sample data for the panel design image.
- Use YWRKMNU (or write a CL program) to create a menu, specifying as a menu option YDSPPNL with OPTION(DSPDTA).

Present the design to the user by letting him display the menu and take the option. (Also provide a printed version prepared with the YDOCPNL and YDOCMNU commands.)

- Use YWRKPNL to revise the panel design to incorporate the user's modifications.
- Use YCRTPNLDDS to generate DDS from the panel design.

Manipulating Panel Designs

Standard functions are available to do the following:

- YCRTDSNF - Create a new panel design file.
- YCPYPNL - Copy a panel design.
- YDLTPNL - Delete a panel design.
- YRNMPNL - Rename a panel design.

A panel design may be converted into a report design, and conversely, by means of the CVTOPT parameter on the Toolkit commands Copy Panel Design (YCPYPNL) and Copy Report Design (YCPYRPT).

Selecting a Panel Design

A select function is available to facilitate the use of panel designs: this enables you to select or add to a list of existing panel designs, or to perform any of the panel design manipulation functions upon a selected panel design. The select function is invoked by specifying a value of PANEL (*SELECT) when using the command Work with Panel Design (YWRKPNL), for example:

```
YWRKPNL PANEL(*SELECT) FILE(YDSNPNL)
```

Generating DDS for a Panel Design

The Toolkit command Create Panel Design DDS (YCRTPNLDDS) is provided to create the i OS Database Description Specifications (DDS) for a display format directly from a Toolkit panel design. The generate function is interactive, and allows you both to specify how a panel design is to be broken up into formats and to give names to the fields on panel designs. Formats may be defined as subfiles.

The source generated is standardized and includes useful combinations of the DDS keywords. You may optionally specify features such as message handling code, references to a field dictionary, and a print key file. Standards can be changed centrally using the Toolkit command Edit Design Defaults (YEDTDSNDFT).

A default field name is assigned to each field. You may specify override values for the field names, and also refer the fields back to reference fields in a data dictionary. Field references may either be to a field of the same name, or a different name, or to a name related by the CA 2E naming convention. The CA 2E naming convention provides a method of systematically assigning meaningful field names. See the *CA 2E Standards Guide* for further details.

Comparison with SDA

The Toolkit panel design aid should be viewed not as a substitute for IBM's Screen Design Aid, but rather as a precursor. The Toolkit utilities allow you to take a much more sketchy approach. Design can initially be done in outline; detailed information need only be added once you have decided the basics. Since less effort is involved in preparing designs, you have greater freedom to try out different alternatives in search of the best; and above all, to precede top down. The Toolkit utilities allow you to design a display then break it up into formats, whereas SDA requires that you build up the panel from already decided formats.

The facilities in Toolkit for presenting system designs to the user are far more extensive than those present in SDA: this highlights the fact that the Toolkit design tools are intended to operate at an earlier stage in the design process than SDA.

Using the Create Panel DDS

The following steps and illustrations give an outline example of using the Create Panel Design DDS command to generate DDS for a Toolkit panel design.

You enter the YCRTPNLDDS program, specifying the name of the panel design that is to be processed, and the name of the source file that is to receive the generated source:

The YCRTPNLDDS program displays the design image that has previously been entered using the Toolkit command Work with Panel Design (YWRKPNL).

When the format area has been marked out, F9 is pressed to add field details for the format.

For each format, press F09 to show a second display that allows you to specify the name of the format and the name of the fields on the format. If you have specified that the format is a subfile (F20), both the subfile control and subfile detail records are shown at the same time.

When you press Enter, you are prompted to confirm that the details are correct. When you confirm the prompt the source for the format is generated.

Use the available inquiry facility to browse the names of fields in existing database files. The field inquiry is invoked by specifying a **?** in any field name on the YCRTPNLDDS field detail display.

Repeat steps 2 and 3 for each format that is to be produced from the specified panel design. One special type of format, a program message queue subfile, is supported.

Refer to the following listing which shows sample DDS source as would be generated by the YCRTPNLDDS utility. In addition to generating code for the format and field details in the design, the YCRTPNLDDS program also generates all the appropriate supporting code and comments, such as a banner, keywords, and indicator text. Code for a program message queue has also been generated. The default values set up with the YEDTDSNDFT command have been used for the attributes of the code that can be standardized, such as the name of the field reference file, and indicator usage.

Using DDS PUTOVR Keyword

There are slight differences in the generated source depending on whether the use of PUTOVR was specified or not on the design defaults display (YEDTDSNDFT command). The example shown previously includes PUTOVR processing.

If the use of PUTOVR is specified, the following differences occur on all formats, except those of subfile records:

- The PUTOVR keyword will be conditioned by the indicator specified in the central design defaults. Otherwise it is conditioned by the indicator associated with the Help function key on the design defaults.
- For input only fields, the DDS OVRATR keyword will additionally be generated. It will be conditioned by the indicator associated with the Help function key on the design defaults.
- For output only and both fields, the DDS OVRATR and OVRDTA keywords will additionally be generated. They will be conditioned by the indicator associated with the Help function key on the design defaults.
- The Time field will be treated differently: instead of the DDS Time keyword being generated, an output field will defined instead. The field will be named according to the value on the design defaults. This is done because the DDS OVRDTA keyword cannot be used in conjunction with the DDS Time keyword.

This diagram shows the main interconnections between the displays of the Toolkit program Create Panel Design DDS.

Retrieving a Panel Design from DDS Source

A Toolkit utility is provided, invoked by the Toolkit command Retrieve Panel Design (YRTVPNLDSN), which can create a Toolkit panel design from the Database Description Specifications (DDS) of an externally described display file.

If the display file contains many formats, the individual display file formats are combined by overlaying: by default, all formats from a given member will be overlaid; the order of precedence for overlaying being the order in which the formats appear in the DDS source. To override which formats are overlaid, and in which order the overlaying is done, an additional Toolkit command Define Panel Design (YDFNPNLDSN) is provided which can be used to direct how the displays are combined.

Comment statements containing the YDFNPNLDSN command can be placed in the display file DDS which is to be retrieved: each statement can name up to ten display file formats that are retrieved as a single design. See the respective command diagrams in the *CA 2E Reference Guide* for further details.

Note that the Retrieve Panel Design facility does not retrieve information about individual field names.

The utility can be used in particular to document existing application systems.

Report Designs

Toolkit provides facilities for the rapid design of reports, the specification of how the report designs connect to menus, and the presentation of the report designs to the user.

Once finalized, the report designs can be converted into i OS Data Description Specifications (DDS), ready for programming.

Report presentation standards, for example, standard layouts and file attributes can be specified centrally. The standards will then be automatically implemented on all reports created; thus both reducing the amount of work required to specify a report, and also improving the standard of presentation throughout your application systems.

General Considerations for Report Designs

Before creating the report designs, consider the following commands you need to manipulate:

- Report designs
- Report design defaults
- Naming and storing conventions
- Report design components

Report Design Commands

The following commands manipulate Toolkit report designs.

Report Design Defaults

Report presentation standards may be specified centrally using the Toolkit command Edit Design Defaults (YEDTDSNDFT). Among the items that can be specified with this command are:

- The representation characters used to represent each type of field on report designs (O, 6). These are shipped as being the same as those used by the Screen Design Aid utility (SDA).
- Standard values used when generating print file DDS source, including the name of the field reference file, default print attributes (page length, width, CPI, LPI, and so on).

Naming Report Designs

Each report design is given a name: the designs are referred to in the Toolkit report designs commands by this name. Report design names must be valid System names (that is, begin with a letter and be up to ten characters long). For example:

```
YWRKRPT REPORT(FRED)
```

Storing Report Designs

Toolkit report designs are stored in database files. If you wish, when using Toolkit Report Design commands, you may also specify the name of the database file containing the report design. For example:

```
YWRKRPT REPORT(FRED) FILE(CHARLOTTE)
```

Default Names for Toolkit Report Design Files

The default name for the report design file on all Toolkit Report Design commands is YDSNRPT. It is recommended that you call all of your report design files YDSNRPT, and make use of your library list to distinguish between different versions of report designs, in the same way as you would use the i OS default source file names.

Creating Toolkit Report Design Files

Toolkit report designs are stored in special database file formats.

Files of the correct type can be created using the Toolkit command Create Design File (YCRTDSNF):

```
YCRTDSNF TYPE(*RPT) FILE(QGPL/YDSNRPT)
```

You may have different sets of report designs in different members in the design file. The Toolkit command Add Design File Member (YADDDSNFM) is provided to give you an easy method of adding additional members to design files. For example, to add a new member to report design file YDSNRPT in library QGPL:

```
YADDDSNFM TYPE(*RPT) FILE(QGPL/YDSNRPT) MBR(EXTRA) TEXT('More reports'
```

Report Design Components

There are three parts to a Toolkit report design:

- Title information
- Layout
- Narrative

Each report design may be one of three different widths:

- 80 Characters (A4)
- 132 Characters (Standard)
- 198 Characters (Big)

Report Design Title

For each report design, you may specify title information: this includes a title, the order in which the report design is to be printed, the report width, and whether the design uses a standard header for the top of the report.

Report Design Layout:

Report design layout information consists of the actual report image: special characters are used to indicate field.

Report Design Narrative:

For each report design, you may enter up to twenty-four lines of narrative information. This narrative text can appear under the report design when the design is printed. Printing is done using the Toolkit command Document Report Design (YDOCRPT).

Working with Report Designs

Toolkit report designs are entered and maintained with the Toolkit utility Work with Report Design (YWRKRPT). The report design utility is an interactive program with many functions to assist in report design, including windowing, line copy, line duplication, block copy, block move, and block save functions. See the Help text of the program for details of these facilities.

The Toolkit utility Work with Report Design is very similar to the Toolkit utility Work with Panel design. The Work with Report Design utility differs in having the added facility to window across a display that can be up to 198 characters wide.

Using a Default Report Design

Make most of your application reports conform to a standard layout that is to have items like the report title, and origination information in the same place on all reports. Doing so not only makes them look better, but also makes the application systems easier to understand.

Toolkit allows you to set up a default report design that will be provided as a starting position on any new report design that you add.

You can specify one default report design for each report width for each member in the report design file. For example, you could name the default report design, #DFTRPTnnn, where nnn is the report width, for example, #DFTRPT132, #DFTRPT198.

An example of the Work with Report Design display is:

```
YWRKRPT REPORT(ALSTSTKVAL) FILE(YDSNRPT)
```

Report title details are entered through a fixed format display.

Selecting 1 from the report title details display giving the report design image. The image of the report is laid out using a free format display.

Selecting a 2 from the title details display gives the text display.

The explanatory narrative for the report design is also laid out using a free format display:

Pressing F03 from any display causes the exit display to be shown. The exit display allows you to make a controlled exit from the Work with Report Design program. The design is not updated to disk until this point

Presenting Report Designs

The Toolkit command Document Report Design (YDOCRPT) will print a report design or specified range of report designs in any order. The designs are printed to a high quality standard, suitable for inclusion in a system specification. Narrative and index can be included.

```
YDOCRPT REPORT(*ALL) PRTINX(*YES) PRTCOM(*NO)
```

Manipulating Report Designs

Standard functions are available to do the following:

- YCRTDSNF - Create new report design file.
- YCPYRPT - Copy a report design.
- YDLTRPT -Delete a report design.
- YRNMRPT - Rename a report design.

A report design may be converted into a panel design, and vice versa, by means of the Toolkit commands Copy Panel Design (YCPYPNL) and Copy Report Design (YCPYRPT), using the CVTOPT parameter.

Selecting a Report Design

A select function is available to facilitate the use of report designs: the facility enables you to select from, or add to, a list of existing report designs; or to perform any of the report design manipulation functions on a selected report design. The select function is invoked by specifying REPORT(*SELECT) when using the command Work with Report Design (YWRKRPT), for example:

Generating DDS for a Report Design

The utility Create Report Design DDS (YCRTRPTDDS), creates the Database Description Specifications (DDS) for a print file directly from a Toolkit report design. The generate function is interactive, and allows you both to specify how a report design is to be broken up into formats, and to give names to the fields on each format. The source generated is standardized, and includes features such as a banner, and a reference to a field reference file.

The Create Report Design DDS utility is very similar to the equivalent Toolkit Create Panel Design DDS utility (YCRTPNLDDS).

1. You enter the YCRTRPTDDS program, specifying the name of the report design that is to be processed, and the name of the source file that is to receive the generated source:

YCRTRPTDDS REPORT(ALSTSTKVAL) SRCFILE(QDDSSRC)

The YCRTRPTDDS program displays the design image that has previously been entered using the Work with Report Design (YWRKRPT) command.

When the format area has been marked out, press F09 to add field details for the format.

For each format, press F09 to show a second display which allows you to specify the name of the format and the name of the fields on the format:

When you press Enter, you will be prompted to confirm that the details are correct. When you confirm the prompt, the source for the format will be generated.

Repeat the process for each format on the report design.

When the format area has been marked out, press F09 to add field details for the format:

1. The following listing shows sample DDS source as would be generated by the YCRTRPTDDS utility. In addition to generating code for the format and field details in the report design, the YCRTRPTDDS program also generates all the appropriate supporting code and comments. The default values set up with the YEDTDSNDFT command are used to obtain such attributes of the code as can be standardized, such as the name of the field reference file.

The following diagram shows the main interconnections between the displays of the Toolkit Create Report Design DDS program.

Retrieving a Report Design from DDS Source

A Toolkit utility is provided, Retrieve Report DDS (YRTVRPTDSN), to retrieve a Toolkit Report Design from the Database Description Specifications (DDS) for an externally described print file. Print file formats are combined in the order that they appear in the DDS source. The utility can be used to document existing application systems.

Chapter 4: Programmer Utilities

This chapter provides an overview of programmer aids including library lists, generic commands, debug aids, compile preprocessor, and edit aids.

This section contains the following topics:

Introduction

Toolkit provides a variety of utilities designed to help programmers. The programmer utilities include generic manipulation aids, compilation aids, edit aids, and debug aids.

Note that many of the other Toolkit utilities, notably the documentation and the menu design utilities, will also be of great use to programmers.

All of the Toolkit utilities are invoked with commands: the commands may be incorporated into your own programs, in particular, you may wish to make use of the list manipulation utilities in this manner.

Overview

The programmer aids are grouped into five main areas:

- List manipulation aids
- Generic manipulation aids
- Debug aids
- Compile preprocessor utility
- Edit aids

List Facilities

The concept of a library list will already be familiar to i OS users. Toolkit also introduces the concept of four other types of lists: object lists, data base file lists, member lists and format lists.

A list is a collection of item descriptions that can be built from existing descriptions of items, and then manipulated, stored, and used to drive other processing.

Toolkit list processing includes the following capabilities:

- Lists can be executed.

  Any CL commands that operate either on objects or on members can be invoked on lists of objects or list of members by means of the Toolkit Execute List commands: (YEXCOBJLST, YEXCMBRLST, YEXCDBFLST). Non-generic commands can be made to work generically, and generic commands can be made to work on selections of items.

- Lists can be stored indefinitely.
- Lists can be edited interactively.
- Lists can be trimmed, either interactively, or in batch.
- Lists can be combined as sets.
- Lists can be converted from one type to another.
- Lists can be transferred from machine to machine.

Using List Processing

Toolkit's list processing capabilities give you significant leverage: many instructions can be executed as the result of a single request. For example:

To move a selected list of objects from one library to another you could:

- Create a list of objects using the Toolkit command Build Object List (YBLDOBJLST).
- Remove the ones you do not want using the Toolkit command Edit Object List (YEDTOBJLST).
- Use the Toolkit command Move Object (YMOVOBJ) to move the objects remaining in the list.

To stop and restart journalling a group of files (perhaps so as to carry out a physical file reorganization using the Reorganize Physical file command (RGZPFM). You could:

- Create a list of the files, using the Toolkit command Build Database File List (YBLDDBFLST).
- Trim the list if necessary, using the Toolkit Edit commands Database File List (YEDTDBFLST), the Filter Database List (YFLTDBFLST), or both.
- Then use the Toolkit command Execute Database File List (YEXCDBFLST), to stop journalling, reorganize the files, and then restart journalling.

If you expand your application, and wish to have additional files journalled, all you have to do is amend the list of files.

To recompile a group of programs that all reference a given program, you could:

- Use the Toolkit utility Scan Source (YSCNSRC) to build a list of the source members of the programs that call the program.
- Change the source, making use of the Toolkit Edit command Member List (YEDTMBRLST), to present each source member in turn for editing with the i OS utility Start SEU (STRSEU).
- Then use the Toolkit command Create Objects (YCRTOBJ), to recompile all of the changed source members.

List Processing and Existing Commands

Here are some other ways in which list processing can be used in conjunction with existing commands:

Housekeeping:

- RMVM: To remove obsolete work members.
- CRTDUPOBJ: To replicate a list.
- CHGPF: To change the attributes of a list of files.
- CLRPFM: To clear a list of physical file members.
- RGZPFM: To reorganize a list of physical file members.

Secofring:

- V. to secofr - to perform a security officer's duties
- YMOVOBJ: To install new programs, or replacements preserving any existing authorities.
- CHGOBJOWN: To change ownership generically.
- DSPDEVD: To list all the device descriptions.

Documentation:

- YSCNSRC: To look for the occurrence of a string.
- DSPDTAARA: To print the contents of a list of data areas.
- DSPDEVD: To print all or some device descriptions.
- DSPJOBD: To print the contents of all the job descriptions.
- DSPUSRPRF: To print the attributes of all user profiles.

Programming:

- STRSEU: To present a list of source members for editing.
- CHGCMD: To change the attributes of a list of commands.
- CHGPGM: To optimize programs.
- ALCOBJ: To reserve a list of objects.
- DLCOBJ: To release a list of objects.
- CHGDSPF: To change display file attributes.

Generic Manipulation Commands

Although it is possible to execute almost any command generically using the list manipulation utilities, certain frequently used commands have, for convenience, been given their own generic versions as part of Toolkit.

In particular, this has been done for commands to move objects and source. Where development and maintenance is carried out in a different library from the live system - as indeed it always should be - there is a common requirement to move lists of new versions of programs, and their source, into the production environment.

Generic move commands are provided within Toolkit; the commands also include additional facilities such as the keeping of a record of what has been moved, the archiving of previous versions of the object, and the preservation of object authorities.

Debug Aids

Toolkit includes utilities that can greatly speed up your testing and debugging:

- A utility to change the data on any database file directly, without creating an HLL program, or DFU. Add, update, copy, delete, positioning, and formatting functions are available.
- A utility to set up debug sessions from directives contained in the source. This gives you the ability to re-instate rapidly a complicated debug environment, after recompilation or interruption. Breakpoints can be specified relative to executable statements, rather than as absolute source line numbers, and several different debug sets can be maintained.
- A utility to take a snapshot of the contents of a list of files. The snapshots can be used to establish check points for system testing, and for recovery.

Work with Database Files Utility

The Toolkit utility Work with Database File utility (YWRKF) enables you to display and change the data on any database file - interactively. It includes the following features:

- Ability to change, add, delete, or copy records.
- Ability to work on single or multi-format files: formats may also be selected within a multi-format logical file.
- Positioning functions
  - If the file has a keyed access path, the display may be repositioned using the key fields.
  - If the file has an arrival sequence access path, the display may be repositioned using the relative record number.
- Ability to specify select/omit criteria, using a separate display
- A two level display:
  - A record level display which enables you to scan through records quickly, and to compare records. Fields may be dropped from the record level display.
  - A field level display that enables you to examine all the fields for a record.
- Alternate field labels - You may specify that fields are to be described either with their DDS field names or their DDS column heading text.
- File, format, and field descriptions.
- Hexadecimal displays. Change values may also be specified in hexadecimal.
- A confirm prompt. When you press Enter after having entered new values for one or more fields, a confirm prompt will be displayed. You must answer the prompt with a Y before the program will update the database. You may switch this facility off or change the default from Y to N if you wish.

YWRKF Example

Examples of the YWRKF displays are given over the next few pages. The first display would be obtained by entering the Toolkit command Work with File upon a nominated file, for example:

```
YWRKF FILE(QGPL/AARPGNL1)
```

The following represents the Work with File - Multi record display.

The following represents the Work with File - Single record display.

Select a single record for update with selection option **5.**

The following represents the Work with File - Extended Field display.

Pressing F17 on a field gives the Extended Field display:

The following represents the Work with File - File/Format Details display.

Pressing F14 causes the following display of format and field definitions to be shown:

The following represents the Work with File - Prompt format display.

You may use YWRKF upon a multi-format logical file, but only one format can be keyed on or added to at a time. You may specify which format is the current format using the following display, which is obtained by pressing F05.

The following represents the Work with File - Select details display.

Pressing F07 on the top level display gives the following display, which allows you to specify select/omit criteria for records.

The following represents the Work with File - Prompt key display.

Pressing F04 allows you to specify positioning criteria for records using the following display:

Note that this screen is only available if the file you are working with is indexed.

The following diagram shows the main interconnections between the displays of the Work with File program:

The following represents the YWRKF - Display connection map.

Start Debug Utility

The Toolkit utility Start Debug (YSTRDBG) enables you to set up a debug session for a program using directives stored in the program's source.

- Identical debug sessions can be entered repeatedly. If you alter the source for a program and recompile it, the line sequence numbers for your breakpoints will automatically be adjusted.
- Many different debug sets are allowed. A debug set consists of one or more breakpoints to be inserted according to the source directives.
- Variables can be specified for display at each breakpoint.

Refer to the YSTRDBG command diagram for details about how to specify breakpoint sets.

The Toolkit command Start Debug helps you to program in an efficient top down manner. Sets of debug directives can be placed at strategic points in a new program as a routine part of your programming. Each debug set can reflect a different control level in your program, for example, main control, key validation, detail validation, update routines. You can then make use of the debug facilities for routine testing of the program and for problem shooting later on.

Utility to Copy a List of Database Files

The Toolkit utility Copy Files (YCPYF) will copy a list of database files. This enables you to take synchronized snapshots of the contents of all the files in a database, that is, to record the overall position. Snapshots are of particular use in the later stages of system testing, when you may want to track the updating of many files through a succession of update stages. The Copy File utility can also be used to restore the contents of a snapshot back into the database: this technique can be of use both to recover from a crash, and to re-run a test to establish that a fix has worked.

- Many snapshots can be kept in the same library**.**
- Journalling is suspended while restoring.

Techniques for Preparing Test Data

The snapshot technique can be used to complement two other techniques of preparing test data on the i OS:

- Use of test pack libraries: you will probably keep one or more data libraries, each containing a complete set of all of the application files required by a system: these alternative libraries can be used in place of the library of live files when testing. Such a library will typically contain the physical files and all necessary dependent logical views. Creating or restoring such a library may be fairly cumbersome and slow, as it must either be done via an off-line media, or by using the i OS command Create Duplicate Object (CRTDUPOBJ**)** in a controlled manner that is on physical files before logical files.
- Use of journalling. Journalling may be of use of the immediate unravelling or tracing of errors. It is of limited use if there has been significant subsequent update of the database.

The Toolkit utility Copy File simplifies the process of creating test data: you need to keep only one or two complete sets of test files. Repeated snapshots can be taken of the contents of selected physical files: you can specify which files by use of the Toolkit facilities Create List and Edit List. The snapshots constitute a full description of the data in the library for the purposes of recovery.

The files within each snapshot can be given an identifying prefix. This allows you to store many snapshots of a given file in the same library.

Suspension of Journalling

The Toolkit utility Copy File will stop the journalling of a physical file before replacing its contents: this makes the copy operation run faster. You may specify that journalling is to be restarted by the YCPYF utility after the copy has finished.

Setting a Break Program

The Toolkit command, Set Break Program (YSETBRKPGM), creates a message queue in the QTEMP library of the invoking job, and sets a break program to receive and execute any messages sent to the queue.

Option five on the system request display (which invokes the i OS command Send Message (SNDMSG) can thus be used to send any arbitrary request string to the message queue at any point during the execution of an interactive job.

This facility can be used, for example, to start debug for a given program when you are already executing the program, or to add additional break points, to call the i OS command Entry Program (QCMD) while executing a program.

Utility to Display a Program's Message Queue

The Toolkit command Display Program Message Queue (YDSPPGMQ) displays the messages on the program message queue of an active program. For example, to display the messages from the YINLPGM program:

```
YDSPPGMQ PGM(YINLPGM)
```

Compile Preprocessor

An aspect of i OS system documentation that is often overlooked is the recording of the object attributes that are specified as compiler overrides on the various create commands (CRTRPGPGM, CRTCLPGM, CRTPF, and CRTDSPF). This information, which in some cases is essential for the successful use of an object, is stored on the object description, and can easily be lost if the object is recompiled.

Here are some examples:

- For a database file - the maximum number of members
- For a program - the profile that the program is to run under
- For a command: the name of the command processing program, and the modes under which the command may run

Toolkit provides a utility, the Compile Preprocessor that ensures the information is not lost; instead it is stored in the source of objects and is automatically added when compilation takes place. The utility also provides several other extremely useful functions:

- Cancellation of previous compilation listings (thus the default listing retrieved by SEU is always the latest version).
- Storing of compile time overrides.
- Synchronization of object text, source titles, and source member text
- Suppression of non-essential cross-reference listings

Overview

The compile preprocessor is a program that can be automatically invoked to run as a preliminary step on batch compiles. It has 256 Compiler cross-reference independent options to:

- Cancel any previous spool file listings. This ensures that only the latest compilation listings are retained, thus making the use of the i OS facility Source Edit Utility (STRSEU) for browsing compilation listings both quicker and easier**.**
- Synchronize source member text, object text and source title. You can ensure that objects and members have the same description.
- Invoke CL command(s) prior to compiler execution. A compilation environment can be created within the batch job. For instance, file overrides can be supplied if the name of a file required by a program differs from the name used in the program, or pre-requisite objects such as data areas can be created in the QTEMP library.
- Add compiler options from source member lines**.** The i OS shipped system requires that any compiler options be entered by hand for each compilation. This process is both time consuming and prone to error; and is especially a problem with maintenance programming done some time after the original development. If objects are recompiled without the correct overrides, the consequences can be far reaching. This facility enables these compiler directives to be stored in the source, and automatically re-applied whenever the object is compiled - and because the directives are applied automatically, it becomes possible to carry out generic recompilations of whole systems, or parts of systems, correctly.
- Suppress XREF listings. Compilations run significantly faster if cross-reference listings of program variables are not producti ve.

Enhanced functionality within the compile preprocessor

The compile preprocessor now allows greater functionality and more flexibility when compiling objects.

The following additional functionality has been included:

Preprocessor directive limit increased to 50

The total number of compile preprocessor directives allowed in a single source member has been increased to 50. This limit includes pre-compilation and post-compilation commands (Y* directives), compilation overrides (Z* directives), source member text overrides (T* directives), exit program calls (P* directives – see below) and external source member definitions (X* directives – see below).

Wider range of substitution variables

The following substitution variables are now valid in compile preprocessor directives:

&A - Source member attribute, e.g. RPGLE
&B - *OBJ (hard coded for backwards compatibility)
&C - Compilation command, e.g. CRTBNDRPG
&D - Source change date in MDY format, e.g. 032105
&E - Current source member (may be external X* member)
&F - Source file name, e.g. QRPGLESRC
&G - Job name, e.g. QPADEV0003
&H - Job number, e.g. 123456
&I - Job user, e.g. HEWRO01
&J - Qualified job description, e.g. MYMDLLIB/QBATCH
&L - Source library, e.g. MYGENLIB
&M - Source member, e.g. UUAJEFR
&N - Object name, e.g. UUAJEFR
&O - Object name, e.g. UUAJEFR
&Q - Qualified job, e.g. 123456/HEWRO01/QPADEV0003
&S - Object type without '*', e.g. PGM
&T - Object type, e.g. *PGM
&X - Source member text
&Y - Compilation date in MDY format, e.g. 032105
&Z - Compilation timestamp e.g. 05-03-21-12:13:45:00000

Separation of pre- and post-compilation commands

Pre-compilation commands and post-compilation commands can be separated through the use of one or more Z* (compilation override) directives.

Note: Due to backwards-compatibility, directives can be in standalone form or in block form. Details of these forms and their different functionality can be located within the 'Integration with CA 2E using the EXCUSRSRC section.

Any Y* or P* directives which appear before the first Z* directive will be processed before the source member is compiled. Any Y* or P* directives which appear after the first Z* directive will be processed after the source member is compiled. If multiple Z* directives appear in the source member separated by other compile preprocessor directives, the first Z* directive will determine the separation of pre-compilation and post-compilation commands (but all subsequent Z* directives will be processed as part of the compilation command).

If both pre-compilation commands and post-compilation commands are required but no compilation overrides are required, a blank Z* directive (a directive containing only a Z* or a /*Z:) or a Z* directive containing only the compilation command itself can be used to separate pre-compilation commands and post-compilation commands, e.g.:

```
Y* SNDMSG MSG('About to compile &N...') TOUSR(*REQUESTER)

Z*
Y* SNDMSG MSG('Compilation completed!') TOUSR(*REQUESTER)
```

or

```
Y* SNDMSG MSG('About to compile &N ...') TOUSR(*REQUESTER)
 Z* CRTBNDRPG
Y* SNDMSG MSG('Compilation completed!') TOUSR(*REQUESTER)
```

In this case, the compile preprocessor will recognize that the command in the second Y* directive should be executed after the compilation command has been executed.

Both pre-compilation and post-compilation commands can contain the full range of substitution variables.

Exit program call functionality

Users can define their own 'exit programs', to be automatically called before or after the source member is compiled. Exit programs are invoked through a new preprocessor directive identifier – the P directive.

In a fixed-format source member (such as RPG, COBOL or DSPF) an exit program preprocessor directive will have the following format:

```
P* [library-name/]program-name
```

In a free-format source member (such as CL), an exit program preprocessor directive will have the following format:

```
/*P: [library-name/]program-name */
```

If the library name is not specified, *LIBL is assumed.

All exit programs have the same parameter format, as follows:

1. Compile command
2. Source member
3. Source file
4. Source library

For instance, if the following compile preprocessor directive is found in an RPGLE source member called MYPGM in file QRPGLESRC in library MYLIB which is being compiled with the CRTBNDRPG command:

```
P* QGPL/EXITPGM
```

then the following command will be executed as part of the compilation process:

```
CALL PGM(QGPL/EXITPGM) PARM('CRTBNDRPG ' 'MYPGM     ' 'QRPGLESRC ' 'MYLIB     ')
```

Note that if an exit program is called before the source member is compiled, it can make changes to the source before the source member is compiled.

It is the responsibility of the user to create and test exit programs thoroughly before deploying them in a production environment.

Global control data area YBRTPXA

A new data area called YBRTPXA is shipped in the product. It is a 42 byte character data area with the following format:

Bytes 1 – 10: Pre-compilation exit program name (default: '*NONE')
Bytes 11 – 20: Pre-compilation exit program library (default: blank)
Bytes 21 – 30: Post-compilation exit program name (default: '*NONE')
Bytes 31 – 40: Post-compilation exit program library (default: blank)
Byte 41: Cancel compilation if error ('1'/'0') (default: '0')
Byte 42: Currently unused (default: blank)

If a global pre- or post-compilation exit program is specified, it must have the parameter format as defined in the "Exit program call functionality" section above) which will automatically be called during the compilation process. If the exit program library is not specified, *LIBL will be used.

The global pre-compilation exit program (if specified) will be called before any exit programs defined using a P* directive and the global post-compilation exit program (if specified) will be called after any exit programs defined using a P* directive.

If byte 41 in the YBRTPXA data area is set to '0' (the default), then if the compile preprocessor encounters an error, the source member will be compiled using the system defaults, with all compile preprocessor directives ignored - this is the default that has been used in previous versions of the compile preprocessor. However, if byte 41 is set to a value of '1', an error in the compile preprocessor will cause the compilation to end immediately. This setting allows the user to immediately identify any problems with, for instance, invalid compile preprocessor directives.

Using the YBRTPXA data area to specify global pre- and post-compilation exit programs in this way allows users to perform automatic source manipulation (such as the addition of color to directives in the source) prior to compilation, without needing to specify a P* directive in every source member.

Use of external source members to hold preprocessor directives

Users can define their own external compile preprocessor source members, which can be used to hold global compile overrides or other compile preprocessor directives. External preprocessor source members are identified with a new preprocessor directive identifier – the X directive.

In a fixed-format source member (such as RPG, COBOL or DSPF) an external source member directive will have the following format:

```
/*
X* [[library-name/]file-name,]member-name
/*
```

In a free-format source member (such as CL), an exit program preprocessor directive will have the following format:

```
/*X: [[library-name/]file-name,]member-name */
```

If the library name is not specified, *LIBL is assumed. If a file name is not specified, the current source member file name is assumed.

For instance, a user could create a source member called RPGDFTOVR with the following directives in it:

```
Z* TGTRLS(V5R1M0)
 Z* DBGVIEW(*LIST)
 Z* OPTION(*NODEBUGIO)
```

and place it in file QRPGLESRC in library QGPL. Then, in all RPGLE source members, the user could simply have the following directive:

```
/*
X* QGPL/QRPGLESRC,RPGDFTOVR
/*
```

and when the program is compiled, the compile preprocessor will retrieve the Z* compilation overrides from the RPGDFTOVR source member and include them in the compilation command.

Any compile overrides specified in an external source member may themselves be overridden by compile overrides specified further down in the source member itself (or in a different external source member which is defined further down in the source member being compiled). So for instance, if the following preprocessor directive directives are coded into an RPGLE source member:

```
/*
X* QGPL/QRPGLESRC,RPGDFTOVR
Z* TGTRLS(V5R2M0)
/*
```

then the source member will be compiled using TGTRLS(V5R2M0), since the Z* directive is after the X* directive.

An external source member can itself contain X* directives pointing to 'nested' external source members (there is a limit of 100 nested source members).

External source members can also contain Y* directives, to perform pre-compilation or post-compilation tasks. For instance, if a user has a number of modules MOD1, MOD2 and MOD3, which are bound into a single service program SRVPGM1, they could have single source member called e.g. SRVPGM1 which contains the CRTSRVPGM command to create the service program, e.g.:

```
Y* CRTSRVPGM SRVPGM(&L/SRVPGM1) +
Y* MODULE( MOD1 MOD2 MOD3) +
Y* TEXT('Service program 1')
```

In each of the modules MOD1, MOD2 and MOD3, the user can include the following directive anywhere after the first Z* directive:

```
/*
X* QGPL/QRPGLESRC,RPGDFTOVR
X* QGPL/QRPGLESRC,SRVPGM1
/*
```

and whenever any of MOD1, MOD2 or MOD3 are recompiled, the module will be compiled using the defaults in the RPGDFTOVR external member and then the SRVPGM1 service program will also be automatically recreated to include the changed module.

Integration with CA 2E using EXCUSRSRC

Prior to this rewrite of the compile preprocessor, Y* directives specified in EXCUSRSRC were automatically inserted into the generated source of functions which call the EXCUSRSRC function prior to any Z* directives (i.e. as pre-compilation commands). In addition, the RPGIV generator allowed users to specify X* directives in EXCUSRSRC, which were automatically inserted into the generated source of functions which call the EXCUSRSRC function as Y* directives after any Z* directives (i.e. as post-compilation commands).

The following should be noted when including compile preprocessor directives in EXCUSRSRC within CA 2E

1. All generators have now been standardized for backwards-compatibility so that any standalone X* directives in EXCUSRSRC will be converted into Y* directives that are inserted into the generated source of functions which call the EXCUSRSRC function after any Z* directives (i.e. as post-compilation commands). Standalone Y* directives continue to be inserted into the source before any Z* directives (i.e. as pre-compilation commands). This only applies to existing stand-alone X* compile preprocessor directives.
2. To use the new compile preprocessor directive types (X* and P*), they must be in a block surrounded by directives which begin with '/*', e.g.:

```
/*
compile preprocessor directives
/*
```

   The /* directives can contain directives, e.g. '/* Start of preprocessor block' but will not be generated into the source. Only a single /* directive should start and end the preprocessor directive block.

   Directives within a preprocessor directive block are copied into the final source member prior to any Z* directives, until a Z* directive is found in the preprocessor directive block. If a non-blank Z* directive is found, it is inserted into the source with the default Z* directives generated into all 2E functions. If a blank Z* directive is found, it will not be generated into the source, but it will be used to delimit pre- and post-compilation commands (see the section on "Separation of pre- and post-compilation commands" for more details). All subsequent preprocessor directives are inserted into the final source member after any Z* directives.

1. Any P* (exit program call) directives will be ignored if they are found outside a preprocessor block. P* directives are only valid within a preprocessor directive block.

   For instance, if an EXCUSRSRC function contains the following code:

```
/* Start of preprocessor directive block
Y* SNDMSG MSG('About to compile...') TOUSR(*REQUESTER)
 X* QRPGSRC,DOCSRC1
P* QGPL/PREPROCRPG
Z* USRPRF(*OWNER)
 P* QGPL/PSTPROCRPG
X* QRPGSRC,DOCSRC2
Y* SNDMSG MSG('Compilation completed!') TOUSR(*REQUESTER)
 /* End of preprocessor directive block
```

Then the final code that would be seen in the source of a function that calls this EXCUSRSRC would be as follows:

```
...
 Y* SNDMSG MSG('About to compile...') TOUSR(*REQUESTER)
 X* QRPGLESRC,DOCSRC1
P* QGPL/PREPROCRPG
Z* <default-2E-compile-overrides>
Z* USRPRF(*OWNER)
 P* QGPL/PSTPROCRPG
X* QRPGLESRC,DOCSRC2
Y* SNDMSG MSG('Compilation completed!') TOUSR(*REQUESTER)
...
```

The first three directives of the EXCUSRSRC (not including the starting '/*' directive) have been inserted into the source. Next come the default Z* directives that are generated for 2E functions, followed by the Z* directive from the preprocessor directive block. Finally, the last three directives in the preprocessor block (not including the ending '/*' directive) are inserted into the source.

General Considerations for the Compile Preprocessor

This topic contains information on invoking and installing the compile preprocessor, routing required data, switching settings to control the preprocessor, and CL syntax.

Invoking the Compile Preprocessor

To have the compile preprocessor invoked on a compilation request, the following conditions must be met:

- The preprocessor utility must be installed in the subsystem in which the compile is taking place. This is done by adding a routing entry to the subsystem.
- The submitted job must have the correct routing data, and also the switch settings to invoke the desired options. Any combination of options may be selected via the job switch settings.
- For the options specifying extraction of directives from the source, the directives must have been coded in the source in the correct format (see Source Directives below).

Installing the Compile Preprocessor

To install the utility in a subsystem, do the following:

1. Terminate subsystem.

   ENDSBS SBS(subsystem-name)

2. Add the following routing entry (An alternative sequence number can be specified if 1111 is already in use).

   ADDRTGE SBS(subsystem-name) SEQ(1111) CMPVAL('YCRTOVR') PGM(YBRTPRC)

3. Restart the subsystem.

   STRSBS SBS(subsystem-name)

The installation only has to be done once per subsystem. Thereafter all jobs submitted to that subsystem with the correct routing data will invoke the preprocessor. The preprocessor can be installed in as many subsystems as desired.

Preprocessor Job Entries

Whether the preprocessor is invoked for a particular job is controlled by the job's routing data. Which preprocessor options are run depends on the job's switch settings. Both of these are described below.

Routing Data Required to Invoke the Pre-processor

Once the utility is installed in a subsystem, the preprocessor runs on all compilation requests (that is, CRTRPGPGM, CRTCLPGM, CRTPF etc.) submitted to the subsystem with the routing data YCRTOVR. Routing data can either be specified on the SBMJOB, TFRJOB or RRTJOB commands, or more conveniently on the job description. For example:

To submit a job that invokes the preprocessor:

```
SBMJOB RTGDTA(YCRTOVR) RQSDTA('CRTRPGPGM DEMO')
```

To transfer a job and invoke the preprocessor:

```
TFRJOB RTGDTA(YCRTOVR) RQSDTA('CRTRPGPGM DEMO')
```

To reroute a job:

```
RRTJOB RTGDTA(YCRTOVR) RQSDTA('CRTRPGPGM DEMO')
```

To change a job description so that all compilations using the job description invoke the preprocessor:

```
CHGJOBD JOBD(QBATCH) RTGDTA(YCRTOVR)
```

The routing data on a job description only has to be changed once. When submitting jobs with the SBMJOB command change the routing data parameter to *JOBD. This will ensure that the routing data defined in the job description is used by the SBMJOB command and the preprocessor will be invoked. This is particularly convenient if compilations are submitted using the i OS Programmer's Menu.

Switching Settings

The options for the preprocessor are controlled by setting the SWS parameter on the job as follows:

- Switch 1 off: SWS(0XXXXXXX**)**. Remove old compilation listings of the object being compiled.
- Switch 1 off, Switch 4 on: SWS(0XX1XXXX). Remove old compilation listings of the object being compiled, but only move those belonging to the current user profile.
- Switch 2 off: SWS(X0XXXXXX). Process source file directives.
- Switch 3 off: SWS(XX0XXXXX). Suppress XREF listings.

The normal default, if no switches are set, is thus that all features will be invoked.

Switches can be set either using the job description or the SBMJOB command.

CL Syntax

- The compile preprocessor can handle CL statements in either i OS or S/38 syntax. It examines the request string to determine the syntax and then If the request string contains qualified names specified in S/38 syntax, that is the form OBJ.LIB, or S/38 keywords, PUBAUT, the string will be deemed to be in S/38 syntax and the request string will be rerouted with routing data QCMD38 which will invoke the i OS Environment request processor QCL. For example:

CRTRPGPGM PGM(FRED.QGPL) SRCFILE(QRPGSRC.QGPL)

- If the request string contains qualified names specified in i OS syntax, for example the form LIB/OBJ**,** or i OS keywords (that is, AUT, REPLACE), the string will be deemed to be in i OS syntax and the request string will be rerouted with routing data QCMDB which will invoke the QCMD request processor. For example:

CRTRPGPGM PGM(QGPL/FRED) SRCFILE(QGPL/QRPGSRC)

Source Directives

Instructions for the compile preprocessor may be coded in source, using special types of comment lines. These instructions are referred to as source directives.

Types of Source Directive

Pre-compile/Post-compile directives can be of three types: Title directives (T), Pre-compile directives (Y), and Compiler directives (Z).

A *Title* directive specifies the text to be used as object text on the compiled object. The text on the source member will also be updated to match that on the object. If the source contains no title, the source member text, if present, will be used instead. If text is specified using the text parameter on the programmers menu (QPGMMENU), that text will be used in preference to the source member text of title text.

Note: Title directives, when specified within EXCUSRSRC functions, do not get generated into the source. This is a feature limitation.

Pre-compile/Post-compile directives specify command requests to be executed before, or after the actual compile is run, or both. The main application of this is to have file overrides in effect during a batch compile, so that an object can be compiled using an existing file: for instance, OVRDBF FILE(X) TOFILE(Z/Y). This might be desirable for say, an RPG III program which doubly defines a file, once as update, once as input. Instead of having to have either a dummy copy of the file with a different name, or to carry out the compile interactively, the required override can be stored in the source. Another possible application is to override a message file at compile time. Yet another could be to create a data area necessary for the compile in library QTEMP. If a Pre-compile/Post-compile directive is placed before the compiler directive, its command request will be executed before compilation; if placed after the compiler directive, its command request will be executed after compilation. The following substitution symbols may be used in post-compile directives:

- &L - Library into which object is compiled.
- &O - Name of object compiled.
- &T - Type of object compiled (for example, *PGM, *CMD)

Compiler directives specify extra parameters to be added to the actual create command string. For example:

- On CRTPF and CRTLF: MAXMBRS (*NOMAX), AUT(*ALL), SHARE(*YES) MAINT(*REBLD), SIZE(*NOMAX), RECOVER(*AFTSTRCPF), LVLCHK(*NO)
- On CRTDSPF: RSTDSP(*YES), DFRWRT(*YES)
- On CRTPRTF: PAGESIZE, LPI, etc.
- On CRTRPGPGM: IGNDECERR(*YES), USRPRF(*OWNER)
- On CRTCLPGM: USRPRF(*OWNER), LOG(*NO)
- On CRTCMD: PGM(PGM), MODE(*BATCH), ALLOW(*BPGM), VLDCKR(CKR), PRDLIB(PRODLIB)

Coding Source Directives

Directives may start immediately after the control character. Title text will be left justified automatically. Note that Y* lines must appear sequentially before Z* lines in the source code.

Source directives in RPG/400 source

Example

Source directives in DDS source

Example

Source directives in CL and CMD source

Example Restrictions

Any source directives for the compile preprocessor must appear within the first twenty lines of the source member.

Using Source Directives to Duplicate Objects

Most application systems will have a certain number of objects, notably files, which are created directly from existing objects by using the command Create Duplicate Object (CRTDUPOBJ), or by the command Copy File (CPYF). The creation of the objects should be recorded as part of the system documentation, so that if the based-on object is changed, the duplicate object can be correctly re-created. One method of doing this is to use a spool reader. Another more flexible approach is to set up a source member for the duplicate object containing documentation and, instead of source, references in the *****Y and *****Z overrides to the based-on object.

Edit Aids

Toolkit Edit Aids include several small, but very useful, aids that help you to program faster, and more accurately. These include:

- A command to rename an object and its source with one instruction
- A command to create a set of source files with one instruction
- A command to search source for the occurrence of a specified search string, or combination of search strings
- A command to search source for the occurrence of a specified search string, and to replace all found instances with another string
- A command to compare source members and list the differences
- A command to retrieve, display and change a data area's contents
- A command to retrieve, display and change a message description
- An abbreviated command for common display functions.
- A command to add a source member of a given type to a source file

Display and Change Commands Manual:

Toolkit contains several interactive aids that can help you by allowing you to examine and override existing values. The aids will retrieve any existing values for the entity that is to be changed, and then display the values as the defaults in the appropriate change command. All you then need do is to override any value, or part of a value, that you wish to change.

Commands exist to retrieve and change:

- Message descriptions: (YEDTMSGD)
- Data area contents:

  -To change any data area: YEDTDTAARA

  -To edit the LDA: YEDTLDA

  -To edit the GDA: YEDTGDA

Message Manipulation Commands

This topic explains the retrieve message file and copy message description commands.

Retrieve Message File Command

The Toolkit command Retrieve Message File (YRTVMSGF) allows you to retrieve a message file into a CL source member. The CL source member will contain the CL instructions necessary to add, change, or remove messages (or a range of messages) from the message file.

This utility can be of use:

- When you need to translate messages into other National languages: it is generally easier to carry out the text editing of messages using SEU rather than the command entry display. This is especially true for IGC versions of i OS, when ideographic message text must be entered using SEU.
- When merging or reconciling messages files, for instance when development has been taking place on two different machines

Refer to the command diagram for an example of the CL which is generated by the YRTVMSGF command.

Copy Message Description Command

The Toolkit command Copy Message Description (YCPYMSGD**)** allows you to copy a message description, either from one message file to another, or to another identifier within the same file. For example:

```
YCPYMSGD FROMMSGID(CPF6801) TOMSGID(USR0001)FROMMSGF(QCPFMSG)
TOMSGF(QGPL/QUSRMSG)
```

Source Manipulation Commands

As well as the object and source manipulation facilities provided by the Toolkit list and generic commands (for example, YMOVOBJ, YMOVM, YRMVM), Toolkit includes aids to carry out six commonly required functions:

- Rename an object and its source (YRNMOBJSRC).
- Create a set of source files in a given library (YCRTSRCPF).
- Scan source (YSCNSRC). A source file member, or members, indicated by a generic name or a Toolkit member list, can be scanned for the occurrence or absence of a search string, or combination of search strings. The output can either be in the form of a report, or be a result list specifying the members containing the string, or both. The list output is especially useful if you wish to subsequently process all items containing the string.
- Compare source (YCMPSRC). A source file member, or members, indicated by a generic name or a Toolkit member list, can be compared against another member or members. A listing of the differences may be produced. If many members are compared a member list may be created of the members which differ or which match.
- Scan and replace source (YSCNRPLSRC). A source file member, or members, indicated by a generic name or a Toolkit member list, can be scanned for the occurrence or absence of a search string, and any found occurrences replaced by another specified string.
- Tidy RPG/400 source (YTDYRPGSRC)**.** The start and end statements of RPG/400 structured programming constructs can be documented automatically.

The following represents the Scan Source prompt.

The above example would scan through all source files whose names begin with the letter Q in library QGPL for occurrences of the string CALL FRED anywhere in columns one to eighty. It would produce a report of every line containing the specified string, as well as a member list of the names of the source members containing the string.

Source Conversion Commands

This topic provides information on the following Toolkit utilities and commands:

- Converting DDS command key usage utility
- Translating with a translation table commands
- Converting source for ideographic support utility

Converting DDS Command Key Usage

The Toolkit utility Convert DDS Command Key Usage (YCVTDDSCKY) can help you to convert your existing DDS to conform to SAA standards. The utility converts DDS source for display files. The utility includes the following features:

- Command key usage is mapped according to the values you specify in a table. For instance you could specify that CF01 should be mapped to CF03, CF03 to CF01. The resulting indicator to be set when the command key is pressed is not changed, so the programs which used the display file will not need altering.
- Mapping of a given key may be fixed - to a specified command key - for example, CF01 to CF03, or variable - to use the first available unused command key.
- As an optional feature you may specify that an attempt be made to convert the command key explanation text, also according to the table, for example, from CMD: 1-Exit to F3= Exit.
- You may also specify that text leaders be converted to SAA standards: any instances of dot leaders ('...:') will then be converted to the SAA standard ('. . :').

The actual mapping is specified by a table which you may edit using an interactive program invoked by the Toolkit command Edit Command Key Conversion Table (YEDTCKYTBL).

Refer to the respective command diagrams for information about the table values, and for examples of converted source.

The following diagram represents the use of DDS conversion utilities.

Translating with a Translation Table

Two Toolkit commands are provided which allow you to translate the contents of database files using a specified translation table. The translation table may either be system-supplied (for example, QSYSTRNTBL or QCASE256) or user-defined by means of the i OS command Create Table (CRTTBL). These commands may be of use:

- When preparing an upper-case only version of an application, for instance to run under the ideographic version of i OS.
- When converting data to run under a different multinational character set.

Translate Physical File Command

The Translate Physical File (YTRNPF) command will translate the data in all the alphanumeric fields in a database file or files.

For example, the following would convert all data fields in Y1MNU to upper case:

```
YTRNPF FILE(Y1MNU) TRNTBL(QCASE256)
```

Translate Source File Command

The Translate Source file (YTRNSRCF) command will translate all the source data in a source file or files.

For example, the following would convert all source files in library QGPL to upper case:

```
YTRNSRCPF FILE(QGPL/Q*) TRNTBL(QCASE256)
```

The command can also be used to strip out Text Management/38 attributes and comments from a source member.

Converting Source for Ideographic Support

A special utility is provided to assist with the preparation of applications to run on ideographic machines. The utility enables you to automatically convert DDS source to run under an ideographic version of i OS, that is, with DBCS (Double Byte Character Set) support. Therefore, in effect, a single set of DDS source files can be used for both environments. This reduces the task of maintenance and simplifies system building.

Conversion is carried out by a Toolkit command Convert DDS to IGC (YCVTDDSIGC). The command can used to convert both to and from IGC format.

Before you use the command on a source file you must mark up the source with additional comments to direct the conversion process. Refer to the command diagram for the Convert DDS to IGC command for further details.

The following represents the use of DDS IGC conversion utility.

Special Display Commands

The Toolkit command, Abbreviated Display (YDSPABR), provides a concise way of displaying the major system displays. For a description of the abbreviated display command refer to the command diagram.

Lists

CA 2E Toolkit list facilities are an extremely powerful extension to the generic capabilities of the i OS operating system.

Lists Concepts

Using the list facilities, whole sets of objects or members may be processed with a single instruction. Toolkit allows you to store a list of items under a single name, and then to subsequently identify all the items in the list just by referring to the list name. Toolkit allows you to create, manipulate, and use, four types of lists:

- Lists of objects
- Lists of database files
- Lists of database file members
- Lists of formats

Each list is made up of list entries containing information about the elements within the list relevant to the list type; for instance:

- For objects - owner, object attribute
- For members - source type, last change date
- For database files - journalling status, database relations
- For formats - file, format identifier

Database file lists can be regarded as a special type of object list.

Note: Toolkit also has facilities for referring to a list of libraries by a single name.

General Considerations for Lists

The list manipulation functions can be incorporated into your own utilities by using the appropriate commands.

LST Type	Create Command	Edit Command	Filter Command	Execute Command	Print Command	Covert Command
OBJ	YPLDOBJLST	YEDTOBJSLT	YFTOBJLST	YEXCOBJLST	YDOCOBJLST	YCVTOBJLST
DBF	YBLDDBFLST	YEDTDBFLST	YFLTDBFLST	YEXCDBFLST	YDOCDBFLST	YCVTDBFLST
MBR	YBLDMBRLST	YEDTMBRLST	YFLTMBRLST	YEXCMBRLST	YDOCMBRLST	YBLDDOC
FMT	YBLDFMTLST	YEDTFMLST	YFLTFMTLST	--	YDOCFMTLST	--

Also refer to the following:

- YADDOLE
- YADDMLE
- YCPYLST
- YINXLST
- YDLTLST
- YOPRLST
- YCHGLST
- YCHKLSTE
- YSCNSRC
- YCVTDBR
- YCVTPGMREF
- YCVTUSRPRF
- YCVTAUTL

The list manipulation functions can be incorporated into your own utilities by using the appropriate commands.

Storing Lists

Lists can be referred to with qualified names, as if they were objects in their own right, for example, OBJLST (QGPL/FRED). Each list name of a given type must be unique within its library.

Lists are actually stored as database file members: a file which contains a list must have the format of the database OUTFILE from the associated i OS command which generates the list. You do not generally need to be concerned with the implementation details of lists; further details are given in the discussion on list parameters in Appendix A of the *CA 2E Toolkit Reference Guide.*

Creating Lists

Lists can be built:

- Directly from objects - YBLDOBJLST, YBLDDBFLST, YBLDFMTLST
- Directly from members - YBLDMBRLST, YSCNSRC, YSCNRPLSRC
- From references to objects - YCVTDBR, YCVTPGMREF, YCVTUSRPRF
- From other list types - YCVTOBJLST, YCVTDBFLST, YOPRLST
- From named items - YADDOLE, YADDMLE
- As output from list processing commands - YCPYLST, YFLTMBRLST, YFLTOBJLST, YCHKLSTE

The building of lists directly from existing items is discussed below.

Building Lists from Existing Items

Each list type has its own Build List command, which allows you to build a list from an existing collection of items. For example:

```
YBLDOBJLST OBJ(QGPL/FRED*) OBJTYPE(*ALL)  OBJLST(QTEMP/FRED)
```

Once the list has been built, the list entries are independent of the items to which they refer. So, for example, an entry will remain in the list even if the corresponding item has been deleted.

For example to use the Build Object List command (YBLDOBJLST) to build a list of all objects that are in library QGPL and whose names begin with the letters FR, see the following diagram:

Building Lists from Named Items

For object and member lists there is another way of adding entries to lists: you may add items to lists one by one using the Toolkit commands Add Object List Entry (YADDOLE) and Add Member List Entry (YADDMLE). For example, to add an entry for a program FRED to list PGMS:

```
YADDOLE OBJ(FRED) LIB(QGPL) OBJTYPE(*PGM) OBJLST(QTEMP/PGMS)
```

For example, to add an entry for a member FRED of source type CL in file QCLSRC:

```
YADDMLE MBR(FRED) LIB(QGPL) FILE(QCLSRC) SEUTYPE(RPG) MBRLST(QTEMP/PGMS)
```

The Add List Entry commands are primarily intended for use in your own CL programs. They can be useful when you wish to build a list of items that do not yet exist.

Editing Lists

All lists can be interactively edited, that is, all the items in the list can be displayed and examined in detail.

The list edit functions are invoked either by using the appropriate Toolkit Edit List command (namely YEDTDBFLST, YEDTFMTLST, YEDTOBJLST or YEDTMBRLST) or by using the EDIT parameter on one of the Toolkit generic commands, for example, Translate Physical File (YTRNPF), Move Object and Source (YMOVOBJSRC), Remove Members (YRMVM).

The list edit functions for database file lists and format lists differ from the list edit functions for object lists and member lists. The two types of list edit functions are described below.

Database File Lists and Format Lists

The list edit functions for database file lists and format lists allow you to display details for an item on the list, and to include or drop list items. You can either specify the items you do want or the items you do not want - whichever is easiest.

Object Lists and Member Lists

The Edit Object List (YEDTOBJLST) and Edit Member List (YEDTMBRLST) commands allow you to work with the entries in a list, and also to work with the underlying objects or members.

Full support is provided for flagging list entries, selecting on flag values and maintenance of an output list. You can use user-defined options set up with the i OS PDM Work with User-Defined Options facility.

Working with the Entries in a List

The first panel displayed by the YEDTOBJLST command allows manipulation of the list entries. Note that a list entry can exist independently of the object it represents. List entries can be added, removed, deleted, flagged or copied to another list.

Working with the Objects in a List

Pressing F10 gives you an alternative panel, which allows you to work with the objects in the list. Objects can be changed, renamed, duplicated, moved, or deleted. Parameters for the object manipulation commands can be entered on the command line. If no parameters are entered, the commands will be prompted.

The YEDTMBRLST command allows member specific operations such as option 2 to invoke the i OS Start SEU command (STRSEU) to edit the source member.

Changing the Defaults for List Commands

Many of the list manipulation commands which can be called from the Work with Object List Entries panel have parameters in common. Default values for these parameters can be supplied from the Change Defaults panel, obtained by pressing F18 from the Work with Object List Entries panel. A user-defined option file, set up using the i OS PDM utility, can also be specified on this panel.

Exit from the Work with Object List Entries Display

On leaving the Work with Object List Entries display, you will have the option of saving the edited list, or abandoning the changes you have just made. You can optionally remove entries which were flagged for deletion.

Filtering Lists

Filtering allows you to remove the items from a list that you do not want, using selection criteria that can apply to whole classes at a time.

- Filtering can be invoked either by pressing a function key from the Toolkit displays Edit List, (Work with List Entries), or by using the Toolkit filter command appropriate to the list type, YFLTOBJLST**,** YFLTMBRLST, YFLTFMTLST, or YFLTDBFLST**.**
- The filtering functions available are different for each list type, reflecting the different attributes possessed by the different types of list items.

The following is a table of filterable attributes.

Filter Type	OBJ List	DBF List	MBR List	FMT List
Object Names	Yes: mask	Yes: mask	Yes: mask	Yes: mask
Library Names	Yes: mask	Yes: mask	Yes: mask	Yes: mask
Member Names			Yes: mask	
Object Type	Object Type	PF or LF	(DBF)	Format Type
Object Text	Yes CT	Yes CT	Yes CT	Yes CT
Attribute	Yes	DATA/SRC	Source Type	File Attribute
Object Owner	Yes			
Create Date	Yes: (1)		Yes: (1)	
Chg. dt. (Obj&Src)	Yes: (1)			
Save file	Yes			
Save File lib	Yes			

Filter Type	OBJ List	DBF List	MBR List	FMT List
Sys Compiler lvl	Yes: (1)			
Object size	Yes: (1)			
Damaged object	Dmgd/Undmgd			
Journal		Yes		
Select/omitused		Select/omit		
Access path		Key/Arrival		
Acc. path maint		Yes		
Alt. Coll. Seq		Yes		
Level Check		Yes		
No. active rcds			(1)	
No. deleted rcds			(1)	
Flag value	Yes: (2)		Yes: (2)	
Last used date	Yes: (1)		Yes: (1)	
Date usage counter	Yes: (1)			
reset				
Days used	Yes		Yes	
Creator	Yes			
System where Created	Yes		Yes	
Object domain	Yes			
User modified	Yes			
Auxilliary storage pool	Yes		Yes	
Compression status	Yes			
Compiler level				

(1) Any of the relational operators *GT, *LT, *EQ, *NE may be used.

(2) Either of the relational operators *EQ, *NE may be used.

- Either selection or omission logic can be used when filtering, that is, either every item meeting the criteria may be selected, or every item that does not meet the criteria may be selected.
- Name selection can be specified; not just to select generic names, but also to select any combination of letters appearing anywhere in the name. For instance, all names containing the letters PGM could be selected. You specify the name selection that you desire by means of a name mask. Rules for specifying name masks are given below.

Name Mask Formation

A name mask is used to specify the names of the list items which are to be selected.

Apart from containing the normal characters of a valid i OS system name, a Toolkit name mask may contain any combination of the following three special characters:

Type	Character	Location in Mask	Scan action
Wild	'?'	Anywhere in Mask	Match on any character
Generic	'*'	At the end of the Mask	Generic name beyond this character
Contains	'*'	Anywhere within the Mask	Floating scan beyond this character

Name Masking Examples:

- A* would select all names beginning with the letter A.
- *A* would select any names containing the letter A.
- *A would select any names ending with the letter A.
- A?C would select any names that begin with the letter A, and that also have the letter C in the third position - any character may be in the second position.
- *A? would select any names with the letter A as the last character but one.
- A*B* would select any names beginning with the letter A, and containing the letter B in any other position.
- Q???SRC will match QRPGSRC and QDDSSRC, but not QCLSRC.
- Q*SRC will match QRPGSRC and QCLSRC, but not QRPGSRC1.
- Q*S?C* will match all QRPGSRC, QCLSRC, QDDSSRC, plus names such as QSXC.

Filtering Examples:

Your example is an object list call FRED in library QGPL with the following items in it:

To select all RPG programs in object list FRED that are owned by user ME and that have object names:

- Either containing the letters FRED anywhere in the name.
- Or beginning with the string M?SH, where ? represents any character.

Enter the following command:

```
YFLTOBJLST OBJ((*FRED*) (M?SH))OBJTYPE (*PGM) OWNER(ME) + OBJLST(QGPL/FRED)
```

Your original list would thus be filtered down to:

Using Lists

You will have built a list because you want to carry out an operation on many items. There are two main ways of using a list to drive processing:

- In Toolkit commands that have the LST or xxxLST parameters
- In any other command by means of the YEXCxxxLST command

Using Commands with the lst or xxxlst Parameter

Toolkit includes versions of certain commonly used i OS commands, the i OS versions of which are not generic. In the Toolkit versions of the commands, you may indicate the items that you wish to process either by specifying a generic name, or by specifying the name of a list which contains the items which are to be processed using the LST parameter, or the OBJLST**,** MBRLST, FMTLST, or DBFLST parameters. For example:

Using a Generic Name

The following will change the ownership of all objects in library QGPL whose names begin with FR:

```
YCHGOBJOWN OBJ(QGPL/FR*) OBJTYPE(*PGM *DSPF) NEWOWN(FRED)
```

Using an Existing List

You indicate that a list is to be used by specifying a special value for the object name, OBJ(*OBJLST)**,** and you indicate which list is to be used by a special parameter, OBJLST(list-name).

The following will change the ownership of all objects in object list HENRY in library QGPL:

```
YCHGOBJOWN OBJ(*OBJLST) OBJTYPE(*OBJLST) NEWOWN(FRED) OBJLST(QGPL/HENRY)
```

Likewise for member lists you would use a MBRLST parameter, and for database lists a DBFLST parameter.

Toolkit generic commands are described in further detail in the section Generic Commands in this chapter.

Using the Execute List Function

One of the most powerful features of Toolkit list handling is the Execute List function (YEXCOBJLST, YEXCDBFLST and YEXCMBRLST commands). These functions will execute on a nominated list of items any request string (including PDM user options) that you specify. You may indicate where in the request string you wish the names of the list items to be inserted. For example, to print the contents of all data areas in a library:

```
YBLDOBJLST OBJ(FRED/*ALL) OBJTYPE(*DTAARA)/* build list */
YEXCOBJLST RQSDTA('DSPDTAARA(&L/&O) OUTPUT(*PRINT)') /* use it*/
```

In other words if the list FRED contained the qualified names of three data areas, XX/AA, YY/BB and ZZ/CC, then invoking the YEXCOBJLST command would be equivalent to specifying the following three commands:

```
DSPDTAARA DTAARA(XX/AA) OUTPUT(*PRINT)
 DSPDTAARA DTAARA(YY/BB) OUTPUT(*PRINT)
 DSPDTAARA DTAARA(ZZ/CC) OUTPUT(*PRINT)
```

By using the Toolkit Execute List functions, commands that are not generic can be made generic, and generic commands can be made to work on lists of items. See the respective command diagrams for further details.

List Execution Errors

An error level can be specified on the execute list commands. Execution will stop if more than a certain number of errors occur. If an error level greater than zero is specified, and errors occur, the list items on which the errors occur will be flagged: thus the list can easily be filtered for re-running once the cause of the errors has been resolved. Alternatively, you may use the UPDLST parameter to specify that list items that have been successfully processed are to be removed from the list or have their entries flagged. The remaining or flagged items may then conveniently be reprocessed once the cause of the error has been determined.

Updating Lists

Updating lists allows you to control the updating of lists and the output lists.

Controlling the Updating of Lists

You can modify the contents of a list as a result of processing it in a Toolkit generic command. A number of the commands have an UPDLST parameter which allows you to do this. The UPDLST parameter controls:

- Whether items in the list are to be flagged as a result of processing.

  - FLAGERR - flag only those list items for which errors occurred.

  - FLAGOK - flag only those list items for which no errors occurred.

- Whether items in the list are to be removed as a result of processing

  - RMVERR - remove list items for which errors occurred.

  - RMVOK - remove list items for which no errors occurred.

- On some commands, the list need not be updated as a result of processing.

  - NONE - do not flag or remove any list items.

UPDLST actions

Actions	Pass	Fail
*None	No Change	No Change
*RMVERR	Leave	Remove
*RMVOK	Remove	Leave
*FLAGERR	No Change	Flag
*FLAGOK	Flag	No Change

Output Lists

Certain of the generic commands allow you to specify an output list: the results of executing the list will be to produce a new list.

In some cases a separate output list is mandatory. For example, in the following command to convert an object list into a member list:

```
YCVTOBJLST OBJLST(SIMBA) MBRLST(CHUUI)
```

In other cases a separate output list is optional. For example, the following command would scan the members named in an input list called NUGU and produce an output list FISI containing just the names of the members containing the search string:

```
YSCNSRC SELN(*IF 'QCAEXEC') MBRLST(NUGU) OUTLST(FISI) UPDLST(*RMVERR)
```

Output Lists and the UPDLST Parameter

The UPDLST action always applies only to the output list. If no separate output list is specified, then the input list is effectively the output list as well. In either case the same logic is used: each item in the list is processed and then its list entry included or flagged into the output list according to the UPDLST parameter.

If an outlist is **not** specified, then for each item processed the input list will be updated according to any actions specified for the UPDLST parameter.

For example, if UPDLST(*RMVERR) is specified, those items for which an error occurred on processing will be removed from the list. Only the items which are successfully processed remain in the input list.

If an outlist is specified, then for each item processed the output list will be updated according to any actions specified for the UPDLST parameter.

For example, if UPDLST(*RMVERR) is specified, those items for which an error occurred on processing will be removed from further consideration: they will not be added to the output list. (If however they are already present in the output list they will not be removed from it). Only those items which are successfully processed are added to the output list.

The following table lists commands with the UPDLST parameter.

Command	UPDLST Value None	UPDLST Value FLAGERR	UPDLST Value FLAGOK	UPDLST Value RMVERR	UPDLST Value RMVOK	OUTFLAGVAL Ass/Opt	OUTFLAGVAL Default Value
YBLYxxxLST			A	A		O	Null
YFLTxxxLST			Y	Y	-(1)	O	On
YEXCxxxLST	Y	Y	Y	Y	Y	O	Fail
YCHKLSTE		Y	Y	Y	Y	O	On
YSCNSRC	-(2)	Y	Y	Y	Y	O	On
YCMPSRC		Y	Y	Y	Y	O	Fail
YCRTOBJ		Y	A	Y	Y	O	Fail (3)
YMOVOBJ		Y	A	Y	Y	A	FailOBJ
YMOVM		Y		Y	Y	A	FialMBR
YMOVOBJSR		Y		Y	Y	A	(4)

UPDLST

: A = Always assumed

: Y = Default, may be overridden

Y = Available

OUTFLAGVAL

:O = Optional, that is chosen value or *NULL

A = Always assumed, value cannot be changed

Flagging Lists

This topic explains the purpose of the list flag value and how to use them.

Purpose of List Flag Values

Each item in a list can have a flag. List item flags can be used:

- To control which items in the list are to be processed
- To determine which items in a list were successfully or unsuccessfully processed

List Flag Values

List flag values may either be one of a number of special values, for example *FAIL, *ON, or *NULL.

Setting List Flag Values

Toolkit List Processing and Generic commands set list flag values. This may be either:

- Implicit - some of the Toolkit generic commands automatically set flags to indicate whether items have been processed successfully or not. For example, the Toolkit command Move Member (YMOVM) flags the list entries for any members which it failed to move, with a *FAILMBR status.
- Explicit - On other Toolkit generic commands you may use the OUTFLAGVAL parameter optionally to specify a flag value to be given to particular items. You must also specify a value of *FLAGOK or *FLAGERR for the UPDLST parameter to indicate the circumstances under which the flag is to be set.

For example, on the Toolkit command Filter Object List (YFLTOBJLST) you could specify that items which meet the filter criteria are to be given a particular flag value:

```
YFLTOBJLST TEXT('USRPGM') OUTFLAGVAL(*ON) UPDLST(*FLAGOK)
```

Similarly, you could use the Toolkit command Scan Source (YSCNSRC) specifying that items which meet the search criteria are to be given a particular flag value:

```
YSCNSRC SELN(*IF 'QCAEXEC' *FALSE')  UPDMBRLST(*FLAGERR) OUTFLAGVAL('X')
```

Using List Flag Values

List flag values may be used to control processing on most of the Toolkit generic commands, which allow the specification of an existing list. In general, only the list items having the specified flag value will be processed by the command.

For example, on the Toolkit command Execute Object List (YEXCOBJLST) you might specify that only those items with a flag value of A are to be processed - the others are ignored. If an error occurs when an item is processed then it will be removed from the list.

YEXCOBJLST FLAGVAL('A') RQSDTA('DSPOBJD @L/@O @T') UPDLST(*RMVERR)

In the special case of the Filter List commands (YFLTxxxLST), items that do not satisfy the flag value may be subject to further processing (depending on the UPDLST parameter).

For example, the following command will remove, from the list, all items that do not have a flag value of A.

YFLTOBJLST FLAGVAL('A') UPDLST(*RMVERR)

The following table shows which Toolkit list commands support the use of flag values. Note that for some commands the use and value of an output flag value is automatically assumed.

Toolkit Command	Test on Input (FLAGVAL)	Set as Output (OUTFLAGVAL)
YBLDxxxLST		O
YADDxLE		O
YFLxxxLST	O	O
YEXCxxxLST	O	O
YCHGLST	O	
YCHKLSTE	O	O
YSCNSRC	O	O
YCMPSRC		O
YCRTOBJ	O	A (Flagerr), O (Flagerr)
YMOVM	O	A
YMOVOBJ	O	A
YMOVOBJSRC	O	A

O = Optional A = Assumed

Convert Object Lists

Conversion functions exist to convert lists of one type into lists of a different type. Examples of reasons why you might wish to do this are given under each conversion function. You may:

- Convert object lists to member lists.
- Convert database relations into an object list.
- Convert program references into an object list.
- Convert user profile references into an object list.
- Convert authorization list references into an object list.
- Convert DBF lists into member lists.
- Convert member lists into text documents.

Converting Object Lists to Member Lists

A list of objects may be converted into a list of source members. The significance of this will be become apparent when you recall that the Toolkit command generic Create Objects (YCRTOBJ) can work from a specified member list.

Recompile a list of objects, selected by name, owner, or indeed any of the object selection criteria such as size or system release level which are available with the Toolkit filter functions. Having built and filtered such a list, use the Toolkit command Convert Object List (YCVTOBJLST) to create a member list from it containing the names of the source members of all the objects in the given object list. The member list can then be fed straight into the Create Object command (YCRTOBJ).

For example, the effect of the following four commands would be to submit recompiles of everything beginning with FRED in library QGPL:

```
YBLDOBJLST OBJ(QGPL/FRED*) OBJTYPE(*ALL) OBJLST(GEORGE) /*bld*/
YFLTOBJLST DATE(*CRT *GT 01/01/87) OBJLST(GEORGE) /*filter*/
YCVTOBJLST OBJLST(GEORGE) MBRLST(GEORGE) SRCFILE(SAUCE/*QDFTSRC)
 YCRTOBJ OBJLIB(NEW) MBRLST(GEORGE) SRCFILE(*MBRLST) JOBQ(QGPL/QGMR) JOBD(QBATCH)
/*recompile */
```

This can be shown in a diagram as follows:

Converting Database Relations into an Object List

The Toolkit command Convert Database Relations (YCVTDBR) converts the output of the Display Database Relations command (DSPDBR) into an object list. This gives you an alternative way of identifying all of the logical files dependent on a physical file or files.

For example, the effect of the following three commands would be to print a list of all logical files which are based on physical files in library QGPL, but which are not themselves in QGPL:

```
YCVTDBR FILE(QGPL/*ALL) /* build list */
YFLTOBJLST FILTER(*OMIT) OBJ((*ALL QGPL)) /*filter list */
YDOCOBJLST /* print list */
```

Converting Program References into an Object List

The Toolkit command Convert Program References (YCVTPGMREF) converts references to a database file, or format, into an object list. All programs that reference the specified file in a specified manner will be included in the list. The resulting list can be used to identify the names of programs that need to be recompiled as the result of a file change.

For example, the effect of the following three commands would be to submit recompiles of all programs in library QGPL that use file FRED:

```
YCVTPGMREF PGM(QGPL/*ALL) FILE(FRED) OBJLST(GEORGE)
 YCVTOBJLST OBJLST(GEORGE) MBRLST(GEORGE) SRCFILE(SAUCE/*QDFTSRC)
 YCRTOBJ OBJLIB(NEW) MBRLST(GEORGE) SRCFILE(*MBRLIST) JOBQ(QGPL/QPGMR) JOBD(QBATCH)
```

Converting User Profile References into an Object List

The Toolkit command Convert User Profile (YCVTUSRPRF) builds a list of objects referenced by a user profile into an object list. The reference may be any one of the following types:

- *OBJOWN - An object list of the objects owned by a profile.
- *CMDAUT - An object list of the commands to which the user is explicitly authorized.
- *GRPMBR - If the profile is a group profile, an object list of all the user profiles belonging to the group profile.
- *DEVAUT - An object list of the device descriptions to which the user is explicitly authorized.
- *OBJAUT - An object list of the objects to which the user is explicitly authorized.

All objects that are referenced by the specified profile in the specified manner will be included in the list. The resulting list can be used to manipulate all the objects belonging to the profile.

For example, the effect of the following two commands would be to change the ownership of all objects owned by user profile FRED to user profile BASIL:

```
YCVTUSRPRF USRPRF(FRED) TYPE(*OBJOWN)
 YCHGOBJOWN OBJ(*OBJLST) OBJTYPE(*ALL) NEWOWN(BASIL)
```

Converting an Authorization List to an Object List

The Toolkit command Convert Authorization List (YCVTAUTL) builds a list of the objects referenced by an authorization list into an object list. The reference may be one of the following types:

- *AUTOBJ - An object list of the objects secured by the authorization list.
- *USRPRF - An object list of the user profiles contained in the authorization list.

All objects that are referenced by the specified authorization list in the specified manner will be included in the list. The resulting list can be used to manipulate all the objects secured by the authorization list.

For example, the effect of the following two commands would be to change the ownership of all objects secured by list FRED to be owned by profile BASIL:

```
YCVTAUTL AUTL(FRED) TYPE(*AUTOBJ)
 YCHGOBJOWN OBJ(*OBJLST) OBJTYPE(*ALL) NEWOWN(BASIL)
```

Converting Database File Lists into Member Lists

The Toolkit command Convert Database File List (YCVTDBFLST) converts a database file list into a member list. It can, like the other list conversion commands, be used to identify the names of source members that are to be recompiled. Note that since the selection functions of a database file list include a based-on file, a physical file and all its dependent logical views can be rapidly identified and recompiled.

Converting Member Lists into Text Documents

The Toolkit command Build Document (YBLDDOC) helps with documentation.

The most convenient and flexible way of arranging documentation for preparation and maintenance is to divide it into small sub-documents. The sub-documents can then be linked together, using a master document that contains imbed references to each component. Thus:

- Different combinations of the same sub-documents can be used for different purposes, for example, manuals for different user departments.
- The separate documents that appear as the Help text for individual menu options and application programs, can be grouped together in any combination and order, and printed as a single document.

The Toolkit function Build Document (YBLDDOC) helps you prepare master documents. It converts a list of members, previously built and edited with the member list build and edit functions, into a master document containing all the necessary references to the documents in the member list.

For example, the help text for each Toolkit display is stored as a document in a file Y1HLPTXT in the Toolkit shipped library. The following three commands would print all the Help text for Toolkit:

```
YBLDMBRLST FILE(Y1HLPTXT) /* build list of documents */
YBLDDOC DOCUMENT(HLPINX) FILE(QGPL/QTXTSRC) TEXT('Help index') /* convert list to document */
QSYS38/PRTDOC SRCFILE(QGPL/QTXTSRC) DOCUMENT(HLPINX) /* print it /
```

Additional List Functions

The following topics lets you manipulate and assign descriptive text on lists.

Manipulating Lists

Toolkit provides various service functions that are useful in manipulating lists. These functions are:

- YCPYLST - Copy a list from one library to another.
- YMOVLST - Move a list from one library to another.
- YDLTLST - Delete a list.
- YINXLST - Add a logical view to a list.

If you are making use of lists in your own commands, the Index List function can be very useful: it performs the commonly required function of adding a logical view on a list. Refer to the command diagram for further details.

Descriptive Text on Lists

Lists can be given descriptive text by using the Change Physical File Member command (CHGPFM).

Printing Lists

Lists may be printed using the Toolkit commands Document List (YDOCOBJLST, YDOCMBRLST, YDOCDBFLST and YDOCFMTLST). For example:

```
YDOCOBJLST OBJLST(FRED)
```

Changing List Entries

The Toolkit command Change List (YCHGLST) allows you to make global alterations to certain of the list values, for example, the library name. For example, to change the source type of all members in list FRED to RPG:

```
YCHGLST LSTTYPE(*MBR) LST(FRED) SEUTYPE(RPG)
```

Such a facility can be useful when you wish to bring an old list up to date, for instance after moving everything in a list from one library to another. It can also be useful when you are comparing lists - see combining lists below.

Checking or Verifying Lists

It is possible that in the interval between building a Toolkit list from existing objects or members, and actually using the list, the original objects or members might be deleted or moved to another library or file. To check whether the entries in a list are still current, you may use the Toolkit command Check List Entry (YCHKLSTE), which will check each entry in a Toolkit list against a specified library, source file, or both. For instance:

```
YCHKLSTE LSTTYPE(*OBJ) LST(QGPL/FRED) CHKLIB(QGPL) CHKFILE(QGPL/*QDFTSRC)
```

The Check List Entry command can be particularly useful for testing whether objects have a corresponding source member or vice versa. It can also be used to compare the contents of two libraries.

Using the UPDLST parameter of the Check List Entry command, you may specify either that the list entries which are found to be in error are to be removed or flagged, or that list entries which are successfully verified are to be removed or flagged. The latter option provides a convenient way of examining a member list to see which source members still need compiling.

For example, the following command would check the entries in member list FRED to ensure that for each member there was a corresponding object in library QGPL. Any entries without objects would be removed:

```
YCHKLSTE LSTTYPE(*MBR) LST(FRED) CHKLIB(QGPL) UPDLST(*RMVERR)
```

You can also use the YCHKLSTE command in your own CL programs to test whether a list is empty: an escape message will be sent if no entries can be found. For example:

```
YCHKLSTE LSTTYPE(*OBJ) LST(FRED)
 MONMSG MSGID(YYY0103) EXEC(RETURN) /* list is empty */
```

Combining Lists

The Toolkit command Operate on List (YOPRLST) gives you the ability to combine or divide lists. Lists can be combined together using set theory operations such as intersection and union. Such a capability is very powerful and can be used in many ways. Here are some examples:

- To compare two libraries and list the differences, build a list of the desired type from the contents of each library, use the Operate on List function to create a list of the differences, and then use the Document List (YDOCxxxLST) function to print the result list.

```
YBLDOBJLST OBJ(A/*ALL) OBJLST(A) /* library A contents */
YBLDOBJLST OBJ(B/*ALL) OBJLST(B) /* library B contents */
YOPRLST LSTTYPE(*OBJ) LSTA(A)
LSTOPR(*DIFF) LSTB(B)
TOLST(C) IGNLIB(*YES) /* differences */
YDOCOBJLST LSTB(C) /* print */
```

- To compare two libraries and list all the objects which have identical names, creation dates/times and change dates/times, use the YOPRLST command with LSTOPR(*INTERSECT).

```
YBLDOBJLST OBJ(A/*ALL) OBJLST(A)/* library A contents */
YBLDOBJLST OBJ(B/*ALL) OBJLST(B)/* library B contents */
YOPRLST LSTTYPE(*OBJ) LSTA(A)
LSTOPR(*INTERSECT) LSTB(B)
TOLST(C) IGNLIB(*YES) IGNCRTDTE(*NO)
IGNCHGDTE(*NO)/* find identical obj */
YDOCOBJLST LSTB(C)/* print*/
```

- To find out if all the objects in a library have a corresponding source member, build an object list of the objects, and convert it to a member list using the Convert Object List function (YCVTOBJLST). The member list so created could then be subtracted, using the Operate on List function from a member list built from the source file itself. The resulting list, which can be printed, represents the objects without source.

```
YBLDOBJLST OBJ(A/*ALL) OBJLST(A) /* library A contents */
YCVTOBJLST OBJLST(A) MBRLST(A)/* library a contents */
YBLDMBRLST FILE(B/Q*) MBRLST(B)/* src files in b contents */
YOPRLST LSTTYPE(*MBR) LSTA(A)
LSTOPR(*SUB) LSTB(QTEMP/B)
TOLST(QTEMP/C) IGNLIB(*YES) /* differences */
YDOCOBJLST LSTB(QTEMP/C) /* print */
```

- To change the ownership of everything in a library, except for a given list of exceptions, build a list of everything in the library, subtract the exceptions, then change the ownership of the remainder.

Valid List Operations

The type of list operation to be used is specified by the list operation (LSTOPR) parameter on the Toolkit command Operate on List (YOPRLST).

The following Venn diagram represents the contents of two intersecting lists: A and B. When comparing lists with the YOPRLST command, you may specify that for the comparison of each item the library name, the creation date, the change date, or all, are to be ignored.

List Selection Functions

When using the interactive Toolkit list commands, such as YEDTOBJLST and YEDTMBRLST, if you do not know the name of the list that you wish to process you may enter a value of *SELECT, which will cause a list of available list names to be shown. For example:

Display Member List Command

The list selection functions are provided by a general purpose Toolkit command, Display Member List (YDSPMBRLST). You may also use the command directly in your own CL programs in order to provide a selection list of available database file members for your users.

For example, the following CL statements would provide a display of all the members in file QTXTSRC in library QGPL, and allow one of them to be selected:

```
DCL VAR(&FILE) TYPE(*CHAR) LEN(10) VALUE(QTXTSRC) /*File */
DCL VAR(&LIB)  TYPE(*CHAR) LEN(10) VALUE(QGPL)  /*Library*/
DCL VAR(&MBR)  TYPE(*CHAR) LEN(10) VALUE(*ALL) /*Member */
DCL VAR(&FILEATR)  TYPE(*CHAR) LEN(10) VALUE('*PHY') /*Type*/
DCL VAR(&TEXT) TYPE(*CHAR) LEN(10) VALUE ('SELECT A TEXT MEMBER') /*Text*/
YDSPMBRLST FILE(&FILE) LIB(&LIB) MBR(&MBR) FILEATR(&FILEATR) TEXT(&TEXT)
```

The name, file name, library and text of the selected member would be returned in the appropriate variables. You may specify that only members from files of a given format type are to be displayed.

Interconnection of the List Utilities

The list ultilities can all be regarded as providing generic connections between system entities, as follows:

Generic Processing

The following section describes the Toolkit generic command concepts and generic commands.

Generic Commands

Through the Execute List facilities that Toolkit provides, it is possible to turn any command, i OS, Toolkit, or user-defined, into a generic command. In addition to this capability, Toolkit provides ready made generic versions of certain commonly used i OS commands.

Toolkit generic command concepts are closely linked with Toolkit list concepts: refer to the section on lists in this manual before reading this section.

Note that the meaning of generic under Toolkit is a considerable enhancement upon the i OS concept. The name masking and attribute selection capabilities of the edit and filter list utilities of Toolkit allow you great flexibility in specifying items - far more than that allowed by the mere specification of a generic name.

General Considerations for Generic Manipulation

The following table shows the Toolkit generic commands that are available.

Toolkit Command	Related i OS Commands	List Type
YCHGCMD	CHGCMD	OBJLST
YCPYF	CPYF, ENDJRNPF, STR, JRNPF	DBFLST
YCRTOBJ	CRTRPGRGM, CRTDSPF, CRTPF, CRTCLPGM	MBRLST
YCRTODUPSBJ	CRTDUPOBJ	OBJLST
YDLTOBJ	DLTPGM, DLTF, DLTCMD	OBJLST
YMOVOBJ	MOVOBJ,GRTOBJAUT	OBJLST
YMOVOBJSRC	MOVOBJ, CPYF, RMVM	OBJLST
YMOVM	CPYF, RMVM	MBRLST
YRMVM	RMVM	MBRLST

Most of the Toolkit generic commands provide additional functions beyond that provided by the simple command; for instance, the Toolkit command Change Object Ownership (YCHGOBJOWN) can revoke authorities as well as change ownership.

The following table shows other Toolkit commands that use Toolkit lists.

Toolkit Command	Related i OS Command	Module	List Type
YBLDDOC		DOC	MBRLST
YCMPSRC		PGMR	MBRLST
YDOCF	DSPFD, DSPFFD, DSPOBJD	DOC	OBJLST
YDOCMSGREF		DOC	MBRLST
YDOCSRS		DOC	OBJLST
YSCNSRC		PGMR	MBRLST
YSCNRPLSRC		PGMR	MBRLST

Using Generic Commands

All of the generic commands can either be used generically, without reference to a list, or make explicit use of the list function. For example:

- Simple generic use of a command to move all programs whose names begin with the letters FR in library QGPL to library FRED:

```
YMOVOBJ OBJ(QGPL/FR*) OBJTYPE(*PGM) TOLIBOBJ(FRED)
```

Note: The parameter usage is exactly analogous with that of the i OS command Move Object (MOVOBJ):

```
MOVOBJ OBJ(QGPL/FREUD) OBJTYPE(*PGM) TOLIBOBJ(FRED)
 MOVOBJ OBJQGPL(FREEZE) OBJTYPE(*PGM) TOLIBOBJ(FRED)
 MOVOBJ OBJ(QGPL/FRIED) OBJTYPE(*PGM) TOLIBOBJ(FRED)
```

- Explicit use of a list to move all objects in object list FRED in QTEMP to library QGPL:

```
YMOVOBJ OBJ(*OBJLST) OBJTYPE(*ALL) TOLIBOBJ(QGPL) OBJLST(FRED)
```

Generic Command Messages

Detailed diagnostic messages are sent to the log for each individual item in the list processed. If you have an appropriate logging level for your job, you can display the messages by using the second level display option F10 from the i OS command Entry Menu **(**QCMD).

All of the generic commands send a completion message giving the total number of items processed and the total number of errors.

Generic Move Commands

The Toolkit generic move commands provide considerably more functions than simply moving an object from one library to another. Options available include:

- Moving the source for an object
- Saving the previous version of the object, source, or both, into an archive library
- Preservation of existing object authorities
- Logging the movement of the object, its source, or both
- Automatic separation of application objects, objects containing data, and source objects, into separate libraries
- Selective movement of only the new objects, or only the objects that already exist in the destination library, or all objects

This can be shown diagrammatically:

Generic Move Example

Say that you have a program and accompanying display file both called CUSCHG**,** as well as a physical file, called CUSPF, in your live system: the program and display file are in a library FREDPGM, the file in a library FREDDTA.

- You wish to replace these with new versions from library FREDNEW.
- You wish also to transfer the source for the objects you are moving to a library FREDSRC, which contains the source for your live objects.
- You want to preserve the previous version of the program and source in an archive library FREDOLD, deleting any existing previous version in FREDOLD.
- You also wish the new versions of the objects to be given the same authorizations that the existing versions possess.
- In addition you wish to keep a record of what you have done, in the form of a message sent to a message queue, MOVLOG.

The following single instruction would perform all of these functions:

```
YMOVOBJSRC OBJ(CUS*) OBJTYPE(*PGM *FILE
TOLIBOBJ(FREDPGM)  TOLIBDTA(FREDDTA) OLDLIB(FREDOLD) MSGQ(MOVLOG)
```

Restrictions on Moving Physical Files

The generic move commands are very powerful: so powerful in fact that Toolkit includes a safety feature to prevent you from doing too much! Namely, physical files cannot be moved without an archive library being specified. Thus you cannot inadvertently destroy data.

Most of the generic commands will also not function on objects in libraries beginning with the letter Q - apart from QGPL and QTEMP.

Generic Compile

The Toolkit generic command, Create Objects (YCRTOBJ), recompiles some or all of the source members in a source file with a single instruction.

For example, the following command will submit compiles of all display file members in QDDSSRC in *LIBL whose names begin with the letters FR. Job description BODJ in *LIBL will be used. Any existing versions will first be deleted.

```
YCRTOBJ OBJLIB(FRED) SRCFILE(QDDSSRC) MBR(FR*)SEUTYPE(DSPF) JOBD(BODJ) CRTOPT(*ALL)
```

The names of the members that are to be compiled can be specified in one of two ways:

- Using a source file name and a generic member name
- Using a Toolkit member list name which can be created in a number of ways:
  - Using the Toolkit command Build Member List (YBLDMBRLST).
  - Using the Toolkit command Scan Source (YSCNSRC), or the command Scan and Replace Source (YSCNRPLSRC).
  - Using the Toolkit functions Convert Object List (YCVTOBJLST), or Convert Database File List (YCVTDBFLST).

You may edit or filter your member list with the Toolkit commands Edit Member List and Filter Member List (YEDTMBRLST, YFLTMBRLST).

Re-compilation Order

If you are compiling source members of many different source types, the Create Object command will order the compilations so as to give the maximum chance of establishing logical dependencies. For example, physical files will be compiled before logical files, database files will be compiled before device files, commands will be compiled before CL programs. Refer to the CRTORD parameter on the command diagram for further details.

Generic Change Ownership

The Toolkit generic command, Change Object Ownership (YCHGOBJOWN), enables you to change the ownership of many objects, by means of a single instruction. All rights of the existing owner can be revoked at the same time. You may also change the ownership of the library containing the objects at the same time.

For instance, the following single instruction would change the ownership to QPGMR of all programs in library QGPL, as well as the ownership of the library itself, and furthermore would revoke the previous owner's rights:

```
YCHGOBJOWN OBJ(QGPL/*ALL) OBJTYPE(*PGM) NEWOWNER(QPGMR) REVOKE(*YES)
CHGLIBOWN(*YES)
```

Generic Remove Member

The Toolkit generic command, Remove Member (YRMVM), enables you to remove some or all of the members in a file, or files, by means of a single instruction. This can be particularly useful for tidying up work files.

For instance, the following single instruction would build a list of all members in file QTXTSRC in library QGPL, display the list for editing and confirmation, and then remove all the members in the edited list:

```
YRMVM FILE(QGPL/QTXTSRC) MBR(*ALL) EDIT(*YES)
```

Generic Delete Object

The Toolkit generic command, Delete Object (YDLTOBJ), enables you to delete a list of objects, by means of a single instruction.

For instance, the following single instruction would build a list of all files in library QGPL, display the list for editing and confirmation, and then delete all the objects in the edited list.

```
YDLTOBJ OBJ(QGPL/*ALL) OBJTYPE(*FILE) EDIT(*YES)
```

Generic Duplicate Object

The Toolkit generic command Create Duplicate Object (YCRTDUPOBJ) enables you to duplicate list of objects into several libraries at a time, by means of a single instruction. The ability to duplicate into several libraries is especially useful when there is a requirement to keep several parallel libraries up to date: for instance, a library of live data plus several alternate versions of test data.

For instance, say that you wish to add a new file and its attendant logical views into three libraries: DTALIB, TSTLIB1 and TSTLIB2. You could build a list of the files, then the following command would duplicate the files in the list:

```
YCRTDUPOBJ OBJ(*OBJLST) FROMLIB(*OBJLST) OBJTYPE(*ALL)     TOLIB(DTALIB TSTLIB1 TSTLIB2)
```

Generic Command Change

The Toolkit generic command, Change Command (YCHGCMD), enables you to change the attributes of one or more commands by means of a single instruction. The attributes which you may change include the product library (PRDLIB), mode (MODE), environment allowed (ALLOW) and text (TEXT).

For instance, the following single instruction would build a list of all commands in library QGPL**,** display the list for editing and confirmation, and then change the product library of the commands to be QGPL.

```
YCHGCMD CMD(QGPL/*ALL) PRDLIB(QGPL) EDIT(*YES)
```

Debug Aids

This section describes CA 2E Toolkit Debug Aid concepts.

There are five Toolkit aids specifically concerned with helping you to debug programs:

- Work with database file utility
- Start debug utility
- Copy a list of files utility
- Break message program
- Display program message queue utility

Compile Preprocessor

This chapter describes the Toolkit compile preprocessor including installing and invoking instructions, required routing data, and information on the source directives.

Edit Aids

The Toolkit Edit Aids enable you to carry out certain commonly required functions more quickly and more accurately. The Toolkit Edit Aids can be divided into three categories:

- Display and change commands
- Source manipulation commands
- Special display commands

Chapter 5: Documentation Utilities

This chapter provides an introduction to Toolkit documentation aides and a brief overview of documentation commands and utilities.

This section contains the following topics:

Introduction

The Toolkit documentation aids are based on simple fourth generation principles:

- Self-documenting systems:

  - Keep documentation on the computer as far as is possible.

  - Use the computer both to edit and to organize the documentation. Tools should be available to search and manipulate the documentation.

  - Make documentation available in the correct context: for example, help text for the displays of a program should be available when using the program; program descriptions should be available in the program source.

  - Make documentation complete and sufficient: it should be possible to re-create a system from source without the addition of further information.

- Structured documentation:

  - Keep only one version of the documentation. The computer can be used to organize it into the various permutations that will be required.

  - Different users will have different needs: for example a system analyst needs to see a different level from a programmer. Ideally the different needs will be supplied by different routes through the same underlying data. Each level should contain information possessing a degree of detail appropriate to the level.

  - Keep documentation to a minimum: this requires the use of standard conventions for common types of documentation.

i OS provides you, in profusion, with the basic information needed to document a system. The Toolkit utilities help you to structure the information into a "documentation pyramid", that has different levels of access.

Note that as far as is possible, Toolkit documentation utilities enable you to generate documentation as a ‘by product’ of your normal activities.

Documentation

The Toolkit Documentation Aids help to improve your documentation in several important ways:

- By providing documentation utilities that will assemble up-to-date descriptions of your actual systems, collated with the relevant descriptive text.
- By providing computer-generated cross-references to index your documentation.
- By providing a method of extracting selected comments from program source into a higher level summary.
- By providing a method of storing information about additional object attributes in the source. For details of this facility, see the section Compile Preprocessor in the chapter "Programmer Aids".
- By providing utilities to present on-line Help text. For further details, see the section on help text in the "User Access Aids" chapter.
- By providing good ‘hard copy’ for all of the Toolkit utilities: for example, menu, panel and report designs, and object, database file and member lists.
- By providing standards that can be used to streamline and structure documentation. The importance of this last point should not be underestimated: a good standard structure removes the need for repetition of information already given by the context.

Your company name can be made to appear on all documentation.

Overview of Documentation Aids

The Toolkit Documentation Aids are discussed here under the following headings:

- System documentation commands, including cross-reference utilities
- Documentation commands for the Toolkit design utilities
- Documentation commands for the other Toolkit utilities

All of the documentation utilities are invoked by commands, so they can be used by your own utilities.

System Documentation Commands

Toolkit includes an extensive range of commands for generating documentation about your existing systems. Information from many disparate sources is collated into a concise and readable format.

Design Documentation Utilities

The different components of the Toolkit design utilities - menus, panels and reports - each have a print command, which generates listings suitable for inclusion in a formal system specification.

Other Documentation Utilities

Commands are supplied to print the information created and stored by all of the Toolkit utilities: for example, the contents of library, member, object, format and database file lists, and user profile extension attributes.

System Documentation

Toolkit provides a comprehensive range of utilities to document your application systems. In particular:

- The Toolkit Summary documentation commands, Document Program (YDOCPGM) and Document File (YDOCF), provide compact, pertinent documents, that describe the salient features of programs and files for reference purposes.
- The Toolkit Cross-reference documentation commands provide concise descriptions of how the entities in your system connect.
- The Toolkit Print key conversion commands (YSTRCVTPRT, YCVTPRT) provide an easy method of obtaining illustrations of actual panel displays for inclusion in your user instruction manuals.
- The Toolkit Source scan commands (YSCNSRC, YSCNRPLSRC) provide listings of source lines containing a specified search string.

The Toolkit List utilities can be used to help with system documentation; for example, to record configuration details.

General Features of the Documentation Utilities

The documentation utilities have the following general features:

- The documentation commands can all be run generically on whole systems.
- Your company name can be made to appear on all documentation.
- The utilities are all command driven and may be invoked either interactively or in batch.

General Considerations for System Documentation

Refer to the following command diagrams to create documentation:

- YDOCAUT
- YDOCEXCREF
- YDOCF
- YDOCMSGREF
- YDOCPGM
- YDOCPGMREF
- YDOCSRC
- YDOCSTRCVTPR

Your Company Name on Documentation

The company name that appears on all system documentation is taken from a data area called YYCOTXA. A copy of the data area is shipped with the system in the Toolkit library. You may change the value of the shipped data area, create additional versions in other libraries with different values, or both, by using the i OS Create Duplicate Object command (CRTDUPOBJ), for instance:

```
CRTDUPOBJ OBJ(YYCOTXA) FROMLIB(Product-library) OBJTYPE(*DTAARA) TOLIB(QGPL)
```

The company name data area can conveniently be changed using the Toolkit command Edit Data Area (YEDTDTAARA) which will display the existing value and allow you to override it:

```
YEDTDTAARA DTAARA(YYCOTXA)
```

A second data area, YYSYTXA, determines the application system name; the data area is used by the Toolkit Create Design DDS commands (YCRTPNLDDS and YCRTRPTDDS) when generating program banners. The data area is also used by the Toolkit Create Source File command (YCRTSRCPF) as a default, when generating descriptive text for new source files.

Toolkit File Documenter

The Toolkit Document File utility (YDOCF) provides a compact listing of all the relevant information about a file. The utility works on database files, device files, or both. Included in the listing generated by the utility are the following items of information:

- Object information
- Access path information, including key fields
- Join file information, including join fields
- Source member information
- Format information
- Database dependency information
- Field information, optionally including offsets, and definition references
- Additional field explanation comments, extracted from the field dictionary

Examples of the output of the YDOCF command are shown in the following two illustrations.

Two levels of list detail are allowed: a basic level (80 characters per line), suitable for inclusion in system specifications, and a detailed level (132 characters per line), suitable as a reference document for programmers.

The command may be run on a file, a generic file name, or on a list of files. For instance, the following command would document all of the physical files in a library:

```
YBLDOBJLST OBJ(QGPL/*ALL)  OBJTYPE(*FILE) /* build list  */
YDOCF FILE(*OBJLST) FILEATR(*PHY) /* Document it */
```

Toolkit Program Documenter

The Toolkit Document Program utility (YDOCPGM) provides a summary description of a program or command, including all relevant linkage information. The level of detail of the YDOCPGM output is ideal for bridging the low level information of source listings, and the high level view of system flowcharts and overviews. The program documenter brings together information from a wide variety of sources, and collates it into an easily assimilable form. Documentation created by the program documenter includes:

- Object information (from object)
- Functional synopsis (extracted from source)
- Source member information (from source member)
- File/format used information (from object)
- Sub-programs used information (from source analysis)
- Entry parameters required (from source analysis)
- "Where called from" information (from source analysis)
- Maintenance notes (extracted from source)
- Warning notes (extracted from source)

Examples of the output of the YDOCPGM command are shown in the next three illustrations.

Documentation Source Directives

Program documentation can include narrative descriptions of what the program does, and why maintenance has been carried out. The narrative descriptions are extracted from special comment lines in the source: they thus stand a good chance of staying up to date.

The special comment lines can be used in all types of source. Four types of documentation comment lines are catered for:

- *H Header source directives
- *M Maintenance source directives
- *T Title source directives;documentation
- W Warning source directives

Program descriptions can be written ahead of the program, so that they serve as program specifications. If programmers fail to provide even a basic description of their programs, the program documenter makes the fact obvious, and so helps you control the quality of your documentation.

For example, the marked comments from the CL program shown following in the first diagram, would be summarized as shown in the diagram that follows it:

Toolkit Program/File Cross-Reference Documentation

The Toolkit Document Program References utility (YDOCPGMREF) provides a cross-reference of which data objects are referenced by which programs, and vice versa.

This allows you to answer, questions such as: "Which programs would be affected if I changed this file?", "Which files does this program use?", "Which files are not used anywhere?".

Options include:

- List by file
- List by format
- List by program
- List unreferenced files

Examples of the output of the YDOCPGMREF command are shown in the following four illustrations.

Execution Cross-Reference Documentation

The Toolkit Document Execution References utility (YDOCEXCREF), provides a cross-reference of which programs, and which commands, are referenced by which other programs, commands, and Toolkit menus. Output is available in both tabular and "explosion" form - the latter provides an across-the-page resolution of references.

This allows you to answer questions such as: "Which other programs would be affected, if I were to change the entry parameters of this program?", "Which sub-programs are needed to run this menu option successfully?".

Options include:

- List by calling object
- List by referenced object
- Explosion listing by calling object
- Explosion listing by referenced object

The utility brings together source, object, and menu file references into a single uniform listing.

Examples of the output of the YDOCEXCREF command are shown in the next four illustrations.

Field Cross-Reference Documentation

The Toolkit Document Field References utility (YDOCFLDREF) provides a cross-reference of field usage by file.

Options include:

- Specification of file name, or generic file name, and file attribute, for inclusion in listing
- Specification of one library, or several libraries

Examples of the output of the YDOCFLDREF command are shown in the following illustration:

Message Cross-Reference Documentation

The Toolkit Document Message References utility (YDOCMSGREF) provides a cross-reference of which messages are referenced by which programs and commands. The command works by analyzing the source of the programs and commands for any use of messages - both impromptu and externally-defined. Refer to the YDOCMSGREF command diagram for details about the rules used to identify messages in source.

This allows you to answer questions such as: "Does a similar message description exist?", "Which messages does this program used?", "Where is this message used?".

Options include:

- List messages by program
- List programs by message
- List messages in alphabetical order
- List unreferenced messages

Examples of the output of the YDOCMSGREF command are shown in the following four illustrations.

Fig L29 - Example of YDOCMSGREF output, part 3

```
Universal Sprocket Company      UNREFERENCED MESSAGES   Op:      FRED  WRKSTN1   04/11/87  16:08:17  Page:  1

Report request details:

                    Source file:   *MBRLST
                  Source Member:   *ALL
                     Message id:   *ALL
                User message file: *LIBL/QUSRMSG
              Prompt message file: *LIBL/PROMPTMSG
      List messages by program...: *YES
      List programs by message...: *YES
      List unreferenced messages:  *YES
                List message file: *YES
    Ignore Impromptu messages:     *NO
      Ignore message variables:    *NO
                  Member list:     PERMLST
                      Library:     PRUP
                                   123456789-123456789-123456789-123456789-123456789-123456789-123456789-123456789-

Message                            Text

CPF5225   *LIBL/QUSRMSG            Value required.
CPF6801   *LIBL/QUSRMSG            Command cancelled.
CPF9801   *LIBL/QUSRMSG            Object &2.&3 not suitable for specified operation.
USR0007   *LIBL/QUSRMSG            File &1 member &2 not found.
USR0006   *LIBL/QUSRMSG            Document processed for &1.&2 member &3.
USR0000   *LIBL/QUSRMSG            ====>OTHER DOCUMENTATION
USR0001   *LIBL/QUSRMSG            Source member analysis complete.

*** END OF REPORT ***
```

Authorization Cross-Reference Documentation

The Toolkit Document Authorities utility (YDOCAUT) provides a compact cross-reference of object authorities, owners, and users.

Options include:

- Selection of object name, object type, object owner, user, or both, for inclusion in listing
- List authorizations by object name
- List authorizations by object owner
- List authorizations by user profile

Examples of the output of the YDOCAUT command are shown in the next three illustrations.

Source Documentation

The Toolkit Document Source utility (YDOCSRC) provides compact source listings of a list of source members.

Features include:

- Optional indentation of RPG III source
- Can run in batch
- Can be list driven

Source Scan Documentation

The Toolkit source search commands Scan Source (YSCNSRC) and Scan and Replace Source (YSCNRPLSRC) can provide listings of all source lines containing a specified search string, as well as a Toolkit member list naming all of the source members containing the search string.

Examples of the print output of the YSCNSRC command are shown in the following illustration. Refer to the respective command diagrams for further details.

Print Key Conversion

The Toolkit Print Key Conversion utility provides an easy method of obtaining illustrations of actual screen displays for inclusion in user documentation.

The convert print command "frames" and labels each display print.

The ensuing text source member can be edited and included into any document, using i OS's SEU utility (STRSEU).

Print Key Conversion Steps

Using the Convert Print utility is a three-step process:

1. You enter "print conversion mode" by means of the Toolkit command Start Print Conversion (YSTRCVTPRT). The command allows you to nominate up to four print files for conversion; for example:

   YSTRCVTPRT PRTF(QSYSPRT YPRTKEY$)

   Any output to the nominated files will be trapped for conversion. Starting print key conversion mode invokes the i OS command entry program QCMD, thus allowing you to call the commands or programs whose displays you wish to document.

2. You create the print output for the displays that you want illustrated. This can be done by pressing the PRINT key while displaying any screen that is to be illustrated: provided the print file used by the print key is one of the nominated print files.

3. You convert the print output into a text source member by means of the Toolkit command Convert Prin*t* (YCVTPRT), for example:

   YCVTPRT PRTF(*ALL) SPLNBR(*ALL) FILE(QTXTSRC) MBR(FRED)

This command would convert all print key output to a spool file member FRED in file QTXTSRC.

The following commands would produce the illustration shown next:

YSTRCVTPRT /* Start print conversion. */

STRPGMMNU /* Call programmer's menu. */

Type in (for example) "YGO *Y" and press PRINT key.....

YCVTPRT /* Convert print output to source. */

You may control the characters used to form the frame by using the Toolkit Edit Design Defaults command (YEDTDSNDFT).

Documenting Designs

Each of the Toolkit design objects has an associated print command. The print output from the commands to document Toolkit menus, Toolkit panel designs, and Toolkit report designs is designed to be compact, comprehensive, and tidy. The prints of the designs can be bound up directly, to form a complete system specification: narrative text is inserted alongside the relevant designs, and indices will be generated automatically.

General Considerations for Documenting Designs

Following are commands to document Toolkit designs

Design type	Print command
Menu	YDOCMNU
	YDOCMNUREF
Panel design	YDOCPNL
Report design	YDOCRPT
Text & Help text	YDOCSRC

Your on-line operator instructions can be printed to form an instruction manual.

Documenting Menus

The Toolkit command Document Menu (YDOCMNU) will print a menu, or menus. The output will fit on 80 column (A4) paper.

Options include:

- An index of menus
- An image of the menu (suitable for including in user instruction manuals)
- A record of the actions invoked by each menu option

The Toolkit command Document Menu References (YDOCMNUREF) prints all menu usage by option.

Documenting Panel Designs

The Toolkit command Document Panel Design (YDOCPNL) will print a panel design, or range of panel designs, in a specified print order. The output will fit on 80 column (A4) paper.

Fig L.11 - Example of YDOCPNL output

Universal Sprocket Company SYNON Panel design utility 03/04/87 22:00

DISPLAY CURRENT STOCK BY SHOP ADSPSHPSTK 1.50

```
                                                    66/66/66/66:66.66  0000000000
DISPLAY    CURRENT    STOCK BY SHOP                 0000000000000000000000000000000

?    Shop  Stock            Stock  Stock
P    Stock No    Description Quantity  Value

!!!!  !!!!!!

B    0000  000000  00000000000000000000000000000  666666666-  66666666.66-
B    0000  000000  00000000000000000000000000000  666666666-  66666666.66-
B    0000  000000  00000000000000000000000000000  666666666-  66666666.66-
B    0000  000000  00000000000000000000000000000  666666666-  66666666.66-
B    0000  000000  00000000000000000000000000000  666666666-  66666666.66-
B    0000  000000  00000000000000000000000000000  666666666-  66666666.66-
B    0000  000000  00000000000000000000000000000  666666666-  66666666.66-
B    0000  000000  00000000000000000000000000000  666666666-  66666666.66-
B    0000  000000  00000000000000000000000000000  666666666-  66666666.66-

  CMD:  1-Exit 2-Previous  Sel:  P-Print,  X-Batches,  L-Lots.
```

Field type/Usage :	Input	Output	Both	'+	Indicates subfile capability
Alphameric :	1	0	8		via use of roll keys.
Numeric :	3	6	9		

This screen provides an inquiry of current stock levels by shop
The stock value will be calculated at run time using the retail price in effect at the time (from file SLPRICAP) as follows:-

VALUE = STOCK QUANTITY X RETAIL PRICE

The calculation should be half adjusted.

Command key usage.

	Row	Col	?	Next panel	Text
01				*EXIT	Exit program
02				*PRV	Previous screen
03				TOTL	If confirmed, display totals screen
05				DSPCUST	Display customers
EN				*SAME	Validate details and redisplay.
RU				*SAME	Rollup.
VL	8	3	L	LOTS	If L, display lot screen.
VL	8	3	X	BATCHES	If X, display batch screen.

Options include:

- An index
- An image of the panel design
- A key explaining the symbols used in the panel design
- The narrative text associated with the panel design
- The branching conditions associated with the panel design

The Panel designs can include sample data in the fields. The sample data is set up with the Toolkit Display Panel Design utility (YDSPPNL) - see the "Design Aids" chapter of this manual.

Use of Panel Designs for Existing Systems

The Toolkit command Retrieve Panel Design (YRTVPNLDSN) allows you to retrieve a panel design from the existing DDS source of an externally- described display file. The facility can be used to document existing systems.

Documenting Report Designs

The Toolkit command Document Report Design (YDOCRPT), will print a report design, or range of report designs, in a specified print order

Options include:

- An index
- An image of the report design
- A key explaining the symbols used in the report design
- The narrative text associated with the report design

Use of Report Designs for Flowcharts

Because it allows for reports up to 198 characters wide, and because it has block move and edit functions, the Toolkit utility Work With Report Design **(**YWRKRPT) can also be used to design system diagrams such as flowcharts.

Use of Report Designs for Existing Systems

The Toolkit command Retrieve Report Design (YRTVRPTDSN) allows you to retrieve a report design from the existing DDS source of an externally- described print file. The facility can be used to document existing systems, which use external print files.

Documenting Utilities

This section describes the documentation available for the Toolkit utilities. Note that the documentation for the design utilities is described in a separate section of this manual.

Documenting Lists

Each of the Toolkit list types has an associated print command:

List type	Print command
Object list	YDOCOBJLST
Database file list	YDOCDBFLST
Member list	YDOCMBRLST
Format list	YDOCFMTLST
Library list	YDOCLIBLST

Documenting User Profiles

User profiles can be documented using the Toolkit Document User profile (YDOCUSRPRF) command.

The Document User profile command can be run generically, or on a single profile. The output includes details, in a compact format, of both i OS user profile attributes and the Toolkit extension attributes.

Chapter 6: Spooled File Utilities

The Toolkit ships with various command which let you work with and manipulate spooled files. This chapter contains an overview of these utilities.

Note: For detailed descriptions of these commands and their parameters, see the *Toolkit Reference Guide*.

This section contains the following topics:

Spooled File Router

The Spooled File Router allows you to predefine rules for spooled files created on your IBM i that control automatic processing occurring for those spooled files.

You can add rules as spooled file routing entries that select spooled files based on any combination of their output queue, name, user, the job that produced them, or their user-defined data attribute, and then process the spooled files that meet the specification criteria. You can apply any spooled file commands for processing. You can also specify that other commands run following the spooled file processing—such as notifying certain users when spooled files are processed.

For instance you can specify rules so that:

- Spooled files arriving on the QPRINT output queue with a USRDTA attribute of MYATTR, produced by user JOHN should be immediately converted into HTML files in the /YSPLFS/User/JOHN/ directory. In addition, user JOHN should be notified that the above processing has occurred.
- Spooled files arriving on the QSYS/QEZJOBLOG output queue, which were produced by user QSPLJOB, should be copied to a specified physical file member and then deleted.
- Any spooled files on any output queues, which have the USRDTA attribute as blank, should have the USRDTA attribute set to the name of the user who produced the spooled file.

You can use any or all of the above examples.

Spooled file routing entries are added, edited, and deleted with the YWRKSPLRTE (Work with Spooled File Routing Entries) command.

Once spooled file routing entries have been created, a spooled file router can be started by using the YSTRSPLRTR (Start Spooled File Router) command and ended using the YENDSPLRTR (End Spooled File Router) command. Additionally the YRSTSPLRTR (Restart Spooled File Router) command is available to restart the spooled file router, if spooled file routing entries have been added, changed, or deleted.

The Spooled File Router can be distributed to any of your IBM i systems, even if the machine in question does not have the full CA 2E Toolkit installed.

Convert Spooled File (YCVTSPLF)

The Convert Spooled File (YCVTSPLF) command lets you convert an IBM i spooled file to a PC file in text, HTML or PDF formats. You can then send the file by e-mail or other PC file transfer methods and manipulate the file by PC software. The YCVTSPLF command is a stand-alone command in the Toolkit product.

When more than one spooled file with the same name exists for the specified job, an interactive window appears so that you can select a spooled file. If duplicate jobs contain the specified spooled file, an error message is sent.

Conversion options are held in a file called YCVTOPTP (prior to 8.0 the options were held in YCVTSPP). This file allows you to specify not only user-specific options, but also file-specific options. When you specify file-specific options, you can specify whether you want header lines to be removed from the spooled file when it is converted. When converting a spooled file, you can now specify the special CVTOPT(*FILE) value, to indicate that you want to use the conversion options with the same name as the spooled file name. You can add or change conversion options by using the YCHGCVTOPT command, which includes on-line help.

You can copy the YCVPSPLF command to any of your IBM i machines, and run it there, even if the machine in question does not have the full CA 2E Toolkit product installed.

Retrieve Spooled File Attributes (YRTVSPLFA)

The YRTVSPLFA (Retrieve Spooled File Attributes) command can only be run from within a compiled CL program or module and allows the program to retrieve the attributes of a given spooled file. Similarly to IBM spooled file commands, the only required parameter is the spooled file name (although the job name and spooled file number are optional input parameters). All other parameters are output parameters that return values into defined CL variables. This command uses the QUSRSPLA API to retrieve the attributes for the specified spooled file.

The YRTVSPLFA prompt screen displays as follows:

```
                  Retrieve Spooled File Attrs (YRTVSPLFA)

Type choices, press Enter.

Spooled file name  . . . . . _____________   Name
Job name . . . . . . . . . . *_____________   Name, *
  User . . . . . . . . . . . ____________    Name
  Number . . . . . . . . . . ______          000000-999999
Spooled file number  . . . . . *ONLY_____    1-9999, *ONLY, *LAST
CL var for RTNJOBNAME   (10) . . _____________   Character value
CL var for RTNJOBUSER   (10) . . _____________   Character value
CL var for RTNJOBNBR     (6) . . _____________   Character value
CL var for RTNSPLNBR   (5 0) . . _____________   Number
CL var for FORMTYPE     (10) . . _____________   Character value
CL var for USRDTA       (10) . . _____________   Character value
CL var for STATUS       (10) . . _____________   Character value
CL var for AVAIL        (10) . . _____________   Character value
CL var for HOLD         (10) . . _____________   Character value
CL var for SAVE         (10) . . _____________   Character value
CL var for TOTALPAGES (15 0) . . _____________   Number

                                                                More...
```

Chapter 7: Technical Considerations

This chapter provides some additional technical information about the CA 2E Toolkit utilities including, installation instructions; system authorization, storage, and backup requirements; print output information; database file descriptions, international character set considerations; and CL commands.

This section contains the following topics:

Installing the Utilities

Instructions for installing the Toolkit utilities are supplied with the Toolkit software. Read the instructions carefully before installing the utilities.

Some versions of Toolkit contain copy protection and will only run on CPUs that are authorized. Although all of the Toolkit utilities are shipped to you, only the modules of Toolkit that you have purchased will be authorized for use. To authorize your copy of the software to run on your CPU you must use the Toolkit command Grant Product Authorization (YGRTPRDAUT) as described in the installation instructions.

Displaying Expiry Date

You may check which modules you are authorized to use and your expiry date by using the Toolkit command Display Expiry Date (YDSPEXPDAT).

System Authorization Requirements

To use a Toolkit utility you must be authorized to each i OS command that the utility uses.

The Toolkit utilities do not give users the right to do anything that they might not otherwise be allowed to do.

For instance, a user who is not authorized to use the Create Physical File command (YCRTPF), will not be authorized to use the Toolkit command Create Design File (YCRTDSNF). The shipped system authorities are sufficient to allow most users to use the Toolkit utilities.

Storage Requirements

The complete Toolkit package takes about 18 MB of disk storage. The package is contained within two libraries, Y1SY and Y1SY38**.** About 3.4 MB of the storage is required for the interactive help text, including command diagrams. If you are short of storage, you may delete the help text without affecting the performance of any of the utilities. This can be done as follows:

```
DLTF FILE(Y1CMDTXT)        /* to delete command diagram text*/
DLTF FILE(Y1HLPTXT)        /* to delete Help text */
```

Backup Requirements

There are several Toolkit objects whose data contents you are likely to change as a result of using the Toolkit utilities, and which therefore may require saving onto an off-line storage medium.

Backup Strategies

Here are four different approaches to backing up Toolkit.

1. Regularly save the entire library Y1SY**:**

   SAVLIB LIB(Y1SY)

   This would allow for recovery in a single step - but requires the unnecessary saving of objects that you have not changed.

2. Save only the items that you have changed since the last full save of the library:

   SAVCHGOBJ OBJ(*ALL) LIB(Y1SY) REFDATE(*SAVLIB)

   This is efficient, but requires two steps to recover: the restoration of the utilities, followed by the restoration of the changes.

3. Move any objects in Y1SY that require saving into other libraries that are already covered by your backup regime.

4. Explicitly save just those objects in Y1SY that require backing up.

If you adopt either of the last two approaches, you will need to bear in mind the backup considerations described below.

Backup Considerations

The Toolkit objects that might require saving can be divided into three categories:

- Design
- List
- System value

Design objects

The following table shows design objects:

Object	*Type	Description	Create command
YDSNMNU	*FILE	Menu design file	YCRTDSNF
YDSNPNL	*FILE	Panel design file	YCRTDSNF
YDSNRPTA	*FILE	Report design file 80	YCRTDSNF
YDSNRPTB	*FILE	Report design file 132	YCRTDSNF
YDSNRPTC	*FILE	Report design file 190	YCRTDSNF
YYSYTXA	*DTAARA	System text file	YCRTDUPOBJ
YMHPFLA	*DTAARA	Name of Help text file	YCRTDUPOBJ
YMHPLBA	*DTAARA	Help text file library	YCRTDUPOBJ

List objects

The following table shows list objects:

Object	*Type	Description	Create command
YOBJLST	*FILE	Object list	YBLDOBJLST
YMBRLST	*FILE	Member list	YBLDMBRLST
YDBFLST	*FILE	Database file list	YBLDDBFLST
YFMTLST	*FILE	Format list	YBLDFMTLST

System value objects

The following table shows system value objects:

Object	*Type	Description	Create command
YLIBLST	*FILE	Library list file	
YUSRPRF	*FILE	User profile file	
YYCOTXA	*DTAARA	Company text	CRTDUPOBJ
YDSCDFA	*DTAARA	Panel design defaults	
YDRPDFA	*DTAARA	Report design defaults	
YDSCDCA	*DTAARA	DDS design defaults	
YPEXCHA	*DTAARA	Substitution character	YEXCxxxLST
YPBXCHA	*DTAARA	Frame characters	YCVTPRT
YPLGOQA	*DTAARA	Name of job log queue	YDSPABR

Different considerations apply to each category of object:

Design objects: you are likely to have several different sets of the design files in different libraries, which will be saved as part of the back-up regimes for those libraries. You will probably keep at least one set of design files per application and it is quite possible that you will not actually keep any data in the versions of the design objects in the Y1SY library. The design files can be regarded as being similar in concept to source files such as QCLSRC and QRPGSRC - and similar considerations apply to using them.

List objects: most of the lists you use will probably be temporary ones stored in QTEMP. When you do create a permanent list you will probably want to create it in a named library other than Y1SY**.** Thus it is again quite possible that you will not keep any data in the versions of the list objects in the Y1SY library.

System parameter values: You will probably want to keep only one version of the objects containing system values. It may be very important that the objects, especially the library list and user profile file, are saved correctly and frequently. The system parameter objects are therefore prime candidates for either moving to a library that you already save regularly (e.g. QGPL), or for saving explicitly by means of the i OS Save changed objects command (SAVCHGOBJ).

Print Output

You may want to change the print file attributes for the print files in the Y1SY library, forms length, lines per inch, etc. This can be done for all files using the command Change Print File (CHGPRTF):

```
CHGPRTF FILE(Y1SY/*ALL) CPI(8) LPI(15)
```

Database File Descriptions

To find out the file descriptions of any of the Toolkit database files used by the Toolkit utilities, such as the user profile file (YUSRPRF), the menu file (YDSNMNU), or the library list file (YLIBLST), run the Toolkit Document File utility (YDOCF) on the file.

For example, to obtain the layout of the Toolkit user profile file:

```
YDOCF FILE(YUSRPRF)
```

International Character Set Considerations

Certain Toolkit commands involve the use of special characters, which may not be easily supported by your National language character sets. You may change the default special values used by some Toolkit commands.

CL Commands

There are seven commands in Toolkit which are intended mainly for internal use by Toolkit, but which may also be of use in your own CL programs. Refer to the respective command diagrams for further details. The commands are as follows:

- YRTVOBJLIB - A command to search the invoking job's library list for a given object and return the library in which it resides
- YCVTBIN - A command to convert a binary number to a decimal number
- YCHKVN - A command to check whether a character string is a valid i OS system name.
- YCHKLIBLST - A command to check whether a library list exists in a given Toolkit library list file
- YCHKMNU - A command to check whether a menu exists in a given Toolkit menu file
- YCHKPNL - A command to check whether a panel design exists in a given Toolkit panel design file

YCHKRPT - A command to check whether a report design exists in a given Toolkit report design file

Appendix A: Toolkit Supply Objects

This appendix describes Toolkit supply objects that you may wish to know more about.

Libraries

Object Name	Description	Crt. Cmd.	Parameters
Y1SY	Toolkit Utility library.	CRTLIB	TEXT('Toolkit Utility + library')
Y1SY38	Toolkit S/38 Utility library.	CRTLIB	TEXT('Toolkit S/38 + Utility library')

Database Files

Object Name	Description	Crt. Cmd.	Parameters

YDSNMNU	Menu file.	YCRTDSNF	TYPE(*MENU) + OPTION(*CREATE) +
YDSNPNL	Panel design file.	YCRTDSNF	TYPE(*PNL) + OPTION(*CREATE) +
YDSNRPT	Report design file.	YCRTDSNF	TYPE(*RPT) + OPTION(*CREATE) +
YOBJLST	Object list file.	YBLDOBJLST	
YMBRLST	Member list file.	YBLDMBRLST	
YDBFLST	Dbf list file.	YBLDFMTLST	
YFMTLST	Format list file.	YBLDFMTLST	
Y1HLPTXT	Help text.	CRTSRCPF	TEXT('Toolkit help text')
Y1CMDTXT	Command diagrams.	CRTSRCPF	TEXT('Toolkit menu text')
Y1USRTXT	Help text for YGO/YDSPHLP	CRTSRCPF	TEXT('YGO/YDSPHLP help text')
YLIBLST	Library list file.	CRTPF	TEXT*Toolkit library + lists')
YUSRPRF	User profile file.	CRTPF	TEXT('Toolkit User + profile extensions')
YPWDVAL	Password value file.	CRTPF	TEXT('YCHGPWD forbidden + passwords')
Y1USRSRC	User source file.	CRTSRCPF	TEXT(Toolkit User src')

Programs

Object Name	Description	Crt. Cmd.	Parameters
YINLPGM	Initial program.	CRTCLPGM	(1)
YINLPGMPWD	Password expiry chk	CRTCLPGM	(1)
YPWDVALPGM	Password validation	CRTCLPGM	(1)
YBRTPRC	Compile	CRTRPGPGM	
YDDSHPR	preprocessor	CRTRPGPGM	
Y1PGMSC	Help text display. Message sender.	CRTCLPGM	(1)

Message Files

Object Name	Description	Crt. Cmd.	Parameters
YYYYMSG	Toolkit messages.	CRTMSGF	MSGF(YYYYMSG) +
Y1USRMSG	Toolkit messages for execution of end user pgms.	CRTMSGF	TEXT('Toolkit messa ges') MSGF(Y1USRMSG) +
Y1USRPMT	Toolkit messages for compilation of end user objects.	CRTMSGF	TEXT('Toolkit messages + for end user programs') MSGF(Y1USRPMT) +
Y1PMTMSG	Toolkit device message	CRTMSGF	TEXT(Toolkit messages + for end user objects) MSGF(Y1PMTMSG) + TEXT(Toolkit messages)

Data Areas

Object Name	Description	Crt. Cmd.	Parameters
YMHPFLA	Default help file name	CRTDTAARA	DTAARA(YMHPFLA) + TYPE(*CHAR) LEN(10) + VALUE('QTXTSRC') TEXT('Default help file + name')

YMHPLBA	Default help file library name	CRTDTAARA	DTAARA(YMHPLBA) + TYPE(*CHAR) LEN(10) + VALUE('*LIBL ') TEXT('Default help file + library name')
YMHPOPA	Default help display	CRTDTAARA	DTAARA(YMHPOPA) + TYPE(*CHAR) LEN(10) + VALUE('*USRPRF ') TEXT('Default help display + ???')
YMHPYKA	Default Keys help label	CRTDTAARA	DAARA(YMHPYKA) + TYPE(*CHAR) LEN(10) + VALUE('FUNCK ') TEXT('Default keys help + label')
YMHPYWA	Default window size	CRTDTAARA	DAARA(YMHPYWA) + TYPE(*CHAR) LEN(7) + VALUE('70 15 Y') TEXT('Default wind ow + size')
YPBXCHA	Frame characters for YCVTPRT command	CRTDTAARA	DTAARA(YPBXCHA) + VALUE('| |— . . ' ') TEXT('Frame characters')
YPLGOQA	Name of job log output queue	CRTDTAARA	DTAARA(YPLGOQA) + TYPE(*CHAR) LEN(10) + VALUE('*LIBL ') TEXT('Job log output + queue name)

YPEXCHA	Alternative substitution character for YEXCxxxLST commands	CRTDTAARA	DTAARA(YPEXCHA) + VALUE(' ') TEXT('Alternative + substitution character')
YWWDBDA	Window borders and attributes	CRTDTAARA	DTAARA(YWWDBDA) + TYPE(*CHAR) LEN(10) + VALUE(' B..::..::') + TEXT('Window + borders and attributes')
YYCOTXA	Company name	CRTDTAARA	DTAARA(YYCOTXA) + TYPE(*CHAR) LEN(40) + VALUE('Company name') TEXT('Company name for + displays and reports')
YYFXNOA	Fix level number	CRTDTAARA	DTAARA(YYFXNOA) + TYPE(*DEC) LEN(4) + TEXT('Toolkit Fix') level number')
YYSYTXA	Application system text.	CRTDTAARA	DTAARA(YYSYTXA) + TYPE(*CHAR) LEN(30) + VALUE('Application name') TEXT('Application name + for reports')
YYPWCKA	Password checking parameters	CRTDTAARA	DTAARA(YYPWCKA) + TYPE(*CHAR) LEN(100) + VALUE('0040Y') TEXT('Password control + values')

Appendix B: Supplied Device Files

This appendix describes the device files used by the Toolkit utilities.

Print Files

File Name	Type	Description	Command
YCMPSRC$	PRTF	Compare source	YCMPSRC
YCVTCKY$	PRTF	Convert DDS command keys	YCVTDDSCKY
YDBFLST$	PRTF	Database file list	YDOCDBFLST
YDOCF$	PRTF	File documentation	YDOCF
YDOCKEY$	PRTF	Command key conversion	YEDTCKYTBL
YDOCMNU$	PRTF	Toolkit menu documentation	YDOCMNURYD
YDOCMNUX$	PRTF	Document menu references	OCMNUREF
YDOCPGM$	PRTF	Program documentation	YDOCPGM
YDOCPNL$	PRTF	Toolkit Panel designs	YDOCPNL
YDOCRPT$	PRTF	Toolkit Report designs	YDOCRPT
YDOCSRC$	PRTF	Source listing	YDOCSRC
YDOCUSR$	PRTF	Document user profiles	YDOCUSRPRF
YEXPBYOBJ$	PRTF	Explode references by object	YDOCEXCREF
YEXPBYREF$	PRTF	Explode objects by reference	YDOCEXCREF
YFILBYPGM$	PRTF	Files by program	YDOCPGMREF
YFILNOREF$	PRTF	Files without a reference	YDOCFLDREF
YFLDREF$	PRTF	Fields by file	YDOCPGMREF
YFMTLST$	PRTF	Format lists	YDOCFMTLST
YLIBLST$	PRTF	Library lists	YDOCLIBLST
YMBRLST$	PRTF	Member lists	YDOCMBRLST
YMSGBYPGM$	PRTF	Messages by program	YDOCMSGREF
YMSGFIL$	PRTF	Document messages references	YDOCMSGREF
YMSGNOREF$	PRTF	Unreferenced messages	YDOCMSGREF
YOBJAUT$	PRTF	Document authority	YDOCAUT
YOBJBYREF$	PRTF	Objects by reference	YDOCEXCREF
YOBJLST$	PRTF	Object lists	YDOCOBJLST
YOWNAUT$	PRTF	Authority by owner	YDOCAUT
YPGMBYFIL$	PRTF	Programs by file	YDOCPGMREF
YPGMBYFMT$	PRTF	Programs by format	YDOCPGMREF
YPGMBYMSG$	PRTF	Programs by messages	YDOCMSGREF
YPRTKEY$	PRTF	Print key file for print key output	All int pg
YREFBYOBJ$	PRTF	Execution references by object	YDOCEXCREF
YSCNRPL$	PRTF	Source scan/replace report	YSCNRPLSRC
YSCNSRC$	PRTF	Source scan report	YSCNSRC
YUSRAUT$	PRTF	Authorizations by user	YDOCAUT
YWRKF$	PRTF	Work with file	YWRKF

Display Files

File Name	Crt. Cmd.	Description	Command
YDDSHPR# (1)	CRTDSPF	FILE(YDDSHPR#) DFRWRT(*YES) SRCFILE(Y1USRSRC) RSTDSP(*YES)Text('YDSPHLP display help text.')	YDSPHLP
YDMNGOI# (1)	CRTDSPF	FILE(YDMNDSI#) DFRWRT(*YES) SRCFILE(Y1USRSRC) RSTDSP(*YES) Text('YGO display menus')	YGO

Appendix C: Authority Required for Object

This appendix contains information about the authorities required to use the Toolkit commands.

Menu

Command	Object rights			Data rights				Additional Rights
	Opr	Mgt	Exs	Read	Add	Upd	Del	
YCPYMNU	X	X		X	X	X	X	READ on lib
YDOCMNU	X			X				READ on lib
YGO	X			X				READ on lib OBJOPR on CRTPF
YRNMMNU	X	X		X	X	X	X	READ on lib
YDLTMNU	X	X		X	X	X	X	READ on lib
YWRKMNU	X			X	X	X	X	READ on lib

Design Files

Command	Object rights			Data rights				Additional Rights
	Opr	Mgt	Exs	Read	Add	Upd	Del	
YCRTDSNF	X	X	X	X				EXIST on obj
YADDDSNFM	X	X	X	X				EXIST on obj
YEDTDSNDFT	X	X	X	X				EXIST on obj

Panel Designs

Command	Object rights			Data rights				Additional Rights
	Opr	Mgt	Exs	Read	Add	Upd	Del	

Command	Object rights		Data rights				Additional Rights
YCPYPNL	X	X	X	X	X	X	READ on lib
YDSPPNL	X		X		X		READ on lib
YDOCPNL	X		X				READ on lib
YWRKPNL	X		X	X	X	X	READ on lib OBJOPR on CRTPF
YRNMPNL	X	X	X	X	X	X	READ on lib
YDLTPNL	X	X	X			X	
YRTVPNLDSN YDFNPNLDSN	X		X	X	X	X	READ on source
YCRTPNLDDS	X		X				OBJMGT on source file.

Report Designs

Command	Object rights			Data rights				Additional Rights
	Opr	Mgt	Exs	Read	Add	Upd	Del	
YCPYRPT	X	X		X	X	X	X	READ on lib
YDOCRPT	X			X				READ on lib
YWRKRPT	X			X	X	X	X	READ on lib OBJOPR on CRTPF
YRNMRPT	X	X		X	X	X	X	READ on lib
YDLTRPT	X	X		X	X	X	X	READ on lib
YCRTRPTDDS	X			X				OBJMGT on source
YRTVRPTDDS	X			X	X	X	X	READ on source

Password Values

Command	Object rights			Data rights				Additional Rights
	Opr	Mgt	Exs	Read	Add	Upd	Del	
YCHKPWDVAL	X			X				READ on lib
YCHKPWDVAL	X			X	X	X	X	OPR+READ only to display values
YCHGPWD	X			X	X	X	X	Enabled by pgm adoption

Lists

Command	Object rights			Data rights				Additional Rights
	Opr	Mgt	Exs	Read	Add	Upd	Del	
YBLDxxxLST	X	X		X	X	X	X	READ on lib
YEDTxxxLST	X			X		X	X	READ on lib
YFLTxxxLST	X			X				READ on lib
YEXCxxxLST	X			X		X		READ on lib
YDOCxxxLST	X			X				
YRNMLST	X	X		X	X	X	X	
YCPYLST	X	X		X	X	X	X	
YCHGLST	X	X		X	X	X	X	
YCHKLSTE	X	X		X	X	X	X	
YDLTLST	X	X		X	X	X	X	
YOPRLST	X	X		X	X	X	X	READ on input lst
YINXLST	X			X	X	X	X	OBJOPR on CRTLF
YCVTDBR	X			X	X	X	X	
YCVTDBFLST	X			X	X	X	X	READ on YDBFLST
YCVTMBRLST	X	X		X	X	X	X	READ on YMBRLST
YBLDDOC	XRX	X		XRX	X	X	X	READ on YMBRLST
YDSPMBRLST								READ on lib & fil

Library Lists

Command	Object rights			Data rights				Additional Rights
	Opr	Mgt	Exs	Read	Add	Upd	Del	
YBLDLIBLST	X			X	X	X	X	
YCHGLIBLST				X	X	X	X	
YDLTLIBLST	X			X	X	X	X	
YEDTLIBLST	X			X	X	X	X	
YCPYLIBLST	X			X	X	X	X	
YRNMLIBLST				X	X	X	X	
YWRKLIBLST				X	X	X	X	
YCHGLIBL	X			X				UPD on jobd
YCHGJOBDLL	X			X				UPD on jobd
YRNMLIB	X			X	X	X	X	OBJMGT on LIB

Data Areas

Command	Object rights			Data rights				Additional Rights
	Opr	Mgt	Exs	Read	Add	Upd	Del	
YEDTDTAARA	X			X	X	X	X	CHGDTAARA
YEDTLDA	X			X	X	X	X	CHGDTAARA
YEDTGDA	X			X	X	X	X	CHGDTAARA

Object Manipulation

Command	Object rights			Data rights				Additional Rights
	Opr	Mgt	Exs	Read	Add	Upd	Del	
YCPYF	X	X	X	X	X	X	X	READ on from file
YCHGCMD	X	X	X					OBJOPR on journal
YDLTOBJ	X	X	X					READ on YOBJLST
YMOVOBJ	X	X	X					READ on OBJLST
YCHGOBJOWN	X	X	X	X	X	X	X	READ on YOBJLST
YCRTDUPOBJ	X	X	X	X				READ on YOBJLST OBJOPR on
YCRTOBJ	X	X	X	X				userprf READ on YOBJLST READ on YMBRLST
YRNMOBJSRC	X	X	X	X				READ on source ADD on lib
YMOVOBJSRC	X	X	XD	X				OBJOPR on jobd.
YRTVOBJLIB	X			XP255D				OBJEXIST on obj MGT on src file. OBJEXIST on obj OBJMGT on src file

Member Manipulation Commands

Command	Object rights			Data rights				Additional Rights
	Opr	Mgt	Exs	Read	Add	Upd	Del	
YMOVM	X	X	X	X	X	X	X	
YRMVM	X	X	X	X	X	X	X	
YSCNSRC	X			X				
YSCNRPLSRC	X			X		X		
YCMPSRC	X			X				
YDOCSRC	X			X				
YDSPHLP	X			X				
YWRKF	X			X	X	X	X	
YSTRDBG	X			X				OBJOPR on
YTDYRPGSRC	X			X		X		ENTDBG
YEDTCKYTBL	X			X	X	X	X	
YCVTDDSCKY	X	X	X	X	X	X	X	
YRTVMSGF	X	X	X	X	X	X	X	
YSTRCVTPRT	X							OBJOPR on MSGF
YCVTPRT	X	X		X	X	X	X	
YADDSRCM	X	X		X	X	X	X	

User Profiles

Command	Object rights			Data rights				Additional Rights
	Opr	Mgt	Exs	Read	Add	Upd	Del	
YDSPUSRPRF	X			X				
YCRTUSRPRF	X			X	X			Authority to use CRTUSRPRF cmd
YDLTUSRPRF	X			X			X	Authority to use DLTUSRPRF cmd
YCHGUSRPRF	X			X		X		Authority to use CHGUSRPRF cmd
YDOCUSRPRF	X			X				
YRTVUSRPRF	X			X				
YCPYUSRPRF	X			X	X	X		Authority to use CRTUSRPRF cmd
YRNMUSRPRF	X			X	X	X	X	Authority to use CRTUSRPRF and DLTUSRPRF cmds

Appendix D: Source Directives

This appendix lists all of the types of directives, and indicates which of the Toolkit utilities use them.

Overview

Several of the Toolkit utilities make use of directives encoded in HLL source statements. All such directives are coded as special types of comment lines in the source.

The source types of HLL languages are classified by Toolkit as either fixed or floating:

- For fixed source types, the directive control characters must always be in a particular column of the source.
- For floating types, the directive control characters may appear in any column position, and the end of the directive should be shown by the use of the end comment characters (**'*/'**).

Fixed Format Source Types

Fixed format source types are: RPG III, DDS, and COBOL. Directives may be specified either in columns six and seven (**'x*'**), or in columns seven, eight and nine (**'*x:'**).

Free Format Source Types

Free format source types are: CL, CMD, PLI.

The following table lists the different types of source directives:

Fixed Fmt Source 123456789	Free Fmt Source	Source Directive Type	Where Used
....*D:	/*D: */	Debug breakpoint statement	YSTRDBG
...D*			
....*H:	/*H: */	Documentation synopsis	YDOCPGM
...H*			
....*M:	/*M: */	Documentation maintenance	YDOCPGM

....M*				
*T .(1) :	/*T: */	Title line	YCRTOVR	YDOCPGM
...T*			(2)	
....*W:	/*W: */	Documentation warning		YDOCPGM
...W*				
....*Y:	/*Y: */	Compile time override	YCRTOVR	YDOCPGM
...Y*			(2)	
....*Z:	/*Z: */	Compiler override	YCRTOVR	YDOCPGM
...Z*			(2)	

(1) For RPG III source, '/TITLE' may also be used

(2) Compile pre-processor directives should appear within the first twenty lines of a source member.

Examples

The following is an example of Toolkit Source directives for RPG III

The following is an example of Toolkit Source directives for CLP

The following is an example of Toolkit Source directives for CBL

The following table lists help text source directives:

Text Source	Source Directive Type	Where Used
. *T: text	Text title statement	YDSPHLP
. *Y*: text	Text index comment	YDSPHLP
. *YI: XXXXXXXXXX	Text index entry statement	YDSPHLP
. *YH: XXXXXXXXXX	Text index label statement	YDSPHLP

The following table lists help text CUA/Text directives:

Window Source	Source Directive Type	Where Used
. *YW: XXXXX	Window size	YDSPHLP
. *YD*: XXXXXXXX	Window format definition	YDSPHLP
. *YF: XX	Window format	YDSPHLP
. *YK: XXXXXXXXXX	Keys help vector	YDSPHLP

For explanations and examples of these directives, see the section Specifying Help Text Directives in the chapter "User Access Aids".

Appendix E: Toolkit Modules

This appendix contains information about the contents of each of the four Toolkit modules:

Toolkit Modules

The four modules are:

- Administrator (*PGMR)

  To execute commands that administer and manipulate source code and objects as follows:

  — Display, amend, or print any database files with selection on field or data values

  — Automatically scan for and replace values in source code

  — Build and change lists of objects and source members, then apply any command to each list item without programming or compilation

  — Submit source members of different types to be compiled and automatically map correct attributes to each new object

- Documaker (*DOC)

  To execute commands that document the Toolkit as follows:

  — Document database and device files to show system-wide field usage

  — Document programs, detailing all files, data areas and other programs used and summarizing program source automatically

  — Cross reference the system with exploded object and object calls, producing a list of unreferenced files

  — Authorization to use the modules is given with the Toolkit Grant Product Authorization command (YGRTPRDAUT).

- Menumaker (*USR)

  To set up and manage as a Toolkit environment as follows:

  — Design and run menus without programming or compilation

  — Automatically create, store, manipulate and replace library lists

  — Manage user environments by extending attributes of CPF user profiles without programming or user disruption

- Designmaker (*DSN)

  To paint design layouts, create prototypes and automatically generate source as follows:

  — Paint panel and report designs with an interactive editor

  — Simulate complete systems with prototype tools

  — Automatically generate panel and report source code

Following are the utilities included with each module. Note that some utilities are included in more than one module.

Administrator (*PGMR)

The contents of the Administrator module are:

YADDHLPTBL	YCPYMSGD	YDSPHLP	YGO
YADDLLE	YCRTDUPOBJ	YDSPLIBLST	YGRTPRDAUT
YADDMLE	YCRTOBJ	YDSPMBRLST	YMOVOBJ
YADDOLE	YCRTSRCPF	YDSPPGMQ	YMOVLST
YADDSCRM	YCVTAUTL	YEDTCKYTBL	YMOVM
YBLDDBFLST	YCVTBIN	YEDTDBFLST	YRMVLLE
YBLDDOC	YCVTDBFLST	YEDTDTAARA	YMOVOBJSRC
YBLDFMTLST	YCVTDBR	YEDTFMTLST	YOPRLST
YBLDLIBLST	YCVTDDSCKY	YEDTGDA	YRNMLLE
YBLDMBRLST	YCVTDDSIGC	YEDTLDA	YRMVM
YBLDOBJLST	YCVTDEC	YEDTLIBLST	YRNMLIBLST
YCHGCMD	YCVTPGMREF	YEDTMBRLST	YRTVOBJLIB
YCHGJOBDLL	YCVTOBJLST	YEDTMSGD	YRNMOBJSRC
YCHGLIBL	YCVTUSRPRF	YEXCDBFLST	YRTVMSGF
YCHGLIBLST	YDLTLIBLST	YEDTOBJLST	YSETBRKPGM
YCHGLST	YDLTLST	YEXCCL	YSCNRPLSRC
YCHGOBJOWN	YDLTOBJ	YFLTDBFLST	YSCNSRC
YCHKLIBLST	YDOCDBFLST	YEXCMBRLST	YTRNPF
YCHKLSTE	YDOCFMTLST	YEXCOBJLST	YSTRDBG
YCHKVN	YDOCLIBLST	YFLTOBJLST	YTDYRPGSRC
YCMPSRC	YDOCMBRLST	YFLTFMTLST	YWRKLIBLST
YCPYF	YDOCOBJLST	YFLTMBRLST	YTRNSRCPF
YCPYLIBLST	YDSPABR	YINXLST	YWRKF
YCPYLST	YDSPEXPDAT		

Documaker (*DOC)

The contents of the Documaker module are:

YBLDDBFLST	YCVTDEC	YDOCMNU	YDSPHLP
YBLDDOC	YCVTPRT	YDOCMNUREF	YEDTDTAARA
YBLDFMTLST	YDOCAUT	YDOCMSGREF	YGO
YBLDMBRLST	YDOCEXCREF	YDOCOBJLST	YGRTPRDAUT
YBLDOBJLST	YDOCDBFLST	YDOCPGM	YINXLST
YCHKMNU	YDOCF	YDOCPGMREF	YRTVMSGF
YCHKVN	YDOCFLDREF	YDOCSRC	YRTVOBJLIB
YCMPSRC	YDOCFMTLST	YDSPABR	YSTRCVTPRT
YCVTBIN	YDOCMBRLST	YDSPEXPDAT	YTDYRPGSRC

Menumaker (*USR)

The contents of the Menumaker module are:

YADDHLPTBL	YCHKLIBLST	YDOCAUT	YGRTPRDAUT
YADDLLE	YCHKMNU	YDOCLIBLST	YINLPGM
YADDSNFM	YCHKPWDVAL	YDOCMNU	YINXLST
YBLDDBFLST	YCHKVN	YDOCMNUREF	YRMVLLE
YBLDFMTLST	YCPYLIBLST	YOCUSRPRF	YRNMLIBLST
YBLDLIBLST	YCPYMNU	YDSPABR	YRNMLLE
YBLDMBRLST	YCPYUSRPRF	YDSPEXPDAT	YRMNLIB
YBLDOBJLST	YCRTDSNF	YDSPHLP	YRNMLIBLST
YCHGJOBDLL	YCRTUSRPRF	YDSPLIBLST	YRNMMNU
YCHGLIBL	YCVTBIN	YDSPUSRPRF	YRNMUSRPRF
YCHGLIBLST	YCVTDEC	YEDTDTAARA	YRTVOBJLIB
YCHGOBJOWN	YDLTLIBLST	YEDTLIBLST	YWRKLIBLST
YCHGPWD	YDLTMNU	YEDTPWDVAL	YWRKMNU

YCHGUSRPRF	YDLTUSRPRF	YGO	YWRKUSRPRF

Designmaker (*DSN)

The contents of the Designmaker module are:

YADDDSNFM	YCRTRPTDDS	YDOCRPT	YGRTPRDAUT
YCHKMNU	YCVTBIN	YDOCSRC	YRNMPNL
YCHKPNL	YCVTDEC	YDSPABR	YRNMRPT
YCHKRPT	YDFNPNLDSN	YDSPHLP	YRTVOBJLIB
YCHKVN	YDLTMNU	YDSPEXPDAT	YRTVPNLDSN
YCPYMNU	YDLTPNL	YDSPPNL	YRTVRPTDSN
YCPYPNL	YDLTRPT	YEDTDTAARA	YWRKMNU
YCPYRPT	YDOCMNU	YEDTDSNDFT	YWRKPNL
YCRTDSNF	YDOCMNUREF	YGO	YWRKRPT
YCRTPNLDDS	YDOCPNL		

Index

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Y