CA 2E

Generating and Implementing Applications

Release 8.6.00

Contact CA Technologies

Contact CA Support

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

- 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.

If you would like 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 4: Generating and Compiling Your Application 173

Chapter 5: Implementing Your Application 197

Chapter 1: Managing Model Objects

This chapter describes the change control facilities provided by CA® 2E for managing changes to your model.

This section contains the following topics:

Components of CA 2E Change Management

The primary components of CA 2E Change Management are:

- Configuration Management:
  - Controlling access to the CA 2E model and its component objects
  - Setting up environments to facilitate the integrity of production, development, and test models
  - Controlling access to Change Control processes
- Impact Analysis:
  - Determining the impact of a proposed or actual change to the design objects in a CA 2E model
  - Ensuring the integrity of a set of changes by inclusion of dependent objects
- Change Control:
  - Tracking changes to objects in your CA 2E model
  - Administering changes to, and relationships between, objects within and across CA 2E models

Overview of CA 2E Change Control Facilities

CA 2E change control facilities are a set of features and functions supplied with CA 2E for managing CA 2E design objects. They include:

- Capability of building lists of CA 2E design objects, known as model object lists, for auditing and input to specialized model list commands.
  - Impact analysis:
  - Component change processing
- Model cross references
- Support for versioning of functions and messages.
- Redirection of functions and messages.
- Copying model objects between models.
- Centrally recorded model object information .

You can use these change control facilities:

- As tools to work with and manage your CA 2E design objects.
- To manually control changes to your design model.
- With CA 2E Change Management (CM) Option to provide an integrated, automated change management solution.

Change Management of CA 2E Environments with CM

CM is fully-integrated with CA 2E and offers a total solution for automated change management in your CA 2E environment. CM manages changes for:

- CA 2E design objects such as, access paths, functions, and fields. These are also known as model objects.
- Implementation objects such as, generated source and compiled objects. These are also known as traditional or 3GL objects.

CM has extensive capabilities for controlling your entire operation including the following:

- Check out and promotion of CA 2E design objects using model object lists
- Automated control of versions for functions and messages
- Automated archiving and roll-back of functions and messages
- Authority to model object lists
- Control of access to CA 2E
- panels
- Authority to access, view, or edit model object types

For more information on using CM to manage changes in your CA 2E environment, see the CA 2E *Change Management User Guide*.

Summary of CA 2E Change Management Features

The following table summarizes CA 2E change management features. For each feature, the table shows whether the feature is provided as part of CA 2E's change control facilities or by CM and where to find more information about the feature.

Feature	CA 2E Change Control Facility	CM Change Management	Where Documented
Work with Model Objects Utility	Yes	Yes	This chapter, Working with Model Object Lists section
Model User Profile	Yes	Yes	This chapter, Model Profile section
impact Analysis Utilities	Yes	Yes	This chapter, Model Object Audit Information section

Feature	CA 2E Change Control Facility	CM Change Management	Where Documented
Model Object Change Tracking			
Model Object List Processing	Yes	Yes	This chapter, Model Object Lists section
Redirect Function References	Yes	Yes	This chapter, Working with Versions of Functions and Messages section, Making a Version Current section
Versions of Functions and Messages	Yes	Yes	This chapter, Working with Versions of Functions and Messages section
Session List	Yes	Yes	This chapter, Model Object Lists section, Session Lists section
Global Browse-only Access to Model	Yes	Yes	*Administrator* Guide
Copying Model Objects Between Models	Yes	Yes	This chapter, Editing Model Object Lists section, Copying Model Objects section, *Admin-istrator Guide,* and the *Command Reference*
Check Out	No	Yes	*CM User Guide*
Automated Version Control	No	Yes	*CM User Guide*
Access Control	No	Yes	*CM User Guide*
Automated Rollback of Functions and Messages	No	Yes	*CM User Guide*

Feature	CA 2E Change Control Facility	CM Change Management	Where Documented
Automated Concurrent Development	No	Yes	*CM User Guide*
Authority to Model Object Lists	No	Yes	*CM User Guide*
Promotion Capabilities	No	Yes	*CM User Guide*
Analysis of Context of Change and Rollback	No	Yes	*CM User Guide*

Model Objects

Model object is another term for a CA 2E design object such as an access path, a function, or a field.

Supported Model Object Types

Following are the types of model objects that are supported in your model.

Model Object	Object Type
Access Path	ACP
Application Area	APP
Array	ARR
Condition	CND
File	FIL
Field	FLD
Function	FUN
Message	MSG

Some model objects, such as external functions and access paths, have corresponding implementation objects, namely, generated source and the compiled object. Model objects having implementation objects are sometimes referred to as *generatable objects*.

Naming and Identifying Model Objects

Within a model, a model object is identified by either of the following:

- **Object Surrogate Number**—A 7-digit number assigned automatically by when the object is created. For example:

OBJSGT(1100897)

  Note that it is generally more efficient to use the surrogate number when possible. You can obtain it using the Retrieve Model Object (YRTVMDLOBJ) command.

- **Model Object Name**—Consists of the owner, name, and type you assign when creating a model object.

  The following table shows the components of the model object name for each of the supported model object types.

Model Object Model Object	Object Owner Name	Object Name	Object Type
Access Path	name'	path name'	*ACP
Application Area	*NONE	area code'	*APP
Array	*ARRAYS	name'	*ARR
Condition	'field name'	'condition name'	*CND
File	*NONE	name'	*FIL
Field	*NONE	name'	*FLD
Function	name'	name'	*FUN
Message	*MESSAGES	name'	*MSG

  For example, the following shows the model object name for the Display Product Details function that is owned by the Product file:

OBJNAM(Product 'Display Product Details' *FUN)

Model Object Description

CA 2E maintains an object description for each model object. Each description contains information such as the object's name and type, the surrogate number, the date and time the object was last changed, and various change management flags.

The object descriptions for all model objects are centrally maintained in the All Objects list, also known as the *ALLOBJ list. Each time an object is changed, regenerated, or imported, CA 2E automatically updates the model object's description in the All Objects list to reflect the change.

For more information on the *ALLOBJ list and model object descriptions, see the All Objects Model Object List section, in this chapter.

Model Object Lists

Model object lists provide a method of grouping CA 2E model objects. A model object list is a named set of references (or pointers) to model objects within a model. Each reference within a model object list is called a list entry.

Each list entry contains information about a model object at the time the list entry was created. In other words, a model object list provides an historic snapshot of a model or a group of model objects. Each list entry contains the following:

- Model object surrogate number
- Model object name consisting of the owner, name, and type of the object
- Date and time the model object was created
- Date and time the model object was last changed

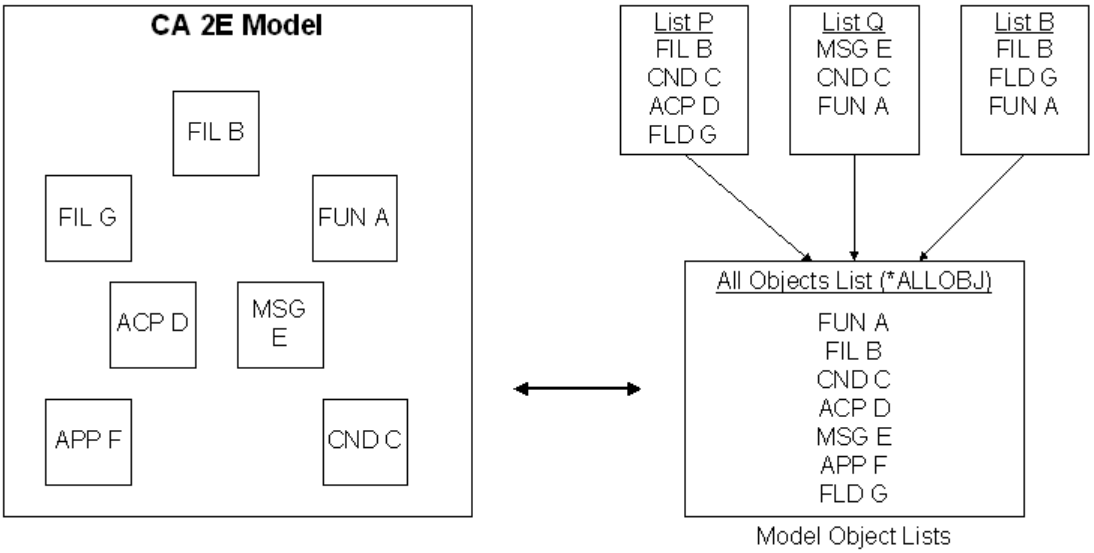
Note: Since model object lists provide an historic record, a model object list can contain list entries that see objects that have been deleted from the model.

Each model list entry also contains a flag selection value that you can set and use for filtering on model object list commands. For example, you can process only model list entries flagged as selected or just those flagged as in error.

For more information:

- On model object list commands and the flag selection value, see the Commands to Manipulate Model Object Lists section in this chapter, and the CA 2E *Command Reference Guide*.
- On commands to create, change, and delete model list entries, see the Commands to Manipulate Model Object Lists section in this chapter.

The following diagram illustrates the model object list and its relation to the CA 2E model.



Model Object Lists

As shown in the diagram, model object lists can include model objects of different object types. You can use a model object list to group related objects into a set in the same way application areas let you group files. This enables you to manipulate or process a set of model objects with a single command or to easily process a series of tasks required as a result of a change to your model.

For example, suppose you need to change the length of a key field such as Customer number. The following list suggests a way to use model object lists to simplify the change process and ensure that all necessary tasks are done:

1. Use impact analysis to automatically build a list of all model objects affected by the change.
2. Use the list of affected model objects as a guide to making the necessary related changes to the identified objects.
3. Convert the list of affected model objects to a job list and submit the list for generation and compilation.
4. Use the same list as a release or PTF and promote it through test and QA to the end user sites.

Note: With CM, this promotion can occur automatically; for example, to multiple remote locations.

These steps are explained in more detail throughout this chapter.

All Objects Model Object List

As shown in the diagram at the beginning of this section, every model contains a special All Objects model object list, or All Objects list for short. The primary purpose of the All Objects list is to provide a central location for model object information and to record change data; for example, date, time, and the user profile of the developer who made a change.

The All Objects list contains a dynamic reference for each supported model object in the model; in other words, whenever a model object is updated, its model object description, or detail, is also updated. When referring to the All Objects list in commands, it has the special name *ALLOBJ.

For more information on the All Objects list, see the All Objects List section in this chapter.

Session Lists

A session list is a model object list to which all objects you change, add, or delete during a session can be logged. Note that logging changes to a session list is optional.

The session list persists across your model sessions; in other words, the session list is cumulative until you clear it. This lets you keep track of changed objects between model sessions. As a result, you will probably want to clear your session list periodically so that it contains only recently changed objects. You can clear your session list using option 9 on the Work with Model Lists panel.

Naming and Activating Session Lists

You assign a name to your session list in the following ways:

- Specify a session list name when entering your model using the Edit Model (YEDTMDL) command. The default name is the one stored on the model profile.
- Specify a session list name on the model profile using the Edit Model Profile (YEDTMDLPRF) command. The default name is the name of your i OS User Profile. Note that this change does not take affect until the next time the model is loaded.

You activate automatic logging of changes to the session list by setting the Log changed objects option in your model profile to Y (Yes).

For more information on the model profile, see the Model Profile section in this chapter.

Using Session Lists

The following are suggestions for ways to use session lists to manage your model.

Administering Your Model

Following are examples of ways you could use session lists to manage your model.

- Assign the same default session list to all developers working on the same change to the model. The shared session list will contain a record of each model object changed by any member of the development team assigned to the project.
- Set up a separate session list for each development project. When you enter the model, specify the session list assigned to the project on which you plan to work.

Using Session Lists with Model List Commands

You can use model list commands to process the model objects contained on your session list. You can use the Edit Model Profile (YEDTMDLPRF) command to set your default model object list for commands (*MDLPRF) to be the same as your session list.

Example

Suppose you need to regenerate all external functions that use model objects contained in the session list. You can produce a list of these functions by using the Display Model Usages (YDSPMDLUSG) command. The following command creates a model object list (*list-name*) consisting of all model objects that use any object contained on your default model list (*MDLPRF), up to and including the first external function:

```
YDSPMDLUSG  MDLLST(*MDLPRF)+
 OUTPUT(*MDLLST)+
 OUTMDLLST(list-name)+
 SCOPE(*EXTFUN)
```

You can then convert the resulting output list to a job list for generation and compilation using the Convert Model List (YCVTMDLLST) command.

The following are more examples of ways you can use session lists with model list commands:

- Produce a printed record of changes to your model for a specific development project by using the session list for the project as input to the Document Model Object List (YDOCMDLLST) command.
- Flag selected entries on the session list to copy using the Filter Model Object List (YFLTMDLLST) command or Edit Model Object List for copy (YEDTCPYLST) panel. Then promote the change to a target model by using the session list as input to the Copy Model Objects (YCPYMDLOBJ) command.
- Delete a session list related to a completed development project by using the Delete Model Object List (YDLTMDLLST) command.
- Create an empty session list by using the Index Model Object List (YINXMDLLST) command.

For more information on model list commands, see the Commands to Manipulate Model Object Lists section in this chapter, and the CA 2E *Command Reference Guide*.

Referencing Model Object Lists in Commands

You specify model object lists on commands using qualified names. For example, you would specify the named model object list, LISTA in MYMDL, as MDLLST(MYMDL/LISTA). You would specify the All Objects list as MDLLST(MYMDL/*ALLOBJ).

How Model Object Lists are Stored

Each model object list name must be unique within a model. Model object lists are actually stored as members in the following physical files within the model library.

- **YMDLOBJRFP**—Consists of a single member containing all supported objects in the model. This is the All Objects list.
- **YMDLLSTRFP**—Consists of multiple members, each of which contains a named model object list.

Model Object List Authority

Users of the model need to have a minimum of i OS *CHANGE and *OBJMGT authority to the YMDLLSTRFP file. This is the default authority for user, *PUBLIC.

Although you cannot assign authority on individual model object lists, you can prevent users of the model from deleting or adding lists by removing *OBJMGT authority on the YMDLLSTRFP file.

Note: List level authority is provided by CM.

All Objects List

Every model contains a special All Objects model object list. The All Objects list provides a central place for model object information and contains a dynamic model object description for each supported model object in the model. When referring to this list in commands, it has the special name *ALLOBJ.

The primary purpose of the All Objects list is to record information related to changes to model objects; for example, the date, time, and name of the developer who made the change. As you create, delete, update, and generate model objects, the All Objects list is also updated to reflect these changes. The All Objects list also contains information for impact analysis, versions of functions and messages, and CM.

The All Objects list differs from a named model object list in a number of important ways. The following table summarizes these differences:

All Objects List (*ALLOBJ)	Named Model Object Lists
Consists of a dynamic object description for each model object in the model.	Consists of static (unchanging) references to all, or a subset of, model objects in the model. These references are known as list entries.

All Objects List (*ALLOBJ)	Named Model Object Lists
A model object's description is updated automatically each time the object is changed, imported, regenerated.	A model object's list entry does not change when the model object changes. It provides a persistent historic record of the object at the time the list entry was created.
Actions to the All Objects list see the actual model objects; you add entries only by creating new model objects.	Actions to named lists affect only the model object list entries and not the actual model objects.
Does not contain model object descriptions for deleted model objects and displays only the current version of functions and messages.	Can contain references to deleted model objects and non-current versions of functions and messages.
You cannot update the All Objects list or model objects using model list commands.	You can update named model object lists and model list entries using model list commands.

Model Object Description

Each object in your model contains an object description in the All Objects list. Each description contains detailed information about the model object, such as the object's name and type, the surrogate number, the date and time the object was last changed, and various change management flags.CA 2E maintains the model object descriptions and automatically updates an object's description each time the object is changed, regenerated, or imported.

To view a model object's description, use either selection option 8 on the Edit Model Object List panel or the Display Model Object Description (YDSPMDLOD) command.

The following two commands let you retrieve and change information within a model object description:

- Retrieve Object Description (YRTVMDLOBJ) command
- Change Model Object Description (YCHGMDLOBJ) command

Model List Commands and the Model Object Description

You can use model object description fields in combination with model list commands. This provides a powerful tool for creating utilities to filter and analyze your model. Examples appear throughout this chapter. Some of the fields most suited to this purpose are:

- Object Type
- Object attribute
- Various dates and times
- Impact processed indicator
- Action required indicator
- Change type
- Group surrogate
- Current version indicator

The remainder of this section groups the model object description fields by function and provides suggestions for using them.

For more information:

- On model list commands, see the CA 2E Command Reference Guide.
- On definitions of each model object description field, see the appendix titled "Change Control Facilities Reference Tables" in this guide.

Basic Model Object Information

When you create a model object,CA 2E records the following information in the model object description:

- Model object surrogate
- Object name
- Object type and attribute
- Copy name
- Creation date and time
- Model object surrogate of the owning object
- Object name of the owning object
- Function type

If the model object is a version of an existing function or message,CA 2E also records the following information:

- Group surrogate number
- Current object indicator
- Version type

The basic information maintained for each model object uniquely identifies the model object in the following two ways:

- Model object surrogate
- Model object name consisting of three fields:
  - Name of owner of object
  - Object name
  - Object type

Use either of these to identify a model object on the model object list commands.

For more information on these model object identifiers, see the Model Objects section in this chapter.

Date and Time Information

CA 2E maintains the following dates and times for each model object as audit stamps for various processes:

- Creation date and time
- Date and time the model object was last changed
- Component change processed date and time
- Generation date and time
- Import date and time
- Check out date and time (applies only to CM)

You can use these date fields in Command Language programs to create lists of model objects requiring specific actions; for example, editing, generation, or copying to another model.

Change Information

When you change a model object, the date, time, and user are logged. In other words,CA 2E automatically updates the following fields in the model object's description:

- Change date and time
- Change user
- Change type
- Impact processed indicator
- Component change processed date and time

CA 2E uses the Change type during component change processing to identify other objects in the model affected by the change. The Component change processed date and time are set to the date and time the change occurred.

Press F11 from the Edit Model Object List panel to view this information for any model object list.

For more information:

- On Change type and changing model objects, see the Model Object Audit Information section in this chapter.
- On component change processing, see the Impact Analysis section in this chapter.

Component Change Processing Information

Component change processing ensures the integrity of your model whenever you change a model object. It does this by identifying which other model objects are affected and the type of change required for the affected objects.CA2E sets and maintains the following fields in the model object description as part of component change processing:

- Action required indicator
- Impact processed indicator
- Component change processed date and time
- Change type

Press F11 from the Edit Model Object List panel to view this information for any model object list.

You can check the Impact processed indicator to determine whether to run the Apply Component Changes (YAPYCMPCHG) command.

The Action required indicator identifies whether a model object that uses a changed object needs to be edited (EDT) or regenerated (GEN). You can use model list commands to examine and filter objects in your model based on the setting of this indicator.

For more information on component change processing, see the Impact Analysis section in this chapter.

Generation Information

When a model object is successfully generated,CA 2E automatically updates the following fields in the model object's description:

- Generation date
- Generation time

Press F11 from the Edit Model Object List panel to view this information for any model object list.

Check Out Information

If you are using CM, CA 2E automatically updates the following fields in the model object description whenever you check out a model object for a change:

- Check out date and time
- Check out user
- Check out status
- Check out list

Press F11 from the Edit Model Object List panel to view this information for any model object list.

For more information on check out using CM, see the CA 2E *Change Management User Guide*.

Commands to Manipulate Model Object Lists

The model list commands are similar both in style and function to the CA 2E Toolkit object list commands. You can use them to manage your model objects or as API (Application Program Interface) building blocks to automate the manipulation of model objects. See the end of this section for examples.

Change Control Facilities Commands

This section lists the change control facilities commands. They are grouped according to the entity on which the command operates; namely:

- Model object lists
- Model object list entries
- Model object description
- Job lists
- Model profile
- Versions of functions and messages

Some commands use model object lists for input, output, or both. A model object list you specify as input must exist before you execute the command; if you specify a model object list as output,CA 2E automatically creates it if it does not already exist.

The model profile contains the name of a default model object list name to be used for commands. This list is often referred to as the default list. Whenever you specify the value *MDLPRF for a model object list on a command,CA 2E automatically uses the model object list specified on your model profile.

For more information:

- On the change control facilities commands, see the *CA 2E Command Reference Guide*.
- On model profiles, see the Model Profile section in this chapter.

Model Object List Commands

Following are change control facilities commands that operate on named model object lists:

Facility	Command	Description
Create	YBLDMDLLST	Build a Model Object List
	YINXMDLLST	Index a Model Object List
Edit	YEDTMDLLST	Edit Model Object List
	YEDTCPYLST	Edit Model List for Copy
Display	YDSPMDLLST	Display a Model Object List
Object Dependencies	YDSPMDLUSG	Display Model Usages

Facility	Command	Description
	YDSPMDLREF	Display Model References
Clear	YCLRMDLLST	Clear a Model Object List
Delete	YDLTMDLLST	Delete a Model Object List
Filter	YFLTMDLLST	Filter a Model Object List
Check and Refresh	YCHKMDLLST	Check Model Object List
Copy	YCPYMDLLST	Copy a Model Object List
	YCPYMDLOBJ	Copy Model Objects
	YSETCPYNME	Set Model Object Copy Name
Print	YDOCMDLLST	Document a Model Object List
Execute	YEXCMDLLST	Execute a Model Object List
Compare	YOPRMDLLST	Operate on Two Model Object Lists
Convert	YCVTMDLLST	Convert a Model Object List to a Job List
Work with	YWRKMDLLST	Work with Model Object Lists

Model Object List Entry Commands

Following are change control facilities commands that operate on list entries in named model object lists:

Facility	Command	Description
Create	YADDMDLLE	Add a Model Object List Entry
Change	YCHGMDLLE	Change Model List Entry
Delete	YDLTMDLLE	Delete a Model Object List Entry

Model Object Description Commands

Following are change control facilities commands that operate on the contents of an object's model object description in the All Objects list (*ALLOBJ):

Facility	Command	Description
Change	YCHGMDLOD	Change Model Object Description
	YCHGMDLOBJ	Change Model Object
Retrieve	YRTVMDLOBJ	Retrieve Model Object Description
Display	YDSPMDLOD	Display Model Object Description
Impact analysis	YAPYCMPCHG	Apply Component Changes

For more information on a model object description, see the All Objects List section in this chapter and the appendix titled "Change Control Facilities Reference Tables" in this guide.

Job List Commands

Following are change control facilities commands that operate on job lists and job list entries:

Facility	Command	Description
Create	YBLDJOBLST	Build Job List
	YCRTJOBLE	Create Job List Entry
Display	YDSPJOBLST	Display a Job List
Convert	YCVTMDLLST	Convert Model List to Job List
	YCVTJOBLST	Convert a Job List to Toolkit Object List
Check and Refresh	YCHKJYSBMMDLCRTOBLE	Check Job List Entries
Submit		Submit Create Request from Model

For more information on job lists, see chapter titled "Preparing for Generation and Compilation" in this guide.

Model Profile Commands

Following are change control facilities commands that operate on the model profile:

Facility	Command	Description
Change	YCHGMDLPRF	Change Model Profile Details
Edit	YEDTMDLPRF	Edit Model Profile
Retrieve	YRTVMDLPRF	Retrieve Model Profile Details

For more information on model profiles, see the Model Profile section in this chapter.

Version Commands

Following are change control facilities commands that operate on versions of functions and messages:

Facility	Command	Description
Create	YCRTMDLVSN	Create Model Version
	YCRTOBJVSN	Create Model Version
Delete	YDLTMDLVSN	Delete a Model Version
Compare	YCMPMDLOBJ	Compare Model Versions
Redirect	YRDRMDLOBJ	Redirect Model Object
Select	YSLTVSN	Select Model Object Version

For more information on versions, see the Working with Versions of Functions and the Messages sections in this chapter.

Using Change Control Facilities Commands

You can use model list commands for generic manipulation of model objects, to process model objects in batch, and to create your own utilities.

Example 1

You can use the following set of commands to compare model objects within a model at two different times; for example, before and after a development project:

1. Use the Build Model List (YBLDMDLLST) command to create a list of all objects in the model by specifying the All Objects list (*ALLOBJ) as input and a named model object list as output.

```
YBLDMDLLST OBJNAM(*ALLOBJ)+
  MDLLST(list-name1)
```

   The output list contains information about each object that existed in the model at the time you ran this command; namely, each list entry contains the Create date and time and the Change date and time of the corresponding model object.

   At some later stage in the development cycle you can build another list of all model objects and compare the new list with the original list as shown in the following steps.

1. Build a new list of all objects in the model, specifying another model object list as output.

```
YBLDMDLLST OBJNAM(*ALLOBJ)+
  MDLLST(list-name2)
```

1. Compare this list with the original list and create a third model object list containing the differences between the two input model object lists.

```
YOPRMDLLST MDLLSTA(list-name1)+
  LSTOPR(*DIFF) MDLLISTB(list-name2)+
  TOMDLLST(list-name3)+
  OPRTYPE(*OBJSGT)
```

Example 2

You can use the following set of commands to compare model objects between two models:

1. Use the Build Model List (YBLDMDLLST) command to create a list of all objects in a model by specifying the All Objects list (*ALLOBJ) as input and a named model object list as output.

```
YBLDMDLLST OBJNAM(*ALLOBJ)+
  MDLLST(list-name1)
```

1. Build another model object list for the second model; e.g., NEWMDL.

```
YBLDMDLLST OBJNAM(*ALLOBJ)+
  MDLLST(NEWMDL/list-name2)
```

1. Copy the model object list just created to the original model and ensure that the surrogate number of each list entry matches that of the corresponding model object in the original model.

```
YCPYMDLLST+
  FRMMDLLST(NEWMDL/list-name2)+
  TOMDLLST(OLDMDL/list-name2)+
  TOUPDOPT(*RFSSGT)
```

1. Filter out any errors. This creates a list of model objects that exist in the new model but do not exist in the original model.

```
YFLTMDLLST FLAGVAL(*ERROR)+
  MDLLST(NEWMDL/list-name2)+
  OUTLST(list-fail)
```

1. Now compare the two lists in the target model and save the differences in another model object list.

```
YOPRMDLLST MDLLSTA(list-name1)+
  LISTOPR(*DIFF) MDLLSTB(list-name2)+
  TOMDLLST(list-diffs)+
  OPRTYPE(*OBJSGT)
```

1. Print the output lists for a permanent hard copy record of the differences between the two models.

```
YDOCMDLLST  MDDLST(NEWMDL/list-fail)
 YDOCMDLLST  MDLLST(OLDMDL/list-diffs)
```

Example 3

You can use the following series of commands to form part of a nightly process to prepare a model for the following day:

1. Optionally run the Synchronize Model (YSNCMDL) command to ensure that the model is synchronized.
2. Run the Apply Component Changes (YAPYCMPCHG) command to ensure that the impact of any changes to model objects are reflected throughout the model.
3. Run the Filter Model Object List (YFLTMDLLST) command over the All Objects list (*ALLOBJ) to select model objects having the Required action indicator set to *EDT. Specify EDTLST as the output model list for programmers to edit the following day using YEDTMDLLST.
4. Run the Filter Model Object List (YFLTMDLLST) command over the All Objects list (*ALLOBJ) to select model objects having the Required action indicator set to *GEN. Specify GENLST as the output model list.
5. Run the Convert Model Object List (YCVTMDLLST) command over the GENLST model list to prepare a job list to generate objects that require generation as a result of a change to a component object.
6. Run the Submit Model Create Requests (YSBMMDLCRT) commands. Specify the job list created in the previous step to generate and compile changed objects.
7. Run the Document Model Object List (YDOCMDLLST) command to document the EDTLST and GENLST model lists for administrative purposes.

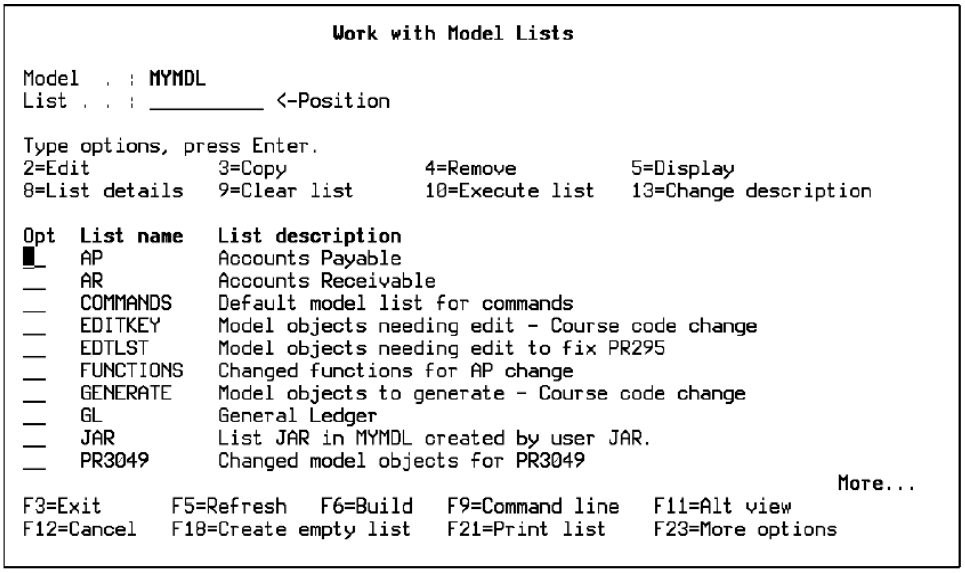
Working with Model Object Lists

The Work with Model Lists interactive panel lets you manage the model object lists in your model. It also provides access to the model object list editing utility (Edit Model Object List panel).

You access the Work with Model Lists panel in either of the following ways:

- Select the Work with Model Lists (YWRKMDLLST) option on the Display Services Menu.
- Enter YWRKMDLLST on a command line.

The following panel displays:



The Work with Model Lists panel displays all the model object lists currently defined in your model library. Press F11 to display the date the list was created and the time and date it was last changed.

From the Work with Model Lists panel, for any model object list shown, you can:

- View a description of the list
- View list entries
- Edit a list
- Create a new model list
- Execute a list
- Clear entries from a list
- Remove a list
- Copy a list
- Change the description of a list

Editing a Model Object List

The Work with Model Lists panel provides a shell for the Edit Model Object List panel. To edit one of the model object lists displayed, type subfile select option 2 against the list and press Enter. You can return to the Work with Model Lists panel at any time during your editing session to select another list or to perform other model object list management tasks.

For more information on the Edit Model Object List panel, see the Editing Model Object Lists section in this chapter.

Creating a Model Object List

You can also use the Work with Model Lists panel to build a new model object list.

1. Press F6 to prompt the Build Model Object List (YBLDMDLLST) command.
2. Specify the model objects to include in the new list using the selection options provided. By default, all objects in the model are included.

   For example, you can select all model objects of a specified object type, a specific model object, or all model objects having the same Object owner. You can also display a selection list of all objects in the model by entering *SELECT for the Object owner option and pressing Enter.
3. Enter the name of the new model object list for the Model object list option.
4. Press Enter.
5. To edit the new list enter subfile select option 2 against it on the Work with Model Lists panel as you did for an existing list.

Note: To create an empty model object list, press F18 from the Work with Model Lists panel to prompt the Index a Model Object List (YINXMDLLST) command. Type the name of the new list for the Model object list parameter and press Enter to accept the defaults and create the new list.

For more information on the YINXMDLLST command, see the *Command Reference Guide*.

Editing Model Object Lists

The Edit Model Object List (YEDTMDLLST) panel is an interactive utility for working with model object lists, including the All Objects list (*ALLOBJ), and model object list entries. This utility has a PDM-like interface and has the following main features:

- Multiple views of current model object list (you can cycle through these by pressing F11.):
  - **Object identification**—Object name, type, attribute, and owner
  - **Implementation details**—Implementation name, date and time of last generation, and if applicable, the function or message type
  - **Component change information**—Date, time, and action required
  - **Audit information**—Change date, user, and type, and the impact processed indicator
  - **Check out information**—Checked out date, model object list name, user, and status

    Note: This alternate view contains data only if the Change Control (YCHGCTL) model value contains a valid library name. The data is set by CM.
- Choice of displaying model object list entries sorted by:
  - Object name within object type
  - Object name within object type within owner
  - Implementation name within object type
- Command line
- Function key to repeat a subfile select option
- Select model objects for processing by a command specified on the command line
- Access to model profile
- Options to work with model objects
- Capability of switching between model object lists, including the All Objects list
- View of detailed description of any model object
- Options and function keys for impact analysis (usages and references)
- Use of user-defined options

This panel serves as an alternate entry point into your model. You can perform most functions available from the Edit Database Relations panel other than editing relations and creating model objects. You can temporarily transfer to the Edit Database Relations panel from the Edit Model Object List panel by entering YEDTMDL or Y2 on the command line. When you finish your editing, press F3 to return to the Edit Model Object List panel.

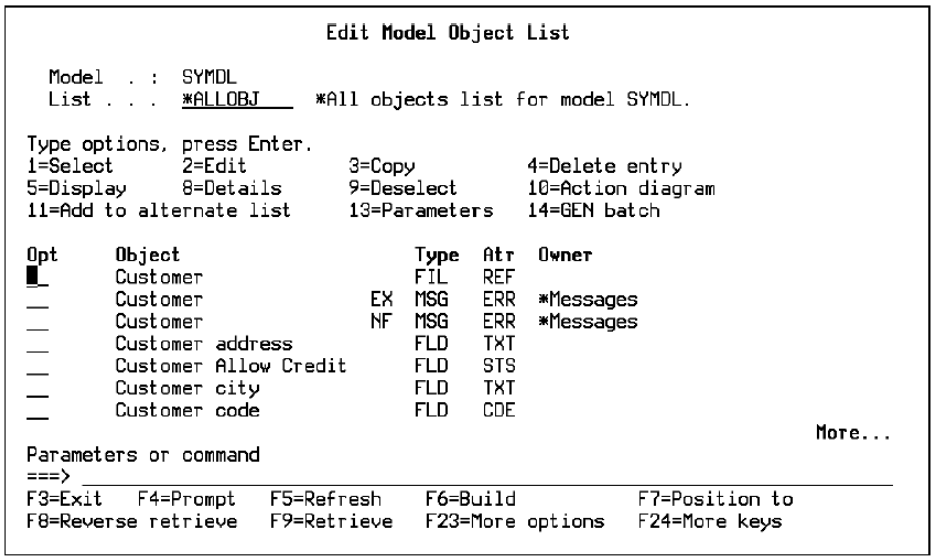
Edit Model Object List Panel

You can use the Edit Model Object List panel to operate on both named model object lists and the All Objects list (*ALLOBJ); however, some functions can be used only for named model object lists.

You can access the Edit Model Object List panel in the following ways:

- Enter YEDTMDLLST from a command line. You can prompt the command or accept the defaults.
- Use selection option 2 from the Work with Model Lists (YWRKMDLLST) panel.
- Select the Work with Model Lists or Edit model list options on the Display Services Menu.

The following is an example of the Edit Model Object List display for the All Objects list:



For more information on the Edit Model Object List panel, see the Working with Model Object Lists section in this chapter.

Many of the subfile select options and function keys shown on this panel are also available on the following interactive panels:

- Display Model Usages
- Display Model References
- Display Model List
- Work with Versions
- Edit Model List for Copy (YEDTCPYLST)

Selecting Another Model Object List

You can select another model object list in the following ways:

- Enter the name of the model object list for the List option in the header and press Enter.
- Enter one of the following special values for the List option in the header:
  - *ALLOBJ to edit the All objects list.
  - *SSNLST to edit your session list
  - *MDLLST to edit the default model list for commands
- Display the Select Model Object List panel by entering *SELECT or *S for the List option in the header, or press F4 with the cursor positioned on the List option.

  This panel lists all named model object lists in your model. Enter * for the List option to also list special lists such as, *SSNLST. Type 1 in the subfile selector to select a model object list and press Enter.
- If you accessed the Edit Model Object List panel using the Work with Model Lists panel, you can press F3 or F12 to return to that panel, select another model list using option 2, and press Enter.

Note: If you have specified subsetting criteria for a model object list, the subsetting is retained when you select another list.

For more information on subsetting a model object list, see the Grouping and Navigation Aids section in this chapter.

Subfile Select Options

You can enter a one- or two-character option for the subfile selection field to perform a large variety of actions on the selected model object or model list entry. Some values are restricted according to certain object types; others are appropriate only when editing a named model object list and not when editing the All Objects list (*ALLOBJ).

User-defined Subfile Select Options

You can define your own subfile select options associated with a command string. User options must be alphabetic; numeric options are reserved for use by CA 2E .

For more information on defining your own subfile select options, see the Performing User-Defined Tasks on Model List Entries later section in this chapter.

CA 2E Subfile Select Options

The following table lists the options supplied by CA 2E.

Note: Some of these options operate on the actual model objects and others operate on *entries* of a named model object list. Be sure to note this distinction when reading the following descriptions and when using the subfile select options.

Option	Action	Meaning	Where Documented
1	Select	Tag the entry in the current list for selective processing by list commands in conjunction with flag value selection (FLAGVAL). Sets the list entry's Object Select flag.	CA 2E *Command Reference Guide*
2	Edit	Edit the selected model object. The CA 2E panel displayed depends on the type of the model object selected.Edit the selected model object. The panel displayed depends on the type of the model object selected.	This section, Editing Model Objects
3	Copy	Create a copy of the selected model object. This applies only to model objects of type FUN and MSG.	This section, Copying Model Objects
4	Delete list entry	Delete the model list entry from the current model object list. A confirmation panel is displayed.	This section, Deleting a Model Object or a Model List Entry
5	Display	Display model object.	This section, Viewing Model Objects
8	Details	Display the model object description for the selected model object and optionally change the model object's Copy name. The information displayed depends on the type of the model object.	This section, Viewing Model Objects and the YDSPMDLOD command section in the CA 2E *Command Reference Guide*
9	Deselect	Removes the list entry's Object Select flag (FLAGVAL). This is the opposite of Select.	CA 2E *Command Reference Guide*

Option	Action	Meaning	Where Documented
10	Action Diagram	Invoke the Action Diagram Editor for the selected function. This applies only to model objects of type FUN.	*Building Applications*
11	Add to alternate list	Add the list entry to another model object list; the default is the list specified in your model profile. Invokes the YADDMDLLE command; you can specify another list on the command line.	This section, Adding Objects to a Model Object List, and CA 2E *Command Reference Guide*
13	Parameters	Edit parameters for the selected model object. Displays the Edit Function Parameters panel. Applies only to functions, messages, conditions, or arrays.	*Building Applications*
14	Generate source in batch	Generate the access path or external function in batch mode. Invokes the YCRTJOBLE command to add the object to the job list. You can specify parameters on the command line; e.g., JOBLST.	*Command Reference Guide*
15	Generate source interactively	Generate the source for an access path or external function interactively. Invokes the YCRTJOBLE command to add the object to the job list. You can specify parameters on the command line; e.g., JOBLST.	CA 2E *Command Reference Guide*
16	Y2CALL	Call the selected function's implementation object using the Y2CALL command. Y2CALL determines the parameters required by an external function directly from details contained in the model. You can provide values for all input-capable fields and you can reuse these values for subsequent calls.	This chapter, Working with Versions of Functions and Messages section and CA 2E *Command Reference Guide*
17	Device design	Invoke the device design editor for the selected model object. Applies only to model objects of type FUN.	*Building Applications*

Option	Action	Meaning	Where Documented
18	Device structure	Invoke the structure design editor for the selected print function. Applies only to model objects of type FUN and attribute PRTFIL or PRTOBJ.	*Building Applications*
19	Work with versions	Work with versions within the model for the selected model object. Applies only to model objects of type FUN and MSG.	This chapter, Working with Versions of Functions section and the Messages section and *CM User Guide*
20	Access path	Display the Display File Access Paths panel for the selected model object. Applies only to model objects of type FUN.	*Building Applications* and *Building Access Paths*
21	Narrative for object	Display the Edit Narrative Text panel for the selected model object.	*Administrator Guide*
22	Narrative for owning object	Display the Edit Narrative Text panel for the model object that owns the selected model object.	*Administrator Guide*
23	Start SEU	Start SEU for the selected model object. Available for model objects of type ACP and FUN only.	—
24	Delete model object	Delete the model object from the model. This applies only to the following object types: ACP, ARR, APP, CND, FUN, MSG; you cannot delete a FIL or FLD in this way.	This section, Deleting a Model Object or a Model List Entry
25	Document function	Invokes the YDOCMDLFUN command for the selected model object. Applies only to model objects of type FUN.	*Building Applications* and the CA 2E *Command Reference Guide*
26	Redirect	Make the selected version active (current) in the model. Applies only to model objects of type FUN and MSG.	This chapter, Working with Versions of Functions and Messages section and the *CM User Guide*
28	Check out	Check out the model object for change. Available only with CM.	*CM User Guide*

Option	Action	Meaning	Where Documented
30	Open function	Open all functions selected with option 30 and then Display the Open Functions panel. Applies only to model objects of type FUN.	*Building Applications*
31	Locks for object	Display locks currently set for the selected model object.	*Administrator Guide*
32	Locks for object owner	Display locks currently set for the owner of the selected model object.	*Administrator Guide*
33	Refresh entry	Refresh the model object list entry from its model object description in the All Objects list (*ALLOBJ).	This section, Viewing Model Objects
34	Compare objects	Invoke the Compare Model Objects (YCMPMDLOBJ) command. Applies only to model objects of type FUN and MSG.	*CA 2E Command Reference Guide*
38	Check action diagram	Scans for error in the selected function, and if errors are found, loads the action diagram at the first error.	*Building Applications*
81	References for object	Display all references within the model for the model object.	This chapter, Model Object Cross References section
82	References for owning object	Display all references within the model for the owner of the model object.	This chapter, Model Object Cross References section
91	Usages for object	Display all usages within the model for the model object.	This chapter, Model Object Cross References section
92	Usages for owning object	Display all usages within the model for the owner of the model object.	This chapter, Model Object Cross References section
/	Merge object with command	Process the selected object with the command on the command line. Serves as a temporary user-defined option.	This section, Command Line

Function Keys

The following table lists the function keys available on the Edit Model List panel. Note that some options are *not* available for the All Objects list (*ALLOBJ).

Function Key	Action	Meaning	Where Documented
F1	Help	Display additional information about the display or option selected.	—
F3	Exit	—	—
F4	Prompt	Prompt a command line or option.	This section, Command Line
F5	Refresh	Updates the panel with current information.	—
F6	Build list	Add selected model objects from *ALLOBJ to the model object list being edited. Applies only to named model object lists.	This section, Adding Entries to a Named Model Object List
F7	Position to	Lets you position the display to a specified model object type, name, owner, and/or implementation name.	This section, Positioning a Model Object List
F8	Reverse retrieve of previous commands	After use of F9, retrieve previous commands in reverse order.	This section, Command Line
F9	Retrieve previous command	Retrieve the most recent command you entered from the command line. Press repeatedly to cycle through all commands you entered during the current session. i OS standard.	This section, Command Line

Function Key	Action	Meaning	Where Documented
F10	Execute list	Perform a specified action on each entry in a model object list. This invokes the YEXCMDLLST command and you can specify parameters on the command line. Applies only to named model object lists.	This section, Executing a Model Object List, and the CA 2E *Command Reference Guide*
F11	Alternate view	Display the following alternate views in sequence: 1. Basic Information 2. Implementation Information 3. Component Change Information 4. Change Information 5. Check out Information	This section, Viewing Model Objects
F12	Cancel	Return to previous menu or panel.	—
F13	Repeat	Repeat the last selection option specified beginning with the last selected model object to the end of the model object list. Press F5 to undo the repeat.	Examples appear within this section
F14	Filter	Filter the displayed model object list. Invokes the YFLTMDLLST command. You can specify an output list for the results; otherwise, the displayed list is changed based on the filter. For named model object lists only.	This section, Filtering a Named Model Object List

Function Key	Action	Meaning	Where Documented
F15	Check list	Check model object list entries for existence, checked out, and lock information, and remove checked entries. Invokes the YCHKMDLLST command; you can specify parameters on the command line. Applies only to named model object lists. For example, use this to refresh list entries with detail from *ALLOBJ.	CA 2E *Command Reference Guide*
F17	Subset	Display only a specified subset of a model object list based on criteria you specify. Similar to subsetting capability of PDM.	This section, Subsetting a Model Object List
F18	Change model profile	Update settings in a specified model profile. Invokes YEDTMDLPRF command.	This chapter, Model Profile section, and CA 2E *Command Reference*
F19	Job list commands	Displays a menu of job list commands to aid in creating and submitting job lists.	This section, Selecting Job List Commands
F20	Usages	Displays the Display Model Usages (YDSPMDLUSG) panel. Applies only to named model object lists.	This chapter, Impact Analysis section
F21	Print	Invokes the Document Model List (YDOCMDLLST) command; you can specify parameters on the command line. Applies only to named model object lists.	*Command Reference Guide*

Function Key	Action	Meaning	Where Documented
F22	References	Displays the Display Model References (YDSPMDLREF) panel. Applies only to named model object lists.	This chapter, Impact Analysis section
F23	More options	Display more subfile select options.	—
F24	More keys	Display more function keys.	—

Command Line

You can use the command line on the Edit Model Object List panel in the following ways:

- Enter or prompt , Toolkit, or i OS commands
- Override the parameters and/or the command invoked by a subfile selection option
- Execute a command that operates on a selected model object
- Enter special command line values to retrieve previously executed commands

Merging Entries with Commands

Many subfile selection options are executed using commands. You can use the command line to override any of the parameters or the command itself. To do so, enter the override details on the command line before you press Enter or F4.

Example 1

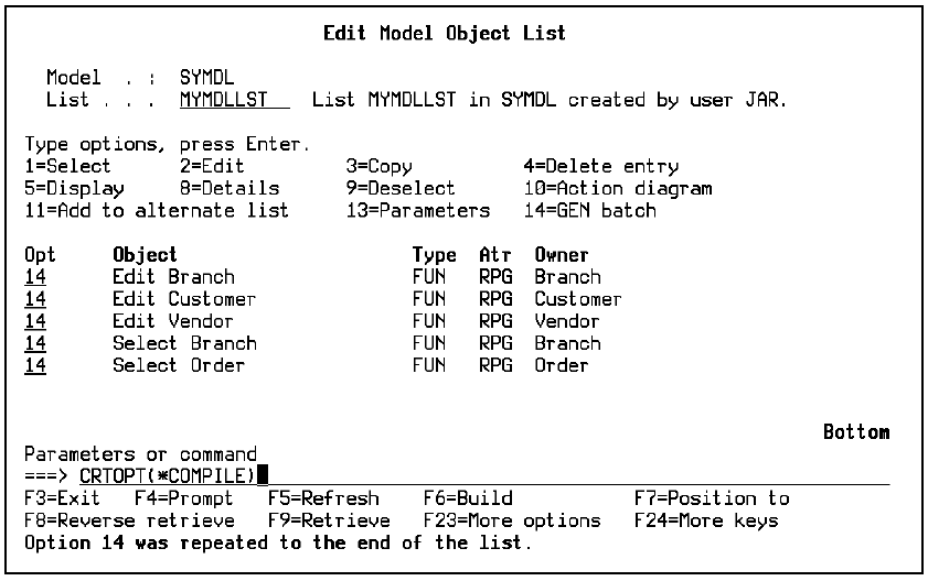
Following is an example of overriding command parameters on the command line. Note that when you override a parameter value you need to specify both the parameter keyword and the parameter override value.

Suppose the functions contained in the MYMDLLST model object list have generated successfully and just require compilation. Create a job list entry for each model object using selection option 14, which invokes the YCRTJOBLE command.

1. Enter 14 for the first model object entry.
2. Press F13 to repeat the option for all list entries. Note the message at the bottom of the panel that states that option 14 was repeated to the end of the list.

   Note: You can press F5 to undo the repeat action.

   Type the CRTOPT(*COMPILE) parameter on the command line as shown to indicate compilation only.



3. Press Enter.

Press F19 to display a list of job list commands and to submit the job list for compilation in batch.

Example 2

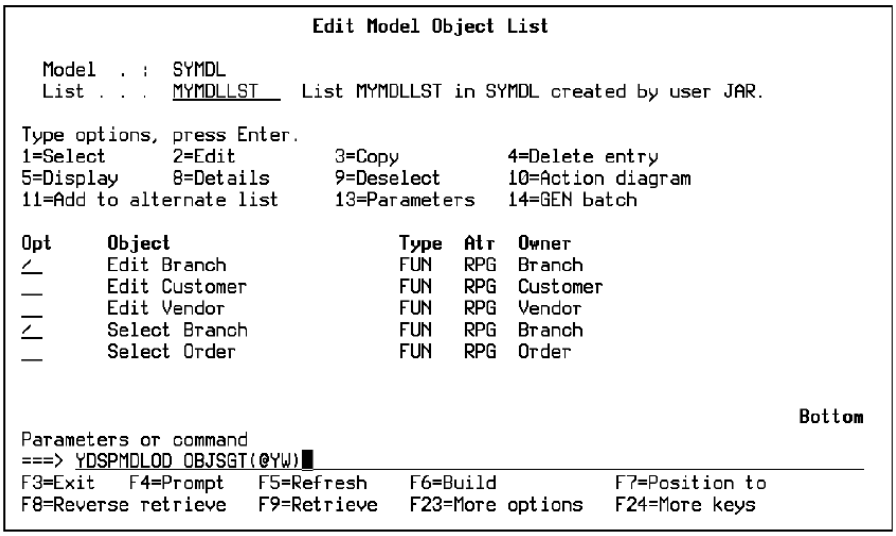
This example uses the "/" selection option to merge model list entries with a command not associated with a subfile selection option. This option serves as a temporary user-defined selection option.

Suppose you want to view the model object descriptions for the owners of one or more model objects. To do so you will use the Display Model Object Description (YDSPMDLOD) command.

1. Enter "/" against each model object whose owner's model object description you wish to view. To select all model objects to the end of the list, type "/" on the first model object you want to select and press F13 as in the first example.

   Type YDSPMDLOD OBJSGT(@YW) on the command line. Note that @YW is a substitution variable that specifies the surrogate number of the owner for each of the selected model objects.

   For more information on substitution variables, see the Performing User-Defined Actions on Model List Entries section in this chapter.



2. Press Enter.

The model object description for the owners of the selected model objects will be displayed one at a time. In this example, the model object description for the Customer file and the Branch file are displayed.

Retrieving Commands

All commands executed in the current job, whether executed from a CA 2E or an i OS panel, are placed in a common command sequence. The F9 and F8 function keys let you step backward and forward through this command sequence.

Using Special Command Line Values to Retrieve Commands

A set of special values that you enter on the command line lets you search for commands containing specified characters and control the contents of the command sequence. You can enter these values in upper or lower case letters and in most cases the space between the special command line value and the following text is optional.

The following table lists the special command line values and their functions:

Special Values	Abbreviations	Function
*SCANF [n] *string*	*SF or >>	Starting from the first command in the command sequence, retrieve the first command containing the specified string. Specify n to retrieve the nth matching command.
*SCANL [n] *string*	*SL or <<	Starting from the last command in the command sequence, retrieve the first command containing the specified string. Specify n to retrieve the nth matching command.
*FIRST [n] *string*	*F or >	Starting from the first command in the command sequence, retrieve the first command beginning with the specified string. Specify n to retrieve the nth matching command.
*LAST [n] *string*	*L or <	Starting from the last command in the command sequence, retrieve the first command beginning with the specified string. Specify n to retrieve the nth matching command.
– command	n/a	Execute the specified command, but do not add it to the command sequence.
+ command	n/a	Add the specified command to the command sequence, but do not execute it.
*CLEAR	n/a	Clear the commands at this invocation of the command line.

When you specify a search string, do not include "*" at the end of the string. You can however specify a '?' as a wild character in any position. To repeat the last search you entered, specify '*' instead of a search string. You can repeat the search in either chronological or reverse chronological order. These concepts are all shown in the following examples.

Examples

1. To retrieve the first previous command that begins with the characters 'wrk', type the following on the command line and press Enter.

*LAST WRK

Note that *last wrk, *L wrk, and < wrk all give the same result.

You can repeat the last search you entered by specifying '*' instead of the search string. To retrieve the next previous command beginning with 'wrk', type the following on the command line and press Enter.

*FIRST *

1. To retrieve the third command from the beginning of the command sequence containing the characters 'uuae', type the following on the command line and press Enter.

>> 3 uuae

Note that *scanf 3 uuae, *SF 3 uuae, and >>3UUAE all give the same result.

To retrieve the next command containing 'uuae', in other words to repeat the search, type the following on the command line and press Enter. Note that in this case the first matching command, not the third, is retrieved.

>> *

You can also use the '*' to repeat the search in reverse chronological order as follows. To retrieve the first previous command containing 'uuae' type the following on the command line and press Enter.

<< *

1. You can use one or more '?' to specify a wild character in the search string. To find the first previous command containing 'uu' followed by any two characters followed by 'srr', type the following on the command line and press Enter.

<<uu??srr

1. To retrieve the first previous command containing the numbers '397', type the following on the command line and press Enter.

<< 1 397

Note that you need to include the '1' to indicate the first occurrence. This is needed to distinguish between the optional numeric value and the numeric search string.

Typing <<1397 gives the same result.

1. To execute a command but not place it in the command sequence, type the following on the command line and press Enter. This is useful to prevent commands you will not need again from cluttering the command sequence.

– dspmsg

Note that **–dspmsg** gives the same result.

1. To place a command in the command sequence but not execute it, type the following on the command line and press Enter. This is useful to prepare complicated or partially complete commands that you can retrieve later.

+cpyf fromfile(myfile) tofile(yourfile)

Full Screen Mode

If you are authorized to change your model profile, you can display the Edit Model Object List panel in full screen mode. In full screen mode the selection options and function keys are not displayed, leaving more room for displaying model objects.

To activate full screen mode, first press F18 from the Edit Model Object List panel to display the Edit Model Profile panel. Set the Full screen mode option to Y and press Enter. To return to regular mode, press F18 again and reset the Full screen mode option to N.

For more information on model profiles, see Model Profile section in this chapter.

Grouping and Navigation Aids

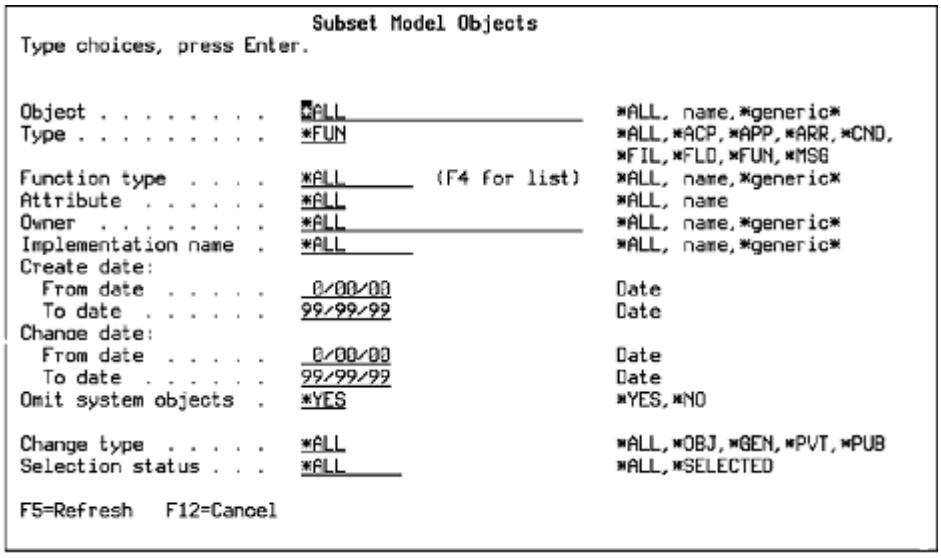
The Edit Model Object List panel provides a variety of tools to help you work with the objects in your model. These include the capability to:

- Subset a model object list.
- Position a model object list.
- Filter a model object list.
- Transfer control to various CA 2E editing panels.

The following sections describe these in more detail.

Subsetting a Model Object List

To display a subset of the model objects in the current list, press F17 from the Edit Model Object List panel. The Subset Model Objects panel displays:



Notes:

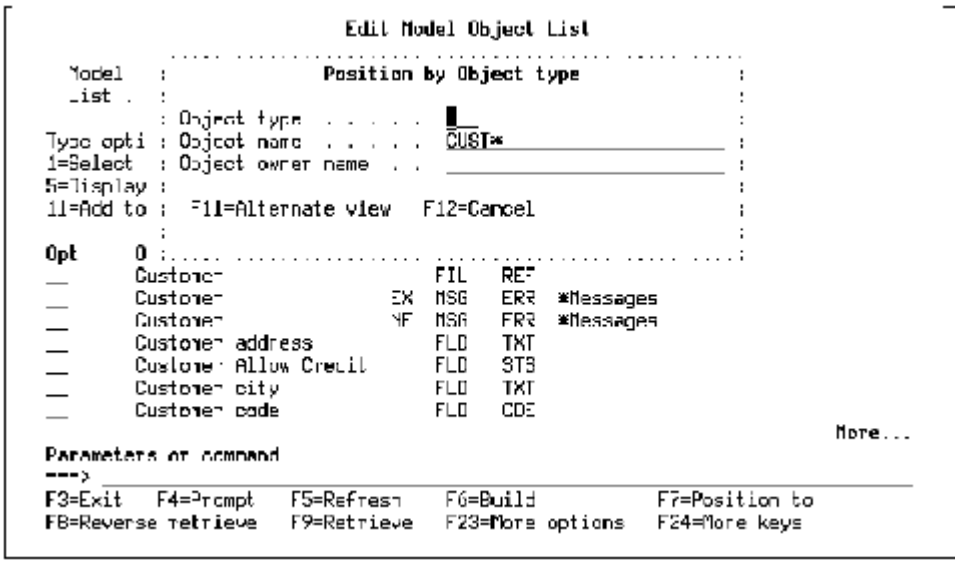
- If you are editing *ALLOBJ, you can also specify whether to display non-current versions of functions and messages.
- If you select another model object list, any subsetting you specified is also applied to the new model list.

Example

To display only Edit File functions on the Edit Model Object List panel type *FUN for the Type option and type EDTFIL for the Function type option. Press Enter twice. Note that the original model object list is not changed by this operation.

Positioning a Model Object List

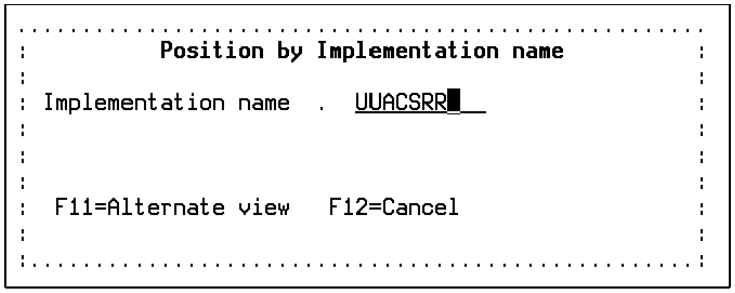
You can position the model list displayed to a specific object type, owner name, object name, or implementation name. Press F7 from the Edit Model Object List panel to display the positioner window.



Use the F11 key to toggle among the four positioning options. To position the model object list, enter values in the appropriate positioner window and press Enter. Note that you can specify partial names for all options except the Type option.

Example

To position the model object list to the model object whose Implementation name is UUACSRR, press F11 until the 'Position by Implementation name' window displays, type UUACSRR.



Press Enter.

You can use this, for example, to solve a problem in program applications where you only know the implementation name of the program in which an error occurred. You would then use impact analysis to identify all programs called by the program in error.

Note: In this case, the Edit Model Object List panel redisplays with the alternate view showing implementation information rather than object type, attribute, and owner. You can switch to other views by pressing F11.

Display Order of Model Objects

CA 2E displays model objects using one of the following key sequences depending on the values you enter in the positioner window.

1. Implementation name —The model objects are listed by implementation name, beginning with the implementation name you entered.
2. Owner name/type/object name—The model objects are listed by owner, for each owner by type, and for each type by object name, beginning with the model object that matches the values you entered.
3. Type/object name—The model objects are listed by type, and for each type by object name, beginning with the model object that matches the values you entered.
4. Object name/type—The model objects are listed by object name, and for each object name by type, beginning with the model object that matches the values you entered.

The following table shows how the display order will be keyed depending on the values you enter in the positioner window. A message displays at the bottom of the panel when the key sequence changes.

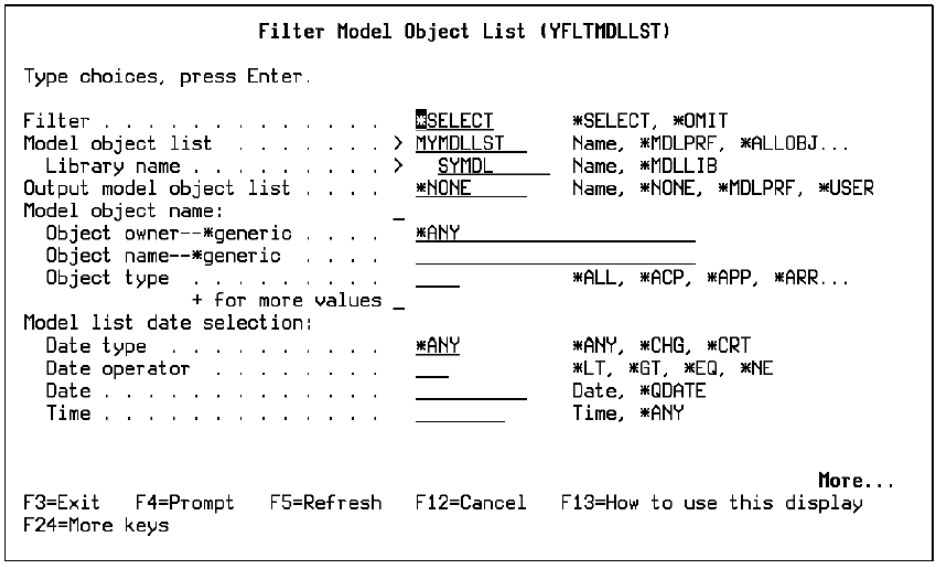
Values Entered in Positioner Window	Display is Keyed by
All fields blank	Object name and Type (the default)
Implementation name	Implementation name; see 1 above.
Owner name	Owner, Type, Object name; see 2 above.
Owner name and Type	Type and Object name; see 3 above.
Owner name, Type, and Object name	Type and Object name; see 3 above.
Type	Type and Object name; see 3 above.
Type and Object name	Type and Object name; see 3 above.
Object name	Object name and Type; see 4 above.

Filtering a Named Model Object List

The filtering panels available from the Edit Model Object List panel provide a powerful tool for selecting, omitting, and testing various criteria for objects in your model.

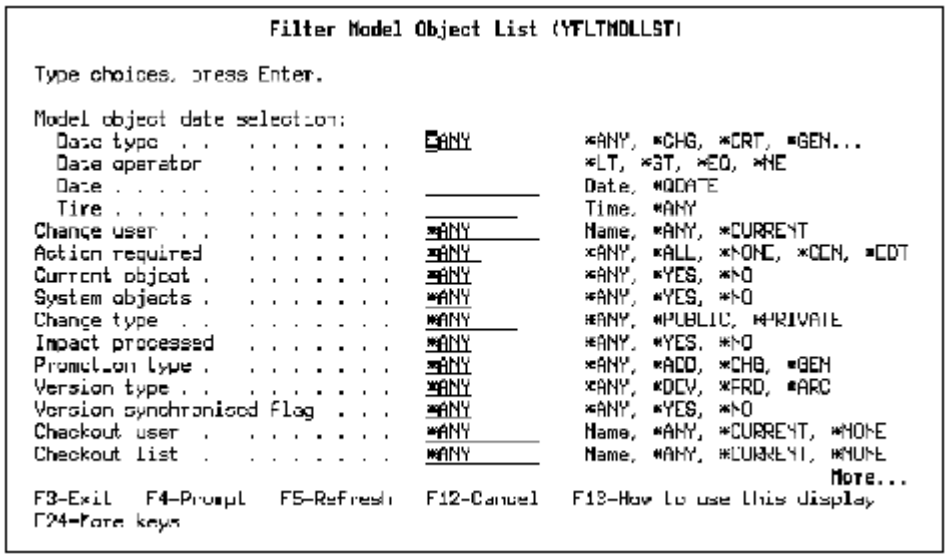
Note: The displayed model object list will be changed based on the filter unless you specify an output model object list. If you do not specify an output list, model object list entries not selected by the filter are by default deleted from the displayed list.

To apply the filtering tools to the current model object list, press F14. The Filter Model Object List panel displays. Press F10 to display the additional parameters.

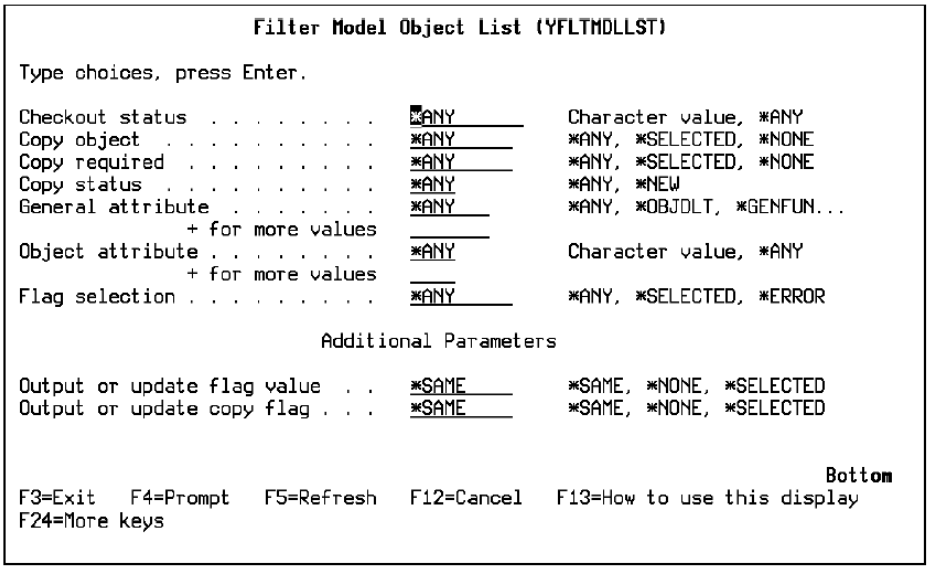


This is the first of three screens of filtering options. You can also change the input model object list and specify an output model object list on this screen. Scroll down for additional filtering options.

The following panel displays:



Scroll down to the third filtering screen:



For more information:

- On the YFLTMDLLST command, see the CA 2E *Command Reference Guide*.
- On the filtering options, see the All Objects List section in this chapter and the appendix titled "Change Control Facilities Reference Tables" in this guide.

Editing Model Objects

Each of the following options on the Edit Model Object List panel access a CA 2E editing panel appropriate to the selected model object:

- **Edit (option 2)**—A general editing option that you can use for any model object type. The specific CA 2E panel invoked depends on the object type of the selected model object as shown in the following table:

Object Type	CA 2E Panel
ACP	Edit Access Path Details
APP	Edit Application Areas
ARR	Edit Array Details
CND	Edit List Condition for LST Edit Field Condition Details for VAL
FIL	Edit File Details
FLD	Edit Field Details
FUN	Edit Function Details
MSG	Edit Message Function Details

- **Action Diagram (option 10)**—Calls the Edit Action Diagram panel for model objects of type FUN.
- **Parameters (option 13)**—Calls the Edit Function Parameters panel for model objects of type FUN.
- **Device Design (option 17)**—Calls the device design editor for model objects of type FUN.
- **Device Structure (option 18)**—Calls the structure design editor for PRTFIL/PRTOBJ functions.
- **Access Path (option 20)**—Calls the Display file access paths panel for model objects of type FUN.
- **Narrative/object (option 21)**—Calls the Edit Narrative Text panel for any model object type.
- **Narrative/owner (option 22)**—Calls the Edit Narrative Text panel for the owner of the selected object.
- **Open Functions (option 30)**—Calls the Open Functions panel for model objects of type FUN.

Viewing Model Objects

For any model object you can view both its edit panel in display-only mode and its model object description. This section explains both.

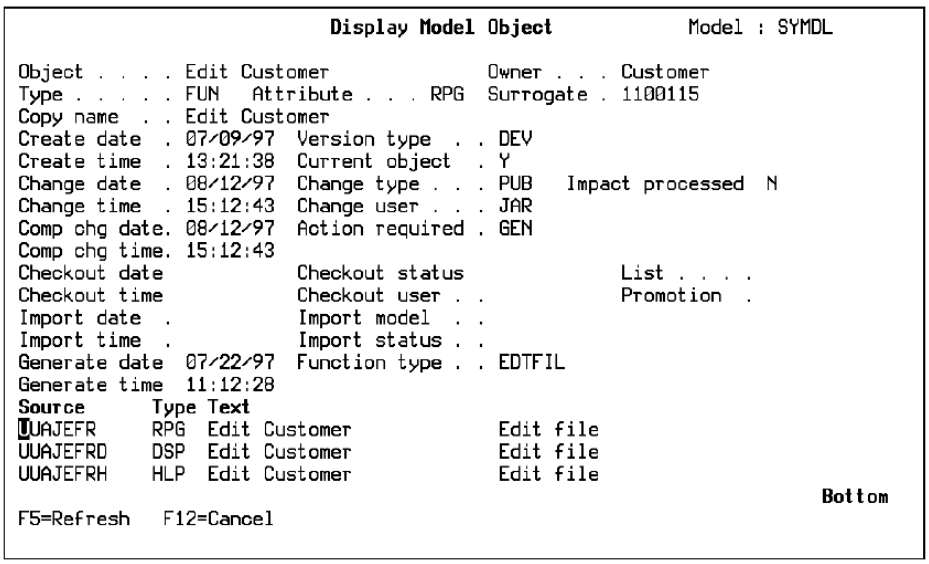
Viewing a Model Object's Edit Panel

From the Edit Model Object List panel, enter 5 to view the edit panel for a selected model object. This option is a general display option that you can use for any model object type. The specific CA 2E panel displayed depends on the object type of the selected model object. The panel displayed is the same as the panel displayed for edit option 2 but in view-only mode.

For more information on the panel displayed for each model object type, see the Editing Model Objects section in this chapter.

Viewing a Model Object Description

From the Edit Model Object List panel, enter 8 to display a model object's description from the All Objects list. You can also use the Display Model Object Description (YDSPMDLOD) command. The following panel displays:



A standard set of information is displayed for each type of model object. Additional information displayed varies according to the type of model object.

Note: If the information for a model object list entry does not match the model object description for the corresponding model object,CA 2E sets the Object select field for the list entry to 8 on the Edit Model Object List panel. You can use subfile select option 33 to refresh the model object list entry.

For more information:

- On the model object description, see the appendix titled "Change Control Facilities Reference Tables" in this guide.
- On model object list entries, see the Model Object Lists section in this chapter.

Creating Model Object Lists

Model object lists can be created in the following ways:

- Explicitly. For example:
  - Enter the name of a new list in the header of the Edit Model Object List panel.
  - Press F6 on the Work with Model Lists panel.
  - Use the CREATE parameter on the Edit Model Object List (YEDTMDLLST) command.

  In each case CA 2E automatically creates a new list with no entries.
- Automatically within a session. For example, you can specify a default session list or model list for commands in your model profile.
- Implicitly using model object list commands. For example, specify a non-existing output list on commands such as, YFLTMDLLST or YDSPMDLREF.

For more information on creating a model object list using the Work with Model List panel, see the Working with Model Object Lists section in this chapter.

Adding Entries to a Named Model Object List

This section explains how to add entries to the current model object list and how to add entries from the current model object list to an alternate model object list. In addition to these two methods, many model object list commands let you add entries to input or output model object lists; for example, the YDSPMDLUSG and YDSPMDLREF commands.

Adding Entries to the Current Model Object List

Press F6 from the Edit Model Object List panel to select and add model objects from the All Objects list (*ALLOBJ) to the named model object list on which you are currently working. This function key is not available for *ALLOBJ.

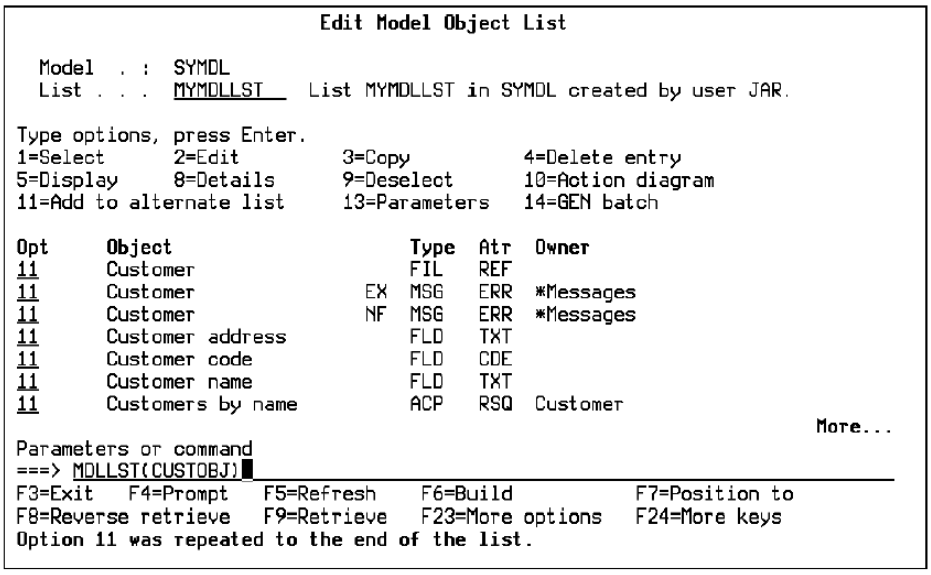
CA 2E displays the Select Model Object panel, listing all model objects from the All Objects (*ALLOBJ) list. You add model objects by entering selection option 1 for all model objects to be added and pressing Enter. The panel continues to display so you can continue selecting objects. Function keys are provided for positioning and subsetting the list.

Adding Entries to an Alternate Model Object List

Use selection option 11 from the Edit Model Object List panel to add selected model objects from the current model object list to an alternate model list. The default output list is the model object list specified in your model profile. This option invokes the Add a Model Object List Entry (YADDMDLLE) command. You can either prompt the command by pressing F4 or you can specify another model object list on the command line.

Example

Suppose you want to add model list entries from model list MYMDLLST to model list CUSTOBJ. Enter option 11 for each model object you want added to the CUSTOBJ list. In this example, enter 11 for the Customer file and press F13 to repeat the option to the end of the list. You can either press F4 to prompt the command, or you can enter the MDLLST(CUSTOBJ) parameter on the command line to specify the target list as shown:



Press Enter. The selected objects are added to the CUSTOBJ model object list.

Deleting a Model Object or a Model List Entry

The Edit Model Object List panel provides two deletion options:

- **Option 24**—Deletes the actual model object from the model
- **Option 4**—Deletes a model object list entry from the currently displayed named model object list.

CA 2E displays a confirm panel for each model list entry that you selected for deletion.

If an entry on a named model object list refers to a model object that has been deleted from the model,CA 2E sets the Object select field for the list entry to X on the Edit Model Object List panel to indicate that the corresponding model object no longer exists. (The Object select field is between the Subfile selector and the Object field.)

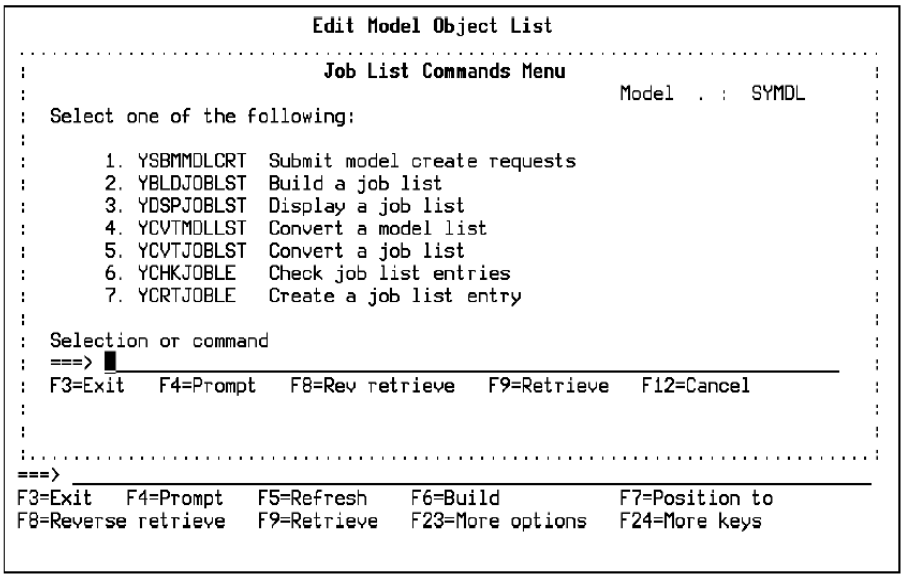
Note: You cannot delete files (FIL) or fields (FLD) using option 24.

You cannot delete a model object if it is used by other objects in the model. Use the impact analysis tools on the Edit Model Object List panel to determine the usages for the model object to be deleted. For example, enter selection option 91 for the model object you want to delete;CA 2E displays a list of the model objects that use it, including an indication of the way in which it is used.

For more information on impact analysis tools and model object usages, see the Impact Analysis section in this chapter.

Selecting Job List Commands

The Edit Model Object List panel provides a menu of job list commands as an aid to preparing and submitting your model object lists for generation and creation. Press F19 to display the Job List Commands Menu window:



Enter the appropriate menu option on the command line to prompt the selected command. Many of the parameter defaults for the job list commands are set in your model profile.

Copying Model Objects

This section discusses change control facilities for creating copies of functions and messages and copying model objects between model object lists and between models.

Creating Copies of Functions and Messages

A copy of a function or message can be either an entirely separate model object or a version.

- Selection option 3 on the Edit Model Object List panel lets you copy functions (FUN) and messages (MSG). This creates a new and independent model object. You will be prompted to enter the following information for the new model object.
  - For MSG, you can change the object name and type. The following values for type are interchangeable: INF, ERR, STS, CMP. When copying a RTV or EXC message the type cannot be changed.
  - For FUN, you can change the object name, access path, and file. For certain functions, you can also change the function type by pressing F8.

  Note: If you change access path, file, or function type, you will probably need to edit the function options, device design, and action diagram to obtain a working function.
- Another way to create a copy of a function or message is to create a version using the Create Object Version command (YCRTOBJVSN). All the versions of a function or message form a version group.CA 2E automatically keeps track of the members of a version group.

For more information on versions, see the Working with Versions of Functions and Messages section in this chapter.

Copying Entries Between Model Object Lists

Use the Copy Model Object List (YCPYMDLLST) command to copy list entries from one model object list to another. The input and output model object lists may be in different models. If the output model object list exists before the copy, you can either replace the model object list or add the copied objects to it. In addition, the copied list entries can be refreshed in the output model object list. You may selectively copy using the flag selection (FLAGVAL) parameter.

Performing User-defined Actions on Model List Entries

This section describes how to:

- Use substitution variables.
- Define user-defined subfile select options associated with a command string.
- Execute a model object list in order to perform an action on each entry of the model list.

Copying Model Objects Between Models

Use the Copy Model Object (YCPYMDLOBJ) command to copy model objects between models. This command requires an input model object list of the model objects you want to copy. This can be an existing model object list (for example, your session list) or you can create one using one of the model object list commands, such as the Build a Model Object List (YBLDMDLLST) command.

In addition, you need to explicitly select the model objects to be copied from the input model object list. Use one of the following to select model objects:

- Use the Edit Model Object List for Copy panel, which you invoke using the Edit Copy List (YEDTCPYLST) command. This panel has many of the same options and function keys as the Edit Model Object List panel.

  Type 1 against the model objects you want to select for copying. To select all list entries, type 1 for the first model object displayed, press F13 to repeat the selection for all list entries, and press Enter. An '*' will appear in the Copy Select field of each selected model object.

  Use option 9 to deselect a selected model object. Option 7 lets you rename a model list entry for the purpose of copying to avoid conflicts with objects with the same owner/name/type in the target model.

  Use the edited model object list as input to the YCPYMDLOBJ command.
- Set the OUTCPYOBJ parameter to *SELECTED on one of the following commands:
  - Build a Model Object List (YBLDMDLLST)
  - Add a Model Object List Entry (YADDMDLLE)
  - Filter a Model Object List (YFLTMDLLST)
- Set the CPYOBJ parameter to *SELECTED on the Change a Model Object List Entry (YCHGMDLLE)

Note: To prepare a list for copying, you should use only the model object list commands; for example, YBLDMDLLST, and/or the YEDTCPYLST command. The Build Copy List (YBLDCPYLST) command is available only for backward compatibility with previous releases of and should not be used.

Selected model objects will contain a "*" in the Copy Select field on the Edit Model Object List for Copy panel. After you run the YCPYMDLOBJ command, any implicitly selected model objects will contain a "!" in the Copy Select field. To view the implicitly selected objects before copying, run the YCPYMDLOBJ command in *PREPASS mode.

For more information on the YCPYMDLOBJ process, see the *Administrator Guide*.

Using Substitution Variables

The following substitution variables let you symbolically pass various list entry attributes to the command or user-option that is to operate on the model object list. You can use these substitution variables with the Execute a Model Object List (YEXCMDLLST) command, on the Edit Model Object List panel Command line, and with user-defined options.

The prefix for the substitution variables may be either '&' or '@'. If you are using PDM to maintain your user option list, you need to use '@' as the prefix. You can also specify an alternate character in the Toolkit data area, YPEXCHA.

Substitution Variable	Corresponding Model Object Attribute
&YN	Object name
&Y@	Object surrogate
&YT	Object type
&YA	Object attribute
&YO	Object owner name
&YW	Object owner surrogate
&YY	Object owner type
&YR	Object group surrogate
&YM	Model list name
&YI	Object implementation name
&YP	Object promotion type
&YS	Object SEU type
&YU	User name
&YL	Model library name
&YG	Change type
&YF	Function type
&YZ	Assimilated file

Note: If you are unable to enter '@' into the command parameter (for example, if it is numeric), you can either use the RQSDTA parameter on the YEXCMDLLST command and enter the command as a string, or you can specify RQSDTA(*USROPT) and specify a PDM user-defined option.

Enclose the &YN, &YO substitution variables in quotes since they can result in text containing blanks.

Defining and Editing User-defined Options

You can define your own subfile select options associated with a command string. User-defined options must be alphabetic; numeric options are reserved for use by CA 2E. You can use these options on the Edit Model Object List panel and you can specify them as the action to be performed on the Execute a Model Object List (YEXCMDLLST) command.

A user-defined option consists of one or two characters to be associated with a command string. This command string may contain embedded substitution variables that see attributes of the model object list entry. The variables supported are the same as those listed above for the Execute Model Object List (YEXCMDLLST) command.

User-defined options are contained in the User-Defined Options (QAUOOPT) file that is shipped with Toolkit. The default names for the user option library, file, and member are contained in your model profile and can be changed. Use PDM to create and edit user-defined options in the file/member specified in the model profile.

Note: You should copy the User-Defined Options file to a user library (for example, your model library) in order to preserve user-defined options when installing a new release of Toolkit.

The following table lists examples of user-defined options:

Option	Command
YC	CHGCURLIB @YL
OD	YDSPMDLOD OBJSGT(@Y@)
YS	DSPSPLF FILE(@YI) JOB(@YI)
JL	DSPJOBLOG
SL	SBMJOB ??CMD(SAVLIB LIB(&N))
WS	WRKSBMJOB
DF	YDOCMDLFUN MDLFILE('@YO') MDLFUN('@YN')

Note: You can use the / option on the Edit Model Object List panel as a temporary user-defined selection option.

For more information on the / selection option, see the Command Line section in this chapter.

Executing a Model Object List

Press F10 from the Edit Model Object List panel to prompt the Execute a Model Object List (YEXCMDLLST) command. This command causes a specified action to be performed on each entry of the model object list. The action can either be a command or a user-defined option.

Note: F10 is not available for the *ALLOBJ list.

The substitution variables listed at the beginning of this section let you symbolically see attributes of the model list entries and pass them to the command or user-option that is to operate on the model object list.

For more information on the YEXCMDLLST command and its parameters, see the CA 2E *Command Reference Guide*.

Example

The following is a control language program showing a method of using model list commands, substitution variables, and the YEXCMDLLST command. Note that this is for illustration only and CA does not warrant usability or functionality of the program.

```
PGM &MDLLST
/*T:  Archive compile listings from given model list                */
/*Z:  CRTCLPGM                                                      */

/*H:  Archive spooled files from compiles of model objects          */
/*H:  referenced on named model object list                         */
/*W: Note: the model library is assumed to be in the library list.  */

/*W: This CL source is supplied solely to illustrate use of         */
/*W: some model object list commands.  CA                           */
/*W: does not warrant usability or functionality of the program.    */

/*---- ------------------------------------------------------------ */
/*M:  MAINTENANCE                                                   */
/*M:  NONE                                                          */
/*---- ------------------------------------------------------------ */
     DCL     VAR(&MDLLST) TYPE(*CHAR) LEN(10)'  /* Model list */
     DCL     VAR(&MSGID) TYPE(*CHAR) LEN(7)
     DCL     VAR(&MSGDTA) TYPE(*CHAR) LEN(50)
     DCL     VAR(&MSGF) TYPE(*CHAR) LEN(10)
     DCL     VAR(&MSGFLIB) TYPE(*CHAR) LEN(10)
/*---- ------------------------------------------------------------ */
     MONMSG (CPF0000 Y2E0000 Y2V0000)EXEC(GOTO  ERROR)
/*---- ------------------------------------------------------------ */
/*H:  1. Remove invalid list entries (no matching objects) and refresh */
/*H:     the information on the list. Preserve the original list    */
/*H:     by output to another list. Do not output entries for which */
/*H:     there is no matching model object.                         */
       YCHKMDLLST MDLLST(&MDLLST) CHKTYPE(*BYSGT) +
         UPDLST(*RFSALL *RMVERR) OUTLST(YARCSPLF01)
       MONMSG  MSGID(Y2E0328)        /* Model Object not found */

/*H:  2.  Filter the list to remove objects which do not have       */
/*H:     associated implementation objects.                         */
       YFLTMDLLST FILTER(*SELECT) +
         MDLLST(YARCSPLF01)  GENATR(*GENOBJ)
```

```
/*H:  3. Archive the spooled files to file COMPILES in library          */
/*H:     ARCHIVES. Remove entries for which the CPYSPLF                 */
/*H:     completed successfully. We assume errors were due to          */
/*H:     non-existent spool files.                                      */
/*H:     Note: the @YI substitution variable specifies to               */
/*H:     substitute with the name of the Implementation object.         */
         YEXCMDLLST CMD(CPYSPLF FILE(@YI)+
           TOFILE(ARCHIVES/COMPILES) +
            JOB(@YI) TOMBR(@YI)) )  MDLLST(YARCSPLF01) +
           ERRLVL(*NOMAX) UPDLST(*RMVOK)

/*H:  4. Make a job list out of the remaining entries which did not     */
/*H:     have spooled files.                                            */
         YCVTMDLLST FRMMDLLST(&MDLLST)  +
           JOBLST(YARCSPLF01)  CRTOPT(*COMPILE)

/*H:  5. Submit the job list for recompile. (Re-run this program        */
/*H:     when complete).                                                */
          YSBMMDLCRT JOBLST(YARCSPLF01) GENLIB(*GENLIB) +
           SRCLIB(*GENLIB) JOBD(*MDLVAL) CRTJOBD(*JOBD)
/*---- ------------------------------------------------------------------------------------- */
ENDPGM:
                  RETURN
/*---- ------------------------------------------------------------------------------------- */
/*H:  99. ERROR HANDLING                                                */
ERROR:   RCVMSG     MSGTYPE(*EXCP) MSGDTA(&MSGDTA)+
           MSGID(&MSGID) MSGF(&MSGF) MSGFLIB(&MSGFLIB)
         MONMSG     MSGID(CPF0000)
         SNDPGMMSG  MSGID(&MSGID)
         MSGF(&MSGFLIB/&MSGF) +
           MSGDTA(&MSGDTA) MSGTYPE(*ESCAPE)
         MONMSG     MSGID(CPF0000)
         GOTO  CMDLBL(ENDPGM)
ENDPGM
```

Model Object Audit Information

CA 2E automatically maintains the following audit stamps in the All Objects list for each model object:

- Creation date and time
- Change date and time
- Component change processing date and time
- Generation date and time
- Import date and time
- Check out date and time

For more information about the meaning of these audit stamps, see the appendix "Change Control Facilities Reference Tables."

Tracking Changes to Model Objects

Whenever you change a model object,CA 2E updates the following information in the model object's description in the All Model Objects list:

- Date and time the object was changed
- User profile of the developer making the change
- Type of change
- Impact processed indicator

To view change information, press F11 on the Edit Model Object List panel until the change information as shown in the following panel displays:

```
                          Edit Model Object List

  Model  . :  SYMDL
  List . . .  MYMDLLST    List MYMDLLST in SYMDL created by user JAR.

Type options, press Enter.
1=Select      2=Edit         3=Copy          4=Delete entry
5=Display     8=Details      9=Deselect      10=Action diagram
11=Add to alternate list     13=Parameters   14=GEN batch
                                 ---------Changed----------   Impact
Opt     Object                        Date      User      Type  Processed
_       Customer                      08/15/97  RMG       OBJ     *NO
_       Customer                 EX             KED               *NO
_       Customer                 NF             KED               *NO
_       Customer address                        KED               *NO
_       Customer code                           KED               *NO
_       Customer name                 07/09/97  KED       PUB     *NO
_       Customers by name             08/14/97  JAR       PUB     *NO
                                                                      More...
Parameters or command
===> ____________________________________________________________________
F3=Exit   F4=Prompt   F5=Refresh   F6=Build          F7=Position to
F8=Reverse retrieve  F9=Retrieve  F23=More options   F24=More keys
```

Notes:

- For a named model object list, the Change date is the date the list entry was created and may not reflect the model object's current status. You can use F15 to update the model object list to contain current information from the All Objects list; this invokes the Check a Model Object List (YCHKMDLLST) command.
- A model object is not considered to be changed in the following circumstances:
- The object is accessed as if to edit but no changes are actually made.
- The object has been copied from another model and has not been changed since copying.

Determining the Change Type

The change type describes the way in which a change to a model object affects other objects that use the changed object. Its purpose is to retain the integrity or functionality of both model objects and implementation objects. It is used primarily during component change processing, a process that ensures that the effect of a change is distributed throughout the model.

CA 2E determines the change type using a shipped table that assigns a change type to each position on a CA 2E panel where you can possibly change a model object.

For more information:

- On component change processing, see the Impact Analysis section in this chapter.
- On the shipped table of change types, see the appendix titled "Change Control Facilities Reference Tables" in this guide.

Following is a list of change types and their definitions. Only private and public changes invoke component change processing.

- **Object Only (*OBJONLY)**—A change that affects only the model object and does not require regeneration. The change has no effect on model objects that use the changed object.
- **Generation Required (*GEN)**—A change that affects only the model object and requires that the changed object be regenerated to effect the change in its implementation object. This change type is used only for access paths and external functions. The change has no effect on model objects that use the changed object.
- **Private (*PRIVATE)**—A change to an object that requires regeneration of the external functions and access paths that use it to effect the change in the implementation objects.

  For example, if you change a file on the Edit Database Relations panel, you need to regenerate all external functions and access paths that use the file. Or, if you change the action diagram of an internal function, you need to regenerate all external functions that call it.
- **Public (*PUBLIC)**—A change to a model object that changes its interface with the model objects that use it. This change also requires the following additional processing:

  1. Model objects that use it may need to be edited.
  2. External functions and access paths that use it need to be regenerated.

  For example, if you change the parameters of an internal function, you need to edit all functions that call the changed function and then regenerate all external functions that contain it.

The four change type values just presented were listed in order of increasing significance:

1. Object Only
2. Generation Required
3. Private
4. Public

In other words, if you make multiple changes to an object, a higher-numbered change overrides a lower-numbered change, and not vice versa. For example, a public change overrides a private change, but a generation required change does not.

Impact Analysis

CA 2E's impact analysis facilities let you determine the impact of a proposed or actual change to model objects in a CA 2E model and ensure the integrity of a set of changes by identifying and including dependent objects. CA 2E impact analysis facilities include:

- Automatic update of audit stamps for various processes including creation, change, copy, and generation.
- Commands and processes to identify model objects that will be affected by a change, including a distinction between changes that:
  - Also affect using objects and require editing.
  - Are internal to the object and require regeneration of using external functions and access paths.
- Support for where-used and referenced objects.
- Expansion across model object types; for example:
- Field, to File, to Access Path, to Function.
- Recursive and multiple-level expansion.
- Filters and controls.
- Full integration with other CA 2E edit and generation facilities.
- Output either to a custom display or to model object lists.

Introduction

This discussion of CA 2E's impact analysis tools are divided into the following sections:

- **Model Object Cross References**—Discusses commands and interactive panels you can use to determine usages and references for any model object.
- **Simulating Changes to Model Objects**—Shows how to simulate a change to a model object to determine the impact of the change on other model objects.
- **Component Change Processing**—Discusses component change processing, an automated impact analysis tool that determines how a change to a model object affects other objects in the model and also records whether the affected objects need to be edited or regenerated.

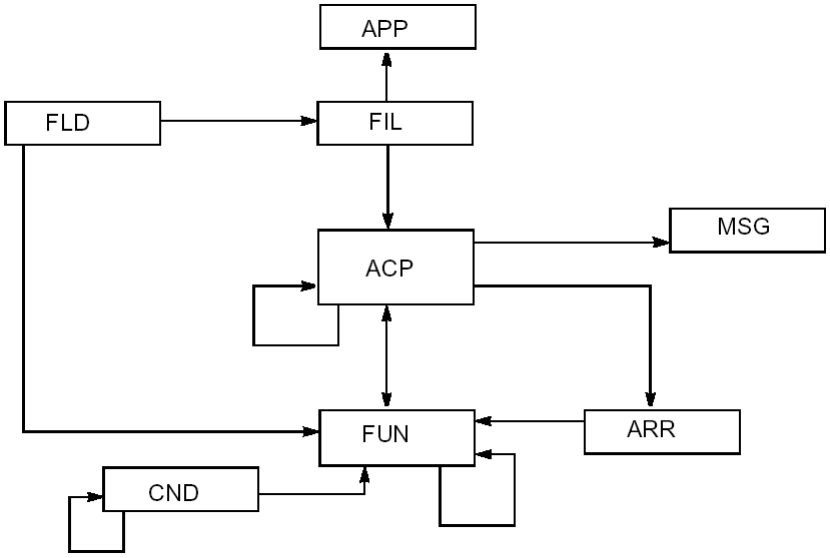
Model Object Cross Reference Facilities

Model object cross reference facilities consist of a set of commands and interactive panels you can use to determine usages and references for any model object.

The process of determining either usages or references for a model object is known as expansion. Using model cross references facilities, you can expand usages or references for a model object across model object types to any depth.

Understanding Model Object Usages

The usages for a model object are all the model objects that use it (or all model objects it is used by). In other words, usages for a given model object are the model objects that require it in order to be complete and are external to the model object. A model object's usages are sometimes referred to as using objects. The following diagram shows the possible usages for each CA 2E model object type:



For example:

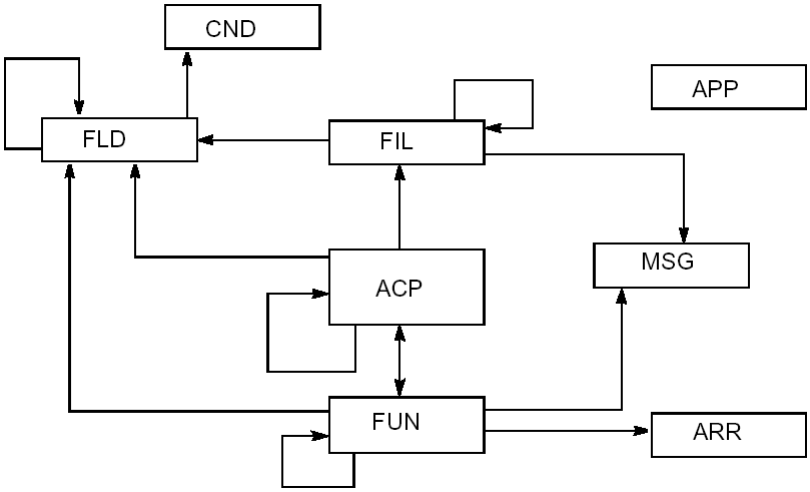
- Database fields (FLD) can be used by database files (FIL) or functions (FUN). In turn, files can be used by access paths (ACP) and application areas (APP).
- Access paths (ACP) can be used by messages (MSG), arrays (ARR), functions (FUN), and other access paths (ACP).

Note: There is no direct relationship between fields (FLD) and access paths (ACP). In other words, a change to a field does not directly affect the access paths that use it.

For more information on the note and the possible ways a given model object type can be used by other model objects types, see the CA 2E *Command Reference Guide*.

Understanding Model Object References

The references for a model object are the model objects that are referenced by the object, or that the original model object refers to internally. In other words, references are the model objects the referring model object requires in order to be complete or to exist. The following diagram shows the possible references for model object types. Note that application areas (APP) have no references.



For example:

- Database fields (FLD) can reference conditions (CND) or other fields (FLD).
- Functions (FUN) can reference messages (MSG), arrays (ARR), other functions (FUN), fields (FLD), and access paths (ACP).

For a table of ways a model object can reference other model objects, see the *Command Reference Guide*.

Interactive Impact Analysis

The model object cross reference facilities provide the following two interactive panels:

- Display Model Usages
- Display Model References

Both interactive panels provide a variety of controls and filters including recursion, scope, selection, and positioning to help you analyze your model and determine the impact of proposed changes.

Following is the header for the Display Model Usages panel showing the fields for controlling the expansion of usages. One way to display this is to use option 91 from the Edit Model Object List panel. The header for the Display Model References panel is similar.

Note: Some of the following screenshots in this section do not show the Include Inactive Code field.

The following discusses each of the controls and filters provided on this panel. Note that the Object and Type fields are positioners; all other fields are record selectors. Also, all values, including your changes, are carried forward each time you invoke this panel.

Using the Level Number

The Level number in the upper-right of the screen shows the number of panels displayed to get to the current panel. In other words, it is the number of times you have invoked the Display Model Usages and/or the Display Model References panel. You can press F15 at any level to return to Level 001.

Using the Gen Objs and Total Counts

The Total specification in the upper-left of the screen shows the total number of model objects displayed; the Gen objs specification shows the number of access paths and external functions (model objects that require generation) included in the total. These counts include all model objects expanded, including those you explicitly exclude from the resulting display or report using selection options on this panel.

Using the Object and Type Positioner Specifications

Use the Object and Type specifications to position the objects displayed to a specified model object, model object type, or both. These appear only when Reason is *FIRST or *ENTRY. In this case, the model objects are displayed sorted by object name and type.

Note: *ENTRY is a special Reason code reserved for use by CA 2E.

Using the Include Inactive Code Specification

This field allows you to control the identification or suppression of objects that have been commented out (deactivated) in an Action Diagram.

Include inactive code: *YES

When Include inactive code is set to *YES, the usage and reference expansion generally behaves as it does at previous releases.

An additional field Wrn (Warning) in the subfile record indicates, with an '*' in the second character of the field, when the corresponding Action Diagram call is deactivated (commented out). Additionally, any object that appears in the usage/reference report *only* by virtue of deactivated code is also marked with an '*'.

Note: The first character of the field Wrn (Warning) in the subfile record is currently unused.

Include inactive code: *NO

When Include inactive code is set to *NO, the usage and reference expansion will consider the deactivated (commented out) status of action diagram calls. This has 2 major effects:

1. Any deactivated action diagram calls within the reference/usage expansion of a target object will be ignored.
2. No objects will be included by virtue of the deactivated call to the object. In other words, the commented out function is not expanded.
3. No objects will be included by virtue of the deactivated call to the object. In other words, the commented out function is not expanded.

Include inactive code: *IGN

When Include inactive code is set to *IGN, the usage and reference expansion behaves exactly as it does at previous releases, and there is no differentiation between active code and deactivated (commented out) code – i.e. all code is treated as active, even if it is commented out.

The additional field Wrn (Warning) in the subfile record is not relevant when this option is used.

Examples

The two examples in this section depict the new impact analysis for objects containing deactivated (commented out) code. Consider the following scenario:

- Function_A calls Function_B.
  Function_A **also** calls Function_C.
  Function_B calls Function_C.
- Function_B's call to Function_C is deactivated (commented out)
- Function C is analyzed for usages.

Example 1—Include Inactive Code = *YES

```
*Scope  . .  *NOMAX  Reason . .  *ALL*
Opt Object                Typ Atr Owner                Lvl Reason  WRN
    Function_C            FUN RPG 17009615             000 *OBJECT
    Function_B            FUN RPG 17009615             001 *ACTION  *
    Function_A            FUN RPG 17009615             002 *ACTION  *
    Function_A            FUN RPG 17009615             001 *ACTION
```

Note: Function_B's record has a '*' in character 2 of the WRN field, to indicate that the action diagram call to Function_C has been deactivated (commented out). Hence the Function_A (LVL 002) which calls Function_B) is also marked with a '*' in the WRN field. However the Function_A (LVL 001) which calls Function_C directly) is not marked with a '*' in the WRN field.

Example 2—Include Inactive Code = *NO

```
*Scope  . .  *NOMAX  Reason . .  *ALL*
Opt Object                Typ Atr Owner                Lvl Reason  Wrn
    Function_C            FUN RPG 17009615             000 *OBJECT
    Function_A            FUN RPG 17009615             001 *ACTION
```

Note: Because Function_B's call to Function_C is deactivated (commented out)* Function_B is not included in the usages report and not expanded, therefore Function_A (LVL 002) is also not included.

Example 3–Include Inactive Code = *IGN

*Scope . . *NOMAX Reason . . *ALL*
Opt Object Typ Atr Owner Lvl Reason WRN
Function_C FUN RPG 17009615 000 *OBJECT
Function_B FUN RPG 17009615 001 *ACTION
Function_A FUN RPG 17009615 002 *ACTION
Function_A FUN RPG 17009615 001 *ACTION

Note: These are the same results that you would see with 8.1 SP2. They also look the same as with Include Inactive Code=*YES but the WRN field is not used, since there is no differentiation between active and deactivated code.

Using the Exclude System Objects and Current Objects Only Specifi

Using the Scope Specification

The Scope specification lets you limit or control the number of levels or depth of the expansion; in other words, it indicates when the expansion process is to stop. The default is *NEXT when you access the panel using subfile selection options; the default is *NOMAX if you use a command to access the panel.

The more you can limit the expansion, the faster the process and the easier to interpret the results. Specifying Scope is especially useful when expanding low-level model objects or model objects that are used frequently by other model objects; for example, conditions (CND), fields (FLD), files (FIL), and access paths (ACP). It is not as critical to limit expansions for functions (FUN) and messages (MSG).

Following are the possible values for Scope:

Value	Meaning
*NOMAX	No limit on the expansion. The maximum level is 999.
*NEXT	Expand only to the next level. This allows you to step through the expansion one level at a time.
*GENFUN	Expand objects up to and including the first external function. Applies only to usages.

Value	Meaning
*GENOBJ	Expand objects up to and including the first object requiring generation; for example, an external function or access path. Applies only to usages.
*EXTFUN	Expand objects up to and including the first external function after the first level. For references, this value is intended for use only with functions. It is a combination of Scope and Filter since only functions are included.
*INTFUN	Expand objects up to and including the first internal function after the first level. For references, this value is intended for use only with functions. It is a combination of Scope and Filter since only functions are included.
Object type	Expand objects until an object of the specified type is encountered. For example, *ACP, *FUN. Use this only when appropriate for the object type you are investigating. Applies only to usages.

Using the Scope Specification

The Scope specification lets you limit or control the number of levels or depth of the expansion; in other words, it indicates when the expansion process is to stop. The default is *NEXT when you access the panel using subfile selection options; the default is *NOMAX if you use a command to access the panel.

The more you can limit the expansion, the faster the process and the easier to interpret the results. Specifying Scope is especially useful when expanding low-level model objects or model objects that are used frequently by other model objects; for example, conditions (CND), fields (FLD), files (FIL), and access paths (ACP). It is not as critical to limit expansions for functions (FUN) and messages (MSG).

Following are the possible values for Scope:

Value	Meaning
*NOMAX	No limit on the expansion. The maximum level is 999.
*NEXT	Expand only to the next level. This allows you to step through the expansion one level at a time.
*GENFUN	Expand objects up to and including the first external function. Applies only to usages.

Value	Meaning
*GENOBJ	Expand objects up to and including the first object requiring generation; for example, an external function or access path. Applies only to usages.
*EXTFUN	Expand objects up to and including the first external function after the first level. For references, this value is intended for use only with functions. It is a combination of Scope and Filter since only functions are included.
*INTFUN	Expand objects up to and including the first internal function after the first level. For references, this value is intended for use only with functions. It is a combination of Scope and Filter since only functions are included.
Object type	Expand objects until an object of the specified type is encountered. For example, *ACP, *FUN. Use this only when appropriate for the object type you are investigating. Applies only to usages.

Using the Filter Specification

The Filter specification lets you limit the type of model objects displayed; it does not affect which objects are included in the expansion. You can specify this before beginning the expansion or afterward to help you analyze the results of the expansion.

Following are the possible values for Filter:

Value	Meaning
*ANY	Select all types of model objects; no filtering is done. This is the default.
*DBFFUN	Select only database functions.
*ERR	Select only error usages; i.e., usages by deleted objects.
*EXTFUN	Select only external functions.
*GENFUN	Select only generatable functions.
*GENOBJ	Select only generatable objects; for example, access paths and external functions.
*INTFUN	Select only internal functions.
Object type	Select only model objects of the specified type; for example, *ACP, *FUN.

Using the Reason Specification

Using the Reason code you can display just those model objects that were included on the display for a specified reason. The Reason code indicates the way in which the displayed object is used by, or referenced by, the expanded model object. For example, if you expand usages for a field (FLD), a reason of *FILENT for a file (FIL) indicates that the field is an entry attribute of the file.

Each time you change the Reason field, usages or references are expanded again for the original model object.

Following are the possible values for Reason:

Value	Meaning
*FIRST	Select just the first using or referenced object. In this case, the model objects are sorted and displayed by object name and type. This is the default.
*ALL	Include all usages or references and display the Reason code for each. In this case the model objects are displayed in the order in which they are encountered by the expansion process. As a result, the level numbers shown reflect the relationships between the listed objects. On the corresponding printed report the levels are shown indented to highlight the relationships more clearly.
Reason code	Select just those model objects that use or reference the original model object in the way indicated by the value specified. Two examples follow this table.

Following are two examples of specific Reason codes:

- Suppose you have expanded usages for a field. Enter *ACTION in the Reason field to display just those functions where the field is used in an action diagram action.
- Suppose you have expanded references for a function. Enter *BASED in the Reason field to display the access path on which the function is based.

For more information on the Reason specification and a list of all Reason values, see the CA 2E *Command Reference Guide*.

Working with Usages Interactively

You can work with usages interactively from the Edit Model Object List panel in the following ways:

- Use option 91 to expand usages for a selected model object. The Display Model Usages panel displays showing usages for the selected model object. Note that the Scope field is set to *NEXT.

  Note: This is the recommended method to work with usages interactively.

- Press F20 to prompt the Display Model Usages (YDSPMDLUSG) command for the named model object list you are editing. In this case usages are not expanded for the model list entries before the Display Model Usages panel displays. Instead, the list is converted as explained in step 1 of the following example. Note that the Scope field is set to *NOMAX.

Example

This example demonstrates the Display Model Usage utility and the use of the Scope specification. The concepts shown also apply to references.

1. From the Edit Model List panel press F20 to display usages. The Display Model Usages command is prompted from which you can specify whether the output is to be displayed (*), printed (*PRINT), or copied to a model object list (*MDLLST).

   Note: Although this is not the recommended method to work with usages interactively, it is included in this example to explain the converted list displayed on the first panel. Starting at step 2, this example shows both methods of working with usages.

   If you choose to display usages, begins by displaying the contents of the list you specified, updated to reflect the current state of each model object from *ALLOBJ. This is indicated by *ENTRY in the Reason column for each model object. The name of the originating model object list is shown in the Converted List field in the upper-left of the screen. Note that the original list is not changed by this process.

```
Gen objs :    17              Display Model Usages        Model  . : SYMDL
Total  . :    41                                          Level  . : 001
Converted list . . :  JAR
                                                Exclude system objs  .  *YES
Scope  . .  *NOMAX    Filter . .  *ANY___   Current objects only .  *YES
Object . .  ____________________________ Type . .  ___    Reason . .  *ENTRY

2=Edit     3=Copy   4=Delete object   5=Display   8=Details   10=Action diagram
13=Parms   14=GEN batch               15=GEN interactive      16=Y2CALL

Opt Object                    Typ Atr Owner                      Lvl Reason
_  Customer                   FIL REF                            000 *ENTRY
__ Employee                   FIL REF                            000 *ENTRY
__ Order                      FIL REF                            000 *ENTRY
__ Order Detail               FIL CPT                            000 *ENTRY
__ Product                    FIL REF                            000 *ENTRY
__ Compute discount           FUN RPG Product                    000 *ENTRY
__ Currency ext -> int        FUN USR *Field attribute types     000 *ENTRY
__ Currency int -> ext        FUN USR *Field attribute types     000 *ENTRY
__ Edit a Customer            FUN RPG Customer                   000 *ENTRY
                                                                      More...
F3=Exit   F5=Refresh   F9=Command line   F12=Previous         F15=Top level
F16=Build model list   F21=Print list    F22=File locks       F23=More options
```

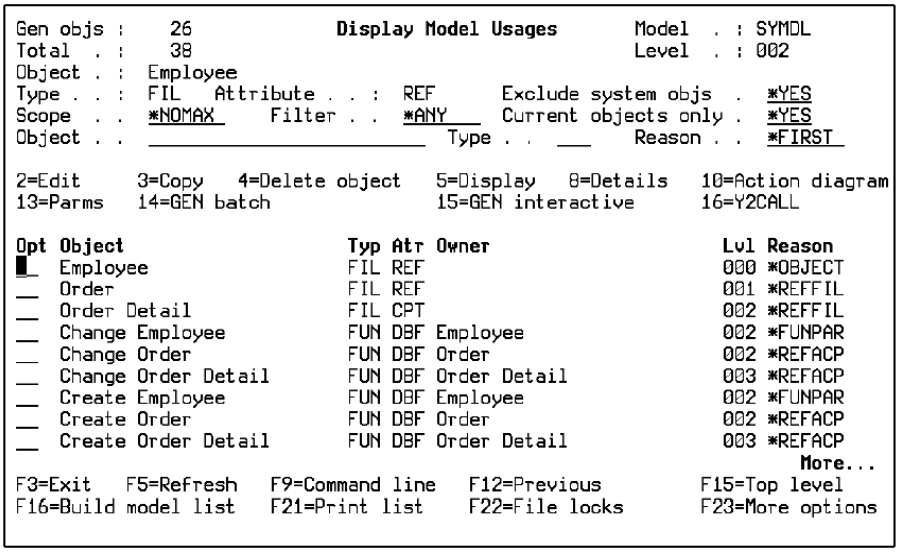
The converted list of model objects displayed differs from the contents of the original model object list in the following ways:

- By default, only model objects that are currently active in the model are displayed. Non-current versions are not displayed. See the Working with Versions of Functions and Messages section in this chapter for more information on the current version.
- Details from the All Objects list are displayed for each model object; for example, if the name of the actual model object differs from that of the model object list entry, the model object name is displayed. In other words, the display reflects the current state of the model.
- Model objects that appear on the model object list, but have been deleted from the model, are not displayed. Any list entries that see the deleted objects are ignored.

Note: Your model object list is not changed by this operation.

You can now use the selection options on any of the model objects displayed. To see additional options, press F23.

2. To display all usages for the internal function, Retrieve Customer, type selection option 91 against Employee and press Enter. The following panel displays:



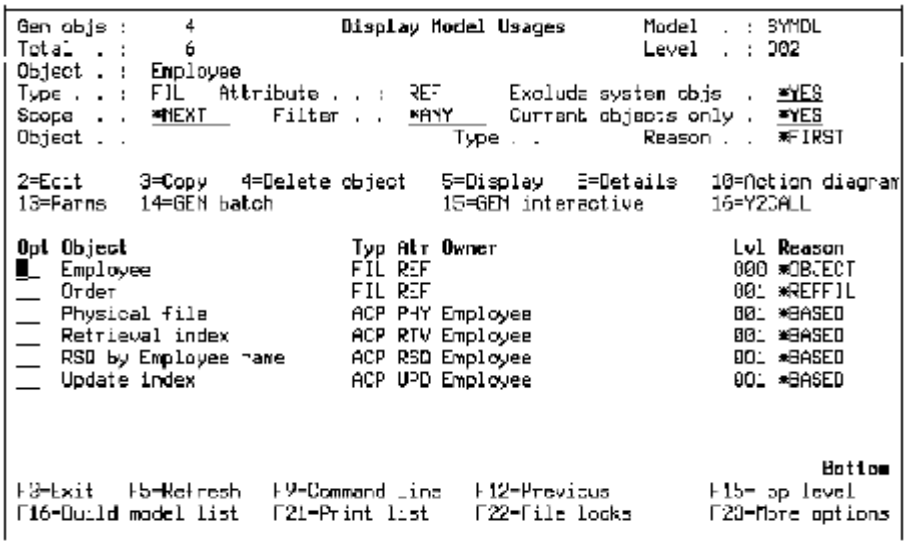
Note the values displayed in the Lvl and Reason columns for each object and how they relate to the diagram that follows. Lvl 000 indicates the object whose usages are shown. This object is used by the Lvl 001 objects, which in turn are used by the Lvl 002 objects. The Lvl 000 object is included so you can edit the originating object as well as the using objects.

3. When a model object is used by many other model objects, it is not always easy to determine the usage structure when Scope is set to *NOMAX, which displays all levels of usages. Instead, you can set Scope to *NEXT to step through the usage expansion one level and one model object at a time.

   Press F15 to return to the Level 001 panel. Change the Scope option to *NEXT and press Enter. Next, enter 91 for Employee and press Enter.

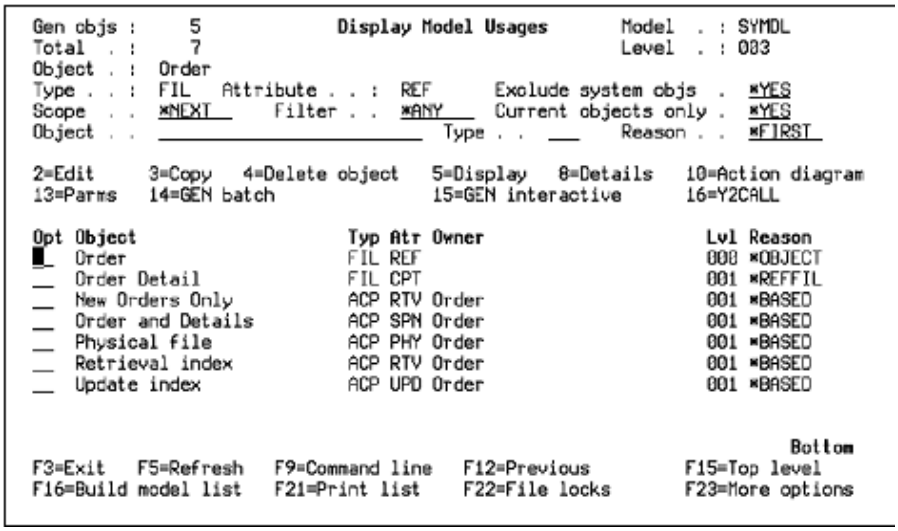
   The following panel displays.

   Note: From here on the process shown in this example is the same for both methods of working interactively with usages; namely, whether you pressed F20 or typed selection option 91 against a model object.

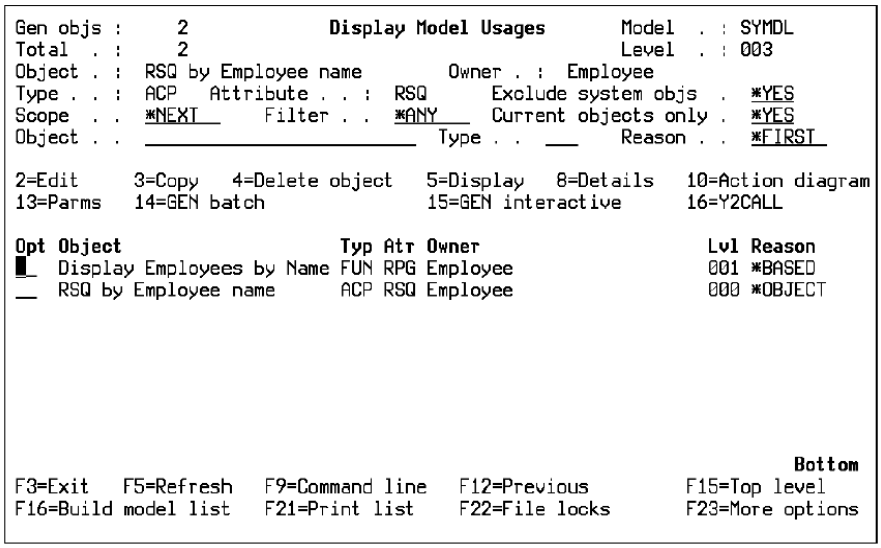


This panel shows only the Lvl 001 model objects that use the Employee file.

4. Enter 91 for the Order file to expand usages to the next level for just that model object. The following panel displays:



5. To expand usages for 'RSQ by Employee name' access path instead, press F12 to return to panel Level 002 and enter 91 against that object. The following panel displays, indicating that the 'RSQ by Employee name' access path is used only by the 'Display Employees by Name' function.



Working with References Interactively

You can work with model object references interactively from the Edit Model Object List panel in the following ways:

- Type 81 against a model object to expand its references. The Display Model References panel displays references for the selected model object. The Scope field is set to *NEXT.

  Note: This is the recommended method to work with references interactively.

- Press F22 to prompt the Display Model References (YDSPMDLREF) command for the named model object list you are editing. In this case references are not expanded for the model list entries before the Display Model References panel displays. Instead, the list is redisplayed with all model objects updated based on information in the All Objects list. The Scope field is set to *NOMAX.

For more information on the panel displayed as a result of using F22, see the Working with Usages Interactively section in this chapter.

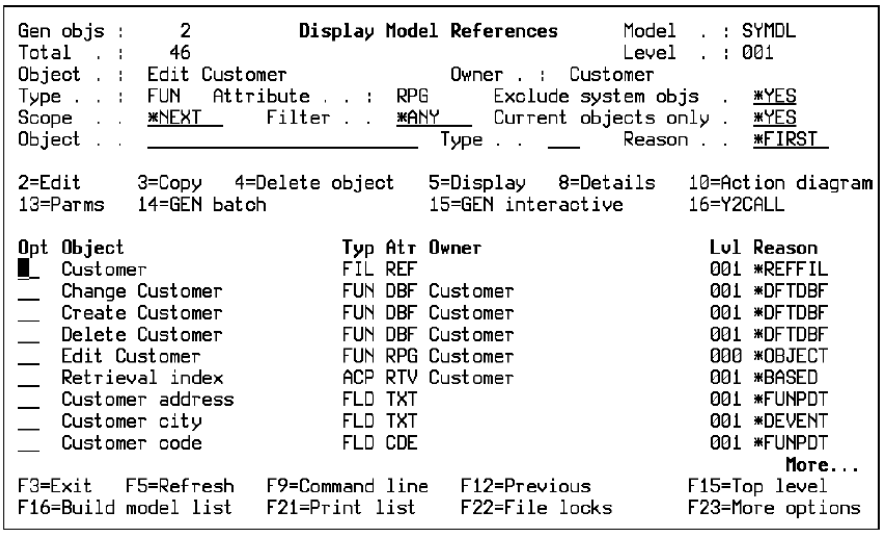
Example

You can use the Display Model References panel to solve problems in program applications. Suppose you only know the implementation name of the program in which an error occurred.

Use the Edit Model Object List panel over the All Objects list (*ALLOBJ). Press F7 to display the positioning windows. Press F11 until the Position by Implementation name window displays:

```
                          Edit Model Object List
                    .....................................................
  Model   :          Position by Implementation name          :
  List .  :                                                   :
          : Implementation name  .  UUAJEFR__                 :
Type opti :                                                   :
1=Select  :                                                   :
5=Display :                                                   :
11=Add to :  F11=Alternate view   F12=Cancel                  :
          :                                                   :
Opt     0 :.....................................................:
__      Edit Customer              FUN   RPG  Customer
__      Edit Employee              FUN   RPG  Employee
__      Employee                   FIL   REF
__      Existing                   CND   VAL  Customer status
__      Former                     CND   VAL  Customer status
__      New                        CND   VAL  Customer status
__      Order                      FIL   REF
                                                          More...
Parameters or command
===> ____________________________________________________________
F3=Exit   F4=Prompt   F5=Refresh    F6=Build          F7=Position to
F8=Reverse retrieve   F9=Retrieve   F23=More options   F24=More keys
```

Enter the implementation name to position the All Objects list (*ALLOBJ) to the function in which the error occurred. Enter selection option 81 for the function to display references for that function. The following panel displays:



1. On the Display Model References panel:

   a. Type *EXTFUN for the Scope option to limit the expansion to include only referenced functions down to and including the first external function call.

   b. Type *ACTION for the Reason option to include only action diagram references.

   c. Press Enter.

   The expanded references are all the functions that comprise the program in which the error occurred. This is a useful starting place for the developers whose task it is to fix the problem.

2. Press F16 to build a model object list containing a list of the functions displayed on your screen. Assign a name to the list that will be meaningful to the development staff.

3. To produce a printed copy of the list use one of the following methods:

   - Press F21 to produce a report that indents the reference levels in order to highlight the relationships among the functions more clearly.
   - Press F9 for a command line and enter the following command to produce a keyed report.

YDOCMDLLST MDLLST(list-name)

The development staff can use either the online model object list or the printed copy as an aid to solving the problem.

Accessing Model Object Cross Reference Facilities

Following is a list of ways you can access model object usages and model object references facilities:

- From the Edit Model Object List panel or the Edit Model List for Copy (YEDTCPYLST) panel, use the following options to display usages or references for any model object displayed:
  - 81=References by Object
  - 82=References by Owner
  - 91=Usages by Object
  - 92=Usages by Owner
- Use the following function keys from the Edit Model List (YEDTMDLLST) panel to display usages or references for a named model object list.
  - F20=Usages
  - F22=References

  You will be prompted to enter the type of output you want: display, print, or to another model object list.
- Use the Display Model Object Usages (YDSPMDLUSG) or the Display Model Object References (YDSPMDLREF) command to display, print, or expand usages or references for a named model object list. You can use these commands interactively or in batch. If using the command interactively, you can then use options 81, 82, 91, and 92.
- Use option U on various panels; such as, Edit File Details and Edit Function Details to display usages for a model object. This displays the Display Model Usages panel from which you can use options 81, 82, 91, and 92.

Working with Model Object Cross References in Batch

Following are examples of using the Display Model Usages (YDSPMDLUSG) and the Display Model References (YDSPMDLREF) commands in batch:

- To add all usages for all objects in model object list WRKLST (in library MYMDL) to model object list OUTLST, including usages up to and including the first external function, use:

```
YDSPMDLUSG MDDLST(MYMDL/WRKLST) +
  SCOPE(*EXTFUN) OUTPUT(*MDLLST) +
  OUTMDLLST(OUTLST) OUTLSTUPD(*ADD)
```

- To print a list of access paths and external functions that are referenced by model objects existing on model object list MYLIST, use:

```
YDSPMDLREF MDLLST(*MDLLIB/MYLIST)+
  OUTPUT(*PRINT) FILTER(*GENOBJ)
```

Simulating Changes to Model Objects

Simulating a change to a model object lets you see how a proposed change impacts other objects in the model before you actually make the change. Simulation identifies which other model objects need to be edited or generated as a result of the proposed change and is an important tool for planning model changes.

When you change a model object, the only objects that can be affected by the change are those that *use* the changed object. As a result, a major part of simulating a change consists of expanding usages for the object to be changed.

Note: The process described here is the same as that used during component change processing when you actually change a model object. However, instead of just displaying the results of the change, component change processing updates the All Objects list for the model objects affected by the change to indicate the additional processing needed.

For more information on component change processing, see the Component Change Processing section in this chapter.

To simulate a change to a model object, follow these steps.

1. Determine whether the proposed change is a private or public change to the model object.

   a. A private change implies that access paths and external functions using the changed object be regenerated in order to implement the change in the application system objects.

   b. A public change implies that first level objects using the changed object may need to be edited and that access paths and external functions using the changed object need to be edited and/or regenerated.

   The type of change depends on which attributes of a model object are changed and is derived internally by CA 2E.

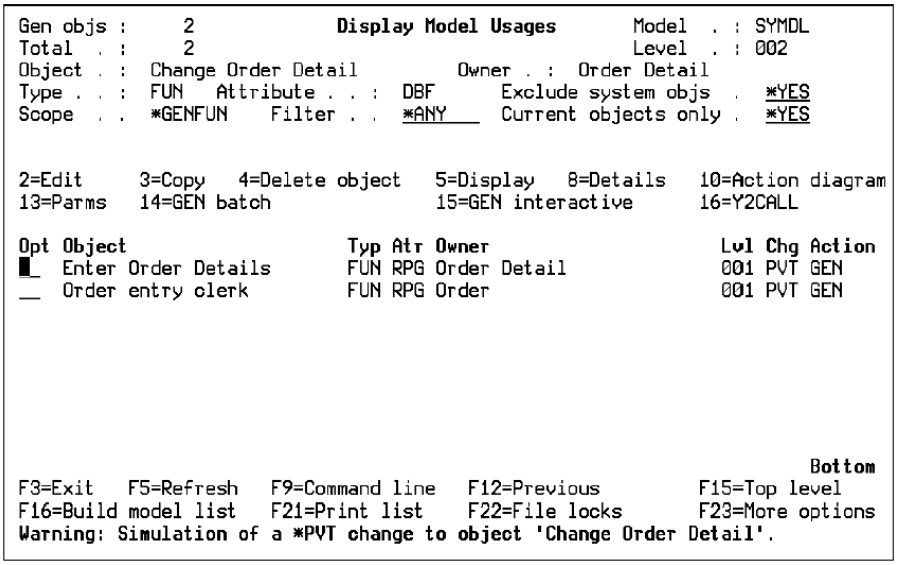
   For more information:

   - On all possible changes and the associated change type for each, see the appendix titled "Change Control Facilities Reference Tables" in this guide.
   - On change type, see the Model Object Audit Information section in this chapter.

2. To simulate a private change, enter option 94 for the object you want to change on either the Display Model Usages panel or the Display Model References panel.

   In this case,CA 2E expands usages for the object to be changed up to the first external function in each sequence of usages. Suppose the object to be changed is the Change Order Detail function.

   The following panel displays:

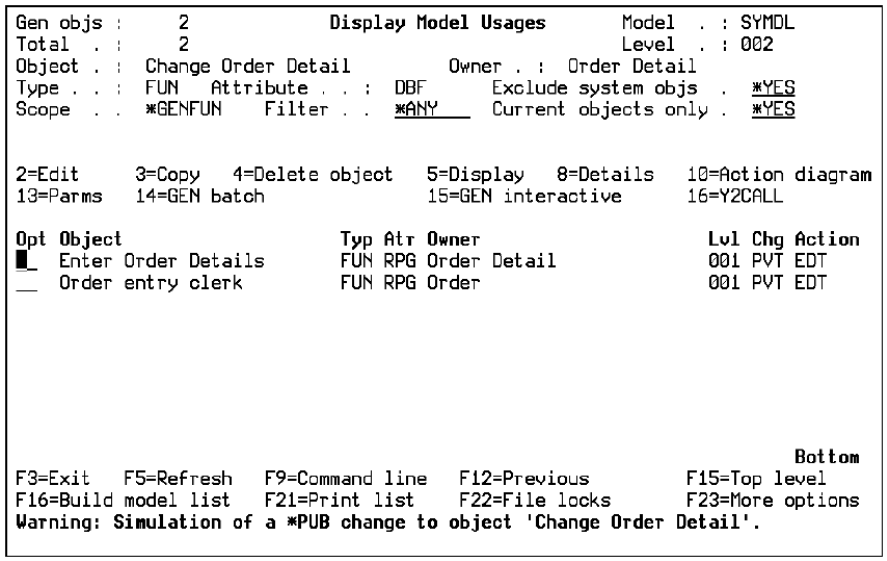


Note that only those objects that need to be generated to implement the proposed change are displayed, *not* all usages. GEN in the Action column indicates that the model object will need to be regenerated when you make the proposed change.

To simulate a public change, enter option 95 for the object you want to change on either the Display Model Usages panel or the Display Model References panel.

As for a private change,CA 2E expands usages for the object to be changed up to the first external function in each sequence of usages. In addition, it identifies objects that need to be edited as a result of the proposed change. Suppose the object to be changed is the Change Order Detail function.

The following panel displays:



Note that only those objects that need to be edited or generated to implement the proposed change are displayed, *not* all usages. EDT and GEN in the Action column indicate that the corresponding model object needs to be either edited or regenerated when you make the proposed change.

3. Press F16 to build a model object list containing a list of the model objects impacted by your proposed change. A window displays where you specify a model object list name and whether to replace or add entries if the model list you specify already exists. If the model list does not exist,CA 2E automatically creates it.

Note: The entries of a model object list always reflect the current state of model objects at the time the list is created unless you refresh them. As a result, the Action required field for entries of the new list will reflect the results of a previous run of component change processing and *not* the results of the simulation. To get a permanent record of the simulation, press F21 to print the results.

Component Change Processing

Component change processing is an automated impact analysis tool that determines how a change to a model object affects other objects in the model and records whether the affected objects need to be edited or regenerated. Objects that use the changed object are referred to as using objects; the changed object is considered to be a component of its using objects.

Component change processing is optional and can be run interactively, by request, or in batch. You control it using the model profile and the Component Change Processing (YCMPCHG) model value.

The simulation options available on the Display Model Usages and Display Model References panels simulate the actions of component change processing. You can experiment with these options to become familiar with the process.

Understanding Component Change Processing

When you change a model object, CA 2E records the type of the change in the All Objects list for the model object. Since the only model objects that can be affected by the change are those that use the changed object, a major part of component change processing consists of expanding usages for the changed object.

If the change was a public or private change, CA 2E expands usages for the changed object, identifies the using objects affected by the change, and updates the detail for the affected using objects in the All Objects list to indicate whether they need to be edited or regenerated.

Specifically, CA 2E determines the impact of a change on other model objects according to the following criteria:

- Object type of the changed object
- Change type of the changed object
- Object type of the using object
- Level of usage at which the using object was encountered.

For more information on change type, see the Model Object Audit Information section in this chapter.

Impact on the All Objects List

Following is the information CA 2E sets and maintains in the All Objects list as part of component change processing:

- **Impact processed indicator**—Set for the changed object and indicates whether component change processing has been run for the last change to the object. In other words, it indicates whether the change is reflected on using objects.
- **Component change processed date and time** —This has two possible meanings:
  - – For a changed object, this is the same as its change date and time.
  - – For an object that uses a changed object, this is the date and time component change processing was run for the change.
- **Action required indicator**—This indicates whether the using object needs to be edited (EDT) or regenerated (GEN). Note that:
  - - For GEN, if the using object is an access path or external function, CA 2E automatically resets this indicator to blank when the object is successfully regenerated.
  - – For EDT, if the object is an access path or external function, CA 2E automatically resets this indicator to GEN when you edit it. For all other model objects, CA 2E resets this indicator to blank when you edit the object.

After you perform the necessary editing, CA 2E performs component change processing again to ensure that the new change is reflected in regenerated and any newly created using objects. As a result, it is recommended that you edit the model object at level 001 first and then proceed upward to edit model objects at continually higher usage levels. This prevents a model object from being flagged EDT more than once. At the end of the process, all objects that require regeneration will be flagged GEN.

- **Change type**—CA 2E does not reset a model object's Change type as a result of running component change processing for the change. It is not reset until you change the model object again. As a result, the Change type always reflects the most recent change to the model object.

You can use the information recorded by component change processing to build model object lists and to create utilities to automate the required additional processing and to help you administer your model.

Examples

Since component change processing can run interactively as you make changes or can be postponed, you can check the Impact processed indicator to determine whether you need to run the Apply Component Changes (YAPYCMPCHG) command.

You can use the Filter Model Object List (YFLTMDLLST) command over the All Objects list (*ALLOBJ), specifying an output list, to build a model object list of just those objects that require editing. You can then give the list to the programming staff to make the required changes.

Similarly, you can build a list of the objects that need to be regenerated, convert the list to a job list using the Convert Model List to Job List (YCVTMDLLST) command, and use the result as input to the Submit Model Create Requests (YSBMMDLCRT) command.

For more information and an example of a Command Language program that uses component change processing, see the Running Component Change Processing in Batch section in this chapter.

Viewing the Results

You can view the results of component change processing or check what additional processing is required using the Edit Model Object List panel over the All Objects list (*ALLOBJ).

- Press F11 to display alternate views. For each model object you can view the Component change date and time, the Required action indicator (GEN or EDT), and the Impact processed indicator.
- Enter option 8 for any model object to view its current detail on the All Objects list.

Simulating a Change

You can also use the simulation options (94 and 95) on the Display Model References and Display Model Usages panels to view the results of component change processing. These options let you see the impact of a proposed private or public change on other objects in the model before you actually make the change. The resulting display identifies which other model objects need to be edited or generated as a result of the proposed change.

For more information on simulating component change processing, see the Impact Analysis section and the Simulating Changes to Model Objects section, both in this chapter.

Setting the YCMPCHG Model Value and the Model Profile

Component change processing is controlled by a combination of details stored on the model profile for the individual developer and the Component Change Processing (YCMPCHG) model value.

Component Change Processing Model Value

The processing of component changes is primarily controlled by the Component Change Processing (YCMPCHG) model value. It can have one of the following values:

- *UNLIMITED —This value lets all developers control the setting of the Component change processing option in their model profile.
- *LIMITED—This value indicates that only developers with *DSNR authority can change the Component change processing option in their model profile.
- *GEN—This value causes component change processing to be invoked automatically during each YGENSRC job submitted for a given model. It ensures that changes are kept up-to-date without incurring the overhead associated with running component change processing interactively. This value also gives *LIMITED capability to change the model profile option.
- *NONE—This value turns off automatic component change processing, both interactive and as a result of YGENSRC jobs. This value also gives *LIMITED capability to change the model profile option.
- Note: To run component change processing in this case you need to use the Apply Component Changes (YAPYCMPCHG) command.

Model Profile Settings

The Component change processing option on the model profile controls whether component change processing runs interactively and automatically as you make changes. Its value has no effect if the YCMPCHG model value is set to *NONE. The possible values are:

- **Y**—This value turns interactive component change processing on. Using objects are flagged when a model object is changed.
- **N**—This value turns interactive component change processing off. Using objects are not flagged when a model object is changed. You can use the YAPYCMPCHG command later to apply component changes throughout the model.

For more information on the model profile in general, see the Model Profile section in this chapter.

Using These Settings to Administer Your Model

Because component change processing requires expansion of usages, it can be costly in processing time. This is especially true when designers (*DSNR) use it interactively. An administrator can use the YCMPCHG model value and the model profile as follows to prevent certain developers from running component change processing interactively, and as a result, minimize interactive overhead:

- Set the YCMPCHG model value to *LIMITED.
- For *DSNRs, set the model profile option to N; for *PGMRs, set it to Y.

Note: If you do not run component change processing interactively, be sure to run the Apply Component Changes (YAPYCMPCHG) command regularly; for example, overnight in batch. This keeps the model changes to be processed down to a manageable level.

When a model object is changed and component change processing is not performed immediately, CA 2E sets the Impact processed indicator to N to indicate this. You can check this indicator to decide when to run the YAPYCMPCHG command by filtering the All Objects list (*ALLOBJ) to determine which and how many objects require processing.

Performance Considerations

The amount of overhead incurred by running component change processing interactively generally depends on:

- The size and complexity of the model.
- The type of editing to be done.

For example, a *DSNR working in a large model negatively affects the system when running component change processing interactively. On the other hand, a *PGMR making changes in a relatively isolated area of the model could effectively use interactive component change processing to quickly identify model objects affected by his changes. Both types of users can use the simulation options to gauge the effect of changes.

Experience will show the impact of running component change processing interactively; however, following are a few suggestions for estimating the likely effect. The amount of interactive overhead depends on:

- The type of the model object you are editing. Changes to low level model objects such as, conditions, fields, and files increase interactive overhead because the expansion process includes more objects.
- The significance of the object in the model. For example, an important internal function that is used by many other internal functions may cause a large number of objects to be expanded and as a result increase interactive overhead.
- The frequency with which you apply component changes. Running component change processing frequently minimizes the amount of work required each time it is run.

Methods of Running Component Change Processing

Following is a list of the ways you can run component change processing.

- Interactively and automatically as you make changes. This occurs when the YCMPCHG model value is set to a value other than *NONE, and the Component change processing option on the model profile is set to Y.
- Using the Apply Component Changes (YAPYCMPCHG) command. If you do not run component change processing interactively, it is useful to run this command in batch to ensure that the effects of changes have been distributed throughout the model.

  The YAPYCMPCHG command requires *PGMR capability. By default you can run this command only within an interactive or batch program to prevent excessive use of interactive resources. You can change this default using the Change Command (CHGCMD) command.
- By request, using the simulation options on either of the following commands:
  - Display Model Usages (YDSPMDLUSG)
  - Display Model References (YDSPMDLREF)

  This is useful for seeing how a proposed change will affect other model objects. Results are only displayed; the All Objects list for the model objects affected by the change is not updated.
- Using the Submit Model Creates (YSBMMDLCRT) command or interactive generation. This occurs automatically if the YCMPCHG model value is set to *GEN.

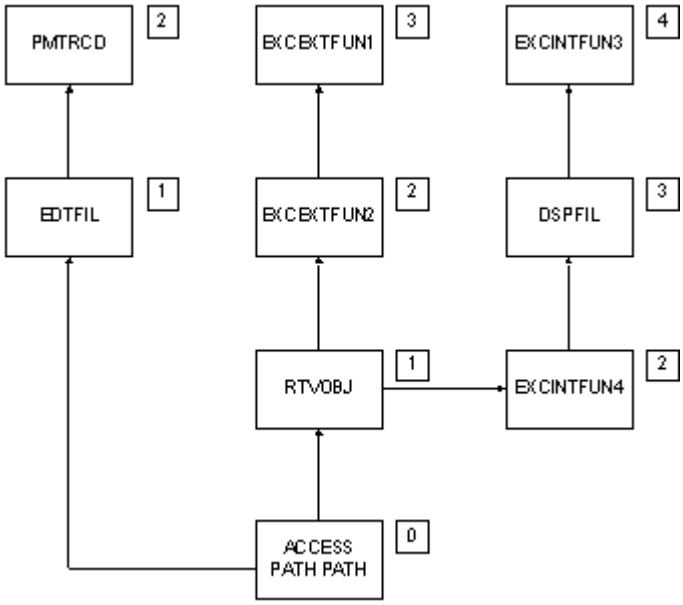
Running Component Change Processing in Batch

You can use the following series of commands to form part of a nightly process to prepare a model for the following day. You can also use these commands interactively to generate lists and process the required changes.

1. YSNCMDL—Optionally included to ensure that the model is synchronized.
2. YAPYCMPCHG—Ensures all changes to model objects are reflected on the objects that use the changed objects.
3. YFLTMDLLST—Run this over the All Objects list (*ALLOBJ), select model objects having the Required action indicator set to *EDT, and output to the EDTLST model list for programmers to edit the following day using YEDTMDLLST.
4. YFLTMDLLST—Run this over the All Objects list (*ALLOBJ), select model objects having the Required action indicator set to *GEN, and output to the GENLST model list.
5. YCVTMDLLST—Run this over the GENLST model object list to prepare a job list to generate objects that require generation as a result of a change to a component object.
6. YSBMMDLCRT—Specify the job list created in the previous step to generate and compile changed objects.
7. YDOCMDLLST—Use to document the EDTLST and GENLST model object lists for administrative purposes.

For more information on model list commands, see the Commands to Manipulate Model Object Lists section in this chapter, and the *Command Reference Guide.*

Component Change Processing Scenario

1. Suppose we have the scenario shown in the following diagram. This example shows the details of component change processing that take place as a result of both a private and a public change to an access path.
2. 

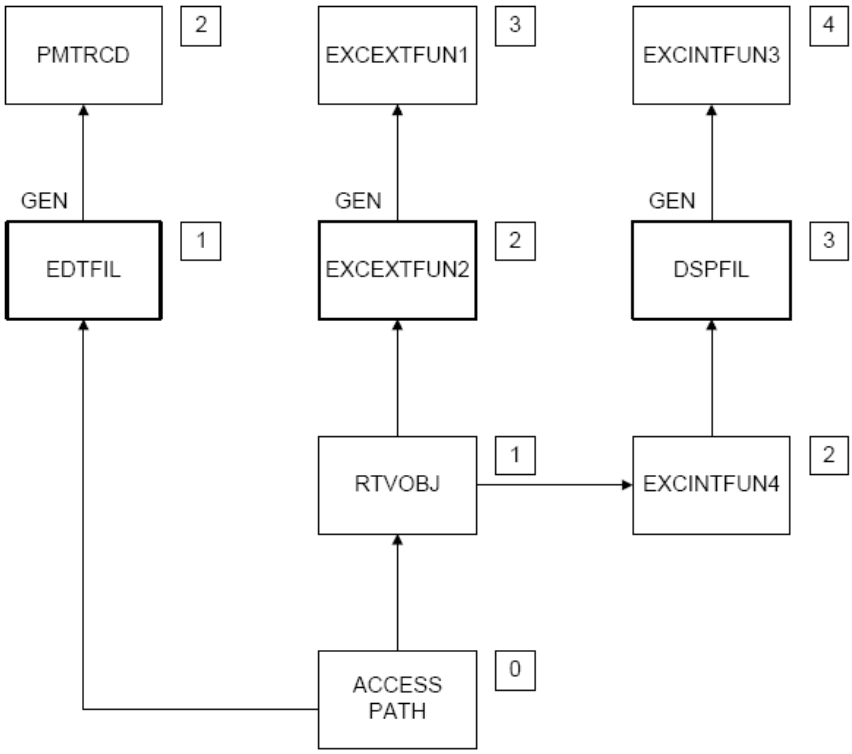
The arrows indicate the usage sequence for the functions that use the access path. The numbers in the small boxes indicate the usage level of each object relative to the access path at level 0.

The following section shows the difference between a private and public change to Access Path.

Private Change to Access Path

Suppose you intend to change the source member name for Access Path using the Edit Access Path Details panel. CA 2E considers this to be a private change to Access Path. As a result, CA 2E:

1. Expands all usages of Access Path until the first external function on any given sequence of usages is encountered. These are shown with darker borders and the word GEN.



Note: That all usage levels beyond the first external function are ignored; in this case, PMTRCD, EXCEXTFUN 1, and EXCINTFUN 3 are not included in the expansion.

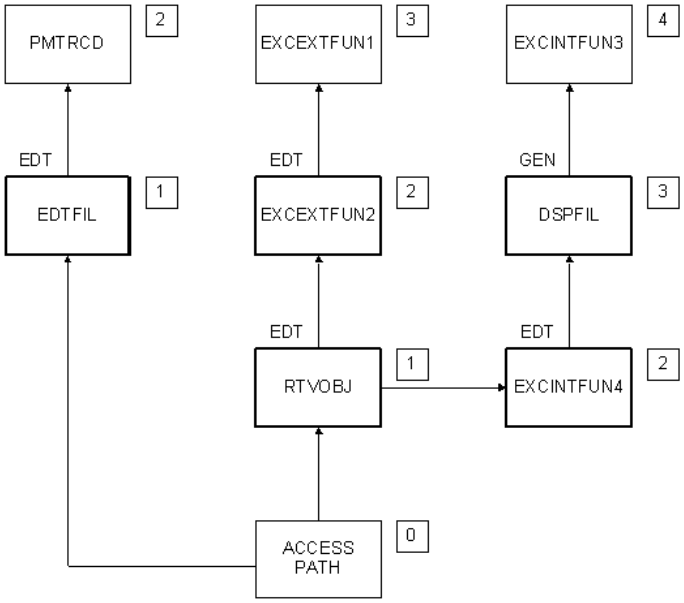
2. Sets the Required Action Indicator for the affected external functions to GEN in the All Objects lists.

Public Change to Access Path

Suppose you intend to add or remove a relation for Access Path using the Edit Access Path Relations panel.CA 2E considers this to be a public change. As a result CA 2E :

1. Expands usages of Access Path until the first external function on any given sequence of usages is encountered as it did for the private change. However, for a public change the level at which each using object occurs is significant.
2. Sets the Required action indicator to EDT in the All Objects list for all using objects that occur at level 1 to indicate that they require editing as a result of the change. Specifically:
   - Public change to Access Path is a private change to EDTFIL and it needs to be edited (Required Action=EDT).
   - Public change to Access Path is a public change to RTVOBJ, because it is used for a parameter definition, and it needs to be edited (Required Action=EDT).

   These are shown in the diagram by the letters EDT and the small box containing a 1.



3. Checks the model objects at level 2. The result depends on the way in which the level 1 object uses Access Path. In this example the RTVOBJ function uses Access Path as the based on access path and for parameter definition. Specifically:

   - Public change to RTVOBJ is a private change to EXCEXTFUN 2; it needs to be edited to accommodate the change to RTVOBJ's parameters (Required Action = EDT).
   - Public change to RTVOBJ is a private change to EXCINTFUN 4 and it needs to be edited (Required Action = EDT).

   These are shown in the diagram by the letters EDT and the small box containing a 2.

4. Repeats the process for the model objects at level 3. The private change to EXCINTFUN 4 is a private change to the external DSPFIL function and it needs to be generated (Required Action = GEN).
5. Repeats this process for each usage level until an external function is encountered.

When you perform the required actions (EDT or GEN), CA 2E automatically resets the Required action indicator as follows:

- For GEN, when an access path or external function is successfully regenerated,CA 2E automatically resets it to blank.
- For EDT, if the object is an access path or external function, after editing it is reset to GEN. For all other model objects, it is set to blank.

After you perform the necessary editing, component change processing is invoked again to ensure that the change is reflected in regenerated and newly created using objects. As a result, it is recommended that you edit the model object at level 1 first and then proceed upward to edit model objects at continually higher usage levels. This prevents a model object from being flagged EDT more than once. At the end of the process, all objects that require regeneration will be flagged GEN.

For more information on how CA 2E distributes the impact of a change throughout the model, see the appendix titled "Change Control Facilities Reference Tables" in this guide.

Model Security

CA 2E lets you access a model as one of three user types: designer, programmer, or user. Your user type determines the limitations placed on you when you access and edit the model.

- A designer (*DSNR) can change any aspect of the model, both data relationships and functional specifications. If the Open Access (YOPNACC) model value is set to *YES, multiple designers and programmers can use the model at the same time; otherwise, only one designer can use the model.
- A programmer (*PGMR) can change functional specifications, but cannot change database files or fields. Multiple programmers can use the model at the same time. However, a programmer cannot use the model at the same time as a designer if the Open Access (YOPNACC) model value is set to *NO.
- A user (*USER) can only view the model and cannot change it. However, a *USER can edit model object lists.

In addition, certain designers have additional locking authority (*DSLK). This authority lets the designer change the Open Access (OPNACC) model value and place permanent, exclusive locks on functions and access paths that can only be removed by a designer.

You can view the authority you have to a model by accessing the CA 2E product menu and specifying the model; namely:

YSTRY2 model-name

A message at the bottom of the screen shows your authority level.

Note: A designer or programmer can access the model in view only mode by setting the View only option to Y in the model profile.

For more information on user authority and the YOPNACC model value, see the Administrator Guide.

Model Profile

The model profile lets you define defaults for various processes and file specifications for an interactive session. For example, the model profile contains defaults for the following:

- Log changes mode
- View-only mode
- Full screen mode for the Edit Model Object List panel
- Component change processing
- Name of session list
- Name of user option library and file
- Default model object list name for commands
- Generation and creation library

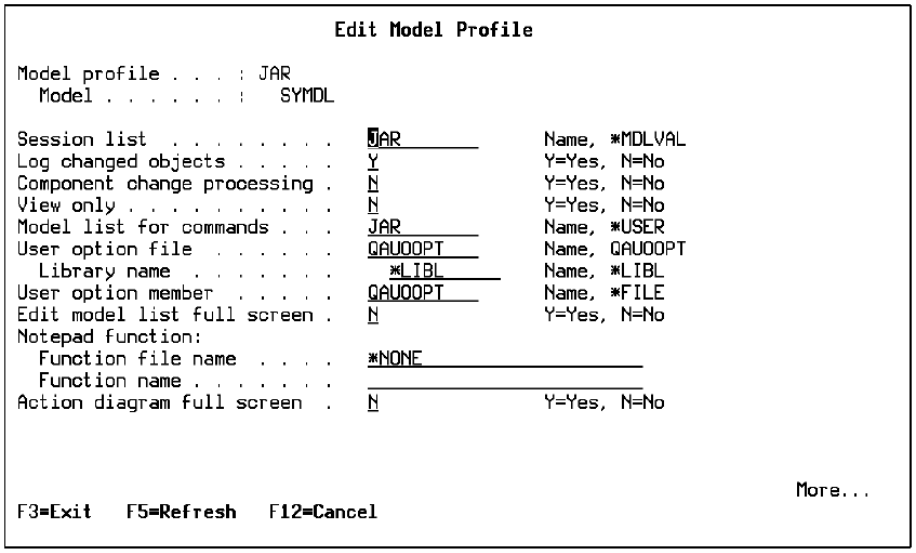
There is a model profile for each user for each model. When a new user is granted access to a model, CA 2E automatically creates a model profile for the user. The defaults for a session are taken from the model profile.

Changing a Model Profile

You can change the model profile in the following ways if you are authorized to do so:

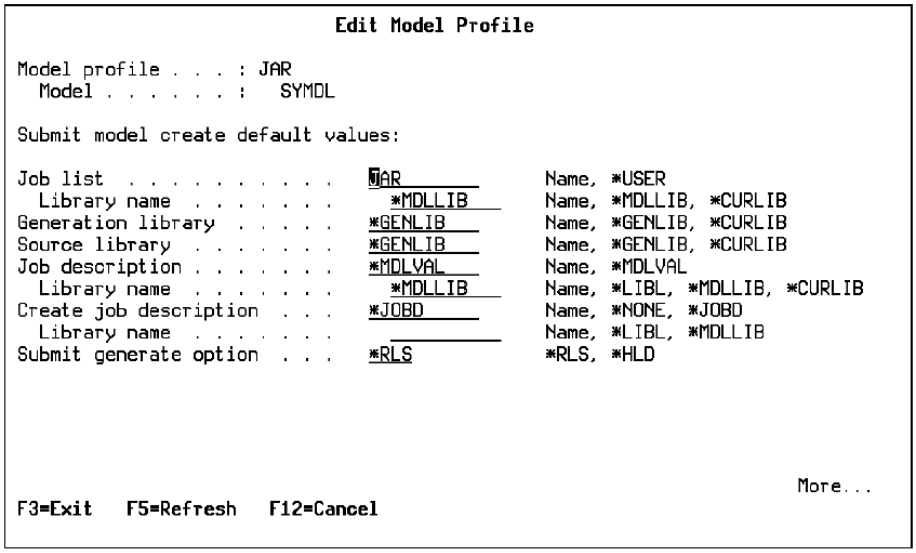
- Press F18 from the Edit Model Object list panel.
- Use the Edit Model Profile (YEDTMDLPRF) command.
- Use the Change Model Profile (YCHGMDLPRF) command.

Suppose you pressed F18 or used the YEDTMDLPRF command. The following panel displays:



These default settings establish the basic working environment for your interactive session. You can reset these to modify your environment for the current session; for example, to use full screen mode on the Edit Model Object List panel or to access the model in view only mode.

Scroll down to view the second screen of the Edit Model Profile panel containing the Submit Model Create Default Values.



These values establish the default generation and creation environment for the current session and are used to preload job list commands; for example, the Submit Model Create (YSBMMDLCRT) command. In addition to changing values on this screen, you can modify the default values when you invoke the command by specifying other parameter values on the interactive command prompt or on the batch command.

The third screen of the Edit Model Profile panel contains the Submit compilation option and the name of the GUI folder for CA 2E Thin Client.

How Model Profiles are Stored

A default model profile is shipped with the null model, Y2SYMDL, in the YSYS record of file YMDLPRFRFP. When you create a new model, this default model profile is automatically copied to the YSYS record of file YMDLPRFRFP in the new model library. You can edit either of these model profiles to create a default model profile tailored to your environment.

Managing Model Profiles

Managing Model Profiles

Following are suggestions for ways you can use model profiles to manage your model.

- As already mentioned, you can create a default model profile suitable for your environment by editing the model profile in the null model or by editing the model profile associated with a particular model.
  - To change the model profile defaults for new models, edit the default model profile in the null model. Note that this will be overwritten when you receive a new version of CA 2E.
  - To change the model profile defaults for new users of a model, edit the default model profile in the model library.
- You can tailor model profile settings for particular developers or groups of developers using the Edit Model Profile (YEDTMDLPRF) command. Some examples are:
  - Assign developers to different session lists based on the project to which they are assigned.
  - Restrict certain developers to view only mode.
  - Turn component change processing off for all designers (*DSNR) to avoid interactive overhead.
  - Set a different generation and creation environment for each ongoing project.
- You can use the Retrieve Model Profile Details (YRTVMDLPRF) command to retrieve model profile settings for a specified user profile. As with other change control facilities commands, you can use this within control language programs to build utilities and model object lists to help manage your model.

Working with Versions of Functions and Messages

A version is a model object that is a copy of either a function (FUN) or a message (MSG) in the same model. A function or message can have an unlimited number of versions. Three benefits of using versions are:

- You can test changes on a version of a function or message without interfering with the functionality of the existing model object.
- When you finish testing a new version of a function or message and make it active in the model, the original model object remains unchanged and can easily be made active again if needed.
- Only the currently active version of a function or message is displayed on CA 2E editing panels. As a result, the panels are not cluttered with inactive versions.

Understanding Versions

Understanding Versions

Versions of the same model object form a version group. Each version in a version group is a unique model object; the term version is used to identify them as being related.

In any group of versions, one and only one of the model objects in the group may be current. The current version is the version that is active in the model; it is the model object that is used by other objects in the model and is the model object that appears on CA 2E editing panels.

If you use CM, each version also has one of the following version types. If you do not use CM, all versions have version type DEV.

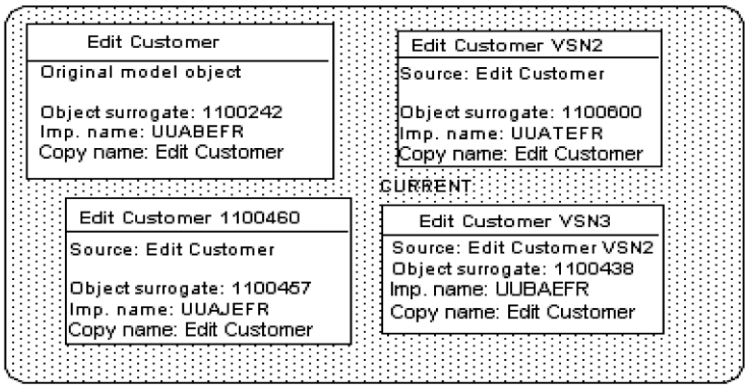
- **DEV**—Development
- **PRD**—Production
- **ARC**—Archive

All versions in a group have the same Copy name. This value is used to initialize the Copy name stored on the model list that the Copy Model Objects (YCPYMDLOBJ) command uses to copy objects between models. Initially the Copy name for a version group is the same as the name of the original model object. You can change the Copy name for all versions in a group using the Change Model Object Description (YCHGMDLOD) command on any one of the versions. New versions will automatically be given the same Copy name as the other members of the group.

For more information:

- On version types, see the CA 2E *Change Management User Guide*.
- On the YCPYMDLOBJ and YCHGMDLOD commands, see the *Command Reference Guide.*

The following diagram illustrates basic concepts relating to versions. The original object in this diagram is the Edit Customer function; it has three versions. These four functions comprise the version group for Edit Customer.



Note the following concepts shown by this diagram:

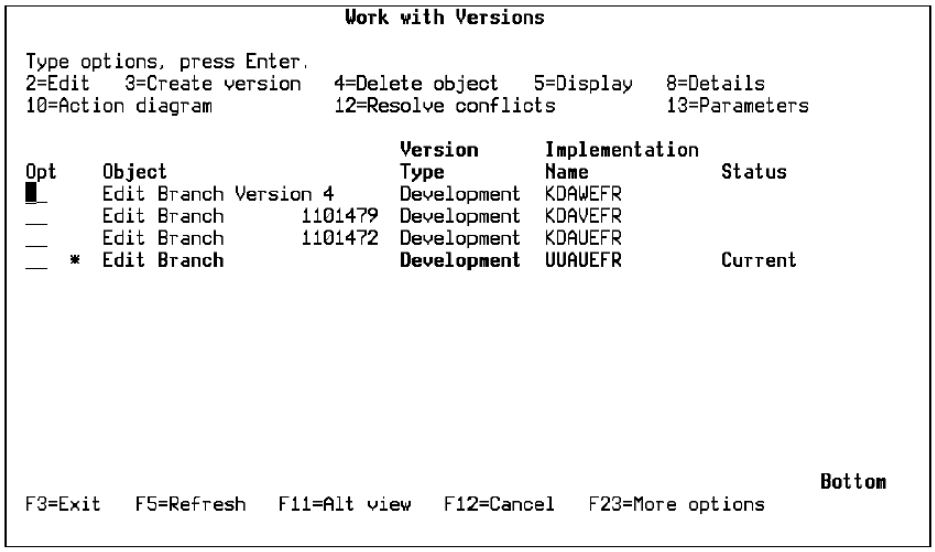
- When you create a version you can assign a name to the version or you can let CA 2E automatically generate it.
  - Edit Customer 1100460 is an example of a version name that was automatically generated by CA 2E.
  - Edit Customer VSN3 is an example of a version name that was assigned by the developer.
- The two versions, Edit Customer 1100460 and Edit Customer VSN2, were created directly from the Edit Customer function; Edit Customer VSN3 was created as a version of Edit Customer VSN2.
- All versions in the group have the same Copy name, in this case Edit Customer. If you change the Copy name for any version in a group using the YCHGMDLOD command, the Copy name is automatically updated for all versions in the group.
- Each version, including the original function, is a distinct model object and therefore has a unique:
  - Object surrogate number.
  - Object name; e.g., Edit Customer, Edit Customer VSN2, Edit Customer 1100460.
  - Implementation name.
- Edit Customer VSN3 is labeled as CURRENT and is the version displayed on CA 2E editing panels.

A Reason Not to Use Versions

When you make a version of a function current in the model,CA 2E globally changes all the model objects that referenced the original function to reference the version instead. If not all of the referencing model objects need the changed functionality, you should create a new function rather than a version. After updating and testing the new function, you would then need to update references to the new function manually.

Working with Versions

From the Edit Model Object List panel, enter 19 in the subfile selector to work with versions for the selected function or message. The following panel displays:



All versions in the version group are displayed in reverse chronological order; the most recently created version appears at the top of the list of entries. The current version is highlighted and has an asterisk to the right of the Subfile selector and a Status of Current.

This panel supports a wide range of subfile selection options and provides alternate views for each version. The options with similar selection values perform the same function as those on the Edit Model Object List panel. For example, from this panel you can:

- Create a version
- Edit a version
- Make a version current
- View detail for a selected version
- Perform impact analysis
- Generate a version
- Delete a version

For more information on the subfile select options, see the Editing Model Object Lists section in this chapter.

Viewing a Version Group

The following views are available from the Work with Versions panel. Press F11 to display the views sequentially.

- Object Data—This view shows various object information including, model object name, implementation name, version type, and status for each version. The current version is highlighted and has an asterisk to the right of the Subfile selector and a Status of Current.
- Creation Data—This view shows the date and time each version was created.
- Change Data—This view shows the date, time, and user name for the last change to each version.
- Check Out Data—This view shows the check out date, list, user, and status for each version. This view applies only if you use CM.

Creating a Version

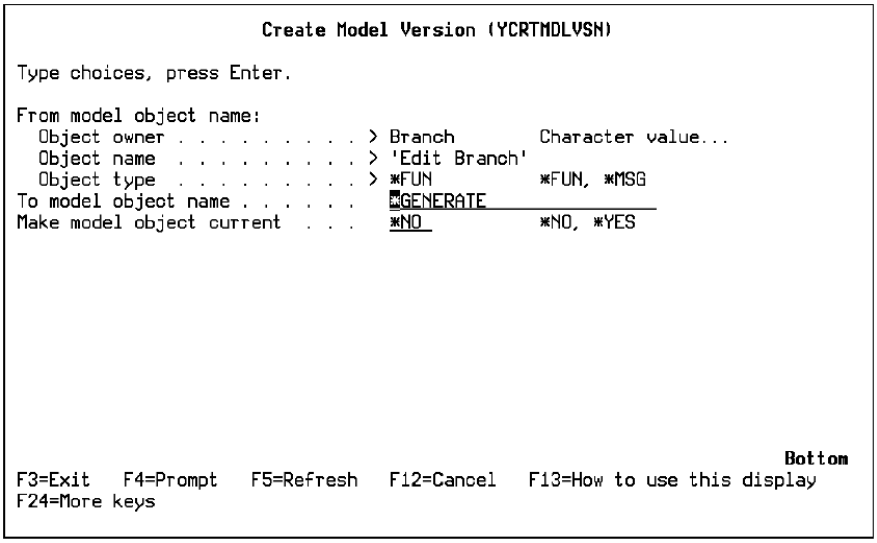
You can create a new version of a function or message by using the:

- Work with Version panel
- Create Model Version (YCRTMDLVSN) command from within a control language program
- Create Object Version (YCRTOBJVSN) command outside control language programs

The new version will be a copy of the original function or message, but will have a different object name, object surrogate number, and implementation name. The object name for the version must be unique within the owning file; the implementation name must be unique within 3GL object type in the model. The new version is given the Copy name used by the version group to which it belongs.

To create a version from the Work with Versions panel:

Enter selection option 3 for the version you want to use as the source for the new version. You can select any version listed; it does not need to be the current version. The following panel displays:



1. Determine the To model object name. You can either let CA 2E generate a new name for the new version, or you can override this default.

   The name CA 2E generates is the original name suffixed by a 7-digit number; the original name is truncated if the new name is longer than 25-characters. For example,

   - Edit Branch 1459353
   - Edit Customer Addr1541577

   Note: You can define your own naming convention for automatic name generation using the exit program YOBJNAMR1C.

2. Determine currency. You can make the new version current by specifying *YES for the Make model object current option. The default is not to make the new version current so you can edit and test the new version before you make it current.

   If you do not make the version current at this time, you can do so later using option 26 on the Work with Versions panel or by using the Redirect Model Objects (YRDRMDLOBJ) command.

3. Determine whether to transfer the object name. You can request that CA 2E exchange the name of the original function or message with the name of the new version by entering *****YES for the Transfer object name option. As a result, the name assigned to the original object will be the name indicated by the To model object name option. The default is *not* to exchange the names.

4. Press Enter to create the new version and return to the Work with Versions panel.

CA 2E adds the new version to your session list and creates a model object description for the new version. The Copy name assigned to the new version will be the Copy name currently being used for the group to which the new version belongs.

You can view the model object description for a non-current version using selection option 8 on the Edit Model Object List panel when editing your session list or any named model object list containing the version. By default only current versions are displayed when you edit the All Objects list (*ALLOBJ). To display non-current versions, press F17 and set the Current objects only option on the Subset Model Objects panel to *YES.

Making a Version Current

The current version in a version group is the one used by other model objects in the model. The current version has the following characteristics:

- Only one version can be current at a time
- Only the current member of a group appears on CA 2E editing panels
- You can select only current versions from selection panels
- You can only reference a current version, for example, in an action diagram construct

A version group need not have a current version. In that case, the only impact is that the function or message will not be visible on CA 2E editing panels. Since a non-current version can appear as an entry of a model object list, you can still view, edit, and work with the versions in the group.

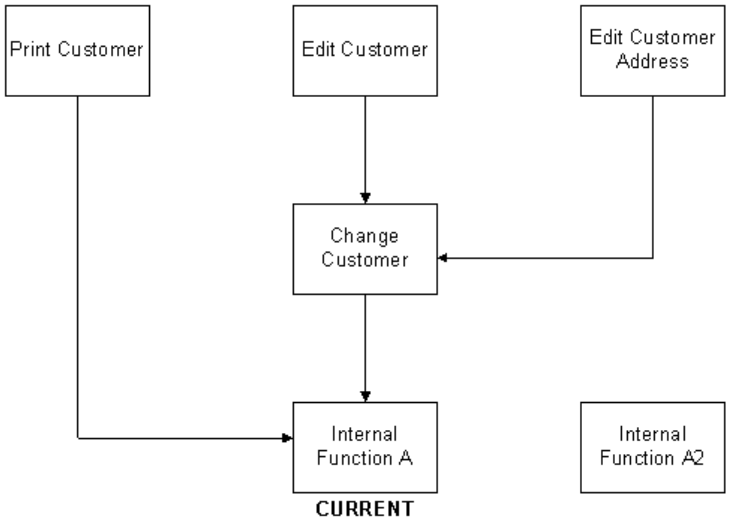
When you make a version current, CA 2E:

- Identifies the model objects that directly use (see) the existing current object. These are the level 001 using objects discussed in the Impact Analysis section.
- Changes all references for the identified using objects to see the new current version.
- Exchanges the implementation names of the existing current version and the new current version. This is the message identifier for a message (MSG) and the source member name for an external function (FUN).
- Optionally exchanges the model object names of the existing and new current versions.

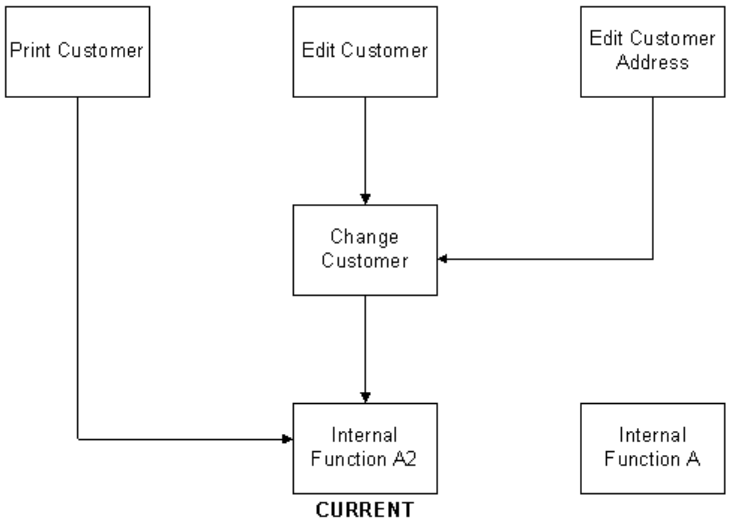
This process is also referred to as redirection because it redirects all model objects that see the existing current version to see the new current version.

Example

Suppose Internal Function A2, a version of Internal Function A, has been edited and tested and you want to make it current. This diagram shows the other objects in the model that see Internal Function A. Note that no model objects see Internal Function A2.



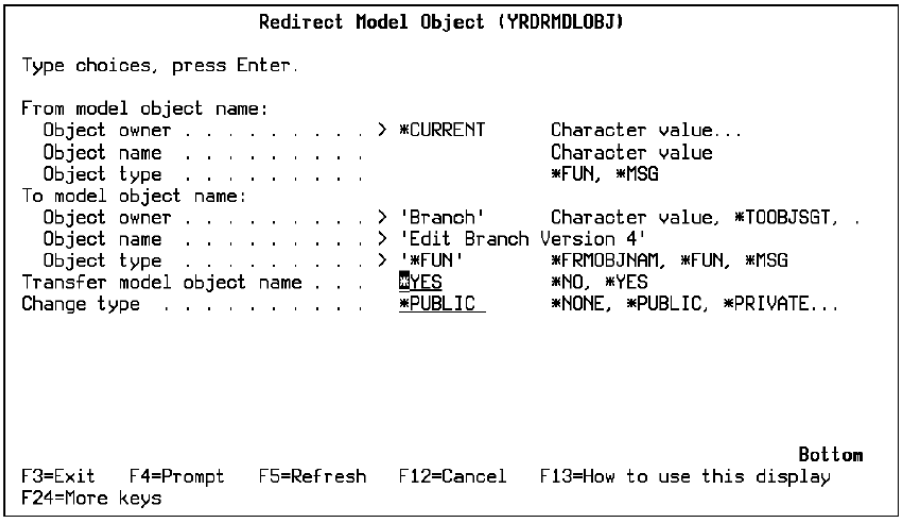
When you make Internal Function A2 current, all model objects that had referred to Internal Function A are changed to use Internal Function A2. This is shown in the following diagram. Note that other model objects no longer see Internal Function A.



If you find errors in Internal Function A2 you can make Internal Function A current again using the same process.

To make a version current:

From the Work with Versions panel enter 26 for the version you want to make current or execute the Redirect Model Object (YRDRMDLOBJ) command from the command line. The following panel displays:



1. Indicate how CA 2E is to handle the object names of the two versions. By default,CA 2E exchanges the name of the original current version and the name of the new current version.

   Be sure to document the exchange of model object names since this could be confusing to others. Also, note that model object list entries that see these versions are not automatically updated to reflect the exchange of names. You can refresh the affected model object lists by pressing F15 from the Edit Model Object List panel to invoke the Check Model Object List (YCHKMDLLST) command.

   You can change the default if you set the Transfer object name option to *NO.

2. Indicate whether redirection is to be considered a significant change to the model. By default, considers redirection to be a public change. If you set the Change type option to *PUBLIC or *PRIVATE, updates component change processing data in the All Objects list for the new current version. If you specify *NONE, component change processing is not performed.

For more information:

- On Change type, see the Model Object Audit Information section in this chapter.
- On component change processing, see the Impact Analysis section in this chapter.

Cautions

When you make a version current, all objects that use the previously-current version are affected. Use this feature with care to prevent unexpected results. The following are some points to consider:

- You can determine the impact of making a new version current in advance using CA 2E's impact analysis tools to view the level 001 usages for the existing current version. For example, use selection option 95 on the Display Model Usages panel to simulate a public change.
- If some of the using objects require the original functionality (in other words, if they need to continue using the existing current version), you cannot introduce your changes by making the new version current. Instead:
  a. Copy the new version to create a new model object, not a version.
  b. Edit the action diagrams of the using objects that require the new functionality to see the new copy.
- To implement and test an internal function or message it must be used by an external function. In other words, it must be made current. Before you make it current, be sure to advise other developers so that your changes to the internal object are not inadvertently incorporated into a change to an application program.

Non-current Versions

The versions within a version group that are not current:

- Can be included as entries on a model object list.
- Can be edited like any other model object.
- Are subject to the same locking restrictions as other model objects.
- Can be documented, copied and deleted.
- Have unique implementation names and can be generated and called (external functions only).

For more information on calling non-current versions of external functions, see the Testing an External Function section in this chapter.

Other Uses for Redirection

In addition to redirecting usages within a version group to make a version current, you can use the Redirect Model Objects (YRDRMDLOBJ) command in the following ways:

- Redirect the usages of functions and messages that are not versions if they are both of the same type. For example, you can redirect usages from a DSPFIL function to an EDTFIL function, but you cannot redirect from a function (FUN) to a message (MSG).
- Redirect usages to a version in another version group. In this case, both versions must be the current version in its own version group.

  The "from" object remains current after the transfer and the implementation name is not transferred. This ensures that there is a current member in the "from" group.

For more information on the Redirect Model Objects (YRDRMDLOBJ) command, see the *Command Reference Guide*.

Using Versions

Following is the basic process for using a version of a function or message.

1. Create a version.
2. Edit the version.
3. For an external function, generate the source and create the program object.
4. Test the version. See the sections following this list for more information.
5. When you are satisfied with your changes, make the version current.
6. If errors occur, make the previous version current again.

Testing an External Function

To test a non-current generatable version, first generate the source and create the program object. Since the version is not current, the program object will not have the correct name and cannot be called by other program objects. To test the program object, use one of the following methods:

- Use the Call a Program (Y2CALL) command. This command determines the parameters required by an external function directly from details contained in the model. You can provide values for all input-capable fields and you can reuse these values for subsequent calls. This command is especially useful when the parameter interface is complex or if it has changed. You can also retrieve and display output parameters when the called program terminates.

  You can also invoke the Y2CALL command using option 16 on the Edit Model Object List, the Display Model Usages, and Display Model References panels.

- If the program interface is the same as for the current version, you can rename the program object in a separate test library, add the test library to your library list, and test the new functionality.

For more information on the Call a Program (Y2CALL) command, see the *Command Reference Guide*.

Testing Messages and Internal Functions

To implement and test an internal function or message, it must be used by an external function. However, only current versions are used by other model objects. Making a version current before testing it can cause unexpected results. To test an internal function or message, follow these steps:

1. Make a copy of the version using the Copy option on the Edit Model Object List panel. This creates a separate function (FUN) or message (MSG).
2. Use CA 2E's impact analysis tools to identify the external functions that use the existing current version of the internal model object.
3. Create a version of one (or more) of the using external functions.
4. Change the action diagram of the version of the using external functions to see the new copy of the internal model object.
5. Test the changes using the Call a Program (Y2CALL) command to execute the version of the using external function. You can also invoke the Y2CALL command using option 16 on the Edit Model Object List, the Display Model Usages, and Display Model References panels.
6. When you are satisfied with your changes, make the version of the internal function or message current. Also, delete the copy of the internal function or message and the version of the external function.

Comparing Versions

Use the Compare Model Objects (YCMPMDLOBJ) command to compare two versions; for example, to identify changes in one version in order to retrofit them to another version. You can use this command to compare functions, messages, or files (FIL). You can invoke this command from a command line or by using selection option 34 on the Edit Model Object List panel.

For more information on the Compare Model Objects (YCMPMDLOBJ) command, see the CA 2E *Command Reference Gide*.

Deleting Versions

You delete versions using option 4 on the Work with Versions panel or the Delete Model Version (YDLTMDLVSN) command. Since a version is a separate model object, you can delete it as you would any other model object; in other words, if it is not used by other model objects. As a result, you can always delete a non-current version since by definition it is not used by other model objects.

Note: If you try to delete a current version that is *not* used by other model objects,CA 2E instead makes the version non-current. You can then repeat the delete operation to delete the now non-current version.

Chapter 2: Generation and Implementation: An Introduction

This chapter gives you an overall understanding of how the CA 2E generation and compilation process works and what you need to do to set up your application for end users.

Note: The information in this manual is for RPG- and COBOL-generated applications.

This section contains the following topics:

What Happens During Generation and Compilation?

The CA 2E generators allow you to generate CA 2E designs in the HLL you choose: COBOL, RPG, or both.

You can generate source for a CA 2E access path or function interactively or in batch (see the Performance Considerations section in this chapter). The CA 2E generators:

- Produce source from the design that you defined in your CA 2E design model.
- Maintain linkages between files, fields, functions, panels, and reports.
- Preserve the integrity of CA 2E database objects.

The type of source generated depends on the object type for which you issue a request for generation. The object types include:

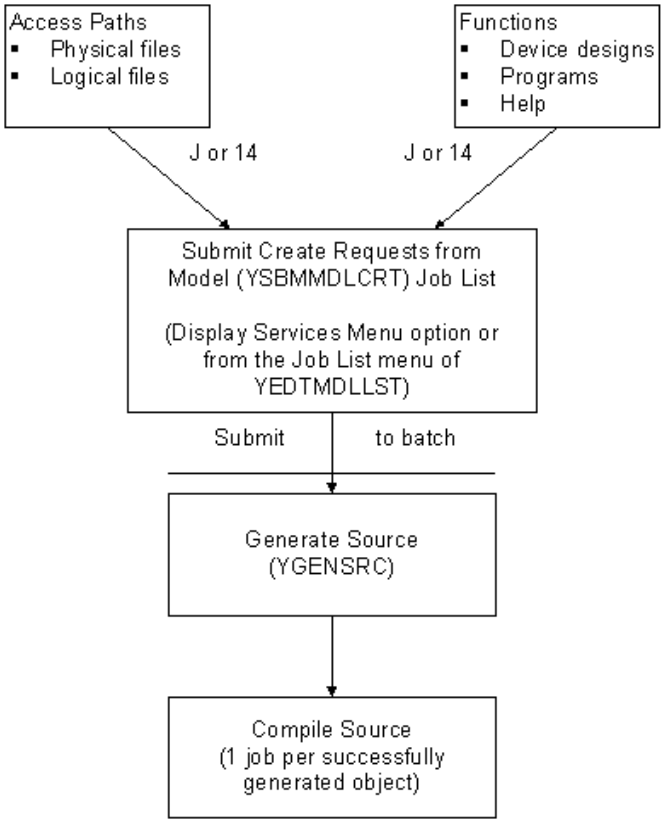
- **Access path**—Produces DDS or SQL for the access path
- **Function**—Produces:
  - HLL program source for external functions, which can include embedded SQL data manipulation language (DML) statements.
  - Device file DDS for device functions.
  - Help text for interactive functions (TM or UIM).
- **Field reference file**—If used, produces DDS for the field reference file

When you submit a request for generation/compilation, the CA 2E generator automatically does the following:

- The member names to be generated are placed in a job list.CA 2E maintains the list during generation and compilation. You can review the list for the status of each item throughout the process, as well as edit the list.
- Source is produced and placed in the appropriate source file in the generation library, according to the object type and HLL that you have chosen. You can name these source files on the Edit Generation Types panel.

  You can override the library for source generation on the Submit Model Create (YSBMMDLCRT) command or by changing the model profile of the user submitting the request.
- The generated source is compiled from the source file according to the object type or HLL that you have chosen.

The following diagram summarizes this process:


Access Paths
Physical files
Logical files
Functions
Device designs
Programs
Help
J or 14
J or 14
Submit Create Requests from Model (YSBMMDLCRT) Job List
(Display Services Menu option or from the Job List menu of YEDTMDLLST)
Submit
to batch
Generate Source (YGENSRC)
Compile Source (1 job per successfully generated object)

Implementation

Once you have generated, compiled, and tested your functions and access paths, you are ready to set up your application and move it into production. Implementation is the process of setting up your application for end users. Your tasks might include the following:

- Using CA 2E Toolkit menu facilities, create menus for end user access to your application.
- To run your application in an environment without the CA 2E product libraries, duplicate necessary CA 2E shipped objects into the library for execution objects. Use the Duplicate Application Objects (YDUPAPPOBJ) command or the Create Generation Objects(YCRTGENOBJ) command.
- Set up test standards and verify that your application works as designed.
- Using Toolkit move commands (and compile commands, as needed) transfer source files, generated application objects, and/or data objects from your development library to the execution library.

You can manage many of these operations using Change Management (CM).

For more information:

- On CA 2E commands, see the *Command Reference Guide*.
- On Toolkit menus, see the *Toolkit Concepts Guide*.
- On Toolkit move commands, see the *Toolkit Reference Guide*.
- On CM, see the *Change Management User Guide*.

Performance Considerations

This section offers tips on efficient generation and compilation.

Batch or Interactive Source Generation?

You have the option to generate source interactively or in batch. Generating in batch allows you to generate many source members together and to have less impact on others using the system. It also permits you to continue to work in your model on other tasks while the generation job is active. Generating interactively is an intensive process that may negatively impact other interactive users.

Generate Several Objects at a Time

Because there is some overhead in starting any generation job, it is better to submit several access paths and/or functions to generate at one time.

Separate Source and Object Libraries

By default, generated source and compiled objects are contained in the library specified by the GENLIB parameter on the Submit Model Create (YSBMMDLCRT) command. You can use the SRCLIB parameter on this command to specify a separate source library. The source library specifies the library into which source is to be generated or that contains the source for a create object request.

If the GENLIB and SRCLIB parameters specify the default value, *MDLPRF, you can also control the specification of the source and object libraries by changing the model profile of the user submitting the request.

Message ID Generation for National Languages

The model value YPMTGEN determines whether constants on devices are generated as messages. You can use the Change Model Value (YCHGMDLVAL) command to change the YPMTGEN model value. If you set YPMTGEN to *MSGID, the generator generates message file descriptions for literal constants. Because generating these descriptions takes time, you may want to block the process during development by setting YPMTGEN to *OFF.

When you finish the development and system test phases, your application is ready for final testing. Then set YPMTGEN to *MSGID if you are externalizing the device text constants for National Language Support (NLS). You must then generate and compile your functions.

Suppressing Help Text

By suppressing help text generation until after all of your modifications to a function are complete, you save generation time. To turn off help generation for the model, set the YGENHLP model value to *NO using the YCHGMDLVAL command. You can also override this model value at the function level using the Generate help function option.

When ready to generate help text for a specific function, you can set the help text function option to Y (*YES) or O (*ONLY) for that function, then generate/compile.

- ***YES**—Generates and compiles help and other function components.
- ***ONLY**—Generates and compiles only the UIM help text for the function. Use the *ONLY option only when all development on the compiled function is complete.

Suppressing Comments in Source Code

The time required to generate a function can be significantly improved if comments are not required for the generated source code. The YGENCMT model value lets you specify whether or not comments are placed in the resulting generated source code.

The possible values for the YGENCMT model value are:

- ***ALL**—All comments are generated into the source for a function
- ***HDR**—Only header comments are generated
- ***NO**—No comments are generated
- ***STD**—The same as *ALL

Model Reorganization

By running the Reorganize Model (YRGZMDL) command, you can reduce the amount of storage needed for the model library by removing old data. Use the RGZOPT(*MDL) option on this command. Running the YRGZMDL command regularly for a model and job lists, based on how often they change, is a good strategy.

Deleting Compile Listings

All compilations result in listings. You may want to keep a listing if errors occurred. You can delete unwanted spool files from the output queue. The Toolkit compile pre-processor also provides an option to cancel any spool file listings except the latest.

Note: For more information on using the Toolkit compile pre-processor, see the *Toolkit Concepts Guide*.

Chapter 3: Preparing for Generation and Compilation

Before creating your application, you may want to review and change aspects of the environment in which your generation and compilation jobs will run. Included in this chapter are guidelines for which model and environment settings to verify before generating and compiling, and how to set up and use your job list.

This section contains the following topics:

Verifying Your Generation Library Setup

CA 2E generates source into source files in the generation library (GENLIB) specified by the YGENLIB model value on the Create Model Library (YCRTMDLLIB) command. In general, your GENLIB contains:

- Source files
- Message files
- Data areas (for example, date format)
- CA 2E-shipped objects (for example, values list display)
- Journal and journal receiver
- The objects you created
- You can create additional generation libraries using the Create Generation Objects (YCRTGENOBJ) command.
- If you change the model value YGENLIB using the YCHGMDLVAL command, you must update the YMPHLBA data area in the new library. This data area contains the name of the library into which Help text is generated. This value is usually the name of the Generation library.
- CA 2E also creates an SQL collection in the SQL library (SQLLIB), if specified by the SQLLIB parameter on the YCRTMDLLIB command.

The SQL library can be created at the same time as the model. If it was not, you can set up an SQL library using the Create SQL Library (YCRTSQLLIB) command. This command also updates the YSQLLIB model value with the SQL library name.

For more information:

- On specifying SQL, see the Administrator Guide.
- on applications that access distributed data, see the chapter titled “Distributed Relational Database Architecture” in this guide.

Defining Source File Names

For your GENLIB, you can define the names of source files at Edit Generation Types. In the shipped product, the names default to i OS default names for each source type. The defaults are:

- DDS - QDDSSRC
- RPG - QRPGSRC
- COBOL'74 - QCBLSRC
- COBOL'85 - QLBLSRC
- SQL DDL - QSQLSRC
- CLP - QCLSRC
- Text Management Help text - QTXTSRC
- UIM Help text - QPNLSRC
- For CA 2E Thin Client:
  - Application Definition Format (ADF) - YADFSRC
  - Screen Definition Format (SDF) - YSDFSRC
  - SDF Instruction Format (SIF) - YSIFSRC
  - Windows Help (RTF) - YRTFSRC

Changing Other System Parameters and Model Values

This section covers the standard source banner and execution displays.

Other implementation options include switching from DDS to SQL and from standard CUA header/footers to windows and action bars (CUA Entry to CUA Text Subset).

For more information:

- On parameters and model values specific to access paths, see *Building Access Paths*.
- On parameters and model values specific to functions, see *Building Applications*.

Changing Text in Standard Source Banner

All CA 2E-generated source contains a standard banner, along with a title line and compiler override line. These lines include the information needed for the CA 2E Toolkit compile pre-processor to compile objects. The banner consists of:

- **Generated by**—The version data
- **Type**—Such as Edit File or Retrieval Index
- **Company name**—From the YCMPTXT model value
- **System**—From the YMDLTXT model value
- **User name**—The user profile name
- **Date**—Includes the job date and time
- **Copyright**—From the YCMPTXT model value

CA 2E retrieves the text for the standard banner in all source types from the messages file Y2MSG. If you want different text in the banner, you can change the message text for these messages using the i OS Work Message Description (WRKMSGD) command.

For more information on WRKMSGD, see the *Application System/400 Programming: Control Language Reference*.

The following table lists the messages used in the standard source banner. These messages are stored in Y2ALCMSG in the language dependent object (LDO) library, Y2SYVENG.

MSGID	Message Text	Shipped Values	Variable
Y2F8463	'Generated by	:&1 Version: &2'	CA generator and version identifier
Y2F8464	'Function type	:&1 Version: &2'	YCMPTXT
Y2F8465	'Company	:&1'	YMDLTXT
Y2F8466	'System	:&1'	Job user
Y2F8467	'User name	:&1'	Job date/time
Y2F8468	'Date	:&1’ Time: &2'	YCMPTXT
Y2F8469	'Copyright:	&1’	

Note: You will need to re-apply any changes to the banner after each CA 2E product upgrade.

Execution Displays

CA 2E provides the following model values for changing certain execution data, using the YCHGMDLVAL command:

- YDATFMT—Date display format. If YDATGEN is MDY, YMD, or DMY, YDATFMT is ignored. If YDATGEN is VRY, CA 2E checks the value of YDATFMT to determine which to use.
- YCMPTXT—Company text on displays.
- **Y2MGFLA**—Message file name.
-
- For more information on YCHGMDLVAL, see the CA 2E Command Reference Guide.

Note: Always use the YCHGMDLVAL command to change model values, rather than using the i OS Change Data Area (CHGDTAARA) command. Changing model values involves more than changing data areas.

Changing Message File Names

When creating a model,CA 2E automatically creates a message file to contain the descriptions that implement execution message functions. The message file resides in your GENLIB as specified by the YGENLIB model value. You can change the message file name that was set at model creation by the MSGVNM parameter on the YCRTMDLLIB command. To change the default message file name, use the YMSGVNM model value with the YCHGMDLVAL command.

Reviewing and Changing Compiler Overrides

You may want to change program and device file properties by specifying compiler overrides.CA 2E stores the values you specify in the source as Z* comments.CA 2E automatically applies the overrides during compilation, using the Toolkit compile pre-processor.

For example, the SEU source listing for an Edit Customer program might include these lines:

```
10    H/TITLE Edit Customer       Edit file
20    H             Y
30    Z*  CRTRPGPGM
40    Z*  USRPRF(*OWNER)
50      *
60    H*  SYNOPSIS:
70    H* -Maintain database file using subfile display
.

160   H* Company      : Your Company
170   H* System       : Your System
180   H* User name    : YOU
190   H* Date         : 07/16/96    Time  :  14:32:17
200   H* Copyright    : Your Copyright
```

The Toolkit compile pre-processor offers many options, such as invoking CL commands and storing compiler directives in source.

For more information on the pre-processor, see the *Toolkit Concepts Guide*.

For Functions

The compiler overrides you select depend on the object type but might include:

- **Printer device files**—Using the i OS Create Print File (CRTPRTF) command to specify form characteristics and spool file scheduling, such as OUTQ(MYOUTQ)
- **Display device files**—Using the i OS Create Display File (CRTDSPF) command, parameter WAITRCD (workstation timeout)
- **RPG programs**—Using the i OS Create RPG/400 Program (CRTRPGPGM) command, with parameter USRPRF set to *OWNER to adopt authority
- **COBOL programs**—Using the i OS Create COBOL Program (CRTCBLPGM) command, with parameter USRPRF set to *OWNER to adopt authority

For Access Paths

Some overrides you can specify for access paths are:

- Physical files, using the i OS Create Physical File (CRTPF) command with such parameters as:
  - **MAXMBRS**—Maximum number of members the physical file can hold
  - **SIZE**—Initial number of records in each member of the file
- Logical files, using the i OS Create Logical File (CRTLF) command with such parameters as:
  - **MAXMBRS**—Maximum number of members the logical file can hold
  - **DTAMBRS**—The physical files and members containing data associated with the logical file

Note: Do not override values that are already specified by access path details, such as MAINT and TEXT. You can review these values on the Access Path Details panel.

For more information on IBM i commands, see IBM's *Application System/400 Programming: Control Language Reference*.

Viewing and Changing Shipped Source

CA 2E shipped source files, stored in files in the Y2SYSRC library, include user-modifiable programs and execution support programs that you can tailor to your needs. You can replace any of these shipped programs with your own.

Important! Make sure you copy the source before changing it.

User-modifiable Shipped Programs

A set of shipped CL programs controls name allocation. Each program names an object type. For example, the program YALCPHYR1C names physical files.

Note: You can change the allocation character that each object type uses from the Edit Generation Types panel.

For more information on name allocation and auto-naming, see the *Administrator Guide*

Execution Support Programs

The following table shows the shipped programs that support execution time processing:

File	Program	Description	Details
QRPGSRC	Y2VLLSR	Values list display	YVLSPX + VLLSR. The DDS for the display is shipped in QDDSSRC in member Y2VLLSR#. The messages describing the display details are in the message file Y2USRPMT.
	Y2VLLWR	Values list display for windows	
QRPGSRC	Y2PSSRR	RPG error handling (*PSSR)	See the Error Routine and Features in RPG not in COBOL sections in this chapter.
QRPGSRC	Y2CLMSG	CLRMSG	Clear messages from a program message queue.
QRPGSRC	Y2SNMGC	SNDxxxMSG	Error, status, information, and completion messages.
QCLSRC	Y2EXMCC	EXCMSG	Execute a message.
QRPGSRC	Y2RVMGC	RTVMSG	Retrieve a message.
QCLSRC	Y2BGCMC	Commitment control	If not already active, start commitment control.

Note: The message handling routines are shipped as RPG programs using i OS Message Application Programming Interfaces (APIs). These replace the original CLP routines. The names have been left with a C extension for compatibility with old applications.

Managing Your Work Environment

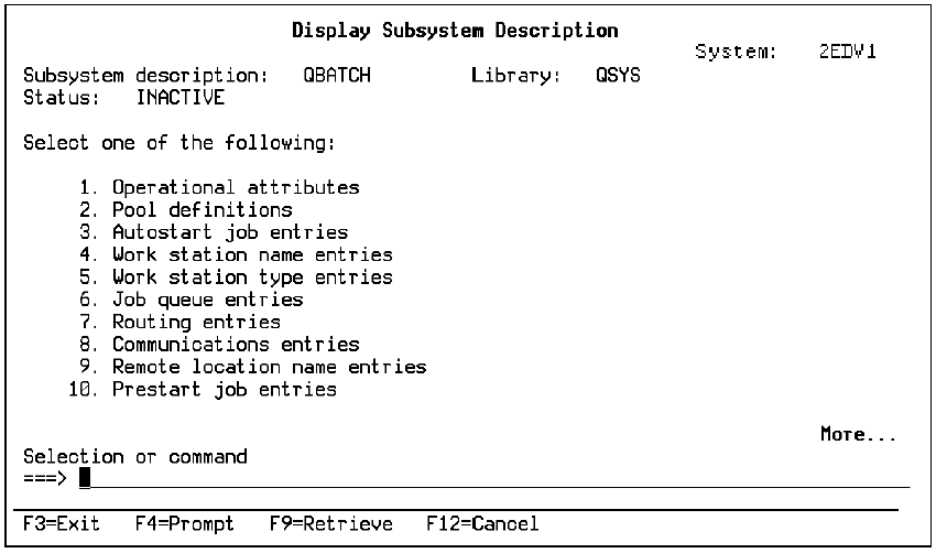
This section covers settings for your work environment. Based on IBM i facilities, the information here is tailored for users.

CA 2E is designed to make use of some of the IBM i subsystem facilities. For successful generation and implementation, an understanding of these IBM i facilities is important. The following information will give you an understanding of how to create and change the subsystem description for use with CA 2E.

Note: All jobs on the IBM i run in a subsystem. You may want to define a subsystem to run similar kinds of jobs, such as interactive or batch.

For more information on IBM i subsystem facilities, see IBM's *Application System/400 Programming: Work Management Guide*.

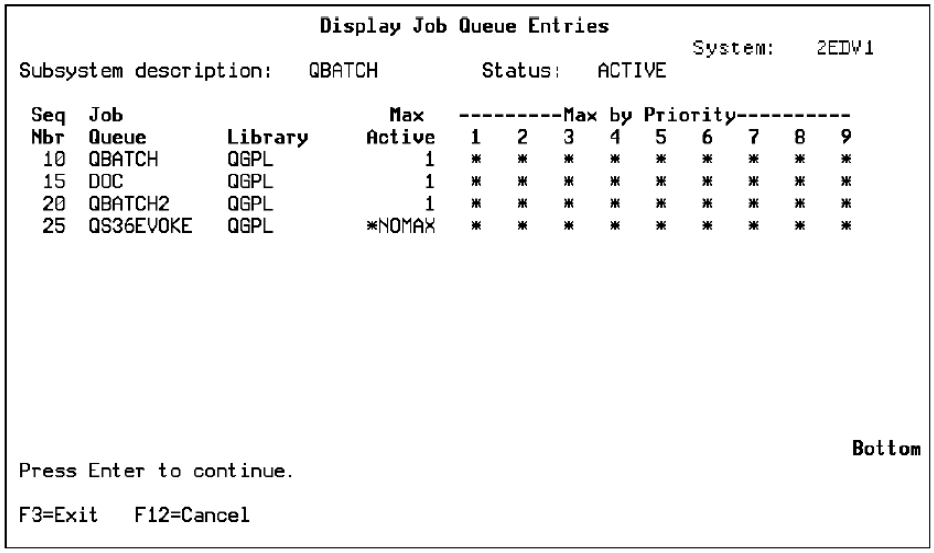
You can display the subsystem definition with either the i OS Display Subsystem Description (DSPSBSD) or Work with Subsystem Description (WRKSBSD) commands.When you run the DSPSBSD command, the following panel displays:



To focus on the job queue and routing entry descriptions, let us use, as an example, the QBATCH subsystem for batch jobs. YSBMMDLCRT uses QBATCH by default. You can change this default and create a different subsystem of CA 2E-related activity.

Job Queue Entries

Option 6, Job queue entries, on the Display Subsystem Description panel displays every job queue that QBATCH uses. When you choose option 6, the Display Job Queue Entries panel displays:



Take note of these two fields:

- **Seq Nbr (sequence number)**—Defines the order in which QBATCH selects the job queues that can start jobs.
- **Max Active (maximum active)**—Defines how many jobs from the corresponding Job Queue you can run at the same time. Ensure that this value is 1 for the job queue (JOBQ) you specify in each model's job description (JOBD).

Routing Entries

Option 7, Routing entries, on the Display Subsystem Description panel defines how CA 2E is to start a job in QBATCH. When you choose option 7, the Display Routing Entries panel displays:

```
                          Display Routing Entries
                                                          System:    2EDV1
Subsystem description:   QBATCH         Status:   ACTIVE

Type options, press Enter.
  5=Display details

                                                                      Start
Opt     Seq Nbr    Program       Library       Compare Value          Pos
 _        300      QCMD          QSYS          'QS36EVOKE'            1
 _        700      QCL           QSYS          'QCMD38'               1
 _       1111      YBRTPRC       A1SY          'ACRTOVR'              1
 _       2222      YBRTPRC       P1SY          'PCRTOVR'              1
 _       3333      YBRTPRC       S1SY          'SCRTOVR'              1
 _       4444      YBRTPRC       Y1SY          'YCRTOVR'              1
 _       5555      YBRTPRC       Z1SY          'ZCRTOVR'              1
 _       6666      YBRTPRC       B1SY          'BCRTOVR'              1
 _       7779      YBRTPRC       M10646        'TCRTOVR'              1
 _       9777      YBRTPRC       M10657        'GCRTOVR'              1
 _       9999      QCMD          QSYS          *ANY

                                                                     Bottom
F3=Exit   F9=Display all detailed descriptions   F12=Cancel
```

CA 2E compares the routing data from the job description to each Compare Value in the list as follows:

- If there is a match,CA 2E calls the associated program.
- If there is not a match,CA 2E uses the last routing entry with the value *ANY. In this example, the job has a routing entry of YCRTOVR, which matches sequence number 4444 and calls the program YBRTPRC.

Initially, the generation job, YGENSRC, uses the Class assigned to the routing entry of YCRTOVR. The program YBRTPRC is associated with the routing entry of YCRTOVR; program YBRTPRC is the pre-processor for YGENSRC. The last step in pre-processing assigns new routing data and issues the i OS Reroute Job (RRTJOB) command. This command reroutes the job, depending on the YCRTENV model value setting. If model value YCRTENV is set to QCMD (IBM i mode), routing data is set to QCMDB.

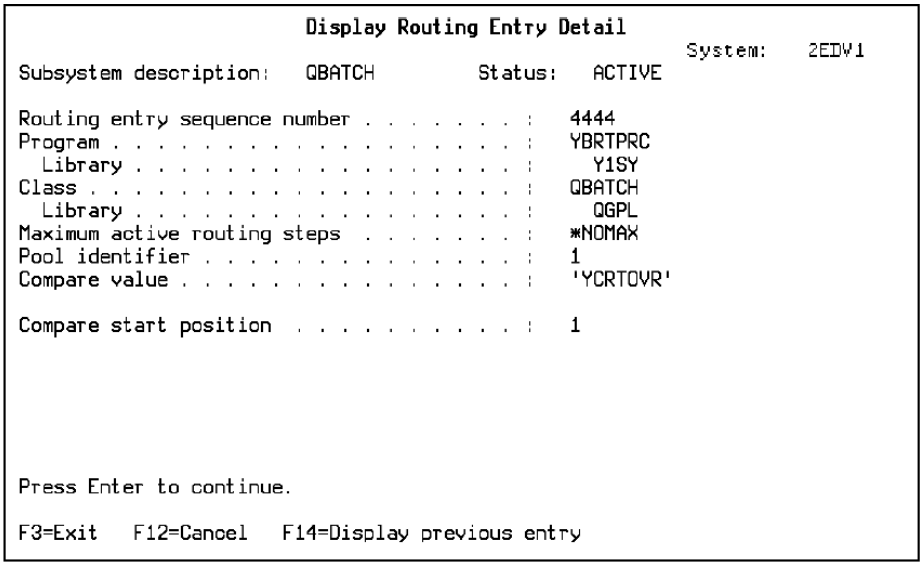
The subsystem compares the new routing data with the Compare Values in the routing entries list.

- If a match is made on QCMD38, job control passes to the QCL program.
- If a match is made on QCMDB or *ANY, job control passes to the QCMD program.

If you want to change the run priority of YGENSRC, modify the Class associated with each of the following routing entries.

- Compare Value YCRTOVR
- Compare Value QCMD and/or QCMD38, depending on your model creation environment (YCRTENV)

To determine the Class, look at the details for the routing entry associated with YCRTOVR. On the Display Routing Entries panel, enter 5 beside the routing entry. This Display Routing Entry Detail panel displays (for this example, the Class is QBATCH).



Let us return to the Display Routing Entries and enter 5 beside the routing entry associated with *ANY. The Display Routing Entry Detail panel displays, with Class QBATCH.

```
                        Display Routing Entry Detail
                                                            System:   2EDV1
Subsystem description:   QBATCH          Status:   ACTIVE

Routing entry sequence number . . . . . :   9999
Program . . . . . . . . . . . . . . . . :   QCMD
  Library . . . . . . . . . . . . . . . :     QSYS
Class . . . . . . . . . . . . . . . . . :   QBATCH
  Library . . . . . . . . . . . . . . . :     QGPL
Maximum active routing steps  . . . . . :   *NOMAX
Pool identifier . . . . . . . . . . . . :   1
Compare value . . . . . . . . . . . . . :   *ANY

Compare start position  . . . . . :

Press Enter to continue.

F3=Exit   F12=Cancel   F14=Display previous entry
```

The run priority defines the job priority (1 through 99). A value of 1 for the run priority gives a job the highest priority in competing for machine resource with other jobs.

To display the Run priority of each program, you can invoke the Display Class Information panel from an i OS command line by entering:

```
DSPCLS Class
```

where Class is the value shown on the Display Routing Entry Detail panel for that routing entry.

Verifying Your Work Environment Setup

To successfully generate and compile your application, verify how your work environment is set up.

Two types of subsystem configuration are possible on the IBM i: QBASE and QCTL. QCTL is recommended for the controlling subsystem.

Before you can verify the settings in your work environment, you need to identify:

- The value of your controlling subsystem; from an i OS command line, type DSPSYSVAL QCTLSBSD and press Enter.
- The model job queue from the model job description. Enter DSPJOBD QBATCH (or your job description name), and record the job queue attached to this JOBD.

The steps below cover both configurations:

- If your controlling subsystem is QBASE and the job queue name in your model job description is QBATCH, follow all the steps.
- If your controlling subsystem is QCTL, or if you are using a job queue other than QBATCH, begin with step 7.

Note: To perform the following steps, sign on as QSECOFR or as a user profile that belongs to the same group profile as QSECOFR.

To manage your work environment:

Steps 1-6 apply only to QBASE subsystem or non-QBATCH job description.

1. Verify that the job queue entry QGPL/QBATCH exists:
   a. On the command line, type WRKSBSD QBATCH. Press Enter.
   b. To display the subsystem description, enter 5 at the Work with Subsystem Description panel.
   c. Verify that QBATCH exists in QGPL at the Display Job Queue Entries panel.
2. If needed, create job queue entry QBATCH2, in library QGPL, for subsystem QBATCH. At a command line, type the following command string and press Enter.

   ```
   CRTDUPOBJ OBJ(QBATCH) FROMLIB(QGPL) +
     OBJTYPE(*JOBQ) NEWOBJ(QBATCH2)
   ```

   Note: QBASE has a job queue named QBATCH. If your job queue is also named QBATCH (the default), the submitted job will go to the QBATCH job queue in QBASE.
3. Stop subsystem QBATCH. Type the following command string and press Enter:

   ```
   ENDSBS QBATCH
   ```
4. Add job queue entry QBATCH2 to subsystem QBATCH. Type the following command string and press Enter:

```
ADDJOBQE SBSD(QBATCH) +
  JOBQ(QGPL/QBATCH2)
```

5. Restart subsystem QBATCH. Type the following command string and press Enter:

   ```
   STRSBS QBATCH
   ```

   Now that you have reviewed and changed your subsystem, complete steps 6-9 to make sure it works correctly in CA 2E.

6. In every model, change the job description to use the job queue name QBATCH2. For each model, type the following command string and press Enter:

   ```
   CHGJOBD JOBD(model-library-name/QBATCH+
     or your-job-description-name) +
     JOBQ(QGPL/QBATCH2)
   ```

   Note: You should check any job descriptions explicitly referenced in the model profiles for the model. You can edit the model profile for a user using the Change Model Profile (YCHGMDLPRF) command or by pressing F18 from the Edit Model Object List panel.

   For more information on compiling right after generation, see the Sending Generations and Compilations to Separate Queues section in this chapter.

7. In every model, do the following:

   a. Verify that the job description library list is correct for the model. Type one of the following command strings, as appropriate, and press Enter to access the Display Job Description panel:

```
DSPJOBD JOBD(QGPL/QBATCH)

DSPJOBD JOBD(model-library-name +
```

You should check the library list for the model using the Edit Library List (YEDTLIBLST) command. You can determine the model's library list using one of the following:

- The Display Model Value (YDSPMDLVAL**)** command to display the YLIBLST model value.
- The Library list options on the Designer (*DSNR) Menu. Enter:

  ```
  YGO DSNR *Y2
  ```

  to display this menu.

  Suggested list:

- QTEMP (must be at top of library list)
- Generation library
- Library for SQL collection (optional)
- Model library
- QGPL

- National Language Support libraries (Y1SYVxxx and Y2SYVxxx) (optional)
- CA 2E product (Y2SY)
- Toolkit product (Y1SY)

a. Verify that the routing data value in the model job description is YCRTOVR. To verify the value, locate it on the Display Job Description panel. If needed, you can enter the following command string to change the routing data value:

```
CHGJOBD +
  JOBD(model-library-name/QBATCH) +
  RTGDTA('YCRTOVR') You should check any job descriptions explicitly referenced in the
```

model profiles for the model.

Note: YCRTOVR must be in capital letters.

1. Verify that the routing entries are correct in the subsystem to which you will submit the CA 2E jobs. The subsystem must contain a routing entry with a value of YCRTOVR to match the routing entry in the job description the model uses. Perform the following to verify routing entries:

   a. Type WRKSBSD QBATCH (or your subsystem name) and press Enter to access the Work with Subsystem Descriptions panel.

   b. To display the subsystem description, at Work with Subsystem Descriptions type 5 and press Enter. The Display Subsystem Description panel displays.

   c. To display routing entries, at the Display Subsystem Description panel select option 7. Your subsystem should contain the following routing entries.

Sequence Number	Program	Library	Compare Value	Start Position
1111	YBRTPRC	Y1SY	'YCRTOVR'	1
9999	QCMD	QSYS	*ANY	

2. 2 . If needed, change or add routing entries using either the i OS command, Change Routing Entry (CHGRTGE) or Add Routing Entry (ADDRTGE), as appropriate. Be sure to terminate the subsystem before adding a routing entry.

Moving Toolkit Data Objects from Y1SY

Some of the Toolkit commands use database files or data areas to store user data, such as library lists, user profile extension attributes, and design defaults. Unless you specified a separate data object library when you installed the product, these data objects reside in library Y1SY. You can choose to put them in another library.

To move data objects from the existing Toolkit library Y1SY to another library, type the following command string:

```
YMOVY1DTA FROMLIB(YSY) TOLIB(NEWLIB)
```

For more information on the Move Product Data Objects (YMOVY1DTA) command, see the *Toolkit Reference Guide*.

Sending Generations and Compilations to Separate Queues

If you set up jobs so that generations and compilations go to the same queue, all generations must complete before any compilation can begin. If you want a job to compile immediately after it generates, you can send compilations to a different queue than generations. The queue must contain the Toolkit compile pre-processor.

To set up and use a separate queue for compilation:

1. Copy the job description in your model into your generation library.
2. Change the job queue of the job description in the generation library to the one you want to use for compilation.
3. When you execute the YSBMMDLCRT command, change CRTJOBD to the job description that you set up for compilation.

   Note: You can execute the YSBMMDLCRT command from an IBM i command line, by selecting option one on the Display Services Menu, or by pressing F19 from the Edit Model Object List panel to display the Job List Commands Menu.

Understanding Job Lists

When you request generation,CA 2E places the names of members to be generated on a job list. The same job list controls generation and compilation. You can review this job list during generation to monitor the process and edit the job list.CA 2E assigns a status to each member on the list. If errors occur,CA 2E flags the specific members in error. Once source is generated for the members,CA 2E automatically submits a request to compile the generated source. A sample of this process follows the brief descriptions below.

Upon receiving your generation request,CA 2E places the members on a job list with a stage flag. The stage flag indicates the next processing step to be performed on the members. This initial flag is based on whether your request is batch or interactive.

- **GEN**—Batch source generation and creation is requested. When batch source generation is completed, the GEN flag changes to CRT.
- **CRT**—Interactive source generation is completed and batch creation is requested.

The processing sequence for batch or interactive mode depends on the value of parameter SBMCRTOPT on the YSBMMDLCRT command.

- ***GENOK**—Default. Submits generated source for compilation only after the source is successfully generated.
- ***IMMED**—Submits source members for compilation with the job that generates source.

After you submit a request, you initiate processing by executing the YSBMMDLCRT command. Once source has been successfully generated,CA 2E changes the stage flag to CRT and submits the source for compilation.

Note: Submission of your request may be deferred until the source generation for related members is complete. If so, a member temporarily may be displayed as CRT *GENSRC.

CA 2E removes from the pending list the members that successfully generate and compile. If members fail to compile, an error message is displayed. You can access source for the members and the compile listings from the job list to identify the problems.

CA 2E assigns each member in the job list one of the following statuses which changes throughout generation:

- ***SBM**—Indicates a job has been submitted to control batch generation and/or submission of compilation.
- ***GENSRC**—Shows that source generation for a member is in progress.
- ***JOBQ**—Shows that generated source for a member has successfully been submitted for compilation;CA 2E deletes existing versions of i OS objects in the generation library.

- ***ACTIVE**—Indicates that the member is being compiled.
- ***ERROR**—Flags an error at any time during generation and compilation.

Note: If you specify CRTJOBD(*NONE) on the YSBMMDLCRT command to indicate that compilation is not required, the process stops after generation.

Sample Job List Series

When you request batch generation, the stage flag (Act) is set to GEN.

?	Member	Type	Act	Status	Text	
	UUAAREP	PF	GEN		Tax	Physical file
	UUABREP	PF	GEN		Item	Physical file
	UUACREP	PF	GEN		Vendor	Physical file

After submission,CA 2E assigns a status to each job on the job list. Immediately after submission, the status is *SBM.

?	Member	Type	Act	Status	Text	
	UUAAREP	PF	GEN	*SBM	Tax	Physical file
	UUABREP	PF	GEN	*SBM	Item	Physical file
	UUACREP	PF	GEN	*SBM	Vendor	Physical file

As CA 2E generates source for a job, the job's status is *GENSRC.

?	Member	Type	Act	Status	Text	
	UUAAREP	PF	GEN	*GENSRC	Tax	Physical file
	UUABREP	PF	GEN	*SBM	Item	Physical file
	UUACREP	PF	GEN	*SBM	Vendor	Physical file

When CA finishes generating the job and submits it for compilation, the activity changes to CRT and the status changes to *JOBQ.

?	Member	Type	Act	Status	Text	
	UUAAREP	PF	CRT	*JOBQ	Tax	Physical file
	UUABREP	PF	GEN	*GENSRC	Item	Physical file
	UUACREP	PF	GEN	*SBM	Vendor	Physical file

When compilation is in progress, the status changes to *ACTIVE.

?	Member	Type	Act	Status	Text	
	UUAAREP	PF	CRT	*ACTIVE	Tax	Physical file
	UUABREP	PF	CRT	*JOBQ	Item	Physical file
	UUACREP	PF	CRT	*JOBQ	Vendor	Physical file

When the job is successfully compiled,CA 2E drops it from the top of the job list.

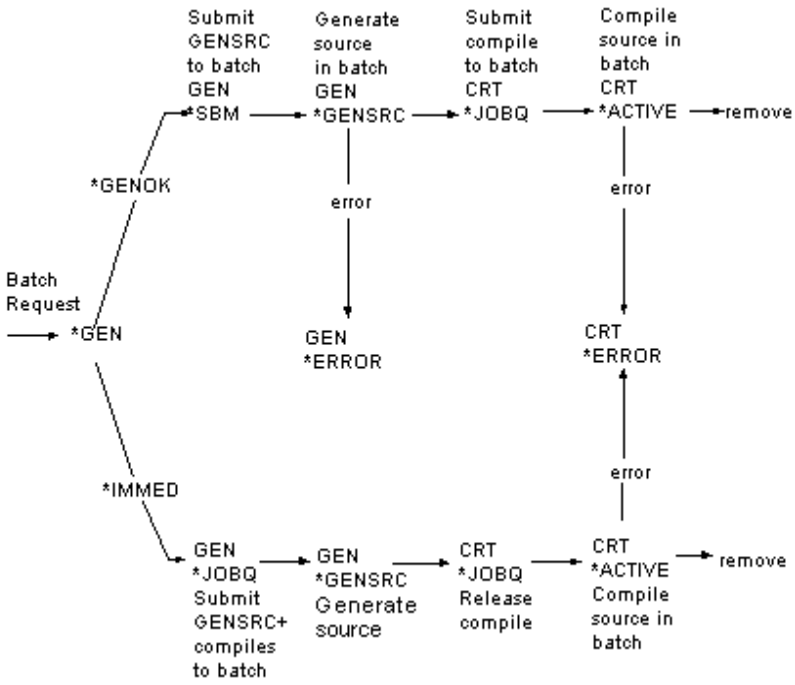
?	Member	Type	Act	Status	Text	
	UUABREP	PF	CRT	*ACTIVE	Item	Physical file
	UUACREP	PF	CRT	*JOBQ	Vendor	Physical file

If an error occurs during generation, the status changes from *GENSRC to *ERROR.

?	Member	Type	Act	Status	Text	
	UUAFREL0	LF	GEN	*ERROR	Customer	Update index

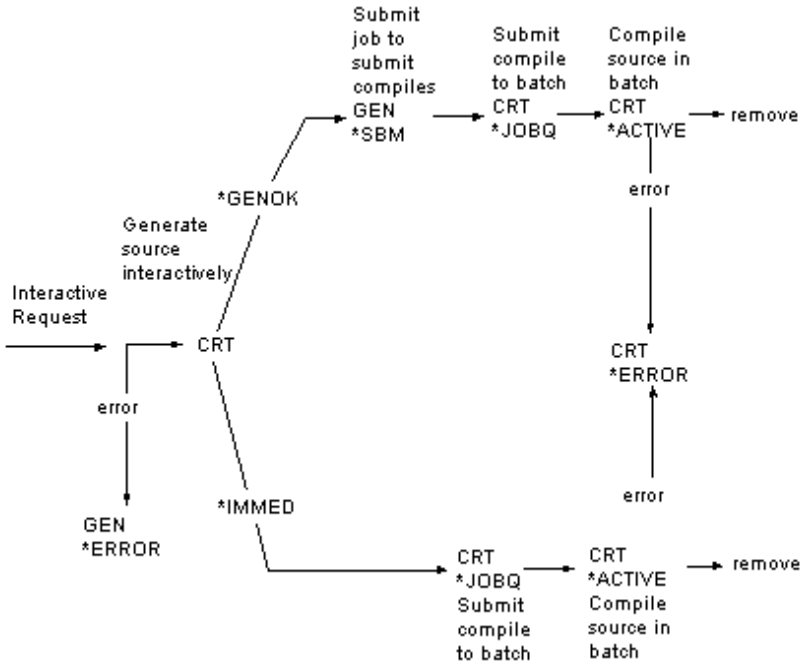
Batch Generation

The following diagram shows the batch generation stages for both *GENOK and *IMMED. Notice that members have a GEN stage flag during generation, which changes to CRT once generation, is successful.CA 2E automatically submits members for compilation that have a CRT stage flag.



Interactive Generation

The following diagram shows the stages for interactive generation for both *GENOK and *IMMED.



For more information on batch versus interactive generation see the chapter titled "Generation and Implementation: An Introduction" in this guide.

Job Descriptions for Batch Generation and Compilation

A job description describes the environment in which a job runs. Parameter JOBD on the YSBMMDLCRT command specifies what job description name CA 2E will use when you submit the job to batch. The value *MDLVAL retrieves the job description name from model value YCRTJBD, usually the QBATCH job description.

Each model has a unique JOBD containing:

- Job queue, which can be any valid IBM i JOBQ
- Routing data, which must be set to YCRTOVR
- Other job attributes, such as the Initial library list and Message logging (Level, Severity, and Text)

You can specify separate job descriptions for generation and compilation, using the JOBD (generation) and CRTJOBD (compilation) parameters on the YSBMMDLCRT command. Both job descriptions default to the YCRTJBD model value.

For compilation, the initial library list of the job description should contain the name of the generation library, specified by model value YGENLIB, and any other libraries containing objects required for compilation, such as the HLL compilers.

Use the Toolkit Edit Library List (YEDTLIBLST) command to maintain both the library list of your model and the associated job description.

Note: Your interactive library list is not used.

Using Job Lists

This section helps you set up and use the job list for generating and compiling your application.

For more information on CA 2E commands covered in this section, see the CA 2E *Command Reference Guide*.

Using More Than One Job List

It is good practice for each programmer or designer using a model to have a job list. You can specify a job list name using the JOBLST parameter on these commands:

- Edit Model (YEDTMDL) command. JOBLST names the list into which CA 2E places requests.
- Submit Model Create (YSBMMDLCRT) command. JOBLST names the list from which CA 2E takes requests.

The value *USER for JOBLST specifies a list with the same name as the user profile of the job invoking the command.

Editing Job Lists

You can edit a job list from the Submit Model Generations & Creates panel. For example, you may want to put a hold on certain members, then release them later.

Note: You can also edit a member's source from Submit Model Generations & Creates by entering E (STRSEU) by the member.

You can access the Submit Model Generations & Creates panel in the following ways:

- From the Display Services Menu, choose the option Submit model create request (YSBMMDLCRT)
- Execute the YSBMMDLCRT command from an i OS command line, setting the EDIT parameter to *YES
- From the Edit Model Object List panel, press F19 and select the YSBMMDLCRT option from the Job List Commands Menu

Building Job Lists

Using the Build Job List (YBLDJOBLST) command, you can create a generic list of objects in a model and use the list to submit regenerations and/or recompilations. You can execute the YBLDJOBLST command from the Display Services Menu. Select the Job list menu option, and then select the Build Job List for Model option.

You can select a job list by specifying:

- Application area
- Generic name of CA 2E files that have the dependent access paths and/or functions to be included
- Generic name of access paths to be included
- Type(s) of access paths to be included
- Generic name of functions to be included

You can specify whether the activity status of the access path and function entries in the list will be *GEN (require generation and compilation) or *CRT (require compilation only). The parameters of the YBLDJOBLST command are:

- **ACPACT**—Access path activity flag
- **FUNACT**—Function activity flag

Alternatively, you can use the Convert a Model Object List (YCVTMDLLST) command to convert an existing model object list into a job list. You can execute this command from the Edit Model Object List panel by pressing F19 and selecting the YCVTMDLLST option from the Job List Commands Menu.

For more information:

- On model objects lists, see the chapter titled "Managing Model Objects" in this guide
- On the YCVTMDLLST command, see the CA 2E *Command Reference Guide*.

Managing Multiple Job Lists

When you request generation/compilation from within a model,CA 2E adds entries automatically to a default job list. Specify the job list using the Edit Model (YEDTMDL) command. This is the same command you use to enter the model; it defaults to *User. If you create multiple job lists, make sure the default job list you are using is the one you want.

Checking Job Lists

Using the Check Job List Entry (YCHKJOBLE) command, you can check the entries in an existing job list to see if source members and/or objects with the same names already exist in a library. You can use this command to sort your job list and drop objects that already exist.

You can execute the YCHKJOBLE command in the following ways:

- Display Services Menu. Select the Job List Menu option and then select the Check Job List for Model option.
- Edit Model Object List panel. Press F19 and select the YCHKJOBLE option.

Parameter UPDLST on the YCHKJOBLE command specifies one of the following actions to take if the object or source does not exist:

- ***RMVOK**—Removes the member from the job list if the object and/or source already exists.
- ***RMVERR**—Removes the member from the job list if the object and/or source does not exist.

Reorganizing Job Lists

As job lists are used, they accumulate old data that you should remove periodically. Use the Reorganize Model (YRGZMDL) command with the RGZOPT(*JOBLST) option.

HLL Implementation Considerations

When converting from one programming language to another, you will need to consider the differences between the languages and the impact on your user source.

Features in RPG Not in COBOL

This section covers differences between RPG and COBOL implementation.

Numeric Parameter Passing

In calling a program and passing numeric parameters to that program, RPG programs first move the parameters to packed variables. RPG programs receive any numeric parameters as packed, and then move them to the associated fields. This ensures that the interface for numeric parameter passing is always packed numeric.

COBOL programs pass numeric parameters using the same definition as the field. If the definition is signed numeric (unpacked), COBOL programs pass the parameter in that way. Similarly, COBOL programs receive numeric parameters according to their definition in the model as packed or signed.

When a COBOL program calls an RPG program with numeric parameter passing, a parameter mismatch error occurs if the COBOL program defines one of the parameters as signed numeric (unpacked).

Exception Monitoring on Program Calls

For RPG implementation, you can monitor a CALL statement to detect exceptions, such as when a called program cannot be found. If an RPG program calls an uncompiled program, an error occurs but processing continues.

For COBOL,CA 2E generates code in the calling function to ensure that the run unit does not terminate.

Closedown Program

For RPG, Closedown Program settings have these results:

- **Y**—Causes RPG to generate a Set on Last Record indicator (LR). This indicator closes all files and ensures a full initialization on the next call.
- **N**—Drops the LR indicator and issues a RETRN. The return allows for a faster subsequent call.

For COBOL, Closedown Program settings have these results:

- **Y**—Issues explicit CLOSE operations to each file.
- **N**—Drops CLOSE operations.

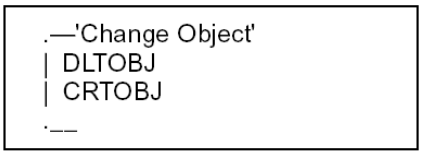
CHGOBJ to Alter Key Values

If you have a model originally implemented in RPG that you want to generate in COBOL, review use of the Change Object (CHGOBJ) function type, described here, and proceed to the steps for converting from RPG to COBOL in this chapter.

A CHGOBJ function implemented in RPG can include code to update key values. However, an equivalent COBOL implementation cannot, since the COBOL REWRITE statement does not allow key value changes.

If you need to change the value of one or more keys using a COBOL program, do not use a CHGOBJ function. Instead, when you define the function that specifies the COBOL function, use separate Delete Object (DLTOBJ) and Create Object (CRTOBJ) functions.

If necessary, you can use an Execute Internal Function (EXTINTFUN) function to provide a dummy CHGOBJ function.



For example, a function that changes the value of a key field in the database has different implementations:

- For RPG implementation, the program could consist of an RTVOBJ function that calls a simple CHGOBJ for each record read.
- For COBOL implementation, the program must consist of separate DLTOBJ and CRTOBJ functions.

For more information on how to change key values using a COBOL program, see *Building Applications*.

Error Routine

For RPG implementation, if you request an error routine, CA 2E generates a *PSSR routine.CA 2E generates any database files in the resulting program, treating each file as user-controlled and specifying a *PSSR routine. The initialization routine includes explicit file opens.

COBOL has no *PSSR routine.

Header Specification

The CA 2E generator obtains an RPG header specification from the model value YRPGHDR.CA 2E uses this header to generate source for RPG programs.

Converting a Model from One HLL to Another

This section gives you some general guidelines for changing HLLs and addresses RPG to COBOL conversions.

User Source Considerations

In changing from one HLL to another, ensure that your user source is in the same HLL as the function that calls it and that the naming convention you use is compatible between HLLs.

Note: If you ever plan on changing the HLL, specify *RPGCBL for model value YHLLVNM.

User Source in Same HLL as Calling Function

An Execute User Source (EXCUSRSRC) function must be the same HLL source type as the function that calls it. For example, the EXCUSRSRC included in a COBOL function must also be COBOL. If you are converting an RPG application to COBOL, you must convert the EXCUSRSRC functions to COBOL.

Note: If the EXCUSRSRC you are converting accesses a data area, be aware that COBOL has no equivalent to the IN and OUT statements in RPG, since COBOL does not support direct access to data areas.

Compatible Names Between HLLs

If you change from one HLL to another, verify that the naming convention you use is compatible for both HLLs. There is not necessarily a one-to-one correspondence between HLLs. For example, COBOL does not accept special characters, and RPG does.

For more information on naming, see the *Administrator Guide*.

Converting from RPG to COBOL

To convert a model to COBOL:

1. If you plan to create more files, fields, or functions, use the YCHGMDLVAL command to change the following model values.
   - Generation Type for new functions, using the command string:

```
YCHGMDLVAL MDLVAL(YHLLGEN) +
  VALUE(*CBL)
```

   - HLL(s) naming convention for new names, using command string:

```
YCHGMDLVAL MDLVAL(YHLLVNM) +
  VALUE(*RPGCBL)
```

Note: If you think you might ever create objects in both RPG and COBOL, set the model value YHLLVNM to *RPGCBL.

If your model was initially created with the naming convention set to *RPGCBL, skip steps 2, 3, and 4.

1. Replace the generation type data in your model library with the COBOL generation types. To replace the data, enter:

```
CPYF +
  FROMFILE(2E-NL-library/YGENTYPPDP) +
  TOFILE(your-model/YGENTYPRFP) +
  FROMMBR(CBL) TOMBR(*FIRST) +
  MBROPT(*REPLACE)
```

   To find out the National Language library (2E-NL-library) name, enter:

```
DSPDTAARA +
  DTAARA(2E-product-library/YLNGxxxSYA)
```

1. Replace the device format data in your model with COBOL formats. To replace the data, enter:

```
CPYF +
  FROMFILE(2E-NL-library/YDEVFMTPDP) +
  TOFILE(your-model/YDEVFMTRFP) +
  FROMMBR(CBL) TOMBR(*FIRST) +
  MBROPT(*REPLACE)
```

1. For existing functions and files, run the Convert Model Names (YCVTMDLVNM) command before regenerating them in COBOL. This command changes existing names to valid COBOL names and creates a report of old names and corresponding new names. To rename function and file names, enter:

```
YCVTMDLVNM MDLLIB(your-model-library)
```

1. Add the COBOL library, Y2SYCBL, to your model and model job description library lists. Be sure to exit and save the changes and update the batch job description. To add the library, enter:

YEDTLIBLST LIBLST(your-model-library/*JOB)

1. Delete previously submitted items from the job list for Submit Create Requests from Model **(**YSBMMDLCRT**)**. You cannot change the source type of a function already on the list.

   For more information on deleting items from job lists, see the Using Job Lists section in this chapter.

2. Change the source type of existing functions to CBL. Enter the model and change the source type. Use either of the following methods to display the Edit Function Details panel:

   - Go to the Display Services Menu and select the Display all functions option. Zoom into each named object on the list.
   - On the Edit Model Object List panel specify option 2 for each function on the All Objects list (*ALLOBJ).

   Note: The model value YHLLCBL determines whether you generate COBOL/74 or COBOL/85.

3. Generate and compile as follows:

   - If the access path format names begin with @, the RPG default, generate and compile all access paths as well as functions. This changes the RPG @ prefix to a COBOL prefix.
   - If you have created a set of access paths with COBOL compatible names, generate and compile only the functions.

For more information:

- On implementation names, see the *Administrator Guide*.
- On CA 2E commands, see the *Command Reference Guide*.
- On recompiling physical files, see the chapter titled "Generating and Compiling Your Application" in this guide.
- On editing the All Objects list, see the chapter titled "Managing Model Objects" in this guide.

Converting from COBOL to RPG

This section covers converting a model you have implemented in COBOL to RPG implementation.

To convert a model to RPG:

1. If you plan to create files, fields, or functions, use the YCHGMDLVAL command to change the following model values:
   - Generation Type for new functions, using the command string:

   ```
   YCHGMDLVAL MDLVAL(YHLLGEN) +
     VALUE(*RPG)
   ```

   - HLL(s) naming convention for new names, using the command string:

   ```
   YCHGMDLVAL MDLVAL(YHLLVNM) +
     VALUE(*RPGCBL)
   ```

Note: If you think you might ever create objects in both RPG and COBOL, set model value YHLLVNM to *RPGCBL.

1. Delete previously submitted items from the job list for Submit Create Requests from Model (YSBMMDLCRT). You cannot change the source type of a function already on the list.
2. Enter the model and change the source type of existing functions to RPG.
   a. Go to the Display Services Menu and select the option Display all functions.
   b. Zoom into each named object on the list.
3. Generate and compile the functions.

For more information:

- On deleting items from job lists, see the Using Job Lists section in this chapter.
- On implementation names, see the Administrator Guide.
- On CA 2E commands, see the *Command Reference Guide*.

Chapter 4: Generating and Compiling Your Application

You must generate and compile the source members for your access paths and functions before you can test your application programs. This chapter covers generating the source for the DDS database files or SQL tables and views, RPG or COBOL programs, DDS display files or printer files, and Help text. It explains how to compile the generated source into objects that will implement the access paths and functions for your application programs. This chapter also covers common errors and regeneration/recompilation.

This section contains the following topics:

Requesting Source Generation

You can request batch or interactive source generation for access paths and functions.CA 2E automatically keeps a job list of the members to be generated and compiled in batch or to be generated interactively and not yet compiled.

Note: If you want to generate and compile in another HLL, you must change the function's attribute *before* you generate the function.

For more information:

- On setting up job lists, refer to the chapter titled "Preparing for Generation and Compilation" in this guide.
- On whether to generate in batch or interactive mode, refer to the chapter titled "Generation and Implementation: An Introduction" in this guide.
- On changing the HLL, refer to the chapter titled "Preparing for Generation and Compilation" in this guide.

Working from the Display Services Menu

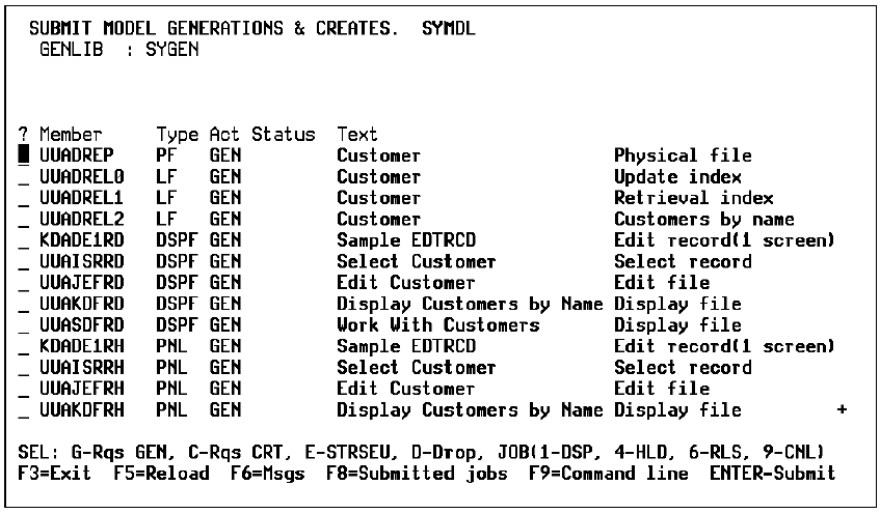
The steps below give you one way to generate specific source members, starting from the Display Services Menu:

1. From the Display Services Menu, display all access paths or functions; if you wish to generate:
   - Access paths, choose the option Display all access paths. The Display All Access Paths panel displays.
   - Functions, choose the option Display functions. The Display All Functions panel displays.

   Note: For functions, ignore items with **N/A under the GEN name column. They are internal functions and will not generate. You can filter these items out of your display by typing *EXT in the Type column, so that only external functions will display.
2. Request generation:
   - For batch, type J beside each item you want to generate, then press Enter.
   - For interactive, type G beside each item you want to generate, then press Enter.
3. Exit to the Display Services Menu. Press F3 (Exit).
4. Submit generations and compilations of all the source members you selected. On the Display Services Menu, you can do this in two ways:
   - Select the Submit model create request (YSBMMDLCRT) option. Press Enter to display the source members you selected or press F4 to change parameter defaults before displaying the list.
   - Select the Job list menu option to display the Job List Commands Menu. Select the YSBMMDLCRT option.

   A job list of the source members you have requested for generation and compilation appears on the Submit Model Generations & Creates panel.

   Review the list before confirming, make any changes desired, and press Enter**.** If the list contains items you do not want, you can drop (D) or hold (H) them. The following panel shows a sample list:


SUBMIT MODEL GENERATIONS & CREATES. SYMDL
GENLIB : SYGEN
? Member Type Act Status Text
UUADREP PF GEN Customer Physical file
UUADREL0 LF GEN Customer Update index
UUADREL1 LF GEN Customer Retrieval index
UUADREL2 LF GEN Customer Customers by name
KDADE1RD DSPF GEN Sample EDTRCD Edit record(1 screen)
UUAISRRD DSPF GEN Select Customer Select record
UUAJEFRD DSPF GEN Edit Customer Edit file
UUAKDFRD DSPF GEN Display Customers by Name Display file
UUASDFRD DSPF GEN Work With Customers Display file
KDADE1RH PNL GEN Sample EDTRCD Edit record(1 screen)
UUAISRRH PNL GEN Select Customer Select record
UUAJEFRH PNL GEN Edit Customer Edit file
UUAKDFRH PNL GEN Display Customers by Name Display file
SEL: G-Rqs GEN, C-Rqs CRT, E-STRSEU, D-Drop, JOB(1-DSP, 4-HLD, 6-RLS, 9-CNL)
F3=Exit F5=Reload F6=Msgs F8=Submitted jobs F9=Command line ENTER-Submit

To drop or hold items:

- If you hold any part of a function, such as a program, a device, or help, you must hold all the parts.
- If you drop any part of a function, also drop the function itself. Generation will fail for a function that uses a device if you do not generate the function and device together.

After you press Enter on the Submit Model Generations & Creates panel, the panel re-displays with the confirm prompt set to Y for confirmation.

5. Confirm the list. Press Enter.

   CA 2E submits the generation/compilation jobs. Messages display at the bottom of the panel to let you know what work is under way, such as:

   - "Job YGENSRC being prepared." YGENSRC is the generation job.
   - "Existing objects are being deleted."
   - "Joblist successfully processed."

6. You can review the list or exit the panel for the status of members as follows:

   - Refresh for the most current status by pressing F5.
   - If you want to exit, press F3. This takes you to the Display Services Menu.
   - To go to the i OS Work with Submitted Jobs (WRKSBMJOB) panel from the Display Services Menu, press F8.

Important! CA 2E orders generation of physical files, logical files, and functions automatically. If you disturb the order by moving dependent items among queues or deleting entries, the job may fail. The logical files must be built over physical files. Therefore, the physical files must be created first.

Each member on the job list initially has a *SBM (submitted) status. As CA 2E generates a member, the status changes to *GENSRC (source member being generated), *ACTIVE, or *JOBQ (source submitted for compilation). A source member no longer appears on the list when its compilation is completed unless the compilation fails and the status changes to *ERROR. For errors, you must resubmit both the display file and RPG or COBOL members.

For more information:

- On generating access paths, refer to *Building Access Paths*.
- On generating functions, refer to the chapter titled "Generating and Compiling" in *Building Applications*.

Using YBLDJOBLST to Submit Jobs

Another way to submit functions and access paths for generation/compilation is to use the Build Job List (YBLDJOBLST) command before running the YSBMMDLCRT command.

For more information on the YBLDJOBLST command, refer to the CA 2E *Command Reference Guide.*

Converting Condition Values

If you have not generated from this model before, or if you have changed or added condition values for status fields since the last generation, you must run the Convert Condition Values (YCVTCNDVAL) command to update the condition values list file. This command moves the values you defined against status fields from the model library to the generation library.

To convert condition values:

1. Choose the option Convert model data menu from the Display Services Menu.
2. Choose the option Convert condition values to database file from the resulting menu

Note: The condition values file should not be in use when running the YCVTCNDVAL command.

For more information:

- On steps for converting condition values, refer to the Enabling Execution Environments section in this chapter.
- On converting condition values for multi-language support, refer to the Changing a Model Language section in the "National Language Support" chapter in this guide.

Generating Your Field Reference File

CA 2E provides the option of using a field reference file, applicable only to DDS, when generating source for an access path and a display file. Because the internal files in CA 2E act as a field reference file, there is not really any need for this external field reference file for CA 2E-generated applications. However, it can be useful when a non-CA 2E application uses CA 2E-defined fields.

If the model uses a field reference file, the DDS for files are generated to use field referencing from this central definition of fields rather than specifying full field definitions in the source.

Before you can generate a field reference file:

- Verify that model value YFRFVNM specifies a name. You must have a field reference file specified for your model. If model value YFRFVNM is *NONE, you cannot generate this file. You can change this setting from *NONE to generate the field reference file and change it back to *NONE provided you do not generate any other files in the interim. Changing model value YFRFVNM has no effect on the model. To change the value of YFRFVNM, use the YCHGMDLVAL command.
- The field reference file must be in a library on the library list.

To generate DDS source for a field reference file, from Display Fields, choose one the following:

- For batch generation, press F21.
- For interactive generation, press F9.

Note: If you are adding a field reference file to an existing model, you must generate its source before submitting functions for generation/compilation.

Two other model values are available for generating a field reference file.

- **YFRFTXT**—Specifies which file text to use to describe the field reference file.
- **YFRFPFX**—Specifies which prefix to give fields in the field reference file.

For more information on model values and the YCHGMDLVAL command, refer to the CA 2E Command Reference Guide.

Enabling Execution Environments

CA 2E provides the option of using a field reference file, applicable only to DDS, when generating source for an access path and a display file. Because the internal files in CA 2E act as a field reference file, there is not really any need for this external field reference file for CA 2E-generated applications. However, it can be useful when a non-CA 2E application uses CA 2E-defined fields.

If the model uses a field reference file, the DDS for files are generated to use field referencing from this central definition of fields rather than specifying full field definitions in the source.

Before you can generate a field reference file:

- Verify that model value YFRFVNM specifies a name. You must have a field reference file specified for your model. If model value YFRFVNM is *NONE, you cannot generate this file. You can change this setting from *NONE to generate the field reference file and change it back to *NONE provided you do not generate any other files in the interim. Changing model value YFRFVNM has no effect on the model. To change the value of YFRFVNM, use the YCHGMDLVAL command.
- The field reference file must be in a library on the library list.

  To generate DDS source for a field reference file, from Display Fields, choose one the following:
- For batch generation, press F21.
- For interactive generation, press F9.

  Note: If you are adding a field reference file to an existing model, you must generate its source before submitting functions for generation/compilation.

Two other model values are available for generating a field reference file.

- YFRFTXT—Specifies which file text to use to describe the field reference file.
- YFRFPFX—Specifies which prefix to give fields in the field reference file.

For more information on model values and the YCHGMDLVAL command, refer to the CA 2E Command Reference Guide

-

Field Condition Values for Status Fields

Field condition values are values end users can enter in input-capable status fields on a panel. To make condition values available to users of your application when they request prompting, you must convert them from the model library where you define them to the generation library where they will be used. The conversion process creates a database file in the library you specify and, if needed, a display file and display program for showing the values on the file. When users prompt a status field that has a check condition,CA 2E displays a list of allowed values.

Note: You can change the names of the display file, display program, and condition values file using the model value YVLSPFX.

For more information

- On converting condition values, see the YCVTCNDVAL section of the *Command Reference Guide*.
- On model value YVLSPFX, see the YCHGMDLVAL section of the *Command Reference Guide*.

Converting Field Condition Values

To convert condition values:

1. Make sure the file that contains your condition values, xxVLLSP (where xx is the value list prefix), is not in use.
2. Invoke the YCVTCNDVAL command from the Display Services Menu or enter it on an i OS command line as follows:

```
YCVTCNDVAL MDLLIB(model-library) +
  GENLIB(generation-library)
```

Note: Be sure to use the appropriate library list containing the generation library into which you want to convert your condition values; that is, use the appropriate model library list.

1. Press Enter to execute the YCVTCNDVAL command.

CA 2E converts the values to a database file. If you invoked the command from the Display Services Menu, CA 2E returns to that menu. The following message displays:

Condition values from model *model name* converted to library *library name*

Converting Condition Values in a Multi-model Environment

There are two ways to convert condition values in a multi-model environment with a common generation library:

- Assign unique prefixes to the condition values list file in each model by using the YCHGMDLVAL command for model value YVLSPFX. When you convert the conditions,CA 2E creates separate condition files and condition display programs for each model.
- If you want to create only one condition file and display program in the common generation library, put all conditions into a common model before conversion. To create a common model, use the Copy Model Object (YCPYMDLOBJ) command.

For more information on multi-model environments, refer to the chapter titled "Setting Up a Multi-Modeling Environment" in the *Administrator Guide*

Converting Model Messages

You can recreate all the message descriptions from a model in a single step using the Convert Model Messages (YCVTMDLMSG) command.

There is generally no need to run this command because changes to message functions in the model are automatically applied to the associated message file in the generation library.

Two reasons you might use this command are:

- To merge the message files for two applications
- To create messages in a separate library

Note: If you want to use the CA 2E shipped default messages, such as *No value selected, in an environment that does not include the CA 2E product libraries, use the Duplicate Application Objects (YDUPAPPOBJ) command.

CA 2E-shipped default messages reside in the message file, Y2USRMSG. User-generated run time messages reside in the file specified by the Message File Name (YMSGVNM) model value. The YCVTMDLMSG command creates both message files automatically if necessary.

For more information on CA 2E commands, refer to the CA 2E *Command Reference Guide*.

Verifying Results

CA 2E sends completion messages for successfully generated source. If errors occur, CA 2E:

- Places a comment line in the generated source with E in column 6
- Flags the item on the job list with *ERROR

For compilation errors, you can look at the spooled file.

The following sections cover finding errors for action diagrams, generation, and compilation. You can display a full explanation of the error using the i OS Display Message Description (DSPMSGD) command. This command looks up the message description of the message in the message file Y2MSG. For example, you might get the following message:

```
Columns  . . . :  1 71          Edit            MYGEN/QDDSSRC
SEU==>                                                APPLSPN

0084.00   *A. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
0085.00   A* Key fields
0086.00   E* Y2V0124 - No keys specified for format 'Account History'
0087.00   A*===========================
```

You would look up message Y2V0124 by typing in the following command string and pressing Enter:

```
DSPMSGD RANGE(Y2V0124) MSGF(Y2MSG)
```

For more information on common errors and recommended actions, refer to the appendix "Troubleshooting.”

Finding Errors Before Generation

You can find errors in the action diagram that would cause generation errors by using the Action Diagram Services panel.

The action diagram editor finds errors such as:

- Missing parameters
- Invalid context
- Invalid domains
- Undefined conditions
- Undefined actions

Note: On the display of errors, you can press F7 to scroll to the next error, and type 3 for the ‘Occurrences to process’ option to scroll to the prior error.

The action diagram editor only finds syntax errors in the current action diagram. It does not find any errors present in embedded internal functions, unless you zoom into those functions first. The action diagram editor does find errors in hidden structures within the same action diagram.

To position to a function or message within an action diagram having a specified source member name or message identifier, respectively, type the implementation name in the ‘Scan for implementation name’ option.

For more information on the Action Diagram Services panel, refer to the Using Action Diagram Services section of the "Modifying Action Diagrams" chapter of Building *Applications*.

Error analysis capability is also available outside the action diagram as follows:

- The new Check Function Action Diagram (YCHKFUNACT) command processes a list of model objects and produces a listing of functions that contain errors. For functions containing errors, the Option parameter specifies whether to print a report (*PRINT) or load the action diagram of the first function containing an error (*EDIT). The action diagram is positioned at the first block that contains an error.
- Subfile option 38 on the YEDTMDLLST panel scans for errors in the selected function. If any errors are found, the action diagram is loaded and positioned to the first error.

Finding Errors After Generation

To find errors that occurred during either batch or interactive generation:

1. Access the Source Entry Utility (SEU). On the Submit Model Generation & Creates panel, place E in front of the program object in error.
2. Enter an 'E* Y2' search string. On the SEU positioner line, type 'E* Y2' and press F16. The editor goes to the line beginning with E* (message ID) on which the error is located in the compiled source. Press F16 again to continue the search.
3. Look for the prior user point in error. User points are identified by the USER comment line. You can use F16 (find) and the string USER to locate the incomplete action diagram statement(s) that caused the error(s).
4. Exit SEU and return to the function's action diagram to make the needed corrections.

For more information on the SEU editor, refer to IBM's *Application System/400 Application Development Tools: Source Entry Utility User's Guide and Reference*.

Interactive Generation Errors

If you need more information, you can look for interactive generation errors as follows:

1. To look at the generation source file, YGENSRCRFP, in QTEMP, use the Toolkit Work with Database File Data (YWRKF) command. Enter:

YWRKF FILE(QTEMP/YGENSRCRFP)

1. Scroll through the file to display the point in the action diagram where generation stopped.
2. Return to the action diagram to make the needed corrections.

For more information:

- On the YWRKF command, refer to the CA 2E Toolkit Reference Guide.
- On action diagram editing, refer to the chapter titled "Modifying Action Diagrams" in Building Applications.

Finding Errors After Compilation

To find compilation errors, display the compiled listing and job log, using one of the following approaches.

From the Display Services Menu

To access the job log from the Display Services Menu:

1. Press F8 (Display submitted jobs).
2. On the Work with Submitted Jobs panel, type 8 (Spooled files) next to the job, and press Enter. The job will have the same name as the associated source member.
3. On the Work with Job Spooled Files panel, type 5 (Display) next to both the compile listing and job log. Press Enter.
4. On the job log, scan for errors of severity 30 or above. This may show you what the problem is. If not, go to the compile listing.
5. On the display of the compile listing, type B (bottom) in the Control field at the top of the screen. Press Enter.
6. Note the message ID for the error message.
7. Use F16 (find) for that ID (without the *) to locate the action diagram statement in error.

Display the Compile Listing

To browse the compile listing using SEU:

1. Access SEU. On the Submit Model Generation & Creates panel, place E next to the program object in error. Press Enter.
2. Select copy/browse mode. Press F15 (Browse options).
3. Display the compile listing. Select Option 2 (Spool file) and press Enter.

   The compile listing displays at the bottom of the split panel.
4. Find each error. Enter *****ERR on the find line at the bottom of the panel and press F16 (Find/Repeat find).
5. Press Help on the message displayed at the bottom of the SEU display.

You can then revisit the action diagram to correct the error(s).

For more information on using SEU, refer to IBM's *Application System/400 Application Development Tools: Source Entry Utility User's Guide and Reference*.

Using Job Logs

For your initial review of the job log, look for:

- What the library list entries are when the problem occurs.
- The error message ID and its text

Resetting Job Log Severity Level

You can set the logging level for interactive and batch jobs as follows:

- For an interactive job, if the logging level is not low enough to isolate the problem, change the level and resubmit the job. To lower the job log severity level from an i OS command line, execute the i OS Change Job (CHGJOB) command. Suggested parameters are:

  CHGJOB LOG(4 00 *SECLVL) LOGCLPGM(*YES)

- For a batch job, you can change the logging level for the job description (JOBD) using the i OS Change Job Description (CHGJOBD) command. Suggested parameters are:

  CHGJOBD +
    JOBD(library-name/job-description- name) +
    LOG(4 00 *SECLVL) LOGCLPGM(*YES)

For more information on the CHGJOB command, refer to IBM's *Application/System 400 Programming: Control Language Reference*.

Accessing an Interactive Job Log

If necessary, you can access an interactive job log. From an i OS command line, type one of the following and press Enter:

```
DSPJOBLOG

SIGNOFF LOG(*LIST)
```

For more information on the DSPJOBLOG command, refer to IBM's *Application/System 400 Programming: Control Language Reference*.

Working with the Output Queue

Use the i OS Work with Output Queue (WRKOUTQ) command to access the output queue for the spool files. This queue is the OUTQ on your JOBD:

1. Type the following and press Enter:

   WRKOUTQ OUTQ(library name/output queue)

2. Find the job (name of the program) and use option 5 (Display) to display the spool file.

For more information on the WRKOUTQ command, refer to IBM's *Application System/400 Programming: Control Language Reference*.

Debug Aids

The Toolkit provides utilities to streamline debugging. These utilities are a set of commands that allow you to change the data on a database file directly, set up debug sessions from directives stored in program source, and take synchronized snapshots of the contents of a list of files. These commands include:

- Work with Database File Data (YWRKF)
- Start Debug and Add Auto Breakpoints (YSTRDBG)
- Copy Files (YCPYF)
- Set Break Program (YSETBRKPGM)
- Display a Program Message Queue (YDSPPGMQ)

The CA 2E documentation utilities are also useful in debugging. You can use them to identify implementation objects impacted by changes you make as a result of debugging.

Note: If you use the i OS debug facilities, ensure that you end debug before entering the model.

For more information:

- On how to use the debug commands, refer to the chapter titled "Debug Aids" in the CA 2E Toolkit Concepts Guide.
- On parameters for the debug commands, refer to the CA 2E Toolkit Reference Guide.
- On using CA 2E documentation utilities, refer to the Documenting Your Generated Application section of the "Implementing Your Application" chapter of this guide.

Generating and Compiling After Changes

This section describes considerations for regeneration and recompilation, and the facilities provided by CA 2E for your assistance.

Impact Analysis

You can use CA 2E's impact analysis tools to determine the impact of a proposed or actual change to a model object and to ensure that all dependent objects are edited, regenerated, and compiled. These tools include:

- Automatic update of date and time for various processes, including creation, change, copy (for example, import from another model), and generation, for each model object.
- Commands and processes to identify dependent model objects. These include:
  - Multi-level model object usages
  - Multi-level model object references
  - Simulation of a proposed change
  - Component change processing
- Distinction between changes that require editing of using objects and changes that are internal and require regeneration of using external functions and access paths.
- Full integration with CA 2E edit and generation facilities.

For more information on CA 2E impact analysis tools, refer to the Impact Analysis section of the chapter titled "Managing Model Objects" in this guide.

The remainder of this section provides a general overview of the idea of dependencies between model objects.

What to Generate/Compile When You Change a Model Object

If you change a model object after generation and compilation, you must generate and compile the dependent access paths and functions. What needs to be compiled depends on what you change.

For more information on how changes to a model object affect other types of model objects, refer to the Impact Analysis section of the "Managing Model Objects" chapter in this guide; the appendix titled "Change Control Facilities Reference Tables" in this guide; and the YDSPMDLUSG and YDSPMDLREF sections of the CA 2E *Command Reference Guide*.

Changes Requiring Generation/Compilation

Following is an overview of dependent model objects that require generation and compilation as a result of changing various model object types:

If You Changed	You Must Generate and Compile
Function options	The function
Device design	The function
Action diagram	The function **Note:** For a non-generated (internal) function, all functions referencing that function must be generated/compiled.
Function message	If parameters in a message are changed, all functions using the message. If no parameters have changed, no generation or compilation is needed.
Function parameters	The function and any other functions that use it. **Note:** If you changed or added parameters, you must first revisit each reference to the function in the action diagram, supplying the new parameters. Use Find Error within an action diagram to find where the new parameters are missing.
Access paths	All functions using the access path and all associated access paths. **Note:** Use function references (F next to an access path).
Condition value	All functions referencing the condition value. **Notes:** If the condition value is used in select/omit logic on an access path, generate/compile that access path. If only the condition value is changed, you do not need to generate/compile the function(s) using the access path. If you change the values list prefix, you must generate/compile all programs that call the values list display program

If You Changed	You Must Generate and Compile
YDATGEN model value	Compile all functions. **Note:** Set YDATGEN to VRY (display/enter date in MDY), then you can make changes at execution time.
Relation	If the relation is changed: All access paths and all functions using those access paths. All access paths that include the changed relation as a foreign key. For example, if the key of a referred to file changes, the files that refer to the relation are affected. This, in turn, triggers function changes. All files, all access paths, and all functions using those files. If the relation is added or deleted, all files using the relation, all access paths, and all functions using those files.

The only time you do not need to regenerate access paths and functions following a change to a file (FIL) is if the access path does not include the changed relation.

The following table shows some examples of changes to files that determine whether you need to generate and compile functions.

	FLD1 FLD2	3.0 2	FLD1 FLD2	3.0 2
	FLD1 FLD2	3.0 4	FLD1 FLD2	3.0 2
	FLD1 FLD2	3.0 2.0	FLD1 FLD2	3.0 2
	FLD1 FLD2	3.0 2	FLD1 FLD2 FLD3	3.0 2 5
	FLD1 FLD2 FLD3	3.0 5 2	FLD1 FLD2	3.0 2

Finding Where CA 2E Objects are Used

The CA 2E usage and reference facilities and several Toolkit convert commands help you identify which objects are used by the object you are looking at, and which objects you need to compile with this one.

Model Object Usages

The CA 2E Where Used facilities within your CA 2E model tell you where a model object is used. You can work with model usages as follows:

- From the Edit Model Object List panel, enter option 91 for a specific model object or press F20 to display the Display Model Usages panel for the model object list you are editing.
- Enter the Display Model Usages (YDSPMDLUSG) command at a command line.
- Enter U against an object where this option is available to display level 1 usages.

In each case, the Display Model Usages panel displays. This panel provides a variety of controls and filters including recursion, scope, and positioning to aid you in analyzing the effect of an actual or proposed change.

When you request usages for a model object,CA 2E displays a list of model objects that use it. Following are examples of model usages according to model object type:

- **Physical (PHY) access path type**—All references to the access path by other non-physical (built over or joined to) access paths
- **Other access path types**—Physical access path(s) to which the access path refers
- **Function**—All functions that reference the function
- **Field**—Files, functions, and other fields that refer to the field
- **Condition**—In an access path to specify selection criteria, in an action diagram, for a field to control the validation and default values
- **File**—Owning file and application area
- **Array**—All functions based on the array or using it for parameter definition

For more information on model usages by model object type, refer to the YDSPMDLUSG section of the CA 2E *Command Reference Guide*.

From the Display Model Usages panel you can request generation/compilation of any of the listed items. You can also edit details such as access path attributes for access paths or action diagrams for functions.

For example, to find out which access paths are dependent on the physical file access path of a specific file in a database:

1. From Edit File Details for the file, enter U against the PHY access path. The Display Model Usages panel displays, listing:
   - Access paths built directly over the specified file
   - Any access path with file-to-file relations to the specified file that result in a join
2. Find where the listed access paths are used. Enter 91 against each access path.

3. Find the functions that use an access path. Enter F against each access path. The Display Model Usages panel displays.

Model Object References

You can also display a list of the model objects a model object refers to from the Display Model Usages panel or Edit Model Object List panel using option 81 or by entering the Display Model References (YDSPMDLREF) command at a command line.

For more information on usages and references, refer to the Impact Analysis section of the "Managing Model Objects" chapter in this guide.

Finding Unreferenced Model Objects

The Document Unreferenced Objects (YDOCURF) command produces a model object list of model objects that are unreferenced in a specified model. The OBJTYP parameter specifies up to six special values that select the object types to be analyzed by the command.

Note: A model object that is identified by the YDOCURF command as unreferenced within a model may be referenced from outside the model by menus, messages, user defined programs, and so on.

For more information on the YDOCURF command, see the *Command Reference*.

Toolkit Convert Commands

Two Toolkit commands, Convert Database Relations (YCVTDBR) and Convert Program References (YCVTPGMREF), allow you to list items you want to compile as follows:

- **YCVTDBR**—Lists files dependent on physical file(s). You can compile any of the listed files using the Toolkit Create Object (YCRTOBJ) command.
- **YCVTPGMREF**—Lists programs that reference a file or format. To compile, first you convert specified items using the Toolkit Convert Object List (YCVTOBJLST). You then run the YCRTOBJ command. The YCVTOBJLST command creates the member list used by the YCRTOBJ command.

For more information on the parameters for these commands, refer to the CA 2E *Toolkit Reference Guide*.

Multi-Programmer Environments

To be able to compile objects that another programmer or designer created, you must be authorized to those objects. One way to achieve this is to use an appropriate group profile.

For more information on setting up authority, refer to the Signing on with the Correct User Profile section of the "Creating and Managing Your Model" chapter of the *Administrator Guide* and the Locking Objects and the Open Access sections of the "Using Your Model" chapter of the *Administrator Guide*.

Retaining Data When You Recreate Physical Files

You can compile a physical file that contains data without losing the data.CA 2E provides model values to name the libraries.

- **YOLDLIB**—Name of the library into which you want old physical files archived.
- **YCPYLIB**—Name of the library from which you want physical file data copied during the generation process.

Note: Usually, these two model values are the same because you store data in a library and then copy the data back from the same library.

To compile, retaining the data in a physical file:

1. Before you compile the physical file, designate library names for model values, YOLDLIB and YCPYLIB, using the YCHGMDLVAL command. Type the following command strings, pressing Enter to execute each one:

   a. YCHGMDLVAL MDLVAL(YOLDLIB) +
      VAL(old-library-name)

   b. YCHGMDLVAL MDLVAL(YCPYLIB) +
      VAL(copy-library-name)

2. When you execute the YSBMMDLCRT, specify the libraries for OLDLIB and CPYLIB parameters.CA 2E will use your library names as defaults. The respective values are:

   - ***OLDLIB**—Library name from YOLDLIB.
   - ***CPYLIB**—Library name from YCPYLIB.

Note: When you execute the YSBMMDLCRT command from the Display Services Menu, the default for both *OLDLIB and *CPYLIB is *NONE.

Chapter 5: Implementing Your Application

Once your application is compiled, you can set it up for end users. This chapter offers guidelines for implementation.

This section contains the following topics:

CA 2E Toolkit Menus

You can create menus for end user access to your application with Toolkit menu facilities. When you create menus or make changes, additions, and copies, the menus are ready to use immediately; no compilation is needed.

This topic offers some step-by-step guidelines.

For more information on menu facilities, refer to the Menus section of the "User Access Aids" chapter of the CA 2E *Toolkit Concepts Guide*.

Creating and Maintaining Menus

To create or change a Toolkit menu, perform the following:

1. From any command line, enter:

YGO *Y1.

   The Toolkit Utilities Main Menu displays.

1. From the Toolkit Utilities Main Menu, enter the Design Aids option.

   The Toolkit Design Aids menu displays.

2. From the Toolkit Design Aids menu, enter the Menu Commands option.

   The Toolkit Menu Commands menu appears.

3. From the Toolkit Menu Commands menu, enter the YWRKMNU Work with Menus option.

   The Work with Menus panel displays.

4. From the Work with Menus panel:

   a. In the Menu name field, enter the name of your menu.

   b. In the Library name field, verify that the library is the name of your generation library (GENLIB).

   c. Press Enter. The Work with Menu Details panel displays.

5. To insert lines on your menu, from the Work with Menu Details panel enter the Insert Mode option.

   While in this mode, Toolkit inserts a blank line each time you press Enter.

6. Enter the information you want:

   - **Prefix**—A word or two that categorizes options (optional).
   - **Opt**—The character or number the user enters to select the menu. N in this field automatically numbers the options.
   - **Description**—Title as you want it to display on the menu.

7. Press Enter twice to end insert mode.

8. Type Z (Details) against each menu option you have defined.

   The Work with Menu Options panel displays for the first option.

9. From the Work with Menu Options panel, change any defaults as needed. You can invoke the command prompter by pressing F4 (Prompt), typing Z (Details) against any option, and pressing Enter.

   Notes:

   - For Option type choices:
     - PGM allows use of ? (question mark) for the name of a program.

– CMD requires that you enter an IBM i command in the Option to be executed field.

- In the Library field, entering *MDL enables the menu to list all functions in the model. From here, you can select one. This automatically sets the parameter list for the function.
- In the Option to be executed field, if you enter a function,CA 2E automatically adds a return code parameter.

10. Exit and save:
    a. From Work with Menu Options, press F3 (Exit) twice. The Work with Menu Details panel displays, followed by the Exit Work with Menu panel.
    b. From Exit Work with Menu, enter the option Exit and replace menu.

       The Toolkit Menu Commands menu displays.

Displaying Your Menu

To display the menu you have just created, perform the following:

1. From the Toolkit Menu Commands menu, key in the option YGO Go to Menu, and press F4 (Prompt).

   The Go to Menu (YGO) panel displays.

2. From Go to Menu (YGO):
   a. In the Menu name field, enter the name of the menu.
   b. In the Library name field, enter your GENLIB name.

3. On the display of your menu, you can:
   - Execute compiled programs.
   - Edit the menu by typing *M (the Work with Menu Details panel displays.)

Note: To be able to execute Toolkit menu options, you must be authorized to the program or command that the option calls.

For more information on setting up user profiles and passwords for Toolkit menu access, refer to the User Profiles section of the User Access Aids chapter of the CA 2E *Toolkit Concepts Guide*.

Setting Up Color Menus

You can specify color menus from Go to Menu (YGO):

1. On a command entry line, type:

   YGO

   and press F4 (Prompt). The Go to Menu (YGO) panel displays.

2. For User options, select *EXTENDED, then press Enter.

Calling a Program

You can execute a compiled program using:

- Toolkit menus.
- The CA 2E Call a Program (Y2CALL) command. This command determines the parameters required by an external function directly from details contained in the model. You can provide values for all input-capable fields and you can re-use these values for subsequent calls.
- This command is useful especially when the parameter interface is complex or if it has changed.
- The ‘Call function’ option on the Action Diagram Services panel.
- The i OS CALL command:

CALL program ' '

where:

- **Program**—Is the name of the source member for the function's program.
- **' ' (single quoted space)**—Is a required dummy parameter that represents the standard return code parameter.

Note: Other parameters may be required. How these are passed depends on how they are defined in the function.

For more information:

- On modifying function parameters, refer to *Building Applications*.
- On calling functions from the Action Diagram Services panel, see the Using Action Diagram Services section of the "Modifying Action Diagrams" chapter in *Building Applications*.
- On the Y2CALL command, refer to the Working with Versions of Functions and Messages in the "Managing Model Objects" chapter in this guide, and the *Command Reference Guide.*

To find the name of the program you want to execute, use one of the following methods:

- For the CALL and Y2CALL commands, go to the Display Services Menu and use the option, Display all functions. Note the source member name of the program you plan to execute.
- From the Toolkit Work with Menu Options panel, enter the following values:

Option	Value
Option type	PGM

Option to be executed	?
Library	*MDL

A selection list of functions displays. Note the source member name (DDS name) of the function you want to execute.

Note: All CA 2E generated external functions have at least one parameter, a return code. In the following logic, checking is being done to show the program ended normally:

```
.—AACASE
 | - PGM.*RETURN CODE is *NORMAL
 | - *OTHERWISE
 |   error handling
 |__ENDCASE
```

Execution Environments

This section gives guidelines for preparing your application for execution.

Duplicating Shipped Application Objects

You can run your application independently of the CA 2E product libraries. To run your application without the product libraries in your library list or to distribute your application to another system, copy shipped application objects into the library for execution objects.CA 2E provides the Duplicate Application Objects (YDUPAPPOBJ) command for this purpose. The YDUPAPPOBJ command also provides options to copy the called programs needed to support action bars, help, and Toolkit menus.

Duplicating Execution Objects

To duplicate generated execution time objects into a non- CA 2E environment, use the Create Generation Objects (YCRTGENOBJ) command. You can optionally invoke the YDUPAPPOBJ command from the YCRTGENOBJ command.

For more information

- On CA 2E commands, see the *Command Reference Guide*.
- On the list of objects required to run the menu and help utilities, refer to the appendix titled " CA 2E Objects Required for Compilation and Execution" in this guide.

Testing

As CA 2E provides the structure for your application, your testing can focus on any logic you have added to action diagrams and the relations between functions. In developing your test plan, be sure to include:

- Test data with known input and output.
- Steps for regression testing.

Before You Begin

Before testing, verify:

- That the application's message file, or a copy of it, is present in a library in the application's library list.
- That all required CA 2E execution modules are present. This is automatically true if you are testing in the same environment in which you developed the application.

For more information, refer to the Execution Environments section in this chapter.

What to Test

Critical details to test in your application:

- Do function calls work?
  - Can every function be called?
  - Can every function that is supposed to call another function do so?

  Note: You can set up a menu, using Toolkit menu facilities, to test function calls.
- Do all the function keys work? If a function key is enabled but does not show on the panel, or does not work and does show on the panel, go to the device design and select the command to refresh the function keys from the action diagram.
  - Ensure the YCUAPMT model value is set to *YES or *CALC.
  - Does the F4=Prompt work for fields in which values should be checked?
  - If F4 does not work, the check values entry for the field may specify the default *NONE. Set the value to *ALL values or an appropriate LST condition.
  - If F4 works but you do not get the right list of values, run the Convert Condition Values (YCVTCNDVAL) command.
  - If F4 in a key field does not give you a Select Record (SELRCD), the SELRCD probably did not exist when you generated the function. You can create the SELRCD and regenerate the application that needs it.
- Are before/after image errors occurring? The file definitions in the model may differ from the compiled versions of the file. Regenerate the files and the functions built over those files.
- Are level checks occurring? Files may have been changed but the functions were not regenerated and compiled successfully.
- Is no data being returned for a Retrieve Object (RTVOBJ) function? You may need to code a *MOVE ALL built-in function from a Database File 1 (DB1) into a Parameter (PAR) context.
- Is an error occurring although the program continues? The program is not properly checking for and handling a return code.

For more information:

- On the SELRCD and RTVOBJ functions, refer to *Building Applications*.
- On YCVTCNDVAL, see the *Command Reference Guide*.

Moving Objects

To carry out testing in a different library than development, move lists of new versions of programs and their source into the test environment. There are two approaches for moving objects with CA 2E facilities.

- Change Management (CM)
- Toolkit generic move commands

Note: Before moving objects and source members, be sure to convert condition values.

CA 2E CM Overview

CM, an automated change management system for the IBM i, includes features that control access and modifications to CA 2E model objects, source code, and executable objects. CM automates and tracks the flow of source and objects between development, test, and production libraries. You define the libraries, including the rules by which you want CM to govern them. CM also implements your IBM i and model object security.

CM's automatic Check Out of model objects that you intend to change protects against conflict with the work of other developers.

The CM Check In feature supports moving objects. This feature includes the following processes:

- **Create Request**—Determines the source and objects for migration, maintaining integrity between CA 2E design objects and source-based objects.
- **Precompile**—Promotes the model objects.
- **Compile Request**—Generates and compiles the requested source-based objects.

  Note: Compile Request is an optional process. You can move objects using CM that were compiled with CA 2E.
- **Move Request**—Moves successfully compiled objects to the target library, including the related logical files for physical files.
- **Distribute Request**—Distributes programming changes to a remote or local target library.

For more information on CM, refer to the *Change Management User Guide*.

Toolkit Generic Move Commands

Move commands have generic versions in Toolkit, which move the source for objects. These generic commands also allow you to log the movement of objects and source, to archive previous versions of source/objects, and to preserve object authorities. You can move all objects or selectively move only new or existing objects.

Note: Toolkit protects your data by requiring an archive library when you move physical files.

The generic move commands include:

- **YMOVM**—Move Members. Use to move a list of source members, including archiving if specified.
- **YMOVOBJ**—Move Objects. Use to move a list of objects, including preserving authorizations if specified.
- **YMOVOBJSRC**—Move Objects and Source. Use to move a list of objects and corresponding source members, including preserving authorizations if specified.

For more information:

- On using the generic commands, refer to the CA 2E Toolkit Reference Guide.
- On an overview of generic moving, refer to the Generic Processing section of the "Programmer Aids" chapter of the CA 2E Toolkit Concepts Guide.

UIM Help Text

The native i OS User Interface Manager (UIM) provides support for defining and running panels, dialogs, and online help information. UIM is a tag-based language related to other tag languages, such as the OS/2 Information Presentation Facility (IPF) and ANSII Standard Generalized Markup Language (SGML).

UIM Panel Groups

CA 2E generation optionally supports UIM help. CA 2E creates the help text with control codes (tags) into a source member to be compiled into an object. This object is called a panel group (*PNLGRP). Each panel group contains help panels called help modules. If you generate UIM help, you must install the panel group, with the display and function, for production.

Before generating and compiling UIM help text for the first time, you must execute the YCVTCNDVAL command. This process will update the Y2USRMSG message file with the UIM message IDs that are used in the generated source.

Note: Because UIM is a tag-based language, do not start a line of narrative text with a period (.). The compiler will assume it is part of a tag and the compile will fail with error CPD5B41 - Control word is missing.

The help modules are referenced in the DDS display file source, using the HLPPNLGRP keyword. These are associated with specific areas on the panel using the HLPARA keyword.

When end users request help:

- i OS automatically selects and displays the appropriate help module, dependent on the cursor location.
- i OS support allows for automatic selection of an appropriate window size, shape, and location for the help, and automatically reformats the help text to fit.
- End users can access all help modules enabled for a format, either at the record or file level.
- Additional facilities are available, such as printing help text and accessing help for help.

Additional UIM tags are available for your use, including:

- Hypertext links to other panel groups by selection of specific fields on a panel
- Indexed lookup by word or synonym
- Display attributes, headings, and column and list numbering
- Import of modules from other panel groups at execution time

For more information on tailoring your help using additional UIM tags, refer to IBM's Application System/400 Guide to Programming Displays.

Note: If you have Text Management (TM) help text that you have modified outside of CA 2E, you can convert it to UIM help text by using the Convert Help Text to UIM Panel (YCVTTMUIM) command. You can then use the resulting panel group with existing functions without the need to generate again.

For more information on the YCVTTMUIM command, refer to the CA 2E Command Reference Guide.

Documenting Your Generated Application

To generate technical documentation, the Document Model Functions (YDOCMDLFUN) command uses the operational text (or functional text if no operational text is available) from your model.

- **Function text**—Creates an introduction to the function.
- **Device design text**—Creates an introduction to the panel.
- **Field text**—Creates a description of each field on the panel.
- **Condition text**—Creates a description of each allowed value for a field.

Note: The Edit Object Text panel initially shows functional text.

Two types of text are allowed for each CA 2E object:

- Functional text that the designer enters to describe the object's purpose; this text may include restrictions and notes on design decisions.
- Operational text that the developer enters to describe functions, panels, fields, and conditions for end users.

CA 2E includes commands to produce hard copy documentation of your model, including dependencies between objects. You can invoke these commands from the Display Services Menu or call them directly.

You can add functional or operational text using the Edit Object Text panel. For access:

- Type N (narrative text) next to the item you want to document. The Edit Narrative Text (Functional) panel displays.
- Press F20 (operational text) to go to the Edit Narrative Text (Operational) panel.CA 2E incorporates the text you enter on this panel into the help text for the generated function.

For more information

- On documenting your model:
  - For fields, refer to the chapter titled "Documenting Your Data Model" in the *Defining a Data Model Guide*.
  - For access paths, refer to the chapter titled "Documenting Access Paths" in the *Building Access Paths Guide*.
  - For functions, refer to the chapter titled "Documenting Functions" in *Building Applications*.

On Toolkit documentation commands, refer to the CA 2E *Toolkit Reference Guide* under commands that begin with the letters YDOC.

Chapter 6: National Language Support

CA2E offers you more than one way to generate applications in different national languages:

- If your whole application is to appear in one language, do the following:
  - Obtain the appropriate national language module from CA.
  - Translate your model.
  - Design each panel in the target language (refer to the Changing a Model Language section in this chapter).
- If your application is to appear in more than one language, such as an application for an international bank with clerks in different countries who access a central database, you will want to show different translated versions of each function.

This chapter describes how to generate in other national languages.

This section contains the following topics:

Understanding NLS Implementation

CA2E provides NLS support that allows applications generated by CA 2E to be language independent. This is done by removing hard-coded constants from panels and reports and placing them in message files. These message files can then be translated into several languages with the appropriate language resolved at execution time. This allows a single set of generated programs and panels to support multiple languages at execution time.

You can implement advanced NLS for a model, function, or field using external message identifiers. Using external messages, you can change from one national language to another, using the same set of application objects. Implementation of *MSGID at the field level would override implementation at the function level, as would function-level implementation override model-level implementation.

When you specify externalized screen constants for a model, function, or field,CA 2E converts screen text literals into messages (MSGID DDS keywords). There can be up to five messages for a literal, depending on the definitions: three for column headings, one for right-hand text, and one for left-hand text.CA 2E puts the messages in the message file specified by model value YPMTMSF at generation time. The generated application retrieves the messages from the message file at execution time.

Note: The default message file is xxPMTMSG in the generation library (GENLIB), where *xx* is the model object prefix defined by YCRTMDLLIB.

In order to facilitate maintenance of multi-language environments,CA 2E provides an exit program that automatically creates translated copies of prompt messages for each supported national language when you copy an NL-enabled device function.

For more information:

- On the exit program, see the Managing Multi-Language Environments section in this chapter.
- On device design, refer to the National Language Design Considerations section of the "Modifying Device Designs" chapter of *Building Applications*.

Translating a Generated Model

This section addresses tasks you need to do before generating your application in another national language. Once you have completed these tasks, refer to the chapter, "Generating and Compiling Your Application" in this guide as necessary.

Choosing Implementation Level

You choose the level at which to externalize constants before generating your application. Each level allows specific controls as follows:

- **Model level**—Full application control of the generating options. Set model value YPMTGEN to *****MSGID using the YCHGMDLVAL command.
- **Function level**—Control of the function. On the Edit Function Options panel in the Screen Text Constants field, enter M (MSGID).
- **Field level**—Flexibility for each constant on a panel or report. At Edit Screen Entry Details in the Screen Text field, enter M (MSGID).

The message ID allocation is a single character prefix followed by six generated characters (xYYYYYY). You can modify the single character prefix from Edit Generation Types.

Note: If you do not have the national language library for your generated language, you will need to translate the message file Y2USRMSG in your translated objects library after YCVTCNDVAL execution, CA 2E-generated help text, and any user-defined help.

Placing Language-Specific Objects in Libraries

After generating your application in the desired language, place the language-specific objects in separate libraries as follows:

- All application objects, such as programs, display files, and printer files, should go in one library
- Externalized, translated objects, such as messages, menus, and help text files, should go in several language-specific libraries

To change the language of an application from one language to another, change the library list to refer to the library that contains the language-specific objects.

Changing a Model Language

Using the CA 2E command Apply Translation to Model (YAPYTRNMDL), you can convert the default model language-specific objects from one language to another. This includes such objects as confirm prompts, function keys, shipped files, and conditions.

Note: Remember, once you convert a model to another national language using YAPYTRNMDL, the default model language is also changed. All subsequent generation will be in the new language.

YAPYTRNMDL does not convert user-modified data. If you use the same model for more than one language, you will need to maintain a set of the following files for each language:

- Condition values
- User messages
- Prompt messages (externalized message IDs)
- Help text
- Panel literals

You can change to another language by following the steps below.

To change the language of your created model:

1. Restore the language libraries Y1SYVnnn and Y2SYVnnn from the product tape, where *nnn* is the language-specific code.
2. If the following libraries exist on your machine, add both language libraries above them on the model job description library list.
   - Y1SY and Y2SY
   - Y1SYVENG and Y2SYVENG
3. Verify that the data area YLNGnnnSYA in library Y2SY exists and contains the name of the national language library Y2SYVnnn. If not, create the data area as follows:

   ```
   CRTDUPOBJ FROMLIB(YLNGENGSYA) +
       FROMLIB(Y2SY) OBJTYPE(*DTAARA) +
       TOLIB(*SAME) NEWOBJ(YLNGnnnSYA)
   ```

4. Change the value of the data area YLNGnnnSYA in Y2SY to Y2SYVnnn as follows:

   ```
   CHGDTAARA DTAARA(Y2SY/YLNGnnnSYA) +
       VALUE(Y2SYVnnn)
   ```

5. To convert default model language objects, run the YAPYTRNMDL command as follows:

   ```
   YAPYTRNMDL MDLLIB(model-library) +
       LNG(*nnn)
   ```

6. Enter the model to verify that the translation was successful.

7. Execute YCVTCNDVAL.

Note: The YAPYTRNMDL command does not change panel literals defined by Edit Database Relations or Edit Functions such as field names and panel titles. However, you can externalize these to messages for translation.

For more information on the YAPYTRNMDL command, refer to the CA 2E *Command Reference Guide*.

Translating User-Modified Data

To translate user-modified data:

- For database files, use one of the following:
  - Toolkit Work with Database File Data (YWRKF) command
  - IBM i file editing utilities
- For user messages, you can use IBM i message editing utilities.
- For Help text, use the Source Entry Utility (SEU)
- For condition names, you can use the YWRKF command on Y2VLLSP, created by the YCVTCNDVAL command. Translate only the user-defined condition names. You can drop all the fields on the file field, using the YWRKF command, except Condition text and External value. Translate the contents of these two fields to get titles and condition names in the desired language.

Generating Help Text

To generate default Help text in another language, place the product libraries for the language higher in the library list than the language from which it was originally generated.

Before generating, be sure to save the original Help text file.

The field names in the generated Help are the ones named in your model's database relations. You can edit the field names in the generated Help to match the column headings of screen fields.

Remember, you will need to translate any operational or functional text you have added.

Managing Multi-Language Environments

In order to facilitate maintenance of multi-language environments,CA 2E provides an exit program that automatically creates translated copies of prompt messages for each supported national language when you copy an NL-enabled device function. The exit program works whether the copy is within a model or between models using the Copy Model Objects (YCPYMDLOBJ) command.

Overview

An NL-enabled function is a device function in which screen or report literals are generated as external prompt messages, which can then be translated into other national languages. Prompt messages are stored in a prompt message file named by the YPMTMSF model value. The prompt message file for each national language is stored in a separate library. Which national language messages are retrieved at execution time depends on which library is highest on the library list.

To implement the exit program, the following objects are supplied in Y2SYSRC:

- Source for the exit program (YPRCMSGR1R).
- Database files consisting of two physical files with a logical file over each.

When these objects are compiled and the program object for YPRCMSGR1R exists in the library list, the exit program is called automatically whenever an NL-enabled function is copied.

National Language Database Files

The following physical files (Y2NLSPP and Y2MSMPP) and their corresponding logical files (Y2NLSPP and Y2MSMPP) are supplied in Y2SYSRC. You need to compile them before you can compile the exit program.

National Language Support File (Y2NLSPP)

This file contains the following details about each national language supported by your application:

- Language code
- Description
- Library that contains the national language prompt message file

Before using the exit program, add a record to this file for each national language you support. For example:

- Use the Toolkit Work with Database File Data (YWRKF) command to add records.
- Retrieve the Y2NLSPP file into your model using the Retrieve Physical Files into Model (YRTVPFMDL) command and design a maintenance program for the file in your model.

Message Mapping File (Y2MSMPP)

This file contains mapping details for each prompt message involved in the copy of an NL-enabled function. In addition to the base language code for the model, it contains the following details for each prompt message:

Existing Message	New Message
Message Id	Message Id
Function surrogate	Function surrogate
Message file name	Message file name
Model name	Model name
Generation library name	Generation library name

Note: You do not need the Message Mapping File if all national language message files are located on the same computer. See the Using the Exit Program section in this chapter for more information.

Exit Program (YPRCMSGR1R)

When the program object for YPRCMSGR1R exists in the library list, it is called automatically whenever an NL-enabled function is copied within a model or copied to another model using the YCPYMDLOBJ command. Before calling the exit program, the copy function allocates new prompt message ids for any prompt messages defined in the source function. The new message is a copy of the original prompt message in the base language for the model.

The YPRCMSGR1R exit program is written in RPG and is comprised of two separate subroutines. You can modify or add code to the exit program to customize it for your use or comment out either subroutine to suit special requirements. See the program source for details on its entry parameters.

Note: If you change the source, save it in a library other than Y2SYSRC.

AAPRMM Subroutine

The AAPRMM subroutine writes a record to the Message Mapping File for each prompt message when copying an NL-enabled function. You can use this to log data for processing later; for example, if your NL prompt message files are located on different computers.

BAADMS Subroutine

For each new prompt message created during copy of an NL-enabled function, the BAADMS subroutine creates a translated prompt message in each NL prompt message file. In other words, BAADMS iterates through the NL support file and creates a duplicate of each new base-language message, already translated into the corresponding national language, and stores it in the NL prompt message file. As a result, the new function is immediately prepared for use in multiple national language environments.

To create a new prompt message in the NL prompt message file, BAADMS internally calls a copy program (YCPYMSGR3I) that:

1. Uses the message id of the original base-language message to retrieve the message details (the translated message) of its counterpart in the NL prompt message file. If this message id does not exist in the NL prompt message file, a new message will not be created.
2. Creates a new prompt message in the NL prompt message file that consists of the message id allocated for the new base-language message and the retrieved translated message.

Using the Exit Program

The following are suggestions for using the exit program:

- Before you use the exit program, you need to:
  - Compile the physical files, the associated logical files, and the exit program.
  - Save the compiled objects in a library in your library list.
  - Add a record to the NL Support File for each supported national language.
- You need to generate the NL-enabled function before copying it in order to create the prompt messages.
- Each national language prompt message file needs to be stored in a separate library, since each message file has the name specified by the YPMTMSF model value.
- Before copying an NL-enabled function, ensure that the NL prompt message files are synchronized; in other words, ensure that the message ids in the base-language prompt message file also exist in each NL prompt message file.
- You do not need the Message Mapping File if all NL prompt message files are located on the same computer. In that case, you can comment out the call to the AAPRMM subroutine and the new messages will be created immediately using the BAADMS subroutine.
- If your national language message files are located on different computers or if you wish to defer the copying of the messages for performance reasons, use the following process:
  a. Comment out the call to the BAADMS subroutine so only the AAPRMM subroutine is called. AAPRMM adds records to the Message Mapping file (Y2MSMPP) but does not copy the prompt messages to the NL prompt message files.
  b. Write a program similar to the BAADMS subroutine that reads records from the Y2MSMPP file and calls the YCPYMSGR3I program for each record read.
  c. Add Y2MSMPP, Y2MSMPL0, and the new program object to the library list on the target computer.
  d. Run the new program on the target computer to copy the prompt messages to the correct prompt message files.
- Following are two suggestions for using the exit program with the YCPYMDLOBJ command. Experiment to determine the best method for your needs.
  - Place the Y2NLSPP and Y2MSMPP files and the associated logical files in the library list for the source model.
  - Create a prompt message file name for the target model using the YPMTMSF model value. This name can differ from that specified for the source model. Store the NL prompt message files for the target model in the same libraries as used for the source model.

For more information on the exit program and the two physical files, see comments in the exit program source.

Double Byte Character Set (DBCS) Applications

Languages containing characters that are ideographic such as Japanese, Chinese, and Korean are called double byte character set (DBCS) languages. A DBCS character is minimally four bytes long, two bytes for the character plus two shift-control characters. Each DBCS character, or string of DBCS characters, must be delimited by shift control characters. The i OS DBCS operating system has special system defaults to standardize handling of DBCS data.

Creating Applications

You can create applications for both single byte character set (SBCS) and DBCS from the same model or from separate models that you maintain in parallel. For either approach, in creating a DBCS application,CA 2E provides the model value YIGCCNV for Ideographic Character (IGC) support in DDS.

If model value YIGCCNV is 1 and the display includes an input-capable, IGC-type field, CA 2E:

- Generates an IGCCNV keyword in the DDS assigned to the function key specified by the *IGCCNV function key condition.
- Implements special processing for IGC-based languages, such as string handling and compile overrides.

Note: Model value YIGCCNV is automatically set to 1 when an IGC language is loaded.

SBCS Machine to Create a DBCS Application

If your model is to be single byte, model value YIGCCNV is set to 0, the default condition; however, you may want to create an application for both single and double byte use. An SBCS system cannot create source, device files, or database files with IGCDTA parameters. This approach requires:

- Recreating these files on a DBCS machine with a parallel model.
- Ensuring that all device constants and input-capable fields to contain ideographic data are no less than four bytes long, the minimum requirement for a DBCS character.
- Ensuring that all text fields have the IGC field attribute which cannot be compiled on an SBCS machine.
- Externalization of messages is also recommended.
- Changing any data in lowercase English to upper case, using the following Toolkit commands:
  - **YRTVMSGF**—Retrieve Message File
  - **YTRNPF**—Translate Physical File Data
  - **YTRNSRCF**—Translate Source File Data
- Porting the model to a DBCS machine for compilation and execution.

Note: If you use the YCPYMDLOBJ command to copy from the source to another model, convert text fields needing the IGC attribute.

DBCS Machine to Create an SBCS Application

By setting model value YIGCCNV to 1 in the source model, you define your model as DBCS. If you do this, you can generate an application that is both SBCS- and DBCS-enabled on the same machine. You can attach both DBCS and SBCS workstations to this machine, with English lower case installed as a secondary operating system.

With an English upper case DBCS operating system:

- System defaults are set to create all source, device files, and database files with ideographic character (IGC) enabling, assuming that you specified database fields with DBCS data types.
- The system value QIGC is set to 1, meaning the system expects to process IGC data. You cannot change this value.
- You can restore source and database files created with the IGCDTA parameter on an SBCS machine. However, you cannot restore source or database files containing ideographic data.

Bi-directional Languages

Bi-directional languages are languages from the Middle East, such as Arabic, in which text is read from right to left and numbers are read from left to right.CA 2E allows you to specify right-to-left cursor positioning for input-capable text fields.

This is done by setting the Field Exit option to R for the field details on device designs.

Chapter 7: Distributed Relational Database Architecture

This chapter introduces the Distributed Relational Database Architecture (DRDA) and covers DRDA support in CA 2E.

This section contains the following topics:

What Is DRDA?

DRDA is IBM's SAA architecture that provides access to data distributed across various systems and platforms. DRDA provides the user, via a high-level programming language, access to relational databases on different systems and platforms and the files that reside on the systems. DRDA is implemented using SQL/400 and the SQL CONNECT statement. Used with embedded or interactive SQL, the CONNECT statement defines the relational database (machine) to be accessed.

Note: While DRDA is implemented using SQL statements, these statements can also operate on DDS access paths, with the exception of Span access paths. To use DRDA with a DDS file only requires that the function be set to generate SQL. This permits most functions to use DDS access, while only those that require DRDA functionality will be generated as SQL COBOL or SQL RPG.

DRDA support at the Remote Unit of Work (RUOW) level is available with i OS Version 2 Release. This phase of IBM support allows for connecting an application requester (the system executing the program) to an application server (the system where data is being accessed).

As supported in CA 2E, DRDA allows the user to specify that a file is distributed and allows CA 2E functions to access the distributed data. DRDA has the constraints applied to the DRDA level of RUOW implemented under i OS Version 2 Release 1.1. The application accesses data on a single machine at a time. At present, support for DRDA is provided for DB2 and i OS platforms.

RUOW restrictions and constraints should be reduced as IBM releases the two higher levels of DRDA, Distributed Unit of Work (DUOW) (i OS Version 3 Release 1) and Distributed Request (DR).

Remote Unit of Work

For Remote Unit of Work:

- Within a single unit of work, users or applications can read and update one system at a time using multiple SQL statements.
- There is one Relational Database Management System (RDBMS) per unit of work.
- There is one unit of work manager or recovery unit.
- The application controls the commit/rollback of data.

At the RUOW level, distribution of processing is limited to a Logical Unit of Work (LUOW) between two systems, the application requester or master, and the application server or slave.

Only one LUOW can be active for each job, restricting the application requester to making contact with one system at a time. However, within the program it is possible for the application requester to contact several different application servers. The program controls commit and rollback, and you must make provisions for them at the program level.

Note: Since the RUOW applies at the job level (routing step), any changes of RDBs within any program will change the RDB of the whole job stack.

Distributed Unit of Work

For Distributed Unit of Work:

- Within a single unit of work, users or applications can read and update data on multiple systems. Each SQL statement can access one system.
- There are several RDBMS's per unit of work.
- There is one RDBMS per request.
- There are several unit of work managers.

Distributed Unit of Work (DUOW) permits the same capability and functionality as RUOW, but allows the system connections to take place on an implicit basis. In other words, the SQL CONNECT statement is not required. The program still controls commit and rollback, and you must still make provisions for them at the program level.

Distributed Request

For Distributed Request:

- This level of DRDA has the highest degree of transparency. Users or applications can, within a single unit of work, read and update data on multiple systems. Each SQL statement can access multiple systems.
- Multiple requests per unit of work are allowed.
- There can be more than one RDBMS per request.
- There can be several RDBMS's per unit of work.
- There can be several unit of work managers.
- One of the unit of work managers coordinates commit/rollback.

Using a Distributed Request (DR), you will be able to access multiple RDBMS's at the same time. The systems update the files. A DR allows the user to perceive all files across all systems as being part of one unified database.

CA 2E Implementation of DRDA

CA 2E's DRDA support makes it possible for you to generate applications that access files distributed over many systems. CA 2E functions can be created with distributed file I/O control set to CA 2E. With this option, the relational databases accessed by the function are controlled by a distributed file configuration table that you can maintain.

The use of a table to control access permits changes to the relational database (RDB) configuration without the need to change application code. This means that you can put the same program objects on several IBM i distributions, with different configuration tables on each one.

CA 2E's DRDA support provides the following features:

- Design of applications for distributed data processing
- Access of the same file name across multiple systems (not as a unified database)
- Implementation-level Distributed File Configuration Table
- CA 2E-generated iteration loops or user-defined control for error handling for CONNECT failure
- User access for specification of the RDB name used in the SQL CONNECT statement
- Access to either DDS or SQL files

Shipped Defaults

In keeping with CA 2E's implementation independent design capabilities, you do not have to modify the design of a function to use DRDA. To generate a DRDA application from an existing design, you set generation options and regenerate the functions, but the design remains the same.

CA 2E ships the default values for the new model values, file options, and function options. You explicitly set DRDA functionality. If you do not specify distributed processing functionality for your model, DRDA functionality does not affect it.

By setting certain flags, you can choose how you wish to generate and implement your distributed applications. The shipped defaults for the options that drive CA 2E distributed function creation are:

- New model values:
  - YGENRDB (*NONE - no DRDA compilation enabled)
  - YDSTFIO (*NONE - no generation of distributed functionality)
- Distributed flag: Distributed - N (No); flag for each CA 2E file in the model
- Function option: Distributed File I/O control - M (MDLVAL); YDSTFIO model value

Use of these defaults is described later in this chapter.

Steps to Implement DRDA

There are two new steps that are required to implement DRDA:

1. For data modeling:
   - Set a flag on a CA 2E file to indicate that it is distributed (refer to the Using Shipped DRDA Values section in this chapter).
   - Set the Distributed File I/O control function option for the function to either S (CA 2E) or U (User) control. Set this option from Edit Function Options, using the Distributed File I/O Control field.
2. If you specified S (CA 2E) control for the function, enter the file locations in the Distributed File Configuration Table. This entry associates the file with the name of the application server and, optionally, the collection name in which it is located, allowing a different configuration at each location on which the application is implemented.

Creation (compilation) of the program is via an extended form of CRTSQLxxx (where xxx = HLL language). Use the YGENRDB model value as the RDB in the CRTSQLxxx command.

Running CRTSQLxxx results in:

- An application program, which resides on the application requester.
- An SQL package, which resides on the application server and contains the SQL statements for accessing the AS database.

Development Environments for DRDA in CA 2E

Using only one machine, you can design and create DRDA applications that are production ready. You can exercise program logic without the use of another machine by simply configuring the files as having one RDB, the local RDB. The generated SQL CONNECT statements are executed and the connection is made to the local system.

By changing the Distributed File Configuration Table, you can modify the collection and relational database names easily, without having to generate or compile the applications again. However, you do need a distributed machine to fully test the distributed data functions of the generated code.

Perform the following tasks to create the correct recommended environment:

1. First create RDB entries in the RDB Directory, using the i OS Work with Relational Database Directory Entries (WRKRDBDIRE) command. The only entry required for one machine is the local RDB name. This same RDB name should be used as the model value for YGENRDB.
2. For a multiple machine environment, describe each RDB in the directory of both the local and remote machines. Each RDB will have SQL collections (libraries) containing the distributed files (tables and views).
3. Using save/restore or communication facilities, such as Send Network File (SNDNETF) or Send SMT Objects (SNDSMTOBJ), distribute SQL packages to the RDBs. Note that the package must reside in a library with the same name as the library of the application program that defined it.

   For example, PGMA resides in a library called DRDAGEN; therefore, the SQL package of the same name (PGMA) also needs to reside in DRDAGEN.
4. Establish appropriate security and communication requirements for the IBM i.

For more information:

- On the i OS commands mentioned above, see *IBM's Application System/400 Programming: Control Language Reference*.
- On DRDA security and communication requirements, see *IBM publications on SNA, such as SNA Concepts and Products*.

Using Shipped DRDA Values

This section specifies DRDA model values and flags.

DRDA Model Values

The model value YDSTFIO (Distributed File I/O control) is input-capable and has the following values:

- ***SYNON**—Synon Control
- ***USER**—User Control
- ***NONE**—No distributed functionality will be generated

If the Distributed File I/O Control function option is M (MDLVAL) for a function, that function inherits the value set for YDSTFIO. Set this option from the Edit Function Options panel.

The model value YGENRDB (Generation RDB Name) is the RDB name used in the i OS CRTSQLxxx command for compiling the CA 2E-generated distributed function. The RDB name indicates the RDB to which the SQL package will be distributed. The RDB name should be the name of the local RDB as defined in the IBM i RDB directory. If the name is not the local RDB, the SQL package is created as part of the compile, and the IBM i will try to move the package to the RDB specified in the CRTSQLxxx command, unless communications limitations prevent it from doing so.

If you use the shipped default value *NONE for YGENRDB, the model is not enabled to compile DRDA applications. Even though code may be generated that contains distributed functionality, there is no RDB to compile against. You must specify an RDB to compile DRDA applications.

Distributed Flag

A new field, Distributed, on the Edit File Details panel, allows you to flag a CA 2E file as distributed. The field can have the following values:

- **N**—No. Default.
- **Y**—Yes.

This flag affects the creation of the Distributed File Configuration Table. Entries for access paths built over CA 2E files are either added (Y) or removed (N) from the table. To do this, execute the Convert Distributed Files to Configuration (YCVTDSTFIL) command.

The entries in the Distributed File Configuration Table can then be modified using the Work with Distributed Files (YWRKDSTFIL) command.

Use of these commands is described in detail later in this chapter.

Note: If you generate a program with DRDA-distributed code but compile it using the model value YGENRDB set to *NONE, the program compiles. However, at execution the program will be unable to perform the connects. Appropriate error messages will be issued. You can override the YGENRDB value for a specific program by using a compiler override to specify a different value for the RDB keyword on the i OS commands, Create SQL RPG/400 Program (CRTSQLRPG) and Create SQL COBOL/400 Program (CRTSQLCBL).

For functions to be distributed with S (CA 2E) for Distributed File I/O Control, the default action diagram of those functions will show the availability of the *Change RDB function and, depending on the function type, the *Previous page function. For example:

```
>Process response
.—CASE
|—CTL.*CMD key is *Exit
|  ...Exit  program
|—CTL.*CMD key is *Next page
|  ...Load next subfile page ===>
|—CTL.*CMD.key is *Previous Page
|—  ...Connect to previous RDB
| —PGM.*Reload subfile = CND.*YES
|—CTL.*CMD key is *Help
|  ...Process help request
|—CTL.*CMD key is *Change RDB
|  ...Change to selected RDB
|—*OTHERWISE
|  ...Process screen
'—ENDCASE
```

Function Options

This section outlines how to use the Distributed I/O option.

Accessing Multiple Systems with the Same File Name

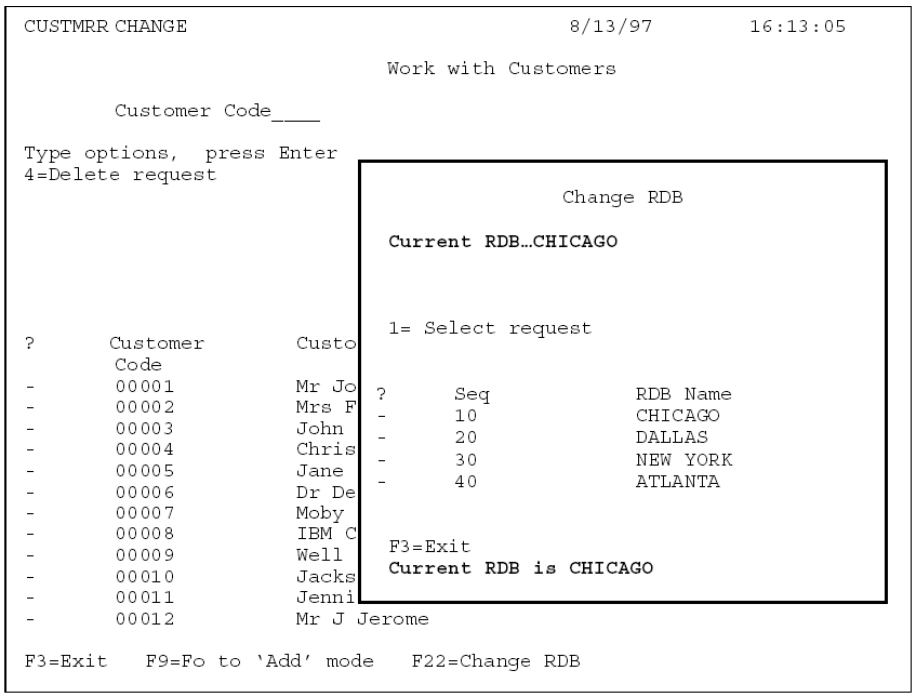
Due to constraints of Remote Unit of Work, the S (CA 2E) Distributed I/O option only applies to the primary access path of the function. There is no CA 2E control for file I/O operations contained within embedded internal functions.

When a function uses the S (CA 2E) Distributed I/O option, initial connections to the primary view of the function will use the configuration table entries specified in the configuration table for that primary view.

Since the scope of a connect affects all files and programs in the current routing step, modifying the RDB for the main file may affect any underlying processing to other database files within the master function and to any other functions called from that master function.

All functions generated under S (CA 2E) control enable function key F22 for their device designs. The default value of this key is stored in a *Change RDB command key list, and can be modified in the model. Function key F22 calls the shipped CA 2E program Y2CRDBR (Change RDB), which appears in a window. The program displays all RDBs in the configuration table for the primary view and allows application users to switch to an RDB by merely selecting one of the entries. This program indicates to users which RDB they are currently connected to.

For example:



Functions with Subfiles

The three functions, EDTFIL, DSPFIL and SELRCD, have subfiles. For these functions, if the end user tries to continue scrolling past the end of the subfile, S (CA 2E) Distributed file I/O control causes an automatic connection to be made to the next RDB at the end of subfile for the RDB. Similarly, if the end user rolls beyond the top of the subfile, a connection will be made to the previous RDB. If a Rollup or Rolldown beyond the current extent is attempted, and the current RDB is the last (or first) in the configuration list, a message appears stating that there are "No more RDBs to connect to".

These automatic connections are subject to a confirm prompt if confirm prompting has been requested for the function. This confirm prompt appears with a message, "Confirm Connect to Next/Previous RDB," as a request to confirm the connection, not the changes. Any changes are lost if not previously confirmed by the end user.

End users can respond N to the confirm prompt to return without connecting, and press Enter to confirm any changes they may have made.

The subfile functions, EDTTRN and DSPTRN, do not have the roll functionality. This is because they have an inherent key screen concept similar to the EDTRCD and DSPRCD functions. Further, an EDTTRN function has infinite roll forwards, as you can add records at the end of an EDTTRN function. A DSPTRN function, while having a defined roll boundary, currently has rolling controlled by i OS, because the whole subfile is built before it is displayed.

Note: All functions with panels, including EDTTRN and DSPTRN, have the F22-Change RDB function under CA 2E Distributed file I/O control.

DRDA Control Fields

Within the action diagram, the programmer has access to the following three fields that control DRDA functionality:

- **JOB.*Current RDB**—The name of the currently attached RDB, maintained by CA 2E. This field can be added to the device design as a function field to inform application users which RDB they are currently connected to.
- **PGM.*Next RDB**—A user-modifiable field. If nonblank prior to the next data access, this field will be used to connect to the named RDB before access. While the application will respond to the values in this field, whether CA 2E or User control is requested, its primary use is for functions generated with User control.
- **JOB.*Local RDB**—-CA 2E -maintained field for the name of the local RDB where the application is executing. This field has a value if the application is connected locally either when invoked or in processing.

With U (User) control, connects are still generated. The application user can modify the RDB by placing values in the PGM.*Next RDB parameter, but there is no F22 support, nor is there any rollup/rolldown support for functions with subfiles. Users receive a message to let them know the connection failed.

Functions such as PRTFIL and EXCEXTFUN, which have no panels, can only be specified as having User Distributed I/O control. Specifying S (CA 2E) control is accepted but ignored and U (User) control is used.

Referential Integrity

Due to an RUOW constraint,CA 2E does not generate CONNECT statements prior to referential integrity checks and therefore validates relations on the RDB to which the user is currently connected. The i OS is unable to switch connections to another RDB in the middle of an SQL FETCH routine. This is fully consistent with the RUOW assumptions that each RUOW is constrained to a single RDB. If a user attempts to switch to another RDB in the middle of a SQL FETCH loop, DRDA closes all cursors and the application fails.

Commands for DRDA

YCVTDSTFIL: Convert Distributed Files to Configuration

The YCVTDSTFIL command allows you to create entries for distributed files in the Distributed File Configuration Table. The command preserves the entries in the table already made for a file in the following ways:

- If the file is distributed and has an entry in the Distributed File Configuration Table, no new entry is required.
- If the file is distributed and there is no entry in the Distributed File Configuration Table, an entry is created with the YGENRDB model value as the default RDB name.
- If the file exists in the Distributed File Configuration Table and is not Distributed, all entries for that file are removed from the Distributed File Configuration Table.

The YCVTDSTFIL command creates the physical file and logical files required for the Distributed File Configuration Table if they do not already exist in the named generation library. It uses the i OS Create Duplicate Object (CRTDUPOBJ) command and copies the template objects from Y2SY.

The YCVTDSTFIL command can be run outside of the model. To access the Display Convert Model Data Menu from the Display Services Menu, select the option Convert condition values to database file. From the Display Convert Model Data Menu, you can select the option to run the YCVTDSTFIL command.

Note: There are two approaches to removing distributed functionality.

- Change generated, distributed files to Distributed = N. Generate and compile the functions to remove distributed functionality. Entries in the Distributed File Configuration Table will be removed when you execute the YCVTDSTFIL command.
- As an alternative to changing the Distributed option for a file from Y to N, you can modify the Distributed File Configuration Table and change the RDB name to be the name for the local RDB. By manipulating the Distributed File Configuration Table, you can avoid changing the generated code, the SQL CONNECT is to the local system, and you do not have to generate the code again.

YWRKDSTFIL: Work with Distributed Files

Two CA 2E files, *Configuration Table and *Distributed File, are present in every model. These files contain functions and access paths shipped for DRDA. When generated, they form the set of objects that make up the configuration table editor environment.

Note: The YWRKDSTFIL command operates outside the model environment. Security for this command, and associated functions and objects, are your responsibility.

You can copy the functions shipped with *Distributed File and *Configuration Table files and make modifications to the copied versions as required. You can also create functions and customize them to your own standards and requirements. You may even want to create an NLS version of the editors by generating NLS applications in CA 2E.

The YCVTDSTFIL and YWRKDSTFIL commands and the shipped RDB loader (Y2LRDBR, Load RDBs for View) use the access paths described in the model built over the *Distributed File and *Configuration Table files. Any changes that impact these access paths may cause these functions to behave incorrectly. The source for Y2LRDBR and the associated display file is shipped in Y2SYSRC.

Note: CA does not recommend adding any fields to either the shipped *Distributed File or *Configuration Table files, nor directly modifying the system functions and access paths.

To begin to use the shipped configuration table editor, generate and compile the following:

- *Distributed File Access Paths:
  - Physical file—Y2DSTFP
  - Update index—Y2DSTFL0
  - Retrieval index—Y2DSTFL1
  - *DSTFNM/*TYPE/*TABVIEWNM—Y2DSTFL2
  - Dist. File - Config Tbl—Y2DSTFL3255
- Functions:
  - Work With Config Table—Y2CFGTR
  - Work With Dist. File—Y2DSTFR
- *Configuration Table Access Paths:
  - Physical file—Y2CFGTP
  - Update index—Y2CFGTL0
  - Retrieval index—Y2CFGTL1
  - *TABVIEWNM/*SEQ/*RDBNM—Y2CFGTL2

Once you generate and compile these objects, run the YCVTCNDVAL (Convert Condition Values) command, then the YCVTMDLMSG (Convert Model Messages) command for your model.

You can then use the YWRKDSTFIL command to invoke the configuration table editor. This command displays a list of the access paths enabled for distribution. These entries are created with the YCVTDSTFIL command and displayed on the Work With Distributed Files panel. On this panel, files are identified as tables or views. A table in SQL is equivalent to a physical file, and a view is similar to a logical file.

You can work with the tables and/or views from this panel. You may wish to define the distributed nature of data at the table level (physical file) or to differentiate the distribution of that at the view (logical file) level.

While you are not required to do so for DRDA support, you can reference the access paths by any valid CA 2E functions.

Note: You cannot remove any of the distributed file table entries, except by running the YCVTDSTFIL command.

Working with Configuration Table Entries for Tables

To work with a Configuration Table entry, take option 5 against the table. The Work with Configuration Table Entries panel displays.

RDB Name

You can add or remove RDB locations for a table. Each RDB name must be unique for a table, regardless of sequence number. You can only add a new RDB for a table. Once you add an RDB, it is propagated to all the views based on the table.

Once you remove an RDB, it is removed from all the views associated with that table. This ensures referential integrity in the configuration table database.

Note: You can only add RDBs directly for views if the RDB exists as an entry for the corresponding table. This prevents any view from being specified on an RDB when its corresponding table is not specified.

Once you add an RDB entry, modifying it at the table level will not cause the views to be updated, as you may have customized certain views.

Seq

The Sequence Number (Seq) on the Work with Configuration Table Entries panel indicates the order in which to process the locations for that particular access path.

The Work with Configuration Table Entries panel automatically resequences in increments of ten. RDBs are initially ordered by the allocated sequence number, then alphabetically within sequence number. This is the order in which the RDBs are retrieved and cycled through in the generated applications which you specify for Distributed I/O control.

Collection

The Collection is optional and serves as a reference. Its presence is compatible with expected enhancements in i OS DRDA support.

Working with Configuration Table Entries for Views

To work with a Configuration Table entry for a view, place 5 against the view. The same Work with Configuration Table Entries panel displays.

You can only create RDBs for views already present as RDBs for the corresponding tables. You can remove any RDBs on which the view may not reside. You can also alter the sequence of access for the view.

Appendix A: CA 2E Objects Required for Compilation and Execution

This appendix lists the CA 2E objects required either to compile or execute applications generated with CA 2E. Source for the required objects is in the shipped library, Y2SYSRC. To compile or execute applications in jobs that do not have the Y2SY library present in their library lists, duplicate the objects to the application libraries.

This appendix also lists the Toolkit objects required to display Help text (Display Help, YDSPHLP) and menus (Go to Menu, YGO).

For more information:

- On duplicating objects, refer to the chapter titled "Implementing Your Application" in this guide.
- On the YDSPHLP and YGO commands, refer to the CA 2E *Toolkit Reference Guide*.

Required CA 2E Objects

This topic covers:

- Required Objects for RPG Compilation
- Required Objects for COBOL Compilation
- Required Objects for Execution

Required Objects for RPG Compilation

Type	Sys Object Name	Obj Type	Obj Att	Description
Cmp	Y2I#DSP	*FILE	PF	RPG III File inf data structure (DEV)
Cmp	Y2I$DSP	*FILE	PF	RPG III File inf data structure (DEV)
Cmp	Y2IUDSP	*FILE	PF	RPG III File inf data structure (UPD)
Cmp	Y2I1DSP	*FILE	PF	RPG III File inf data structure (UPD)

Cmp	Y2PGDSP	*FILE	PF	RPG III PGM STS data structure

Required Objects for COBOL Compilation

Type	Sys Object Name	Obj Type	Obj Att	Description
Cmp	Y2IOPEN	*FILE	PF	COBOL
Cmp	Y2IDSPFIO	*FILE	PF	COBOL Display file IO area (DEV)
Cmp	Y2IPRTFIO	*FILE	PF	COBOL Print file IO area (DEV)
Cmp	Y2IMAJMIN	*FILE	PF	COBOL Major/Minor codes (DEV)
Cmp	Y2PGDSPK	*FILE	PF	COBOL PGM STS data structure

Required Objects for Execution

Type	Sys Object Name	Obj Type	Obj Att	Description
Exc	Y2USRMSG	*MSGF		Default messages (Y2xnnnn)
Exc	Y2CLMSC	*PGM	RPG	Clear a message queue
Exc	Y2CPMSC	*PGM	CLP	Copy a message queue
Exc	Y2EXMGC	*PGM	CLP	Retrieve and execute a message QCL
Exc	Y2EXMSC	*PGM	CLP	Retrieve and execute a message QCL
Exc	Y2EXMCC	*PGM	CLP	Retrieve and execute a message QCMD
Exc	Y2RVMGC	*PGM	RPG	Retrieve a message
Exc	Y2RVMSC	*PGM	CLP	Retrieve a message QCL
Exc	Y2SNMGC	*PGM	RPG	Send a message
Exc	Y2SNMSC	*PGM	CLP	Send a message QCL
Exc	Y2VLLSR	*PGM	RPG	Values list display
Exc	Y2VLLWR	*PGM	RPG	Values list display window
Exc	Y2BGCMC	*PGM	CLP	Begin a commitment control
Exc	Y2RCRSC	*PGM	CLP	Reclaim resources

Exc	Y2QLNMR	*PGM	RPG	Qualify a name CPF syntax (.)
Exc	Y2RTDAC	*PGM	CLP	Retrieve a data area
Exc	Y2RTJBR	*PGM	RPG	Retrieve job attributes
Exc	Y2RVCNR	*PGM	RPG	Retrieve condition name
Exc	Y2RVCNR	*PGM	RPG	Retrieve condition name
Exc	Y2WSTOC	*PGM	CLP	Workstation time out: Program
Exc	Y2WSTOC#	*FILE	DSPF	Workstation time out: Device file
Exc	Y2WSTOM	*MSGQ		Workstation time out: Message queue
Exc	Y2PSSR	*PGM	RPG	*PSSR exception handler
Exc	YCMDEXEC	*PGM	CLP	Platform-independent user command execution
Exc	YWRTLCI	*PGM	PLI	Calculate DDS window location
Exc	YWRTLCI2	*PGM	PLI	Calculate DDS window location subprogram
Exc	YWWDDRR	*PGM	RPG	Retrieve cursor location for automatic positioning of window
Exc	YWWDCKI	*PGM	RPG	Retrieve device type subprogram
Exc	YWWDDR#	*FIL	DSPF	User-defined data stream for YWWDDRR

Toolkit Required Objects

The following table includes the objects required to run Toolkit menu and help utilities on the IBM i:

Object	Type	Text
YDDSHPC@	*PGM	YDSPHLP/YDSPMNU Display Help Text CPF
YDDSHPR	*PGM	YDSPHLP/YDSPMNU Display Help Text
YDMNGOC	*PGM	YGO Go to a menu. Execute an option
YDMNGOC@	*PGM	YGO Go to a menu CPP
YDMNGOI	*PGM	YGO Go to (display) menus
YDMNNSR	*PGM	YWRKMNU/YDSPMNU Menu selection display
YDMNSOI	*PGM	YDSPMNU Retrieve user profile signoff option

YSRTOLC@	*PGM	YRTVOBJLIB Retrieve Object Library
YYCKAUC	*PGM	YDSPMNU Check object authority
YYCVBNR@	*PGM	YCVTBIN Convert binary (2B) to decimal (5.OP)
YYQLN2R	*PGM	Qualify a name
YYRCMSC	*PGM	Receive a request message from specified program message queue
YYRTMGC	*PGM	Retrieve a message from a message file
Y1PGMSC	*PGM	YDSPHLE/YDSPMNU Send message to a given message queue
Y1USRMSG	*MSGF	Y1 Execution messages
Y1USRPMT	*MSGF	DDS MSGCON Message prompts for shipped source
YDDSHPR#	*FILE	PRD YDSPHLP Display help text
YDMNGOI#	*FILE	PRD YGO Go to menus
YDNMWKR#	*FILE	YWRKMNU/YDSPMNU Display screens
Y1USRTXT	*FILE	Y1 LDO ENG P1 Help Text Members: YDMNGOI YDMNNSR
YCVTBIN	*CMD	Convert binary (2B) to decimal (5.OP)
YDSPHLP	*CMD	YDSPHLP Display help text
YGO	*CMD	YGO Go to a menu
YMHPFLA	*DTAARA	YGO Name of help text file
YMHPLBA	*DTAARA	YGO Name of help text file library
YYSLV1A	*DTAARA	Selection value mapping

Appendix B: Troubleshooting

This appendix covers some of the most common source generation errors.

Source Generation Errors

Some examples of source generation errors are as follows:

Display File and Program in Error (*ERROR)

Problem: The entry for the display file and the program appear on the job log with an *ERROR status. The most frequent cause is a device design that exceeds the panel size.

Solution: Enter the device design to correct this problem. You can use F4 to move to the right and look for labels or fields exceeding the device size. The size depends on the header options.

One easily missed area is the function key text lines and subfile selection prompt and text. These fields are 78 characters long and should begin at the left margin. To correct the problem:

1. Press F17 and zoom into the Subfile Control format.
2. Zoom into each *SFLSEL field, and make the Spaces before equal to 1.

Action Diagram Un-determined Action

Problem: An **!!!Undetermined action** within an action diagram causes source generation failure.

Solution: Use the Find Error facility in the action diagram editor, or follow these steps:

1. Scan through the generated source to find the comment line containing !!!.
2. From the Submit Model Generation & Creates panel, invoke the source editor, STRSEU, by placing E in front of the program object in error.
3. To find the incomplete action diagram statement, at the SEU positioner line, enter the search string **!!!** and press Enter.
4. Return to the action diagram for the function to correct or remove the incomplete statement.

Context Not Found

Problem: Context Not Found occurs if:

- You changed the access paths on a function and did not revisit the action diagram to re-verify user statements.
- You removed function fields from the device design and did not revisit the action diagram to re-verify user statements.
- You changed parameters on called functions and did not revisit the action diagram containing the call to re-verify the parameters.
- You copied sections of code between user points or between action diagrams containing contexts that are not valid at the target.

Solution: Use the Find Error facility in the action diagram editor, or find the errors by looking for ??? in the source listing. You can roll backward to identify both the function that is called and the user point in which the function is located.

When you revisit the parameters for a statement or a function, always prompt the field and reselect the values.CA 2E does not automatically perform the validation on the statement shown if you do not make a change to the displayed parameter values.

Appendix C: CA2E--Change Control Facilities Reference Tables

This appendix contains the following change control facilities reference tables:

- **Model Object Description**—This table describes the most significant fields contained in the model object description for each design object in a CA 2E model.
- **Default Change Types for Component Change Processing**—These tables list the default change type for each possible change you can make to a model object. There is a separate table for each model object type.
- **Component Change Processing Propagation Table**—This table shows the effect of each change type on objects that use the changed object.
-
- For more information on CA 2E change control facilities, refer to the chapter, “Managing Model Objects”, in this guide.

CA2E--Model Object Description

CA 2E maintains a model object description for each object in your model and automatically updates the model object descriptioneach time the object is changed, regenerated, or imported.

Use selection option 8 from the Edit Model Object List panel to view the model object description for an object. A standard set of information is displayed for each model object type; for example, name, type, surrogate, and so on. Additional information displayed depends on the model object type as shown in the following table:

Object Type	Information Displayed	Examples
ACP	Attribute	PHY, RTV, UPD
	Source Member	Name and description
	Source Details	Date/time of last successful generatio
	Auxiliaries	Implementation object name and description
APP	Application Area Code	Three-character name
ARR	–	–

Object Type	Information Displayed	Examples
CND	Atrribute	VAL, LST
FIL	Atrribute	REF, CPT
FLD	Usage	ATR, CDE, USR
	Atrribute	STS, DTE, TXT
	Implementation Name	DDS name
FUN	Function Type	CRTOBJ, EXCINTFUN
	Access Path	Name and description
	Source Member	Name and description
	Source Details	Date/time of last successful generation
	Attribute (SEU Type)	RPG (if appropriate)
MSG	Atrribute	INF, ERR, STS
	Implementation name	Message identifier

CA 2E--Definitions of Model Object Description Fields

The following table lists the most significant fields contained in the model object description. You can use model object list commands to analyze your model and create model object lists and reports based on values in the model object description.

For more information:

- On model objects, model object lists, and model object list commands, refer to the chapter titled “Managing Model Objects” in this guide.
- On retrieving model object description fields, refer to the YRTVMDLOBJ section of the Command Reference Guide.

-

Field	Meaning
FunctiModel object surrogate number (OBJSGT)	A number that uniquely identifies an object in a model; it is assigned automatically when the model object is created. CA 2E uses the object surrogate to identify objects internally. You can use the surrogate on model object list commands as an alternate to the owner/name/type identifier.
Object name (OBJNAM)	Descriptive name of the model object assigned by the developer. This is the name part of the object's owner/name/type identifier.
Object type (OBJTYP)	The model object's type; for example, ACP for access path. Possible values are: ■ ACP ■ APP ■ ARR ■ CND ■ FIL ■ FLD ■ FUN ■ MSG This is the type part of the object's owner/name/type identifier.

Field	Meaning
Object attribute (OBJATR)	The model object's subtype; the value depends on the object type. For example, for an object of type ACP, RTV means Retrieval access path. This field is blank for APP and ARR object types.
Surrogate of the owning object (OWNSGT)	The model object surrogate for the owner of the model object; for example, for an access path or function, this is the surrogate of the owning file (FIL). Not all objects have owners.
Name of owner of object (OWNNAM)	Descriptive name of the owner of the model object. This is the owner part of the object's owner/name/type identifier.
Function type (FUNTYP)	The type of function for model objects of type FUN; for example, DSPFIL for Display file and RTVOBJ for Retrieve object.
Implementation name (IMPNME)	The implementation or 3GL name for the model object. The value depends on the object type; it is blank for ARR and CND. ■ ACP—Source member name ■ APP—Application area code ■ FIL—Format prefix ■ FLD—DDS name ■ FUN—Program source member name or blank ■ MSG—Message ID
Copy name (CPYNME)	The Object name of the corresponding model object in a target model. It is used by the YCPYMDLOBJ command when copying model objects between models. For new objects, the Copy name is initially the same as the Object name; for new versions (FUN or MSG), the Copy name is initially the same as that used by the version group to which it belongs.
Creation date and time (CRTDTE/CRTTME)	The date and time the model object was created.

Field	Meaning
Change date and time (CHGDTE/CHGTME)	The date and time the model object was last edited. CA 2E updates this field automatically. It is not updated when: ■ The model object is accessed as if to edit, but no changes are actually made. ■ The model object has been copied from another model or model object and has not been changed since copying.
User profile of developer that changed the object (CHGUSR)	The name of the user profile that last updated the model object.
Change type (CHGTYP)	The type of the most recent change to the model object. It is used by component change processing to identify other objects in the model affected by the change. The possible values are: ■ PUB—Public ■ PVT—Private ■ GEN—Regenerate ■ OBJ—Object only
Impact processed indicator (IPCPRC)	This field indicates whether component change processing has been run for the last change to the model object. It is set by component change processing.
Component changed processing date and time (COMPDTE/COMPTME)	The meaning of this field depends on whether the model object itself or a component of the model object was changed. ■ For a changed object, this is the same as its Change date and time. ■ For an object that uses a changed object (a component changed), this is the date and time component change processing was run for the change.

Field	Meaning
■ Action required indicator (ACTRQD	The type of change required to the model object as a result of a change to one of its components. ■ GEN—Generation required ■ EDT—Edit required
Generation date and time (GENDTE/GENTME)	The date and time the model object was last successfully generated.
SEU Type (SEUTYP)	The system SEU type of the model object; for example, RPG or PF. It contains a value only for source-based model objects.
Import date and time (IMPDTE/IMPTME)	The date and time the model object was last copied from another model using the Copy Model Object (YCPYMDLOBJ) command
■ Group surrogate number (GRPSGT)	For a model object that is not a version of another model object, this is the same as the Model object surrogate. For a version of a model object, this is the group surrogate number of the model object from which the version was created. All versions of a model object have the same value in this field and together comprise a group of versions.
Current object indicator (CUROBJ)	For a version of another model object, this indicates whether the version is the one being used by other model objects in the model; in other words, whether the version is current. Only one version of a group of versions can be current. A model object that is not a version of another model object is always current.
Version type (VSNTYP)	■ The version type for the model object. Only one version in a group can be of type PRD. This value is always DEV unless you are using Advantage 2E CM. The values are: ■ DEV—Development ■ PRD—Production ■ ARC—Archive

Field	Meaning
Archive surrogate number (ARCSGT)	The model object surrogate of the archive object that this object replaced when it was promoted. This field contains a value only for archive versions. This field is used only for Advantage 2E CM.
Check out date and time (CHKDTE/ CHKTME)	The date and time the model object was last checked out for a change. This field contains a value only if you are using Advantage 2E CM.
Check out user (CHKUSR)	The name of the user profile that checked out the model object for a change. This field contains a value only if you are using Advantage 2E CM.
Check out status (CHKSTS)	Status information used by Advantage 2E CM.
Check out list (CHKLST)	The name of the model object list to which the model object has been checked out. This field contains a value only if you are using Advantage 2E CM.
Promotion type (PRMTYP)	The promotion method for the model object; namely, whether the object is new to the target environment and whether the model object is to be promoted. This field contains a value only if you are using Advantage 2E CM. ■ *CHG—The object is to replace an existing version in the target ■ *ADD—The object is to be added to the target ■ *GEN—The object is to be generated from the target; the design object is not to be promoted ■ *DLT—The object is to be deleted from the target

-
-

CA2E--Default Change Types for Component Change Processing

Component change processing ensures the integrity of your model by identifying which other model objects are affected and the type of change required whenever you change a model object. CA 2E uses the default change type table internally to set the change type in the object description for the changed model object. You can use model object list commands to analyze your model based on this change type setting.

For more information on component change processing, refer to the Impact Analysis section of the “Managing Model Objects” chapter in this guide.

The tables in this appendix list the possible changes that can be made to each model object type and the default change type for each change. There is a separate table for each supported model object type. Each entry in each table corresponds to a position on a CA 2E panel where you can change a model object.

In cases where these tables cannot be used to determine the change type, the Override column in the table explicitly specifies the change type. A change type of *CREATE or *DELETE in the Override column are special cases of *PUBLIC. As a result, they do not require component change processing because there are no using objects.

CA2E--Access Paths - ACP

Panel	Change	Change Type	Override
Edit Access Path Auxiliaries	Query physical name	*PRIVATE	
		*OBJONLY	
	Query physical text		
	Query program name	*PRIVATE	

Panel	Change	Change Type	Override
	Query program text	*OBJONLY	
Edit Access Path Conditions	Sequence number	*GEN	
	Field	*GEN	
	Condition	*GEN	
Edit Access Path Details	Access path name	*OBJONLY	
	Allow select/omit	*GEN	
	Generation mode	*PRIVATE	
	Source member name	*PRIVATE	
	Source member text	*OBJONLY	
	Duplicate sequence	*GEN	
	Unique key	*GEN	
	Maintenance	*GEN	
	Alternate collation table	*PRIVATE	
	Format text	*OBJONLY	
	Format name	*PRIVATE	
	Assoc. retrieval access path	*PRIVATE	
	Sequence number	*PUBLIC	
Delete Access Path	Delete Access Path		*DELETE
Edit Access Path Format Entries	Key number	*PUBLIC	
	Ascending/descending	*GEN	
	Alt. Coll. Sequence	*PRIVATE	
Display Access Path Formats	Add new format for SPN access path		*PUBLIC
Edit Physical File	Sequence	*GEN	
	DDS name	*PRIVATE	
	Type	*PRIVATE	

Panel	Change	Change Type	Override
	Length	*PRIVATE	
Edit Access Path Relations	Add/remove relations	*PUBLIC	
	Referenced access path	*PRIVATE	
	Select record function	*PRIVATE	
Edit Access Path Select/Omit	Select/omit details	*GEN	
	Sequence	*GEN	
	Text description	*OBJONLY	
Edit Access Path Relation Virtual Fields	Add/remove virtual fields	*PUBLIC	
No Panel	Default access path creation program		*CREATE

CA2E--Arrays-ARR

Panel	Change	Change Type	Override
Edit Array	New array details		*CREATE
Delete Array	Delete array		*DELETE
Edit Array Details	Number of elements	*PRIVATE	
	Sequence	*PRIVATE	
	Unique	*PRIVATE	
	File/field	*PRIVATE	
	Access path/field	*PRIVATE	
	Sequence	*PRIVATE	
	Array name	*OBJONLY	
Edit Array Entries	'+' Select field	*PRIVATE	
	'-' Drop field	*PRIVATE	
Edit Array Key Entries	Key number	*PRIVATE	

CA 2E--Conditions - CND

Panel	Change	Change Type	Override
Display Field Domain Conditions	Delete condition		*DELETE
Edit Field Condition Details	Create condition		*CREATE
	Condition name	*OBJONLY	
	Relational operator	*PRIVATE	
	Internal value	*PRIVATE	
	External value	*PRIVATE	
Edit List Condition	Create condition		*CREATE
	'+' Add condition	*PRIVATE	
	'-' Remove condition	*PRIVATE	
	Change condition name	*OBJONLY	

CA 2E--Files - FIL

Panel	Change	Change Type	Override
Define Objects	New file/field		
	Record not found MSG		
	Record exists MSG		
Edit File Entries	Replace field	*PUBLIC	
Edit File Details	File name	*OBJONLY	
	Attribute	*OBJONLY	
	Document sequence	*OBJONLY	
	GEN format prefix	*PRIVATE	
	Distributed	*PRIVATE	

CA 2E--Fields - FLD

Panel	Change	Change Type	Override
Edit Field Details	Field name	*OBJONLY	
	Document sequence	*OBJONLY	
	Type	*PUBLIC	
	Ref type	*NONE	
	Ref field	*PUBLIC	
	Field usage	*OBJONLY	
	Internal length	*PUBLIC	
	Internal decimals	*PUBLIC	
	Data type	*OBJONLY	
	Gen name	*PUBLIC	
	K'bd shift	*PRIVATE	
	Lowercase	*PRIVATE	
	Old DDS name	*OBJONLY	
	Text	*OBJONLY	
	Left hand side text	*OBJONLY	
	Right hand side text	*OBJONLY	
	Column headings	*OBJONLY	
	Default condition	*OBJONLY	
	Check condition	*OBJONLY	
	Mandatory fill	*OBJONLY	
	Valid system name	*OBJONLY	
	Modulus 10/11	*OBJONLY	
	Edit codes Screen I/P	*OBJONLY	
	Screen O/P	*OBJONLY	
	Report	*OBJONLY	
	Translate condition values	*OBJONLY	
	External Length	*PUBLIC	
	External decimal	*PUBLIC	
	DDS name	*PUBLIC	

Panel	Change	Change Type	Override
	Prompt function	*OBJONLY	
	Display as multi-line edit	*OBJONLY	
	NPT height	*OBJONLY	
	NPT width	*OBJONLY	
Display Unreferenced Fields	Delete unreferenced fields		*DELETE
Field Mapping Function Parameters	Ctx	*OBJONLY	
	Field	*OBJONLY	
	Permit override	*OBJONLY	

CA 2E--Functions – FUN

Panel	Change	Change Type	Override
Display All Functions	Change access path	*PRIVATE	
Edit Functions	Function	*OBJONLY	
	Function type	*OBJONLY	
	Access path	*PRIVATE	
Edit Function Details	Function name	*OBJONLY	
	Source name	*GEN	
	Target HLL	*GEN	
	Text	*OBJONLY	
No panel	Default functions for a new file		*CREATE
Edit Screen Format Relations	Select record override function		*PRIVATE
Edit Screen Constant	Lines before		*PRIVATE
	Spaces before		
	Screen text		
	Length		

Panel	Change	Change Type	Override
	Constant		
Edit Function Field	Delete function field		*PRIVATE
Screen Field Mapping Parameters	Source context	*PRIVATE	
	Source field	*PRIVATE	
Create Model Version	From & to functions		*OBJONLY
Edit Function Options	Subfile select	*GEN	
	Header/Footer	*GEN	
Edit Function Parameters (see note below)	File/field	*PUBLIC	
	Access path/field	*PUBLIC	
	Passed as	*PRIVATE	
	Sequence	*PRIVATE	

Note: The changes to the Edit Function Parameters panel apply also to arrays (ARR) and fields (FLD).

CA 2E--Messages – MSG

Panel	Change	Change Type	Override
Edit Message Function	Add message		*CREATE
	Change message		*PRIVATE
	Delete message		*DELETE
Copy message	New message		*CREATE
Edit Function Parameters	File/field	*PUBLIC	
	Access path/field	*PUBLIC	
	Passed as	*PRIVATE	
	Sequence	*PRIVATE	
Create Model Versions	From & to messages		*OBJONLY

CA 2E--Component Change Processing Propagation Table

Component change processing ensures the integrity of your model by identifying which other model objects are affected and the type of change required whenever you change a model object. The component change processing propagation table shows the effect of *PRIVATE and *PUBLIC changes to a model object on other objects that use the changed object.

For more information on component change processing, refer to the Impact Analysis section in the “Managing Model Objects” chapter in this guide.

Changed Object Type	**Change Type**	**Using Object Type**						
		ACP	**ARR**	**CND**	**FIL**	**FLD**	**FUN**	**MSG**
ACP	*PUBLIC	*PUBLIC	*PUBLIC				*PUBLIC	*PUBLIC
	*PRIVATE	*PRIVATE1	*PRIVATE				*PRIVATE	
ARR	*PUBLIC						*PRIVATE2	
	*PRIVATE						*PRIVATE	
CND	*PUBLIC	*PRIVATE		*PUBLIC			*PRIVATE2	
	*PRIVATE	*PRIVATE		*PRIVATE			*PRIVATE	
FIL	*PUBLIC	*PUBLIC			*PUBLIC			
	*PRIVATE	*PRIVATE						
FLD	*PUBLIC	*PUBLIC	*PRIVATE		*PUBLIC	*PUBLIC	*PRIVATE2	
	*PRIVATE						*PRIVATE	
FUN	*PUBLIC	*PRIVATE				*PRIVATE	*PRIVATE	
	*PRIVATE					*PRIVATE	*PRIVATE	

Changed Object Type	Change Type	Using Object Type						
		ACP	ARR	CND	FIL	FLD	FUN	MSG
MSG	*PUBLIC				*PRIVATE		*PRIVATE	
	*PRIVATE							

Notes:

1. 1.The using access path is flagged *PRIVATE only when you change a physical access path.
2. 2. If the change affects the parameters of the using function, the using function is flagged *PUBLIC instead of *PRIVATE.

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y