Repository User's Guide

Version 10.3.3

SYNERGY DE

Printed: May 2016

DCN RE-01-10.3_04

Synergex
2330 Gold Meadow Way
Gold River, CA 95670 USA

http://www.synergex.com
phone 916.635.7300
fax 916.635.6549

Contents

2 Working with Structures

3 Working with Fields

7 Subroutine Library

Preface

About this manual

This guide describes the general philosophy, concepts, and usage of Synergy/DE® Repository. It includes information on setting up and maintaining your repository, as well as details on Repository utilities, Synergy Data Language, and the Repository subroutine library.

Manual conventions

Throughout this manual, we use the following conventions:

- In code syntax, text that you type is in `Courier` typeface. Variables that either represent or should be replaced with specific data are in *italic* type.
- Optional arguments are enclosed in *[*italic square brackets*]*. If an argument is omitted and the comma is outside the brackets, a comma must be used as a placeholder, unless the omitted argument is the last argument in a subroutine. If an argument is omitted and the comma is inside the brackets, the comma may also be omitted.
- Arguments that can be repeated one or more times are followed by an ellipsis…
- A vertical bar (|) in syntax means to choose between the arguments on either side of the bar.
- Data types are **boldface**. The data type in parentheses at the end of an argument description (for example, (**n**)) documents how the argument will be treated within the routine. An **a** represents alpha, a **d** represents decimal or implied-decimal, an **i** represents integer, and an **n** represents numeric (which means the type can be **d** or **i**).
- Scenarios are in *italic* type.
- To "enter" data means to type it (or highlight it, in the case of a selection window entry) and then press ENTER. ("ENTER" refers to either the ENTER key or the RETURN key, depending on your keyboard.)
- This grid indicates on which platforms and in which environments a routine, statement, etc., is supported: in traditional Synergy on Windows (WT), in Synergy .NET on Windows (WN), on UNIX (U), or on OpenVMS (V). By "supported" we mean that the item performs a useful function on that platform or environment. For example, an unsupported routine may cause a compiler error or it may just not do anything.

WT	WN	U	V

Other resources

- The Repository release notes, **REL_RPS.TXT**
- *Getting Started with Synergy/DE*
- *UI Toolkit Reference Manual*
- *Synergy DBL Language Reference Manual*
- *xfODBC User's Guide*

Product support information

If you cannot find the information you need in this manual or in the resources listed above, you can reach the Synergy/DE Developer Support department at the following numbers:

800.366.3472 (in the U.S. and Canada)
916.635.7300 (in all other locations)

To learn about your Developer Support options, contact your Synergy/DE account manager at one of the above numbers.

Before you contact us, make sure you have the following information:

- The version of the Synergy/DE product(s) you are running
- The name and version of the operating system you are running
- The hardware platform you are using
- The error mnemonic and any associated error text (if you need help with a Synergy/DE error)
- The statement at which the error occurred
- The exact steps that preceded the problem
- What changed (for example, code, data, hardware) before this problem occurred
- Whether the problem happens every time and whether it is reproducible in a small test program
- Whether your program terminates with a traceback, or whether you are trapping and interpreting the error

Synergex Professional Services Group

If you would like assistance implementing new technology or would like to bring in additional experienced resources to complete a project or customize a solution, Synergex® Professional Services Group (PSG) can help. PSG provides comprehensive technical training and consulting services to help you take advantage of Synergex's current and emerging technologies. For information and pricing, contact your Synergy/DE account manager at 800.366.3472 (in the U.S. and Canada) or 916.635.7300.

Comments and suggestions

We welcome your comments and suggestions for improving this manual. Send your comments, suggestions, and queries, as well as any errors or omissions you've discovered, to doc@synergex.com.

1

Welcome to Repository

What Is Repository?

The Repository application provides a centralized location for your data definitions. It orders and defines your data structures, files, and attributes. Repository can supply this information to UI Toolkit for processing script files and building input windows at runtime, to the Synergy™ compiler for compiling source code files, to Workbench for generating context-sensitive field lists, and to ReportWriter for creating reports. Repository data definitions are also used by xfODBC to create system catalogs and by xfServerPlus–xfNetLink applications.

Using Repository as the center of your software development environment has the following benefits:

- **Increased productivity.** Developers know where to find the data definitions, which can be accessed from your Synergy DBL source code.
- **No duplication of information.** Data is defined only once, and can be shared by all components of your application. Your data is easier to maintain without potential loss of integrity.
- **Easy maintenance.** Modifications to one definition can update all the elements of your application executable.

Repository consists of four main components:

- The **Repository user interface**, where you define your data fields, their attributes, and their organization in your database.
- The **utilities**, which enable you to generate record definition files from your repository structures, verify your repository or print it to a file, create new repositories, set the current repository, and generate and load repository definitions using Synergy Data Language.
- **Synergy Data Language**, which is a set of statements that describes the contents of a repository. You can use it to make mass modifications to your repository, convert non-Synergy/DE repositories to Synergy/DE format, and examine your repository.
- The **subroutine library**, which contains routines that enable your Synergy applications to access Repository information about structures, files, templates, fields, keys, and so forth.

Using Repository with other Synergy/DE components

Synergy DBL

You can use the REPOSITORY qualifier with the .INCLUDE compiler directive to instruct the compiler to include structures from your repository. You can either generate a definition file that contains structure definitions and .INCLUDE that file, or you can include directly from the repository, in which case the Synergy compiler creates structures that contain elements from the repository and includes them in your program. See .INCLUDE in the “Preprocessor and Compiler Directives” chapter of the *Synergy DBL Language Reference Manual* for more information about including from a repository.

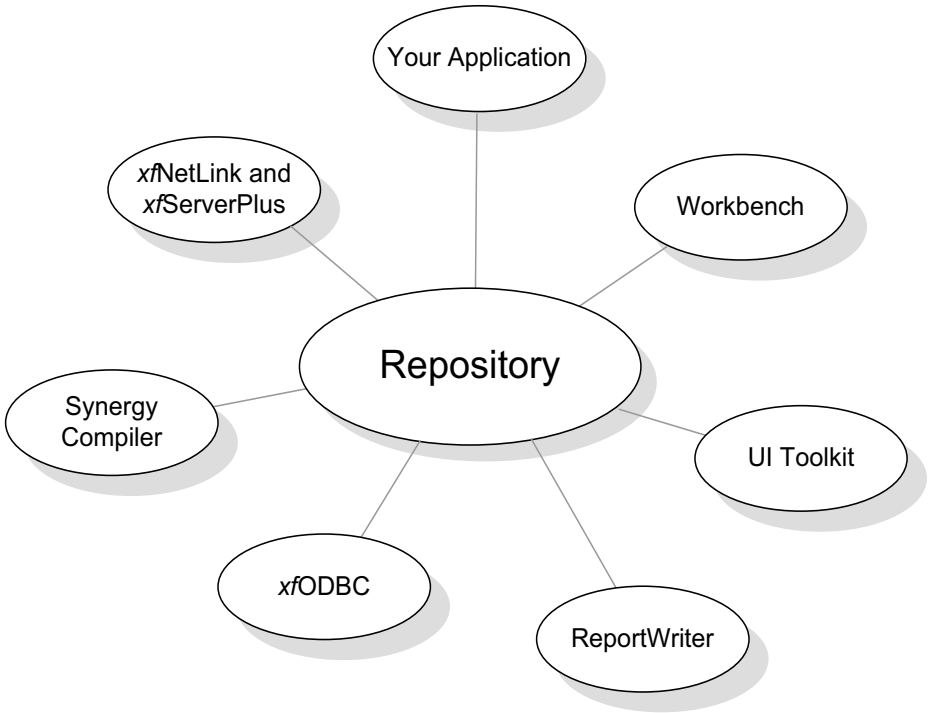


Figure 1-1. Repository provides a centralized location for data definitions.

And when you are working in Professional Series Workbench, the language-sensitive visual editor enables you to select fields from repository structures that have been .INCLUDEd in your routine.

UI Toolkit

Repository is fully integrated with UI Toolkit, enabling you to define all input field qualifiers in your repository. As a result, not only are data definitions consistent from one application to another, but the user interface is consistent as well. In addition, you don't need to duplicate information from your repository in a Toolkit window script.

The fields you define in Repository can also be read into Composer as predefined input fields. See "Defining User Interface Characteristics" in the "Setting Up Your Repository" chapter of *Getting Started with Synergy/DE*.

ReportWriter

Repository supplies information to ReportWriter for creating reports. When defining fields for use with ReportWriter, you can also define the way the field will be used and displayed in a report, the keys that determine the relationships between files, how those keys will be linked with keys from other structures, and which files use which structures. See "Defining Files for ReportWriter" in the "Setting Up Your Repository" chapter of *Getting Started with Synergy/DE*.

xfODBC

xfODBC uses repository definitions to generate system catalogs, which provide the information needed by the xfODBC driver to provide third-party ODBC access to Synergy data files. Repository includes a number of xfODBC-specific settings that enable you to control what is stored to the system catalog. For example, you can exclude or provide an alternate name for a field. See "Setting Up a Repository" in the "Preliminary Steps" chapter of the *xfODBC User's Guide* for more information.

xfNetLink and xfServerPlus

xfNetLink Java and xfNetLink .NET clients can pass repository structures defined as parameters, including structures with embedded structures (that is, groups or struct data types) or embedded arrays, as well as arrays of structures. In addition, you can pass enumerations defined in the repository as parameters or return values, as well as include them as fields in structures passed as parameters. See the xfNetLink Java Edition and the xfNetLink .NET Edition sections of the *xfNetLink & xfServerPlus User's Guide* for more information.

Your application

Any information stored in Repository can be used by your application and accessed through the Repository subroutine library. (See chapter 7, "Subroutine Library.")

Getting Started

Before you can use Repository, you must install Synergy/DE Professional Series, including Repository, on your system.

If you're a current ICS version 3 user or Repository version 6 user, you'll probably want to continue using your existing data definitions. See the release notes file **REL_RPS.TXT** for instructions on converting version 3 dictionaries or version 6 repositories to the current version. For information on new features and changes by version, see the "Repository" chapter of the *Updating Synergy/DE* manual.

Starting Repository

On…	Do this…
Windows	From within Workbench, select the Repository toolbar button or Open a Command Prompt window and enter `dbr RPS:rps`
UNIX	At the command prompt enter `dbr RPS:rps`
OpenVMS	At the command prompt enter `run RPS:rps`

Setting up your repository

In Repository, you create structure definitions and then, within those structures, you define fields and their attributes, such as field names, sizes, headings, formats, and so forth. Then you can create file definitions and assign structures to them. You can define segmented keys for a file and indicate how your data is ordered, and then link the keys for one file with the keys of another to easily access additional related information.

To maintain consistency throughout your structures, you can create template definitions that define basic field characteristics. A template can be assigned to one or more fields or to another template. For use with UI Toolkit or ReportWriter, you can create display formats for specific fields, such as phone numbers.

Figure 1-2 illustrates the steps you might follow to populate your repository. We recommend the order shown here, although you have some latitude of course. For example, as you are defining fields, you may realize a template for a particular type of field would be a useful or you might want to define aliases for structures as you go along.

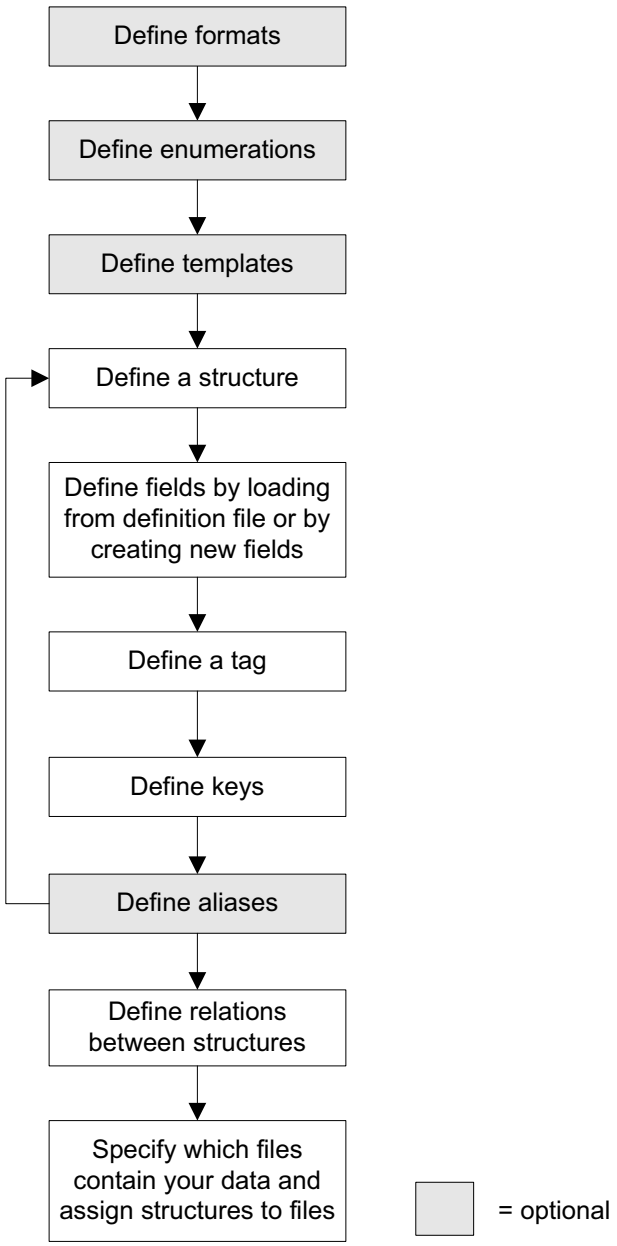


Figure 1-2. Creating your repository.

- **Define display formats for fields**

  If you know what display formats you're going to want for frequently used field types, go ahead and define them first. Displays formats are most commonly used with Toolkit and ReportWriter. You can define global formats or formats specific to a particular structure, and then select these formats when defining display information. Follow the instructions in

  "Defining Field Formats" on page 3-32
  "Display information" on page 3-12

- **Define enumerations**

  If you want to reference enumerations from fields, you must define them before defining fields. Follow the instructions in

  "Defining Enumerations" on page 3-44

- **Define templates**

  If you're going to define fields from scratch, you may want to define template fields for commonly used field types. Follow the instructions in

  "Defining Field Templates" on page 3-36

- **Define structures**

  Structures are a combination of field and key definitions, so you'll need to define structures before defining fields. Follow the instructions in

  "Defining a New Structure" on page 2-3

- **Define fields**

  If you're working from existing record definition files, you'll want to load your fields into Repository and then make sure all of your fields are defined as you'd like them to be. Follow the instructions in

  "Loading Fields from a Definition File" on page 3-30

  If you're defining fields from scratch, follow the instructions in

  "Defining a New Field" on page 3-4

- **Define tags to associate with structures**

  If you will assign multiple structures to a file, you'll need to define tags to associate with your structures so that they can be distinguished one from the other. Follow the instructions in

  "Defining Tags" on page 2-6

- **Define keys**

  Your next step is to define the keys to your data. Follow the instructions in

  "Defining Keys" on page 4-10

- **Define aliases**

  Aliases are alternate names for structures, which can be useful when including a structure in a source file. Follow the instructions in

  "Defining Aliases" on page 2-9

- **Define relations between structures**

  Once you've defined more than one structure, you can define relations between structures so that xfODBC and ReportWriter will be able to access additional information from other related files. Follow the instructions in

  "Defining Relations between Structures" on page 4-18

- **Define files and assign structures to them**

  To use xfODBC or ReportWriter, you'll also need to specify which files contain your data and assign the structures you've defined to those files. Follow the instructions in

  "Defining Files" on page 4-3
  "Assigning Structures to Files" on page 4-8

Using the Repository Interface

When you first start Repository, the Modify menu is pulled down. (See figure 1-3.) Once you select the area you want to work in (structures, files, etc.), additional menus will display.

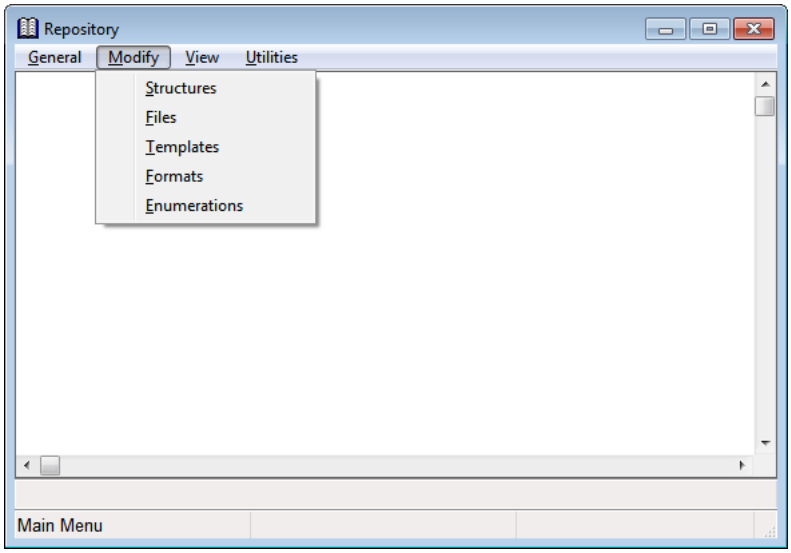


Figure 1-3. The Repository main menu.

At the very bottom of the screen is the *footer*. It contains the name of the current menu or function (on the left), processing messages (in the center), and the name of any structure, template, or other element that is currently selected (on the right).

Just above the footer is the *information line*. It contains information such as whether the "Find Entry" function is operating in forward or reverse mode and also displays a brief field-level help message when you are in an input window.

Getting information about the Repository version

To display information about your version of Repository, select General > About. The current Repository version number is displayed, along with the compile date and the versions of Synergy DBL and UI Toolkit under which Repository is running. To exit the About Repository window, press ENTER.

Making a menu selection (UNIX and OpenVMS)

You can select a menu entry using the arrow keys, quick-select characters, or shortcuts. To use the arrow keys or quick-select characters, you must first activate the menu bar by pressing the process-menu key, CTRL+P. The menu bar is deactivated after you select a menu entry. You can also deactivate it by pressing the process-menu key again.

- **Arrow keys:** Use the LEFT and RIGHT ARROW keys to move across the activated menu bar to select a menu. When you move to a menu, it drops down, displaying the entries. Use the UP and DOWN ARROW keys to move among the menu entries. Press ENTER to select a highlighted entry.
- **Quick-select characters:** A quick-select character is a single character that accesses a menu entry. When a menu is dropped down, type the quick-select character to highlight the menu entry and select it. If the quick-select character is not unique in the menu, it simply highlights the entry, and you will need to press ENTER to select the highlighted entry.
- **Shortcut keys:** A shortcut is a key or key sequence that is associated with a menu entry, and which enables you to execute a function directly without selecting it from a menu. (The exception to this rule is arrow keys: arrow keys can be used as shortcuts only when the menu is *not* enabled.) The shortcut keys are listed to the right of the menu entry and vary depending on the type of terminal. Not all menu entries have shortcuts. See "Appendix E: Distributed Shortcuts" for a list of the most common shortcuts as they are provided with your original Repository distribution. (Note that shortcuts can be reassigned by your system administrator.)

Entering data

In this manual, we refer to both the ENTER and RETURN key as ENTER. After you have finished typing data for a particular field, pressing ENTER causes the information to be accepted.

Repository converts most of the data you enter into uppercase letters. Descriptions or headings can be entered in either uppercase or lowercase.

Getting field-level help

For each input field, the information line displays a brief help message telling you what data to enter. For more detailed information, select General > Help to display a help window for the field. To close the help window, press ENTER.

Accepting or overriding default values

In some Repository fields, the program enters a default value for you. You can override the default by typing your own data, or you can accept the default by pressing ENTER. For optional fields, you can override the default and leave the field blank by pressing the spacebar or the BACKSPACE key and then pressing ENTER.

Moving between fields

On Windows, press the TAB and SHIFT+TAB keys to move between fields in a window. You can also use the mouse to move directly to a specific field.

On UNIX and OpenVMS, you can use the Previous Field and Next Field entries on the Input menu to move between fields.

Skipping a field

To skip optional fields, simply press ENTER. To remove a value from an optional field and leave it blank, press the spacebar or the BACKSPACE key and then press ENTER.

Editing a field (UNIX and OpenVMS)

Changing the cursor direction

When you begin entering text in a field, the cursor direction is set to forward. To toggle between moving forward and backward, use the Direction option (on the Edit menu). This setting does not actually change the direction in which the cursor moves as you type; rather, it is honored by some of the options on the Edit menu. (Refer to the tables on the next page for details on which options honor cursor direction.) The current direction is displayed on the right side of the information line.

Switching between insert and overstrike modes

When you begin entering text in a field, the editor is in insert mode. Any data you enter is placed at the current cursor position, and the existing data is moved to the right as you type. To change to overstrike mode, so that the data at the cursor is *replaced* with the data you type, use the Insert/Overstrike option (on the Edit menu). The current mode is displayed on the right side of the information line.

Editing a multi-line text field

When you start to type in a multi-line text field, the Edit menu appears on the menu bar. See the table below for an explanation of the available options on this menu.

To edit existing text in a multi-line field, use the right arrow key to move the cursor before you begin typing; the Edit menu will display on the menu bar, and then you can use the up and down arrow keys to move to the line you need to edit. (When you first access a multi-line text field that already has text in it, the entire field is selected. If you simply start typing, all existing text will be deleted. If this happens, you can abandon changes to recover the text.)

To…	From the Edit menu, select…
Move up one line	Up One Line. The cursor stays in the same column position.
Move down one line	Down One Line. The cursor stays in the same column position.
Move left one character	Left One Character.

To…	From the Edit menu, select…
Move right one character	Right One Character.
Move to the next word	Move One Word. Cursor direction must be set to **Forward**.
Move to first character of the current word	Move One Word. Cursor direction must be set to **Reverse**. If the cursor is already on the first character of the word, it will move to the first character of the previous word.
Move to the beginning of the line	Beginning of Line. If the cursor is already positioned at the beginning of a line, it will move to the beginning of the previous line.
Move to the end of the line	End of Line. This moves the cursor past the last character in the current line. If the cursor is already positioned after the last character on the line, movement depends on the cursor direction setting. ▸ Forward: the cursor moves to the end of the next line. ▸ Reverse: the cursor moves to the end of the previous line of text.
Re-wrap text (join lines)	Join Lines. This function rewraps the text from the cursor location to the end of the paragraph.

Deleting the data in a field

You can erase the contents of the current field by selecting Input > Clear Field.

To delete…	From the Edit menu select…
The current character	Delete Character.
From the cursor to the beginning of the next word	Delete Word. Cursor direction must be set to **Forward**. Note that this deletes all characters up to the beginning of the next word, including the current character and the space before the next word.
From the cursor to the beginning of the current word	Delete Word. Cursor direction must be set to **Reverse**. All characters to the left of the cursor in the current word are deleted.
From the cursor to the end of the line	Delete to End of Line. The end-of-line character is not deleted.

To delete...	From the Edit menu select...
From the cursor to the end of the line (including the end-of-line character)	Delete Line. Cursor direction must be set to **Forward**. If the next line contains text, that text moves up to the current line.
From the cursor to the beginning of the line	Delete Line. Cursor direction must be set to **Reverse**. Note that if the cursor is positioned on the first character of a line, this deletes the previous line of text, including the end-of-line character, and the remaining text moves up one line.

Moving between tabs in a tabbed dialog

Figure 1-4 is an example of a tabbed dialog.

- On Windows, click the tab you want to display or press CTRL+TAB to cycle through the tabs. To move back to the previous tab, press CTRL+SHIFT+TAB.
- On UNIX and OpenVMS, press TAB until the desired tab is displayed. To move back to the previous tab, press F8.

If you haven't completed the required fields on the primary tab in a tabbed dialog, you may not be able to move to the other tabs.

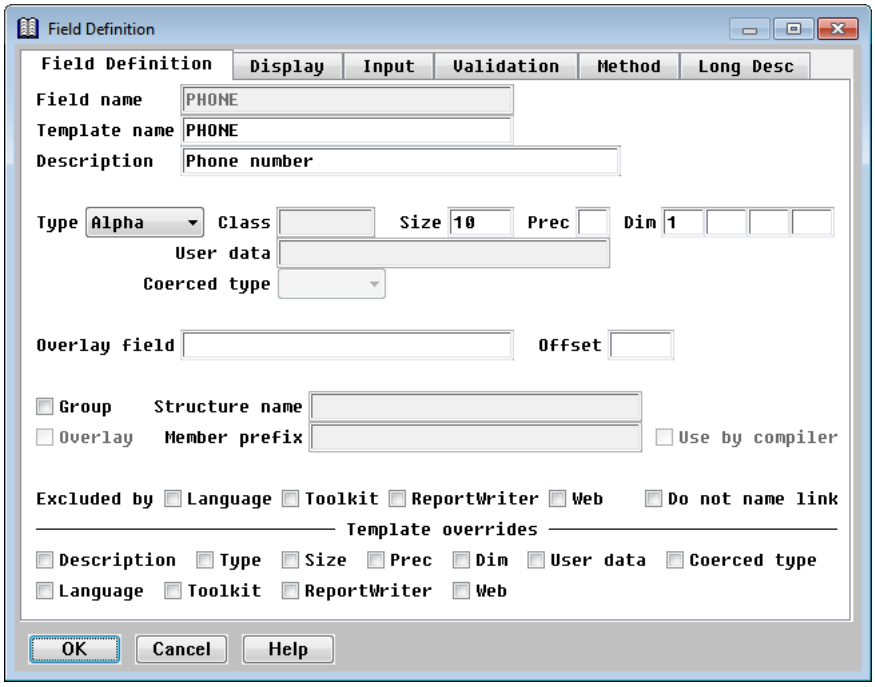


Figure 1-4. Tabbed dialog in Repository.

Abandoning changes

To reset a field to its original value, select Input > Reset Field (UNIX and OpenVMS).

To abandon changes to all fields in a window, select General > Abandon. In a tabbed dialog, this abandons all changes to all tabs. On Windows, you can also click the Cancel button to abandon changes for the current window or for all tabs in a tabbed dialog.

Using lists

Repository uses two types of lists: modifiable and non-modifiable.

Modifiable lists

Modifiable lists contain entries, such as fields or structures, that you can select and edit. You can also add, delete, and sometimes move entries in the list. The Relation Definitions list in figure 1-5 is a modifiable list.

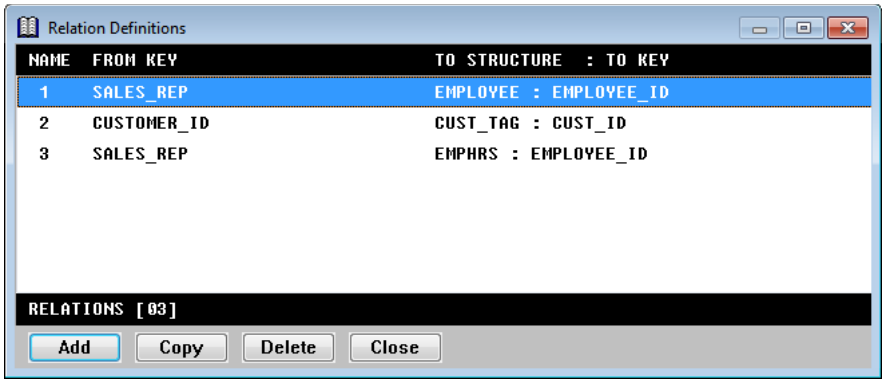


Figure 1-5. Modifiable list in Repository.

To edit an entry in a modifiable list, highlight the entry and press ENTER. On Windows, you can also double-click an entry to edit it.

To exit a modifiable list, press the Exit shortcut or (on Windows) click the Close button.

Because Repository supports multi-user access, a list that you are viewing may change when another users adds or deletes definitions. You can refresh the definition lists for structures, files, templates, formats, and enumerations by pressing F5.

Non-modifiable lists

Non-modifiable lists contain entries that you can select, but cannot change, add, or delete, such as the Available Structures list in figure 1-6. By default, non-modifiable lists display entry names. To display entry descriptions instead, select List > Toggle View. If you toggle the view for a format name, the format string is displayed. Since aliases do not have descriptions, if you toggle the view on a structure list that contains aliases, you will no longer see the aliases.

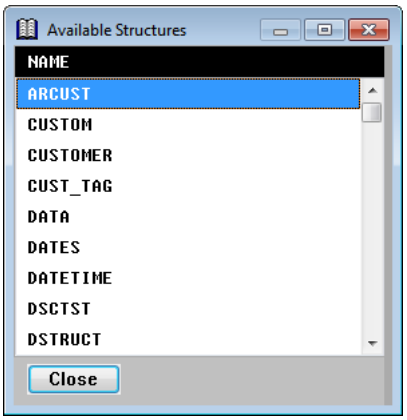


Figure 1-6. Non-modifiable list in Repository.

To select an entry in a non-modifiable list, highlight it and press ENTER. On Windows, you can also double-click.

To exit a non-modifiable list without making a selection, press the Exit shortcut or (on Windows) click the Close button.

Searching for a list entry

In both modifiable and non-modifiable lists, select Find > Find Entry to search for a list entry. At the prompt, enter the name or partial name of the entry you wish to find, and select whether you want to find items that *start* with the search text or *contain* the search text. (The "contains" option will also find entries that start with the search text.) Indicate if you want to search forward (down the list) or in reverse (up the list) and click OK to search.

If Repository finds a match, it highlights the first matching entry in the list. Select Find > Find Next to continue the search. The search will wrap when it gets to the end of the list.

Using selection windows

Like a non-modifiable list, a selection window (often referred to as a drop-down list) contains a list of items from which you can choose.

To select an item on UNIX and OpenVMS, press any key to display the list, use the arrow keys to highlight an item, and then press ENTER. On all systems, you can type the first letter of the item to highlight it, and then press ENTER. On Windows, you can also select an item with the mouse.

To redisplay a selection window on UNIX and OpenVMS once you've made a selection, press any non-shortcut key at the prompt. If the key you press is the first letter of one of the selection window entries, that entry is highlighted when the selection window is displayed.

Exiting the current function

To save your changes and exit the current input window or list, select General > Exit. You are returned to the previous window or menu. In a tabbed dialog, you can exit the window from any tab. The next time you return to that dialog, that tab will be displayed.

On Windows, you can click the OK button to save your changes and exit the current input window.

Viewing Repository definitions

The options on the View menu enable you to view Repository definitions without modifying any data. Although this mode is referred to as "view mode", you can enter new data, but all changes are ignored when you exit the window.

You can view the following definitions in view mode: structures, files, templates, formats, enumerations, fields, keys, relations, tags. You can also view long descriptions, assigned structures, and so forth. When you're in view mode, the footer displays "View…" rather than "Modify…".

1. In the View menu, select the type of definition you want to view. You'll see a reminder that you are in view mode and that changes will not be saved. Select OK.
2. In the Definitions list, highlight the definition you want to view and press ENTER. The definition is displayed for viewing.

   To quit viewing the definition and return to the current Definitions list, press the Exit shortcut.

Customizing the display

Prompts, help messages, shortcuts, menu column headings, and menu entry text reside in the window library file **rpsctl.ism**. You can modify the contents of this file to change shortcuts or translate prompts and help messages to another language.

Do *not* modify the following items:

- .INPUT, .WINDOW, and .SELECT commands
- Entry names in .ENTRY and .ITEM commands
- Field names or the **select** qualifier entry list in .FIELD commands

We also recommend that you not modify TAB and arrow key shortcuts, as these keys are explicitly referenced throughout the Repository documentation.

The window library file is located in the directory pointed to by the RPS environment variable (i.e., the main Repository directory). If RPS is not defined, Repository attempts to open the file in the current directory.

To modify the Repository window library file,

1. Make a backup copy of **rpsctl.wsc** and **rpsctl.ism** (and **rpsctl.is1** on Windows and UNIX) before you make any changes.
2. Move to the RPS directory and open the window script file **rpsctl.wsc**. (The window library file is created from this script file.)
3. Modify the **rpsctl.wsc** file as desired.
4. Run the Toolkit script compiler (Script) to create the window library file, **rpsctl.ism**. See "Compiling Scripts" in the "Script" chapter of the *UI Toolkit Reference Manual* for a more information.

Exiting Repository

There are a couple of ways to exit Repository:

- Press the Exit shortcut to back out of each function until you get to the Repository main menu. Then, press the Exit shortcut again or select General > Quit.
- Select General > Quit to exit Repository immediately—regardless of what function you're currently in.

Understanding Repository Files

A repository consists of two ISAM files: a repository main file and a repository text file. The standard names for these files are **rpsmain.ism** and **rpstext.ism**. You can name them anything you like, but see the tip below. An empty repository (using the standard filenames) is included in your distribution. You can create additional repositories using the Create New Repository utility. (See "Creating a New Repository" on page 5-22.)

Although you can name the repository main and text files anything you like, we recommend that you include "main" and "text" in the filenames. Not only does this help to identify the files as repository files, it also enables you to take advantage of the filename defaulting that occurs in all main and text file fields in Repository dialogs: After you enter or select the name of a repository main file and exit the field, Repository enters a default repository text filename by copying the main filename and changing the last occurrence of the characters "main" to "text".

Determining the repository files used

You can specify the location of your repository files using the RPSMFIL and RPSTFIL environment variables or, if you are using the standard filenames (**rpsmain.ism** and **rpstext.ism**), you can specify the directory in which the files are located with RPSDAT. By default, RPSDAT is set to point to the rps\rpsdat directory below the Synergy/DE installation directory.

Repository searches for the repository files to use as follows:

- If the RPSMFIL environment variable is defined, the main repository filename is the value of RPSMFIL. Likewise, if the RPSTFIL environment variable is defined, the text repository filename is the value of RPSTFIL.
- If RPSMFIL or RPSTFIL is not defined, Repository attempts to open the files as **RPSDAT:rpsmain.ism** or **RPSDAT:rpstext.ism**, respectively.
- If Repository can't open **RPSDAT:rpsmain.ism** or **RPSDAT:rpstext.ism**, it attempts to open **rpsmain.ism** or **rpstext.ism** in the current directory.

You can temporarily change the repository files being used while Repository is running. This does not reset RPSMFIL or RPSTFIL; the files defined by those environment variables will be used the next time you start Repository. See "Setting the Current Repository" on page 5-23.

For more information, see the environment variables RPSMFIL, RPSTFIL, and RPSDAT in the "Environment Variables" chapter of *Environment Variables & System Options*.

Temporary work files

When repository files are opened, various temporary work files (named RPS_SEQ*) are created. By default, they are created in the current directory on OpenVMS and in the location specified by the TEMP environment variable on Windows and UNIX. If TEMP is not set, the default directory is the current directory. You can specify a location for these files with the RPSTMP environment variable. See RPSTMP in the "Environment Variables" chapter of *Environment Variables & System Options*.

Record locking

Repository supports multi-user access, with locking occurring at the record level. When one user begins modifying a record, no other user can write to it. This could affect a definition, such as a structure or template, or an internal reference record that maintains information about relationships between definitions. The lock is removed as soon as the user that has the record locked saves or abandons changes.

In most cases, it will be clear why you are getting a "record locked" error, such as when you try to delete a definition that is in use by another user. If you attempt to modify a structure, file, template, format, or enumeration that is in use, you'll see a "record locked" message and be given the option to view the definition instead.

But there may be cases where it is not so clear why a record is locked. Following are a couple of examples of these less obvious locking situations:

- If you are editing a template definition, and another user is editing a structure that includes a field that references that template, a "record locked" message will display when you attempt to save your template changes.
- If you are editing a file definition, and a structure assigned to that definition is locked by another user, a "record locked" message will display when you attempt to save your file changes.

Any time you encounter a "record locked" message, you'll have to wait until the other user is done before you can complete your task. If you are attempting to save changes, and you don't want to wait, you have the option of canceling the operation by selecting the Cancel button after exiting the "record locked" message dialog.

Because other users may be adding or deleting records while you are working in the repository, definition lists could change. You can refresh the definition lists for structures, files, templates, formats, and enumerations by pressing F5 or selecting List > Refresh.

Moving Repository files

Once you set up all of your repository definitions, you can copy the repository files to other systems. This enables you to do all of your development on one system and then move your repository to other systems as necessary.

To move your repository files to another system, do either of the following:

- Copy the repository main and text files following the general guidelines for moving data files between operating systems in "Moving Database Files to Other Systems" in the "Synergy DBMS" chapter of *Synergy Tools*.
- Use the Generate Repository Schema utility and the Load Repository Schema utility. First, use the Generate Repository Schema utility to generate a Synergy Data Language description (schema) of your repository to a file. Then, copy this file to another system and run the Load Repository Schema utility to convert the contents of the Synergy Data Language file into a new repository. This method works between any two systems, regardless of operating system, because the file generated by the Generate Repository Schema utility is a text file. See "Generating a Repository Schema" on page 5-13 and "Loading a Repository Schema" on page 5-19.

Moving repository cross-reference files

If you are using a cross-reference file (see "Generating a Cross-Reference File" on page 5-24), when you move the repository you must also move the cross-reference file. To move this file, follow the general guidelines for moving data files between operating systems in "Moving Database Files to Other Systems" in the "Synergy DBMS" chapter of *Synergy Tools*.

Converting Repositories to Another Language

You can unload text from your repository so that it can be translated to another language. The manner in which the repository is divided into two files (main and text) simplifies customization for other languages: while the repository main file contains all non-textual information, the repository text file contains all text strings.

1. Once your repository is complete, copy the repository main and text files to another directory and then move to that directory.
2. Use the UNLOAD option in the **isload** utility to unload all text strings from the repository text file into a sequential file. For instructions on using this utility, see isload in the "Synergy DBMS" chapter of *Synergy Tools*.
3. Modify the text strings as desired.

Each record must remain 511 characters long, and you should *never* modify information in the first 41 bytes of the record.

4. Use the CLEAR option in **isload** to clear the repository text file.
5. Use the LOAD option in **isload** to reload the repository text file from your modified sequential file.
6. If you are using the standard filenames (**rpsmain.ism** and **rpstext.ism**), set RPSDAT to the current directory. Otherwise, set the RPSMFIL environment variable to the name of the repository main file, and set RPSTFIL to the name of the repository text file.
7. Run Repository and check all text entries (definition descriptions, user text strings, field headings, and script-related text strings).

Do not try to use a repository main file with more than one repository text file.

2

Working with Structures

Structures Overview

A structure is a record definition or compilation of field and key characteristics for a particular file or files. Each structure must have a unique name. The maximum number of structures that can be defined in a repository is 9,999.

To display the Structure Definitions list, which lists the structures currently defined in your repository, select Modify > Structures. For each structure, the following information is displayed:

STRUCTURE. The unique structure name.

DESCRIPTION. A descriptive identifier for the structure.

TYPE. The file type:

DBL	DBL ISAM
ASC	ASCII
REL	relative
USE	user defined

The total number of structures in your repository is displayed at the bottom of the list. (See figure 2-1.)

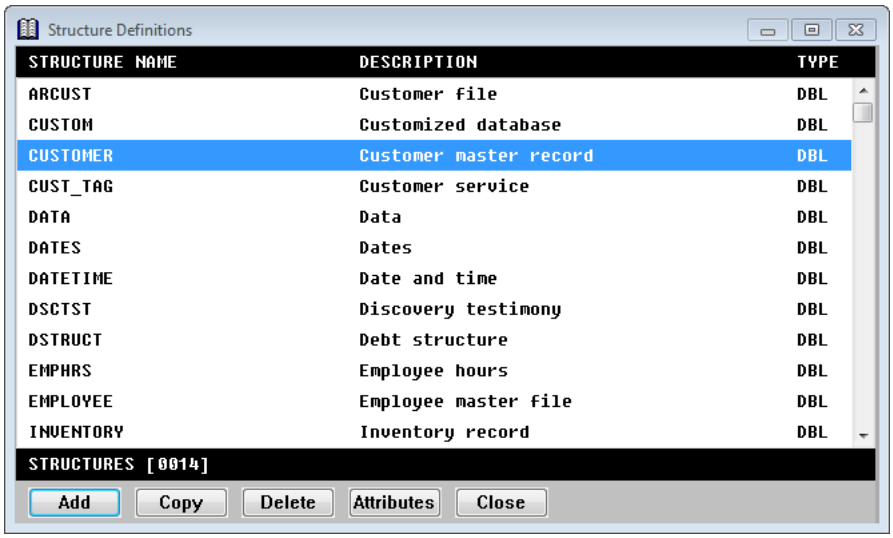


Figure 2-1. The Structure Definitions list.

Defining a New Structure

You can define a new structure from scratch or by copying and modifying an existing structure.

1. From the Structure Definitions list,

   - To define a structure from scratch, select Structure Functions > Add Structure.
   - To define a structure by copying, highlight the structure you want to copy, and then select Structure Functions > Copy Structure.

   The Structure Definition input window is displayed. (See figure 2-2.)

Figure 2-2. Defining a structure.

2. Enter or modify data in each field as instructed below.

   Structure name. Enter a unique structure name. The structure name can have a maximum of 30 characters and must begin with a letter. The remaining characters can be letters, digits, underscores (_), or dollar signs ($).

   File type. Select the file type to which you want to be able to assign this structure. The structure can be assigned only to files of this type. The available types are

   ASCII
   DBL ISAM
   RELATIVE
   USER DEFINED

   Description. Enter a description for the structure, with a maximum of 40 characters. This description is available when Repository displays a list of structures. If you are using ReportWriter, enter a unique description, so that ReportWriter can use it, along with the file description, to identify your files.

   Tag type. Tags are associated with structures and used when multiple structures are assigned to one file. The tag uniquely identifies one structure (or record type) from another. Tags are also used by xfODBC and ReportWriter to filter records. Select the tag type:

None	The structure is not tagged. (default)
Field	The structure is to be identified by specific field comparison criteria. These criteria are defined when you define the structure's attributes.

Size The structure is to be identified by its record size. (This is useful only for variable-length record files.)

See "Defining Tags" on page 2-6 for information on defining Field type tags.

3. You'll probably want to define the attributes for your new structure before you exit the Add Structure function. You can define fields, keys, relationships between structures, redisplay formats used by the fields in the current structure, tags, and aliases. Refer to these sections for more information:

   "Working with Fields" on page 3-1
   "Defining Keys" on page 4-10
   "Defining Relations between Structures" on page 4-18
   "Defining Field Formats" on page 3-32
   "Defining Tags" on page 2-6
   "Defining Aliases" on page 2-9

   If your structure has a file type of relative, Repository has already created one key for you. This key defines the relative record number as the access method. See "Defining Keys" on page 4-10 for more information about relative file keys.

4. Exit the window to save the new structure.

   If you've selected any of the attribute functions from the menu, you are prompted

   Structure has been modified. Do you want to save changes?

   Select Yes to save your new structure definition and all its attributes in the repository. If you select No, the new structure is not saved.

Assigning a long description to a structure

You can assign an 1,800-character description to a structure. This enables you to store more detailed information about the structure and its use.

1. In the Structure Definitions list, highlight the structure to which you want to assign a long description.
2. Select Structure Functions > Edit Long Description.
3. Enter a long description.
4. Exit the window to save your changes.

Assigning a user-defined text string to a structure

You can associate a 60-character user-defined text string with a structure to store additional information you want to access at run time with the Repository subroutine library.

1. In the Structure Definitions list, highlight the structure to which you want to assign a user-defined text string.
2. Select Structure Functions > Edit User Text.
3. Enter a user text string.
4. Exit the window to save your changes.

Defining Tags

A tag is associated with a structure and uniquely distinguishes one structure (or record type) in a file from another. You will want to use structure tags when you've assigned multiple structures to one file or when multiple record types are stored in one file. (For information on assigning a structure to a file see "Assigning Structures to Files" on page 4-8.) A structure tag can be either the record size or a particular field in the structure and its associated comparison values.

xfODBC and ReportWriter use structure tag information to filter records when I/O is performed to a file. When processing a structure that contains tag information, xfODBC and ReportWriter test each record in the file against the tag criteria as it is read.

Creating a record size tag

You can use record size tags when the records in your file are different sizes (i.e., they can be distinguished one from the other based solely on size).

1. From the Structure Definitions list, highlight the structure for which you want to create a tag and press ENTER. The structure's definition is displayed.
2. In the Structure Definition dialog, select **Size** in the Tag type field.
3. Exit the window to save your changes.

Creating a field type tag

Field type tags enable you to use a field to distinguish records.

1. In the Structure Definitions list, highlight the structure for which you want to create a tag.
2. Select Structure Functions > Edit Attributes.
3. Select Attributes > Tags. The Tag Definitions list (see figure 2-3) displays the following information for each tag criterion:

   NAME. The unique tag name.

   FIELD. The name of the field used in the comparison criterion, optionally preceded by a connector to the previous criterion.

   VALUE. The comparison operator and value used in the comparison criterion.

   The total number of tag definitions (comparison criteria) for this structure is displayed at the bottom of the list.

Figure 2-3. The Tag Definitions list.

4. Define a new tag from scratch or by copying and modifying an existing one. If tag definitions already exist, new tags are inserted below the highlighted entry.

- To define a tag from scratch, select Tag Functions > Add Tag.
- To define a tag by copying, highlight the tag you want to copy and select Tag Functions > Copy Tag.

The Tag Definition input window is displayed. (See figure 2-4.)

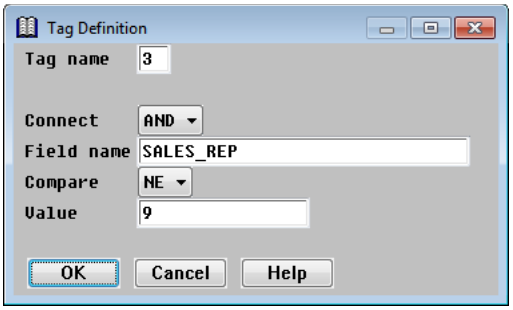


Figure 2-4. Defining a structure tag.

5. Enter data in each field as instructed below.

Tag name. The tag name is the way the tag is identified. A tag must have a unique, numeric name within the current structure. A default tag name is displayed; you can change it if desired.

Connect. If this is not the first tag criterion and you want this structure tag to be a range of values (for example, > = 10 and < = 15), select AND or OR to join two comparison values.

Field name. Enter a field name from the current structure. To display a list of available field names, select Edit Tag Functions > List Selections.

Compare. Select one of the comparison operators from the displayed list:

EQ	Equal to
NE	Not equal to
LE	Less than or equal to
LT	Less than
GE	Greater than or equal to
GT	Greater than

Value. Enter a specific value with which to compare the contents of the field.

6. Exit the window to save your new tag definition. The tag type of the structure is set to **Field** when you save your changes.

Although you can define up to 10 tag criteria, ReportWriter supports only 2 criteria that test the same field.

Modifying a tag

1. From the Tag Definitions list, highlight the tag you want to modify and press ENTER. The Tag Definition input window displays the selected tag definition.
2. Modify data as necessary. You can modify all fields except Tag name. If you need assistance with particular fields, see step 5 on page 2-7.
3. Exit the window to save your changes.

Reordering tags in the Tag Definitions list

1. Highlight the tag you want to move.
2. Select Tag Functions > Reorder Tags. The highlighted tag is enclosed in square brackets ([ ]).
3. Use the UP and DOWN ARROW keys to move the bracketed tag to another location in the list.
4. Select Reorder Tags again to exit move mode. The tag is inserted at the new location.

Deleting a tag

1. Highlight the tag in the Tag Definitions list.
2. Select Tag Functions > Delete Tag.
3. At the prompt, select Yes to delete the tag or No to cancel the deletion. The structure tag type will be set to None when you save your changes.

Defining Aliases

An alias is simply an alternate name for a structure. A single structure can have more than one alias. The alias can be used instead of the structure name when including a structure in a source file or when referencing a structure for a field that is a struct data type. All structure names, whether real or alias, must be unique. (Consequently, when you copy a structure, its aliases are *not* copied.)

For example, aliases are useful when you are converting an application that uses short, cryptic identifier names to use the Repository. To use longer, more meaningful names in the repository, you can create aliases to simplify updating your Synergy code. Aliases can also be useful when you want to define structfields, but still need that repository structure to be included as a record in your Synergy code. You can create an alias and reference that in your structfield definition.

When you include a structure from the repository in your Synergy code, the compiler first searches for a structure that has the name specified in the .INCLUDE statement. If it can't find one, it searches for an alias with the same name.

To display the Alias Definitions list:

1. Highlight the structure in the Structure Definitions list.
2. Select Structure Functions > Edit Attributes.
3. Select Attributes > Aliases to display the Alias Definitions list.

Defining an alias

1. From the Alias Definitions list, select Alias Functions > Add Alias.
2. Enter a name for the alias. The name must be unique among all structure and alias names in the repository. The alias name can have a maximum of 30 characters and must begin with a letter. The remaining characters can be letters, digits, underscores (_), or dollar signs ($).
3. Exit the Alias Definition window to save the alias and return to the Alias Definitions list. Aliases are added alphabetically to the list.

You can also define an alias for a field, but you must use Synergy Data Language to do so. See "ALIAS" on page 6-9.

Deleting an alias

You cannot delete an alias if it is referenced by a structfield.

1. Highlight the alias in the Alias Definitions list.
2. Select Alias Functions > Delete Alias.
3. At the prompt, select Yes to delete the alias or No to cancel the deletion.

Modifying a Structure

Generally speaking you can modify all fields for a structure except the structure name. See below for other restrictions that may apply depending on how the structure relates to other components in your repository, such as files and keys.

1. Highlight the structure in the Structure Definitions list and press ENTER.
2. Modify data in the fields as necessary. See step 2 on page 2-3 for detailed information on the fields.

   If the structure has already been assigned to a file, you can't modify the file type because a structure's file type must always match the file type of *all* files to which it is assigned. If you attempt to modify the file type, an error message is displayed when you attempt to exit the window. You must disassociate the structure from all files before you can modify the file type. (See "Disassociating a structure from a file" on page 4-9.)

   Even if the structure is not assigned to a file, other restrictions may apply. You cannot change a file type to RELATIVE if access keys already exist for the structure, because relative structures can have only one access key (the record number). If the original file type is relative and the record number key has not been deleted, the only file type to which you can change is USER DEFINED. The reverse situation is also true: if the user-defined file type was originally relative and you haven't deleted the record number key, the only file type to which you can change is RELATIVE.
3. If you want to modify the attributes for the current structure, select Structure Functions > Edit Attributes. You can also edit a structure's attributes from the Structure Definitions list by highlighting that structure and pressing the "Edit Attributes" shortcut.
4. Exit the window to save your changes and return to the Structure Definitions list.

Deleting a Structure

A structure can be deleted only when all of the following conditions are true:

- The structure is not assigned to a file.
- The structure is not referenced within an implicit group definition or as a structfield.
- The structure is not referenced by its alias as a structfield
- None of the structure's keys are being used in a relation defined by another structure.
- None of the structure's fields are being used in a key defined by another structure.

When you delete a structure, the structure and all of its field, key, relation, format, alias, and tag definitions are deleted.

1. Highlight the structure in the Structure Definitions list.
2. Select Structure Functions > Delete Structure.
3. At the prompt, select Yes to delete the structure or No to cancel the deletion.

3

Working with Fields

Fields Overview

Field definitions are associated with each structure. The order in which you specify the fields determines the order in which they will exist within the structure. (You can reorder fields if necessary; see "Reordering fields in the Field Definitions list" on page 3-3.) The maximum number of fields that can be defined in one structure is 999.

Before defining a field, you may want to first define field templates and formats. A template is a set of field characteristics that can be assigned to one or more field definitions; it can also be assigned to other template definitions. A field format defines the way the field displays when used in UI Toolkit and ReportWriter.

To define an enumerated field, first create the enumeration definition (see "Defining Enumerations" on page 3-44), and then, select Enum as the field type and select the enumeration by name when you define the field.

You can also load fields from a definition file into an empty structure. Once loaded, you can make changes to the fields or add fields manually. See "Loading Fields from a Definition File" on page 3-30 for more information.

The Field Definitions list

To display the Field Definitions list,

1. Highlight the structure in the Structure Definitions list. (You can also define fields while you're defining a new structure by select the Attributes button in the Structure Definition dialog.)
2. Select Structure Functions > Edit Attributes.
3. Select Attributes > Fields to display the Field Definitions list. (See figure 3-1.)

The following information is listed for each field:

FIELD NAME. The unique field name.

TYPE. The data type: A (alpha), AS (autoseq), AT (autotime), BL (Boolean), BN (binary), D (decimal), DT (date), E (enumeration), I (integer), S (structure), TM (time), U (user).

SIZE. The data size.

PREC. The precision value. (The number of characters to the right of the decimal point in an implied-decimal field.) This column does not contain a value if the number of characters is zero.

DIMENSION. The number of dimensions in the array, if the field is an array. Arrays can have up to four dimensions. Entries in this column are displayed in the form [*n,n,n,n*] where *n* represents the number of elements for that dimension.

OVERLAY. Displays "Y" if the field defines an overlay.

GROUP. Displays "Y" if the field defines a group.

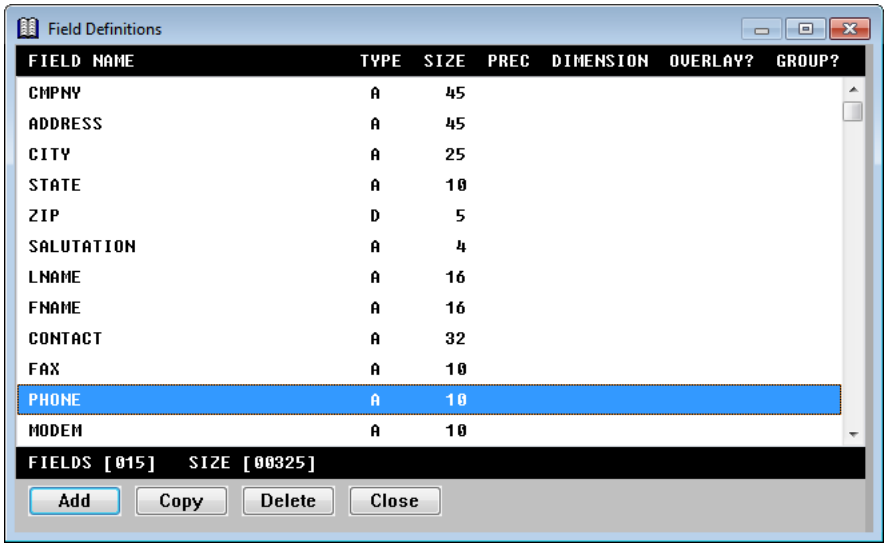


Figure 3-1. The Field Definitions list.

The total number of fields in the structure and the structure (record) size are displayed at the bottom of the list. (Arrays and overlay field definitions each count as one field.) If the structure contains any group fields, both the number of fields at the structure level and the total number of fields in the structure are displayed.

Reordering fields in the Field Definitions list

1. Highlight the field you want to move in the Field Definitions list.
2. Select Field Functions > Reorder Fields.

   The highlighted field is now enclosed in square brackets ([ ]).
3. Use the UP and DOWN ARROW keys to move the bracketed field to another location in the list. (You are not allowed to reorder the fields in a way that makes an overlay specification become invalid.)
4. Select Reorder Fields again to exit move mode. The field is inserted at the new location.

Defining a New Field

You can define a new field from scratch or by copying and modifying an existing field. If field definitions already exist, new definitions are inserted below the highlighted entry.

This is a different place

From the Field Definitions list,

- To define a field from scratch, select Field Functions > Add Field or Add Group (if you want to add a group field).
- To define a field by copying, highlight the field you want to copy, and then select Field Functions > Copy Field.

The Field Definition window displays with tabs on which you can define field information.

- **Field Definition** tab, for defining basic field information; see "Basic field information" on page 3-4.
- **Display** tab, for defining how you want the field to display in a Toolkit input window or ReportWriter report; see "Display information" on page 3-12.
- **Input** tab, for defining how field input is handled in a Toolkit input window; see "Input information" on page 3-18.
- **Validation** tab, for defining how field input is validated in a Toolkit input window; see "Validation information" on page 3-22.
- **Method** tab, for associating methods (that are called by Toolkit) with fields; see "Method information" on page 3-26.
- **Long Desc** tab, for assigning a long description to a field; see "Assigning a long description to a field" on page 3-29.

Basic field information

1. Enter or modify data in each field as instructed below.

   Field name. Enter a field name. The field name is used to identity the field in your definition file and program, and it is one of the ways the field is identified in ReportWriter. The field name must be unique within the current structure or group. (See Group on page 3-9 for more information.) The field name can have a maximum of 30 characters and must begin with a letter. The remaining characters can be letters, digits, underscores (_), or dollar signs ($). See also Alt name on page 3-15.

   Template name. If desired, enter the name of a template to use to create the specified field. Up to 6,000 fields can use the same template. To display a list of available templates, select Edit Field Functions > List Selections.

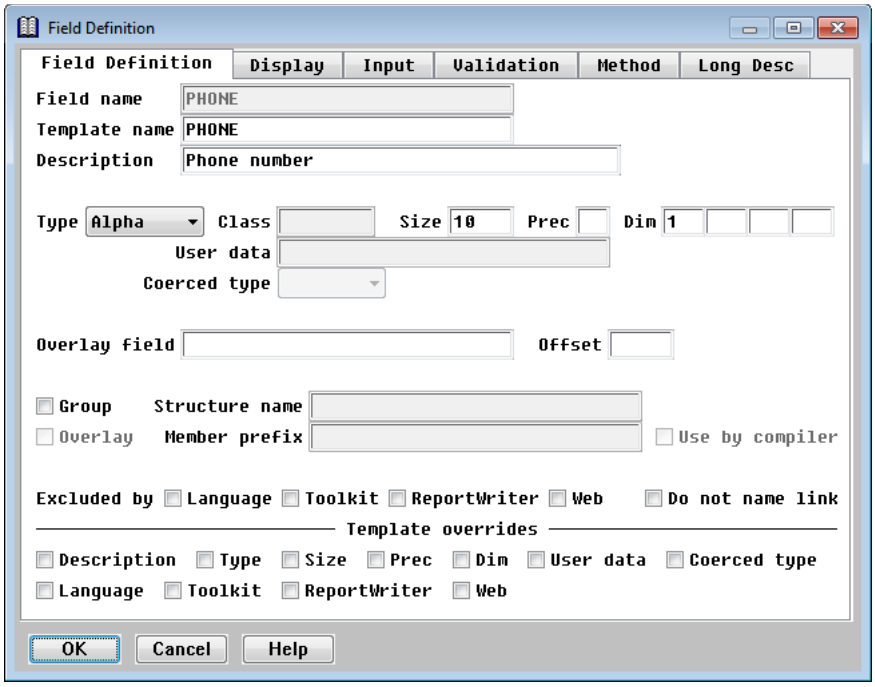


Figure 3-2. Defining basic field information.

All attributes of the template are copied to the current field, including display, input, validation, and method information. You can override any of the template attributes simply by specifying new values. If the template is later modified, only the attributes that have not been overridden are copied to the field. The check boxes at the bottom of the window indicate the template attributes that have been overridden. If an attribute is checked, the value for that attribute has overridden the value that came from the template.

Repository sets the template override flags when you leave the Field Definition tab, but you can also set them manually. You might want to do this when an attribute matches the template and you don't want it to be changed later if the template changes. By default, these fields are read-only. To modify them, select Edit Field Functions > Access Template Overrides. This menu entry is a toggle: select it a second time and the override fields revert to read-only. The Access Template Overrides setting remains in effect for *all* field and template definitions until you change it.

Description. Enter a description for the field, which a maximum of 40 characters. We recommend that you make the description unique because it can be used to identify fields in ReportWriter and Repository. The description also appears as the comment for the field when a definition file is generated by the Generate Definition File utility. In addition, if the structure that this field belongs to is included in an xfNetLink Java JAR file or xfNetLink .NET assembly, this description is included in the generated source code as a comment for the property or field.

Type. Select the type of data the field will contain:

Alpha
Decimal
Integer
Date
Time
User
Binary
Boolean
Enum
Struct
AutoSeq
AutoTime

If you select **Date**, **Time**, or **User**, the cursor moves to the Class field. If you select **Enum**, the cursor moves to the Enumeration field (see Enumeration on page 3-7). If you select **Struct**, the cursor moves to the Structure field (see Structure on page 3-7).

In Synergy DBL, the Binary data type is treated as an alpha.

In xfNetLink Java (when **genjava** is run with the **-c 1.5** option) and xfNetLink .NET. a Binary data type field in a structure is converted to a byte array on the client, and can be used, for example, to store an RFA. For xfNetLink Synergy clients, a Binary data type field is converted to a string.

In xfODBC, a Binary data type field is described as a binary field (SQL_BINARY). This is also true of a User type field with a class of binary (see Class below), but in this case you can use the routines for user-defined data types in xfODBC to manipulate the data read from the ISAM file and return it as a binary field to the ODBC-enabled application.

Class. If you selected **Date** or **Time** in the Type field, specify the storage format in the Class field:

- Date fields

YYMMDD	two-digit year, month, day
YYYYMMDD	four-digit year, month, day
YYJJJ	two-digit year, Julian day
YYYYJJJ	four-digit year, Julian day
YYPP	two-digit year, period
YYYYPP	four-digit year, period

- Time fields

HHMM	hour, minute
HHMMSS	hour, minute, second

If you selected **User** in the Type field, specify the user subtype in the Class field. User subtypes are used by the xfODBC user-defined processing routines and are available in the **gs_inpfld** structure within UI Toolkit's user-defined processing routines. Subtypes also affect data type mapping for xfNetLink Java and xfNetLink .NET; see "Appendix B: Data Type Mapping" in the *xfNetLink & xfServerPlus User's Guide* for details. The available subtypes are

Alpha
Numeric
Date
Binary

Additional date storage formats are supported by xfODBC. See "Appendix B: Date and Time Formats" in this manual for more information. See the note above for information on the binary subtype.

User data. If you selected **User** in the Type field, specify a string of up to 30 characters to identify your user-defined data type.

In a UI Toolkit input window, a user type field flags the runtime input processor to call the ECHKFLD_METHOD, EDSPFLD_METHOD, and EEDTDSP_METHOD subroutines for additional processing. The user data string is passed to these subroutines to be used as a control code. The user subtype (class) is available in the **gs_inpfld** structure.

User type fields also flag ReportWriter to call a user-overloadable subroutine (for example, RPS_DATA_METHOD, which formats the data for display). See the "Customizing ReportWriter Routines" chapter of the *ReportWriter User's Guide* for more information about the user-overloadable subroutines called by ReportWriter.

User type fields also flag xfODBC to call user-overloadable subroutines to process the data for those fields. The user data string is passed to these routines. See the "Creating Routines for User-Defined Data Types" chapter of the *xfODBC User's Guide* for more information.

To save your user data string, press ENTER.

Enumeration. If you selected **Enum** in the Type field, enter the enumeration name or select Edit Field Functions > List Selections and choose it from the list.

The Enum data type is not supported by UI Toolkit. To use an enumerated data field with an allow list or selection list or window, use the Enumerated field as described in Enumerated on page 3-25.

Structure. If you selected **Struct** in the Type field, enter the structure (or alias) name or select Edit Field Functions > List Selections and choose it from the list. If you use the list, note that aliases are listed at the bottom, after the real structure names. See the note under Group on page 3-9 for an explanation of the difference between referencing a structure as a Struct data type vs. as an implicit group.

Coerced type. If the structure that this field belongs to is included in anxfNetLink Java JAR file or an xfNetLink .NET assembly, you can optionally specify a non-default data type for the field to be coerced to on the client side. Type coercion is available when Type is one of the following: Decimal, Integer, Date, Time, User. Note the following:

- For Decimal (with or without precision) and Integer types, select **Default** to use the default xfNetLink type mapping.
- Date types can be coerced when the format is one of the following: YYMMDD, YYYYMMDD, YYJJJ, YYYYJJJ.
- User types can be coerced only when the user subtype (i.e., the Class field) is Date and the User data field contains ^CLASS^=YYYYMMDDHHMISS or ^CLASS^=YYYYMMDDHHMISSUUUUUU (case sensitive).
- For Date, Time, and User types, the default coerced type is DateTime.

See "Appendix B: Data Type Mapping" in the *xfNetLink & xfServerPlus User's Guide* for more information on data type mapping and coercion in xfNetLink.

Size. Enter the maximum number of characters the field can contain. Note the following:

- The maximum size of an alpha, binary, or user field is 99,999.
- The maximum size of an implied-decimal field is 28.
- Valid sizes for integer fields are 1, 2, 4, and 8.
- If the data type is date or time, the size is automatically set when you select a storage format and cannot be modified.
- If the data type is Boolean or Enum, the size is automatically set to 4 and cannot be modified.
- If the data type is Struct, the size is automatically set when you select the structure name.
- If the data type is AutoSeq or AutoTime, the size is automatically set to 8 and cannot be modified.
- The size of a group field is optional.

Precision. If the field is implied-decimal, enter the number of characters to the right of the decimal point. This value must be between 1 and 28, inclusive, and must be less than or equal to the size of the field.

Dim1–4. If the field is an array, enter the number of elements in each dimension. The maximum number of dimensions is 4. The maximum number of elements per dimension is 999.

Overlay field. If the field is an overlay to another field or fields, enter the name of the overlaid field at which the overlay begins or select Edit Field Functions > List Selections and choose it from the list. The overlaid field must be a field that precedes the current field. For example, the year, month, and day might be overlays for a date field.

Offset. If you want the overlay to begin at an offset position, enter the number that should be added to the starting position of the field being overlaid (as specified by Overlay field above). The default offset is 0.

For example, if you wanted to overlay the DATE field, you would enter "DATE" in the Overlay field, and your overlay offset might be **0** for the year, **4** for the month, and **6** for the day. This would make the year start at position 1 (0 added to a starting position of 1 equals 1), the month start at position 5, and the day start at position 7.

Group. Select this field to indicate the field is a group. Group is set by default if you selected Add Group when on the Field Definitions list.

There are two types of groups: explicit and implicit. An *explicit* group is one whose members are defined explicitly within the Field Definitions list. An *implicit* group is one whose members are defined implicitly by referencing another structure. The members of that structure define the members of the group. To define an implicit group, see the Structure name field below.

You can reference a structure as a field either as an implicit group or as a Struct data type. In both cases, you are required to enter the structure name and the field is maintained as a reference to that structure. Any modifications made to a referenced structure affect all fields that reference it.

The difference between the two is that a structure referenced as an implicit group is represented as a group in your Synergy code and in definition files; it uses the group keyword and you will see the group members (fields) listed. In contrast, a structure referenced as a Struct data type is represented as a structfield in code and definition files. That is, the data type of the field is the name of the referenced structure and you do not see the members listed. (See "Structure" in the "Defining Data" chapter of the *Synergy DBL Language Reference Manual* for more information on structfields.)

You can specify a structure by its alias when referencing a structure as a structfield, but not when referencing it as an implicit group.

In Toolkit, xfODBC, and xfNetLink, a Struct data type is treated the same as an implicit group.

After setting the Group field, select Edit Field Functions > Edit Group Members to define explicit group members. (On Windows, you can also click the drilldown button.) This displays another Field Definitions list in which you can define the group members. See "Modifying group members" on page 3-48 for more information about defining and modifying groups and for the rules regarding groups.

If Size is not specified for a group field, its size is determined by the size of its members.

Overlay. If the field is a Group, select this field if you want the group to overlay the last non-overlay field or group.

Structure name. To specify an implicit group (see the Group field above), enter the structure name in this field or select Edit Field Functions > List Selections and choose it from the list.

Once you have entered a structure name in this field, you can select Edit Field Functions > Edit Group Members to view the members of the group. (On Windows, you can also click on the drilldown button to the left of the Structure name field.) Implicit groups are maintained as a reference to a structure; therefore, the list of members is read-only. The members are not copied into the group definition. See also the note above.

If explicit group members exist, the Structure name field is disabled. To change a group from explicit to implicit, you must first delete all explicit group members.

Member prefix. If Group is set, you can specify an optional prefix to be added to group member names when they are accessed by Toolkit and xfODBC. The member prefix name must begin with a letter. The remaining characters can be letters, digits, underscores (_), or dollar signs ($).

In Repository and Synergy DBL, field (and group) names have to be unique only within their parent. In DBL, path specifications can be used to uniquely reference group members. UI Toolkit and xfODBC, however, do not have a way to uniquely identify group members. The Member prefix field enables you to construct unique field names for Toolkit and xfODBC access. (Repository does no validation to ensure that the specified prefix is sufficient for unique field access.)

Use by compiler. Set this option to indicate that the Member prefix specified will be added to all group member fields when referenced by the Synergy compiler. This optional behavior enables you to use consistent names throughout your Synergy and UI Toolkit programs.

Excluded by Language. This value determines whether a field is available to the Synergy compiler. Select this option if you *do not* want the field to be available to the compiler. Excluded by Language is cleared by default, which means the field *will* be included when using the .INCLUDE compiler directive to reference the structure to which this field belongs, and *will* be included in any definition files generated by the Generate Definition File utility. This feature is useful when your repository contains overlay fields defined solely for the purpose of referencing group elements from within ReportWriter.

Excluded by Toolkit. This value determines whether a field is available to UI Toolkit. Select this option if you *do not* want to be able to reference the field from Toolkit. Excluded by Toolkit is cleared by default, which means the field can be referenced by the Script compiler, Composer, and the IB_FIELD subroutine.

Excluded by ReportWriter. This value determines whether a field is available in ReportWriter as a selectable field. Select this option if you *do not* want this field to be selectable in ReportWriter. Excluded by ReportWriter is cleared by default, which means the field can be selected for inclusion in a report. This flag can also be honored when generating a system catalog in xfODBC; see "Setting catalog generation options" in the "Preliminary Steps" chapter of the *xfODBC User's Guide* for details on including and omitting fields.

Excluded by Web. This value determines how the field is treated by xfNetLink. The Excluded by Web flag should be used *only* to control how fields in an overlay are handled. If this field is not part of a structure that contains overlays, do not select this option. Select Excluded by Web if you *do not*

want this template field to be included in a Synergy JAR file or assembly. Excluded by Web is cleared by default, which means that all fields are included in the Synergy component. For details on using this flag to control how overlays are handled, see "Passing Structures as Parameters" in the "Preparing Your Synergy Server Code" chapter of the *xfNetLink & xfServerPlus User's Guide*.

Do not name link. ReportWriter can use name links you establish in Repository to access related files. The value in this field determines whether the field is name linked to its template (if one exists). By default, Repository will use the name of the template to generate name links. Select this field is you want Repository to use the field's name when generating name links. (See also "Generating a Cross-Reference File" on page 5-24.)

Template overrides. If the current field references a template, the Template overrides section indicates the template attributes that are overridden. If these fields are read-only, select Edit Field Functions > Access Template Overrides to make them active. (See the description of Parent template on page 3-38 for more information about template override flags.)

2. To define display, input, validation, or method information, or to assign a long description to the field, go to the desired tab or select the entry from the Field Functions menu. Refer to the following sections for instructions:

   "Display information" on page 3-12
   "Input information" on page 3-18
   "Validation information" on page 3-22
   "Method information" on page 3-26
   "Assigning a long description to a field" on page 3-29 (Long Desc tab)

3. When you have finished defining the field, exit the window to save the new field definition and return to the Field Definitions list.

   When you exit, Repository validates the display, input, validation, and method information. If an error exists, correct it and then exit the window again.

   The new field definition is highlighted in the field list, and the number of fields displayed at the bottom of the list is updated. The SIZE field at the bottom of the window is updated to reflect the new size of the structure.

   If the field being modified has group members, but the Group field is not set, before returning to the Field Definitions list, you are prompted

   Field "*NAME*" is a group. Clearing the "Group" field will delete all group members. Do you want to continue?

   If you select Yes, the group members are *not* saved with the field definition. If you select No, you are returned to the Field Definition tab.

Display information

The Display tab enables you to define how you want the field to display in a Toolkit input window or ReportWriter report.

When the Toolkit script compiler accesses a field in the repository, the default is to use all display information stored with that field. However, you can still use .FIELD qualifiers in your script file to override one or more of the repository attributes. (A field's type, size, precision, and dimensional data cannot be overridden.) For more information on the use of these script qualifiers, see .FIELD in the "Script" chapter of the *UI Toolkit Reference Manual*.

1. From the input window where you're defining the new field or template, go to the Display tab or select Edit Field Functions > Edit Display Information or Edit Template Functions > Edit Display Information. (If you're in the Field Definitions or Template Definitions list instead of an input window, select Field Functions > Edit Display Information or Template Functions > Edit Display Information.) (See figure 3-3.)

If the current field or template uses a global format that no longer exists, a warning message is displayed. You must select another format, or no format, at the Format name prompt before your modifications can be validated.

2. Enter or modify data as instructed below. The name of the field or template cannot be modified.

We've modified display information for the PHONE field as an example. (See figure 3-3.) Let's assume that we want the data in the PHONE field to have the field header "Telephone number," have the display format (@@@)@@@-@@@@, and be left-justified in a report. We highlighted **PHONE** *in the Field Definitions list and pressed the "Edit Display Information" shortcut.*

If you are defining display information for a template, any references to "fields" in the remainder of this section refer to fields that use the current template.

Position. This value specifies whether a position is associated with this input window field and, if so, whether the position is absolute or relative. Select the position:

None	No position is associated with the field. (default)
Absolute	Designates a specific position.
Relative	Specifies the number of rows and columns that the current position will change.

If you select None, the Row and Col fields are cleared, and the cursor moves to the Field pos field.

If a prompt is defined for this field, the prompt begins at the specified position. If no prompt is defined, the input field itself begins at the specified position. If no position is specified, the position of the prompt defaults to one column past the last position used in the window.

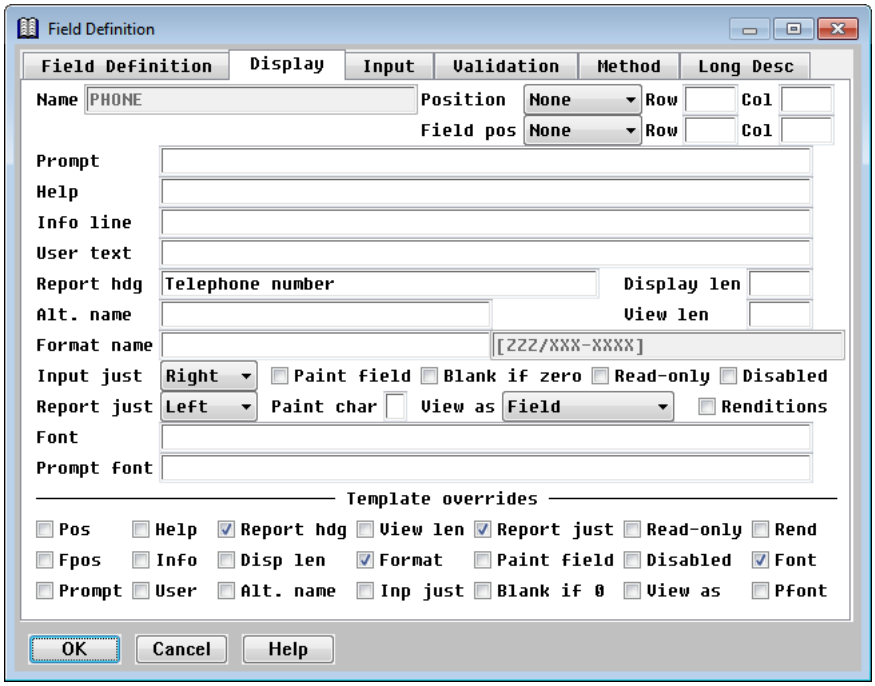


Figure 3-3. Defining display information.

Row. If you selected a position of **Absolute**, specify the row position. If you selected a position of **Relative**, specify the number of rows that the current position will move. This relative movement value can be positive or negative.

Col. If you selected a position of **Absolute**, specify the column position. If you selected a position of **Relative**, specify the number of columns that the current position will move. This relative movement value can be positive or negative.

Field pos. The field position specifies the position of a field independent of its prompt. If no field position is specified, the position of the field defaults to the position of any prompt, with the length of the prompt added to the column position. Select a field position:

None	No position is associated with the field. (default)
Absolute	The row and column values are the absolute coordinates for the field relative to the input window.
Relative	The row and column values specify a change from the last position occupied. If you've specified a prompt, the change is relative to the prompt position.

If you select None, the following Row and Col fields are cleared and the cursor moves to the Prompt field.

Row. If you selected a position of **Absolute**, specify the row position. If you selected a position of **Relative**, specify the number of rows that the current position will move. This relative movement value can be positive or negative.

Col. If you selected a position of **Absolute**, specify the column position. If you selected a position of **Relative**, specify the number of columns that the current position will move. This relative movement value can be positive or negative.

In the next four fields (Prompt, Help, Info line, and User text), you can enter only a 60-character text string, although the maximum string length is 80 characters. To enter a longer string, select Edit Field Functions > Edit Entire Text. If you later need to edit a string that exceeds 60 characters, you must do it using the Edit Entire Text function. (If you edit it in the field, the portion that is not displayed will be lost. Similarly, deleting the 60 characters that are displayed in the field will delete the entire string.)

Prompt. Specify a fixed or variable prompt. A *fixed prompt* is a string that is displayed in the input window to prompt the user for input. The prompt string must include any spacing that you want between the prompt and the input. If the prompt string contains trailing spaces, or if you want to use only digits in the prompt string, enclose the string in quotation marks.

To specify a *variable prompt* enter a numeric value without quotation marks. This value is used by the Toolkit I_PROMPT subroutine as the length of the variable prompt supplied to it. See I_PROMPT in the "Input Routines" chapter of the *UI Toolkit Reference Manual* for more information.

Help. Specify a help identifier. The help identifier is passed as an argument to the Toolkit EHELP_METHOD subroutine. See EHELP_METHOD in the "Environment Routines" chapter of the *UI Toolkit Reference Manual* for more information.

Info line. Specify a text string that will display on the information line when input is being processed for this field. Information line strings that contain trailing spaces must be enclosed in quotation marks. When processing an input window, if you don't specify an information line string, the previous information line remains in effect. If new information line text is displayed, the information line reverts to the text it contained previously once field input occurs.

User text. Specify a user-defined text string associated with this input field, if desired. The size of the user field data is fixed by the size of the string used during the input window's compilation; writing data out to the string won't change the size of the string. You can access this string at runtime with the Toolkit I_USER subroutine. User text strings that contain trailing spaces must be enclosed in quotation marks.

Report hdg. Modify the column heading text if desired, and press ENTER. Maximum size is 40 characters.

This field is used as the column heading by ReportWriter. If no heading exists, ReportWriter uses the field description as a column heading. If no description exists, ReportWriter uses the field name. Note the following:

- Modifying a heading does not affect a field that has already been selected for printing in existing ReportWriter reports. However, it does affect any *new* selections of that field for inclusion in a report.
- If you want the heading to be split into multiple lines when it is printed, insert a caret character (^) into the heading string to designate a line break. You can include up to two carets, for a total of three heading lines. If you want the heading to contain an actual caret character, precede the caret with a backslash (\). Two backslashes in a row cause a backslash character to be printed in the heading.

The Report hdg field is also used by xfNetLink .NET when this structure is included as a DataTable class in a Synergy assembly. The value in this field is used for the column caption in the DataTable. If no heading exists, the field name is used as the column caption. For more information, see "Using DataTables" in the "Calling Synergy Routines from .NET" chapter of the *xfNetLink & xfServerPlus User's Guide*.

Display len. Enter the maximum number of characters that you want to be displayed in the field. This value overrides the default display length computed by UI Toolkit. Valid values are 0 through 65,535.

Display length cannot be specified if the "View as" field is set to Radio buttons or Check box, or the Selections field (on the Validation tab) is set to Window or List, or if the field is a text field (a multi-dimensional alpha field).

Alt name. Enter an alternate name to be used instead of the field name. The alternate name can have a maximum of 30 characters and must begin with a letter. The remaining characters can be letters, digits, underscores (_), or dollar signs ($).

In xfODBC, the value in this field can be used for the column name for the field in the ODBC system catalog; if no alternate name is specified, the field name is used as the ODBC column name. See "Renaming columns for clarity" in the "Preliminary Steps" chapter of the *xfODBC User's Guide* for more information.

In xfNetLink Java and xfNetLink .NET, this value can be used for the property name in the Java class or C# class; if no alternate name is specified, the field name is used as the property name. See "Passing Structures as Parameters" in the "Preparing Your Synergy Server Code" chapter of the *xfNetLink & xfServerPlus User's Guide*.

View len. Enter the number of characters that you want to use to determine the width of the field on the screen (i.e., the width of the area on the screen that will display data for the field). This value overrides the default width as determined by Toolkit. Valid values are 0 through 9,999. View length cannot be specified if the "View as" field is set to Radio buttons or Check box.

On Windows, this value is multiplied by the width of the sizing character for the current font to determine the field width. If view length is less than display length, the field will be scrollable up to the display length. On UNIX and OpenVMS, the width of the field is set to the number of characters specified. If view length is less than display length, the field will be truncated to fit in the view length.

Format name. If you want this field to be associated with a format, enter the name of an existing format. If a format was previously assigned to this field or template, its name is displayed, with the actual format displayed next to it. To display a list of available formats, select Edit Field Functions > List Selections or Edit Template Functions > List Selections. See "Defining Field Formats" on page 3-32 for details on the format feature.

Input just. Select the justification for the text in the field when this field is used in a Toolkit input window:

Left	Left-justification; the default for alpha, date, time, and user type fields.
Right	Right-justification; the default for decimal, implied-decimal, and integer fields. Not valid for text fields (multi-dimensional alpha fields).
Center	Center-justification. Not valid for numeric or text fields (multi-dimensional alpha fields).

Report just. Select the justification for the data within the column when this field is used in a ReportWriter report. The default is left-justified for alpha, user, date, and time fields and right-justified for decimal, implied-decimal, and integer fields. Center-justification is allowed only for alpha and user fields.

The width of the column in which the data will be justified is determined by either the column heading or the field length, depending on which is longer. (The field length is either the length of the format, if one exists, or the length of the field.)

Paint field. Set this field to indicate that the specified paint character (or blanks) are used to "paint" the empty field where the user has typed input. If Paint field is not set, any paint character specified for the input window will be used.

Paint char. If you set the Paint field, you can optionally specify a display character to "paint" the empty field to indicate where the user types input. Repository does not have a default paint character.

Blank if zero. If Set this field to indicate that a decimal, implied-decimal, or integer field should remain blank when the user enters a value of **0**.

Read-only. Set this field to indicate that the field is read-only. Unlike a disabled field, a read-only field can receive focus. Read-only can be set only when View as is set to Field (see below). If the data type is AutoSeq or AutoTime, the Read-only flag is set automatically, as these data types are always read-only.

The Read-only field is honored by xfNetLink Java and xfNetLink .NET when the structure is included in a JAR file or assembly. For xfNetLink Java, you must use the **genjava -ro** option to indicate that you want the read-only flag honored. For xfNetLink .NET, you must choose to generate structure members as properties rather than fields for the read-only flag to take effect. Properties that are flagged as read-only will have a "get" method, but no "set" method.

Disabled. Set this field to indicate that the field is disabled and cannot receive focus.

View as. Select how you want the field displayed in the input window:

Field	Display as a standard input field. (default) To display the field as a selection list field, define an associated selection list or window.
Radio buttons	Display as a set of radio buttons on Windows. On UNIX and OpenVMS, display as a selection list field. This option is allowed only if the field has an associated selection list or window.
Check box	Display as a check box on Windows. On UNIX and OpenVMS, display as a one-character field ("X" if non-zero, or space if zero). This option is allowed only when the field's data type is decimal, implied-decimal, or integer. Since a check box is implicitly an enumerated field, you cannot select **Check box** when the Enumerated value is **Yes**.

See "Validation information" on page 3-22 for more information on defining selection lists and windows.

Renditions. Set this option to define a color palette number and one or more attributes for the field. By default, an input field inherits its renditions from the input window to which it belongs. To override the window's color or attributes, click on Renditions or press the spacebar. When you select this field, a pop-up window displays the following fields:

Color	Enter a color palette number between 1 and 16 (inclusive) to associate with the field. This overrides any color specified for the input window.
Attributes	Select this option to override the input window's attributes for the field. *All* four attributes will override the corresponding input window attributes. When Attributes is set, the other four check boxes are enabled.
Highlight	If set, the input field will be highlighted.
Reverse	If set, the input field will be in reverse video.
Blink	If set, the input field will blink (on Windows, it displays in italic typeface).
Underline	If set, the input field will be underlined.

Exit the window to save the rendition modifications and return to the Display tab.

Font. Enter the name of the font to use for displaying the contents of the input field on Windows. This name must be defined in the font palette (specified in the [FONTS] section of the **synergy.ini** or **synuser.ini** file).

Prompt font. Enter the name of the font to use for displaying the input field prompt on Windows. This name must be defined in the font palette (specified in the [FONTS] section of the **synergy.ini** or **synuser.ini** file).

The font names in the Font and Prompt font fields can each have a maximum of 60 characters and must begin with a letter. The remaining characters can be letters, digits, underscores (_), or dollar signs ($).

Template overrides. If the current field references a template, this section indicates the template attributes that are overridden. If these fields are read-only, select Edit Field Functions > Access Template Overrides to make them active. (See the description of Parent template on page 3-38 for more information about template override flags.)

3. Continue to define field information on the other tabs or, if you are finished defining the field, exit the window to save the field definition and return to the Field Definitions list.

Input information

The Input tab enables you to define how you want field input to be handled when the field is used in Toolkit input windows.

When the Toolkit script compiler accesses a field in the repository, the default is to use all input information stored with that field. However, you can still use .FIELD qualifiers in your script file to override one or more of the repository attributes. (A field's type, size, precision, and dimensional data cannot be overridden.) For more information on the use of these script qualifiers, see .FIELD in the "Script" chapter of the *UI Toolkit Reference Manual*.

1. From the input window where you're defining the new field or template, go to the Input tab or select Edit Field Functions > Edit Input Information or Edit Template Functions > Edit Input Information. (If you're in the Field Definitions or Template Definitions list instead of an input window, select Field Functions > Edit Input Information or Template Functions > Edit Input Information.) (See figure 3-4.)
2. Enter or modify data as instructed below. The name of the field or template cannot be modified.

If you are defining input information for a template, any references to "fields" in the remainder of this section refer to fields that use the current template.

Uppercase. If Uppercase is set, lowercase input characters are converted to uppercase on alpha and user type fields. All input is displayed in uppercase. Uppercase can be set only when the field's data type is alpha or user. Note that only input is uppercased; if your program loads data directly into this field, you must uppercase that data manually.

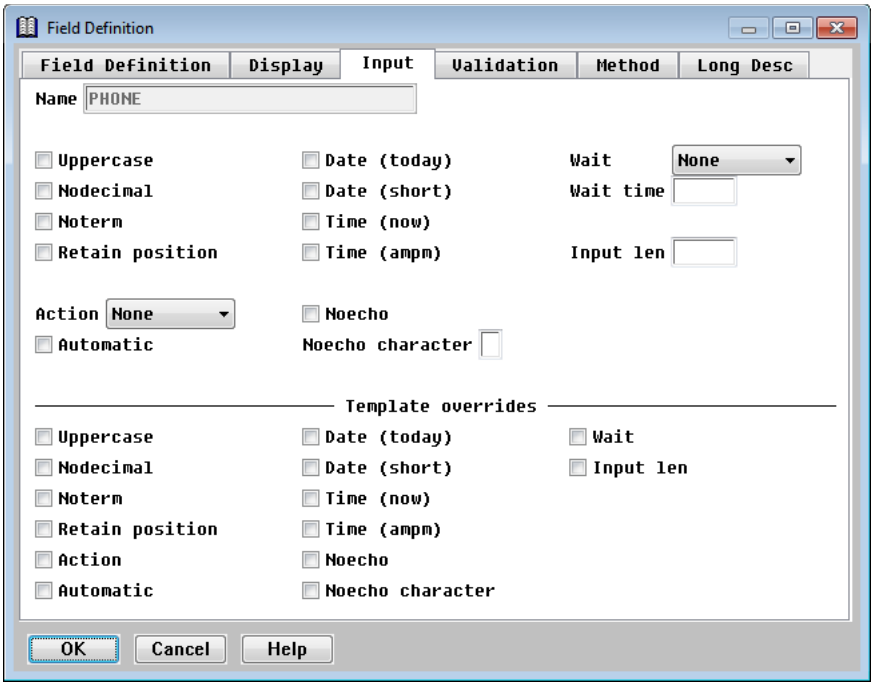


Figure 3-4. Defining input information.

Nodecimal. If Nodecimal is set, the user does not need to type a decimal point when placing input in a numeric field. The input is stored right-justified, with the number of decimal places inferred from the storage format. An explicit decimal point overrides the Nodecimal value. For example, an entry of 12.34 always yields a result of 12.34, regardless of whether Nodecimal is set. Nodecimal can be set only when the field's data type is decimal, implied-decimal, or integer.

Noterm. If Noterm is set, the field is automatically terminated when filled. Field input is normally terminated only when you press ENTER; if a field is filled and the user tries to enter more characters, the terminal bell rings for each extra character typed. However, when you set Noterm, the field is automatically terminated when the field is filled. (In other words, the user doesn't have to press ENTER.)

Retain position. If Retain position is set, the position within the text field is to be retained on subsequent re-entry to the field, until the field is reinitialized or redisplayed. This field is valid only for text fields (multi-dimensional alpha fields).

Action. The Action value determines whether a default value should be displayed in the field or whether a default action should occur. Select an option:

None	No default action occurs. (default) The Automatic field (see below) is set to No and the cursor moves to the Date (today) field.
Default	The specified default value is displayed in the input field. A second window in which you can enter a default value for the input window field is displayed. (See the description of the Default value field below.)
Copy	Copies the value from the data area that corresponds to this field (as defined by the structure) if the field is empty.
Increment	If the user does not enter a value the first time a decimal, implied-decimal, or integer field is processed, the last value in the field plus one is used.
Decrement	If the user does not enter a value the first time a decimal, implied-decimal, or integer field is processed, the last value in the field minus one is used.

Default value. If you selected **Default** in the Action field, an input window is displayed so you can enter a default value for the field. Enter a default value that will be automatically displayed in the field and press the ENTER key. The user can accept the default by pressing ENTER, edit the default, or type a new value. For date and time fields, the default value must be specified in storage form (Class), rather than input or display form. For example, to specify "January 31, 1999" as the default value for a date stored in YYYYMMDD form, you would specify "19990131".

Automatic. If set, the default action specified in the Action field—Default, Copy, Increment, or Decrement—occurs automatically when an empty field is processed. Automatic can be set only when you've specified a default action.

Date (today). If set, Date (today) defaults the date to today's date if the user presses ENTER without entering anything in a blank date field.

Date (short). If set, Date (short) displays a date type field in fewer than the normal 11 characters. A short period date (a date type field with a storage format of *YYPP* or *YYYYPP*) has a display length of five characters. All other short dates have a display length of eight characters.

Time (now). If set, Time (now) causes the time to default to the current system time if the user presses ENTER without entering anything in a blank time field.

Time (ampm). If set, Time (ampm) specifies that the display format of a time field is 12-hour time, followed by an AM or PM indicator.

Noecho. Set Noecho to prevent the text that the user types from being displayed in the field. If you want, you can specify a character to be displayed for each typed character. (See the Noecho character field, below.) Noecho can be set only if the field's data type is alpha or user.

Noecho character. If you set Noecho and the field's data type is alpha or user, you can optionally specify a display character to be displayed for every character of input the user types. The display character also fills any remaining spaces at the end of the input after the user presses ENTER, so other users can't see how many characters were entered.

Wait. The Wait value specifies whether a time-out will occur if the user doesn't complete input in an allotted amount of time. Any Wait value overrides the value in the Toolkit **g_wait_time** field. Select a Wait option:

None	No wait specified. The Toolkit **g_wait_time** field defines the global wait time. (default)
Wait time	Toolkit should wait the specified number of seconds for input processing to complete.
Immediate	Immediate user response is required; do not wait.
Global	The global wait time (defined by the Toolkit **g_wait_time** field) should be used.
Forever	Designates that Toolkit should wait until input processing is complete.

Wait time. If you selected **Wait time** in the Wait field, enter the number of seconds to wait for input processing to be complete.

Input len. Enter the maximum number of characters you want the user to be able to enter in the field. This value overrides the default input length computed by Toolkit. Valid values are 0 through 65,535.

Input length cannot be specified if the "View as" field (on the Display tab) is set to Radio buttons or Check box, or the Selections field (on the Validation tab) is set to Window or List, or if the field is a text field (a multi-dimensional alpha field).

Template overrides. If the current field references a template, the Template overrides section indicates the template attributes that are overridden. If these fields are read-only, select Edit Field Functions > Access Template Overrides to make them active. (See the description of Parent template on page 3-38 for more information about template override flags.)

3. Continue to define field information on the other tabs or, if you are finished defining the field, exit the window to save the field definition and return to the Field Definitions list.

Validation information

Validation information is the data associated with a field or template that affects how field input is validated when the field is used in Toolkit input windows.

When the Toolkit script compiler accesses a field in the repository, the default is to use all validation information stored with that field. However, you can still use .FIELD qualifiers in your script file to override one or more of the repository attributes. (A field's type, size, precision, and dimensional data cannot be overridden.) For more information on the use of these script qualifiers, see .FIELD in the "Script" chapter of the *UI Toolkit Reference Manual*.

1. From the input window where you're defining the new field or template, go to the Validation tab or select Edit Field Functions > Edit Validation Information or Edit Template Functions > Edit Validation Information. (If you're in the Field Definitions or Template Definitions list instead of an input window, select Field Functions > Edit Validation Information or Template Functions > Edit Validation Information.)
2. Enter or modify data as instructed below. The name of the field or template cannot be modified.

If you are defining validation information for a template, any references to "fields" in the remainder of this section refer to fields that use the current template.

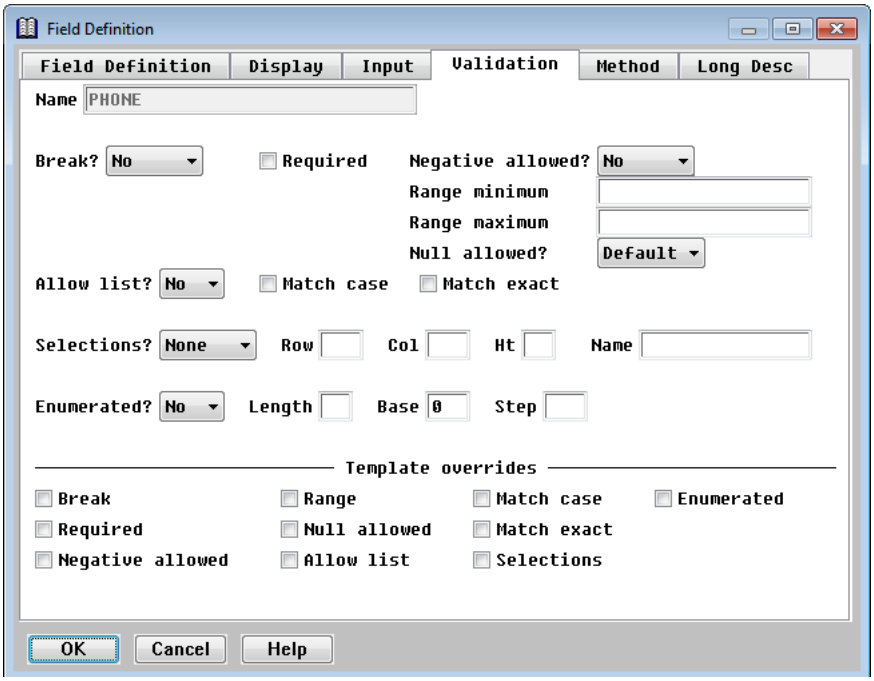


Figure 3-5. Defining validation information.

Break. Break triggers a break in input set processing on a field. Sometimes you'll want to interrupt input set processing on a field, perform external operations, and then continue with the input set. Select a break option:

No	No break occurs. (default)
Yes	A break occurs after input to this field has been processed (that is, whenever the field's data changes).
Always	A break always occurs, regardless of whether input has been processed (for example, even if the field is accessed via the I_NEXT/I_PREV menu entries).
Return	A break occurs when you press ENTER, but not when the field is accessed via the I_NEXT/I_PREV menu entries.

Required. Set this option to indicate that a non-blank alpha or non-zero numeric entry is required in the field.

Negative allowed. The value in this field determines whether negative values are allowed on decimal and implied-decimal fields. Select an option:

No	Negative numbers are not allowed as input. (default)
Yes	Negative numbers are allowed as input. The user can place a negative or minus sign (–) either before or after a number. The size of the input field is one character larger than the size of the data field.
Only	Only negative numbers are allowed as input.
OrZero	Only negative numbers or 0 are allowed as input.

Range minimum. You can specify a range of values for decimal, implied-decimal, integer, date, and time fields. Enter the minimum value for the range in this field. The range minimum must be less than or equal to the range maximum. You cannot specify a range value in conjunction with an allow list or a selection list or window. For date and time fields, Range minimum and Range maximum must be specified in storage form (Class), rather than input or display form. For example, to specify "June 1, 1900" as the range minimum for a date stored in YYYYMMDD form, you would specify "19000601".

Range maximum. If you are specifying a range of values for the field, enter the maximum value for the range in this field. The range maximum must be greater than or equal to the range minimum. See the Range minimum field above for more information.

Null allowed. This field indicates whether a field can be null. It is used by xfODBC to determine the null property for the column in the system catalog. If the data type is AutoSeq or AutoTime, this field is set to No. For details see "Preventing null updates and interpreting spaces, zeros, and null values" in the "Preliminary Steps" chapter of the *xfODBC User's Guide*.

Allow list. An allow list specifies the valid entries for the field. Set this value to Yes if the field has a list of valid entries associated with it; the default is No. If you select Yes, you may also want to select Match case and/or Match exact; see below for details.

If you select **Yes** for Allow list, a window will display in which you can enter the list entries. See Allow List/Selection List Entry below for instructions. If you select No, the cursor moves to the Selections field. You cannot specify an allow list in conjunction with selection lists or with the Noecho attribute.

Allow List/Selection List Entry. If you selected **Yes** in the Allow list field, specify the entries that you want to appear in the Allow List/Selection List Entry window. You can enter a maximum of 99, each with a maximum length of 80 characters. The entry input window displays the entry number to the left of the data entry field. Type the entry text in the field, then select Edit Field Functions > Edit Next Entry or Edit Template Functions > Edit Next Entry (or press CTRL+L for either fields or templates) to enter another entry. To edit the previous entry, select Edit Previous Entry. To define blanks as an allowable entry, you must enclose them in quotation marks (for example, “ ”).

If your entry in the Allow List/Selection List Entry window is longer than 40 characters, it will wrap. When a word wraps to the second line, it may leave several spaces at the end of the first line. These spaces will become part of the allow list entry. To avoid these unwanted spaces in the entry, when you reach the end of the first line, type a space, and then continue typing the word on the second line. (The space you type at the end of the line only serves to break the word in the Allow List/Selection List Entry window; it will not become part of the allow list entry.)

When you have finished specifying allowable entries, press the Exit shortcut. If this is an alpha or user field, the cursor moves to the Match case field. If this is not an alpha or user field, the cursor moves to the Selections field.

Match case. If you selected **Yes** in the Allow list field, select Match case to specify that the value in an alpha or user field must match the case of a specified allowable entry. For example, the entry “no” matches with “no,” but it does not match with “NO” or “No.” If Uppercase is set (on the Input tab), Match case is ignored. Match case can be set only when the field’s data type is alpha or user.

Match exact. If you selected **Yes** in the Allow list field and the field’s data type is alpha or user, select Match exact to specify that the alpha or user field input must match all characters in the specified allowable entry. If Match exact *is not set*, Toolkit utilities look for a match based on the shortest string. For example, if you are looking for a match with “Ann,” the utilities will find a match with “Ann,” “Anne,” and “Annette.” If Match exact *is set*, the input must match the allowed qualifier exactly for the full length of the input. For example, “Ann” will match *only* with “Ann.”

Selections. Select how you want selection windows placed on the screen:

None	No selection windows are placed on the screen. (default) If you select None, the Row, Col, Ht, and Name fields are cleared, and the cursor moves to the Enumerated field.
Window	When this input field is processed, a selection window is placed on the screen. You must also specify the position for the window with the Row and Col fields and the name of the window in the Name field (see below). Window is similar to List, except that the selection window already exists.

List — When this input field is processed, a selection window is placed on the screen. You must specify the position for the placement of the window, along with one or more text entries that should appear in the window. A second input window, in which you can enter a selection window entry for this field, is displayed.

Allow List/Selection List Entry. If you selected **List** in the Selections field, specify the list entries that you want to appear in the window. You can enter a maximum of 99, each with a maximum length of 80 characters. The entry input window displays an entry number to the left of the data entry field. Type the entry text in the field, then select Edit Field Functions > Edit Next Entry or Edit Template Functions > Edit Next Entry to enter another entry. To edit the previous entry, select Edit Previous Entry. Entries may include leading blanks, but to define a completely blank entry, enclose the blanks in quotation marks (for example, “ ”). When you have finished specifying entries, press the Exit shortcut.

If your entry in the Allow List/Selection List Entry window is longer than 40 characters, it will wrap. When a word wraps to the second line, it may leave several spaces at the end of the first line. These spaces will become part of the allow list entry. To avoid these unwanted spaces in the entry, when you reach the end of the first line, type a space, and then continue typing the word on the second line. (The space you type at the end of the line only serves to break the word in the Allow List/Selection List Entry window; it will not become part of the allow list entry.)

Row. If you selected **List** or **Window** in the Selections field, enter the screen row, relative to the beginning of the data field, at which the upper-left corner of the selection window should be placed.

Col. If you selected **List** or **Window** in the Selections field, enter the screen column, relative to the beginning of the data field, at which the upper-left corner of the selection window should be placed.

Ht. If you selected **List** in the Selections field, you can optionally enter the maximum number of rows in the selection window. Data is organized by column, top to bottom and then left to right. For example, if the height is 3 and there are eight entries, the window will contain three columns (with three entries, three entries, and two entries). If you don’t specify a value in the Ht field, the height of the window is the total number of entries.

Name. If you selected **Window** in the Selections field, enter the name of an existing selection window.

Enumerated. Select Yes to specify an enumerated data field type. An enumerated data field returns a decimal value for a displayed text entry. This field is valid only when used in conjunction with a Toolkit allow list, selection list, or selection window (including both existing windows and windows built on-the-fly with the Toolkit S_SELBLD subroutine). An enumerated field must be a numeric data type, and it must be large enough to handle the largest number that might be returned.

The default Enumerated value is No. If you select Yes, the cursor moves to the Length field. You must specify the Length, Base, and Step values.

Length. If you selected **Yes** in the Enumerated field, enter the length of the displayed field. (Note that the length of the displayed field and the length of the actual input field are not necessarily the same.)

Base. If you selected **Yes** in the Enumerated field, enter the return value assigned to the first item in the allow or selection list.

Step. If you selected **Yes** in the Enumerated field, enter the value added to each successive item in the allow or selection list.

Template overrides. If the current field references a template, the Template overrides section indicates the template attributes that are overridden. If these fields are read-only, select Edit Field Functions > Access Template Overrides to make them active. (See the description of Parent template on page 3-38 for more information about template override flags.)

3. Continue to define field information on the other tabs or, if you are finished defining the field, exit the window to save the field definition and return to the Field Definitions list.

Method information

The Method tab enables you to associate method names with a field or template. These methods are called by Toolkit at the appropriate time.

When the Toolkit script compiler accesses a field in the repository, the default is to use all method information stored with that field. However, you can still use .FIELD qualifiers in your script file to override one or more of the repository attributes. (A field's type, size, precision, and dimensional data cannot be overridden.) For more information on the use of these script qualifiers, see .FIELD in the "Script" chapter of the *UI Toolkit Reference Manual*.

On Windows, you can launch Workbench from this tab to define methods for field or template definitions. See "Launching Workbench from the Method tab" on page 3-28.

1. From the input window where you're defining the new field or template, go to the Method tab or select Edit Field Functions > Edit Method Information or Edit Template Functions > Edit Method Information. (If you're in the Field Definitions or Template Definitions list instead of an input window, select Field Functions > Edit Method Information or Template Functions > Edit Method Information.) (See figure 3-6.)
2. Enter the method name in the appropriate field, as described below.

   On Windows, you can click the drilldown button to enter the default method name. (The default name is *Fieldname_Methodname*, e.g., PHONE_ARRIVE.) If you have Workbench installed, see "Launching Workbench from the Method tab" on page 3-28.

If you are specifying methods for a template, any references to "fields" in this section refer to fields that use the current template.

Arrive method. Specify the name of the subroutine (method) to be called before the field is processed by the Toolkit I_INPUT subroutine.

Leave method. Specify the name of the subroutine (method) to be called after the field is processed by the Toolkit I_INPUT subroutine.

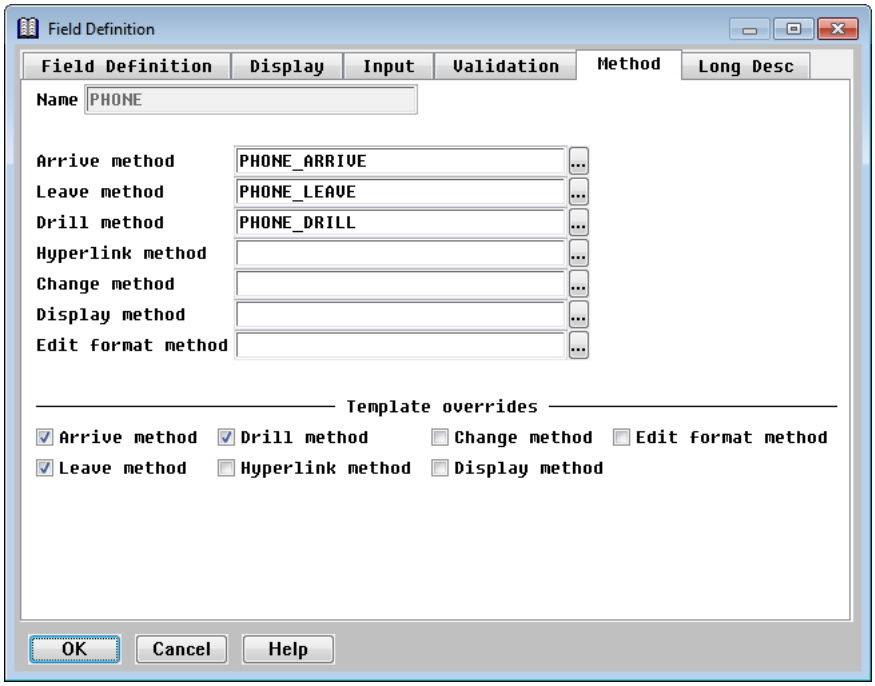


Figure 3-6. Defining method information.

Drill method. Specify the name of the subroutine (method) to be called when either a drilldown button is clicked or the I_DRILL menu entry is selected. On Windows, if a drill method is specified, a drilldown button is placed to the right of the input field. When the user clicks the button, the drill method is invoked.

Hyperlink method. Specify the name of the subroutine (method) to be called when either the prompt text is clicked or the I_HYPER menu entry is selected. On Windows, if a hyperlink method is specified and the field is a member of an input set being processed by the Toolkit I_INPUT subroutine, any prompt text associated with the field will be highlighted. When the user clicks on the highlighted text, the hyperlink method is invoked.

Change method. Specify the name of the subroutine (method) to be called after this field is validated by the Toolkit I_INPUT subroutine.

Display method. Specify the name of the subroutine (method) to be called whenever the field is about to be displayed by Toolkit. This method is called after Toolkit has formatted the display according to its own rules. Display method cannot be specified when the View as value is **Radio buttons** or **Check box**, or when a selection list or window has been specified.

Edit format method. Specify the name of the subroutine (method) to be called by Toolkit whenever text in the field is being formatted for editing purposes. This method is called after Toolkit has formatted the display according to its own rules. Edit format method cannot be specified when the View as value is **Radio buttons** or **Check box**, or when a selection list or window has been specified.

3. If the current field references a template, update template overrides if necessary. The Template overrides section of the Method tab indicates the template attributes that are overridden. If these fields are disabled, select Edit Field Functions > Access Template Overrides to enable them. (See the description of Parent template on page 3-38 for more information about template override flags.)
4. Continue to define field information on the other tabs or, if you are finished defining the field, exit the window to save the field definition and return to the Field Definitions list.

Launching Workbench from the Method tab

On Windows, you can use the drilldown buttons to launch Workbench, where you can write or edit methods.

If the method is not yet written,

1. Launch Workbench and display the Choose Method File dialog by doing one of the following:
   - Type a name for the method in the appropriate method field and click the drilldown button.
   - Click the drilldown button to enter the default name, which is *Fieldname_Methodname*, e.g., PHONE_ARRIVE.)
2. To add the method to an existing file, select the desired file from the list of available files. If necessary, click the Browse button to find the file.
3. To create a new file, type the filename in the field at the top of the dialog and click OK.
4. Workbench opens (or creates) the file and generates template code for the method type, placing it at the end of the file.
5. Write the method, save the file in Workbench, and return to Repository.

If the method is already written,

1. Enter the method name in the appropriate method field.
2. If you need to edit the method, click the drilldown button. Workbench will launch, open the correct file (or display the Choose Method File dialog if the file is not in the source files for the current project), and place the cursor at the method. Edit it and save the file before returning to Repository.

Assigning a long description to a field

On the Long Desc tab, you can assign an 1,800-character description to the field. This enables you to store more detailed information about your field and its use.

1. From the input window where you're defining the new field or template, go to the Long Desc tab or select Edit Field Functions > Edit Long Description or Edit Template Functions > Edit Long Description.

   (If you're in the Field Definitions or Template Definitions list instead of an input window, select Field Functions > Edit Long Description or Template Functions > Edit Long Description.)

2. Enter your long description.

3. Continue to define field information on the other tabs or, if you are finished defining the field, exit the window to save the field definition and return to the Field Definitions list.

Loading Fields from a Definition File

When a structure contains no field definitions, you can load definitions from an existing definition file (also referred to as an *include file*). An include file contains field definitions, optionally preceded by the RECORD, STRUCTURE, or COMMON statement. A single file can contain multiple record definitions.

If your include file uses .DEFINEs within field definitions, for example,

```
custnm ,a NAMELEN
```

you must substitute those definitions with the actual values.

When an include file is processed, information is loaded into the field definitions in the following manner:

- The field name is set to the field identifier. For unnamed fields, the field name is set to NONAME_*nnn*, where *nnn* is a number starting with 001 and incrementing.
- The field's short description is set to the comment on the field definition line. If there is no comment on that line, it is set to the field name.
- The field's long description is set to all contiguous comment lines following the field definition, including the comment on the field definition line, if any.
- The data type is set to the data type. The Load Fields function does not support data types Enum and Struct. (Because the data type of Enum and Struct fields is the enumeration or structure name, Load Fields cannot tell them apart.) Fields of these types will generate an error.
- The data size is set to the length of the field.
- The precision information is set if the field type is implied-decimal.
- The dimension is set if the field is arrayed.
- The group flag is set if the field defines a group.
- Overlay information is set if the field is an overlay.
- The exclusion flags for Language, Toolkit, and ReportWriter are not set for named fields, but are set for unnamed fields.
- The exclusion flag for Web is not set for either named or unnamed fields.
- Justification is set to left for alpha (**a**) fields and right for numeric (**d** or **i**) fields.

1. On the Structure Definitions list, highlight the structure that you are going to load the fields into.
2. Select Structure Functions > Edit Attributes.
3. Select Attributes > Fields.
4. From the Field Definitions list, select Field Functions > Load Fields.

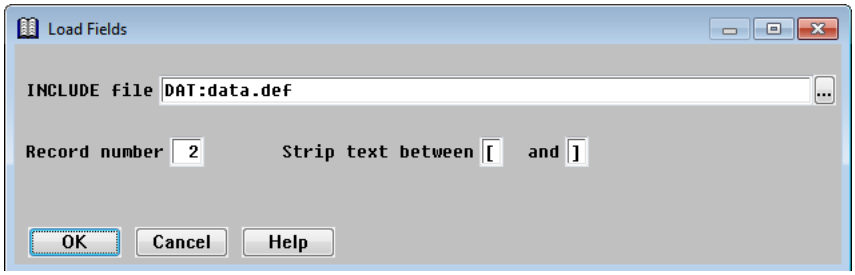


Figure 3-7. Loading fields from an .INCLUDE file.

5. Enter data in each field as instructed below.

   INCLUDE file. Enter the name of the include file from which to load the fields. You can include a full path or logical name. On Windows, you can click the drilldown button to browse for a file to select. If you don't specify an extension, it defaults to **.def**.

   Record number. If the .INCLUDE file contains more than one record, specify which record to use. For example, if your .INCLUDE file contains three records and you want to use the second one, enter **2**. The default record number is 1.

   Strip text between. Comments in the .INCLUDE file are loaded into the field's short and long description fields as described above. You can strip irrelevant data, such as record offsets, by specifying the two characters between which all text should be stripped. The specified characters are stripped as well. Both characters must be non-blank.

   For example, a line in your .INCLUDE file might look like this:

   ```
   csname          ,a30    ;[10,40]Customer name
   ```

   To strip the information in brackets, enter "[" and "]" in the Strip text between fields, then the short description stored for the field will be "Customer name."

6. Exit the window to load the fields.

   If an error occurs while fields are being loaded, you'll get an error message indicating which field caused the error. No additional fields are loaded. You must delete all fields that were successfully loaded before attempting to load from the .INCLUDE file again.

Defining Field Formats

You can define field formats and then select them in the Format name field on the Display tab when defining a field or template. (The information on this tab defines how you want the field to display in Toolkit or ReportWriter; see "Display information" on page 3-12.) For example, you might want to define a format for a date or customer ID.

Repository has two types of field formats: *global* and *structure-specific* (or local).

- Global formats can be used by field definitions in any structure and by templates. You can define a maximum of 9,999 global field formats.
- Structure-specific formats are defined for a particular structure, and only the fields in that structure can use them. You can define a maximum of 250 structure-specific field formats in each structure.

The Format Definitions list

To display the Format Definitions list for global formats, select Modify > Formats from the main Repository window.

To display the Format Definitions list for a structure-specific format, do the following:

1. Highlight the structure in the Structure Definitions list.
2. Select Structure Functions > Edit Attributes.
3. Select Attributes > Formats.

The Format Definitions list displays the format name, type (A for alpha; N for numeric), and the actual format string. The total number of global formats or structure-specific formats is displayed at the bottom of the list. (See figure 3-8.)

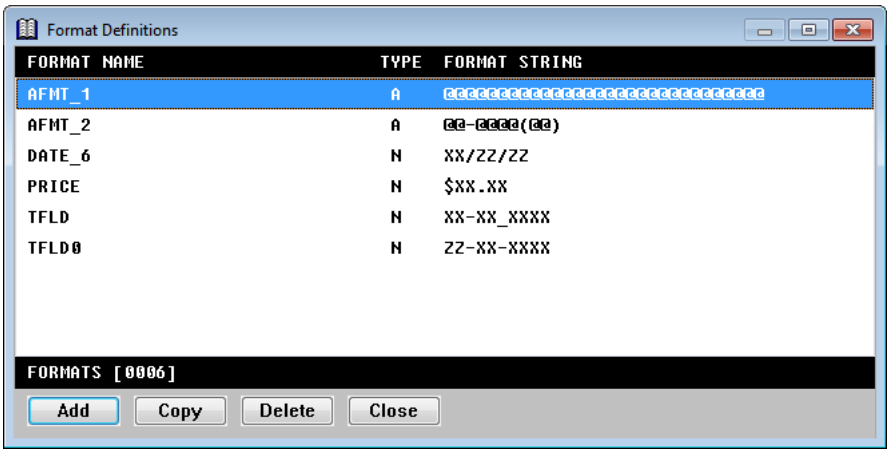


Figure 3-8. The Format Definitions list.

Defining a new format

You can define a new format from scratch or by copying and modifying an existing one. The process is the same whether you are creating a global format or a structure-specific format.

1. From the Format Definitions list,

   - To define a format from scratch, select Format Functions > Add Format.
   - To define a format by copying, highlight the format you want to copy, and then select Format Functions > Copy Format.

   The Format Definition input window is displayed. The example in figure 3-9 shows a display format for telephone numbers, with the area code enclosed in parentheses and the prefix separated from the last four digits by a hyphen.

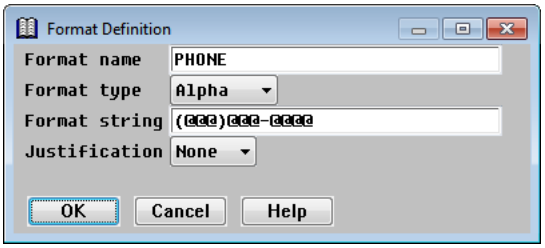


Figure 3-9. Defining a format.

2. Enter or modify data in each field as instructed below.

   Format name. Enter a format name. This name identifies the format when it's assigned to a field or template. For a global format, the name must be unique among all formats, while for a a structure-specific format, the name must be unique within the structure. The format name can have a maximum of 30 characters and must begin with a letter. The remaining characters can be letters, digits, underscores (_), or dollar signs ($).

   Format type. Select the format type: Alpha or Numeric.

   Format string. Enter a format string with a maximum of 30 characters. For an alpha format, enter "at" signs (@). Each @ stands for an alphanumeric character. For a numeric format, enter the correct Synergy DBL data formatting character as shown in "Appendix D: Data Formats". For both alpha and numeric formats, you can also enter formatting characters (such as dashes, backslashes, and so forth) wherever you want them.

Justification. Select how you want the format justified. The default justification is None. The format justification defines how a format is truncated before being applied to a field in ReportWriter.

For example, if the numeric format is

$$$,$$$,$$$.XX-

and you want to use that format with a **d10.2**, a **d8.2**, and **d2.2** field, a right-justified format for the **d8.2** field would look like this:

$,$$$,$$$.XX-

and a right-justified format for the **d2.2** field would look like this:

$$$.XX-

3. Exit the window to save your changes.

Reordering structure-specific formats

1. Highlight the structure-specific format you want to move.
2. Select Format Functions > Reorder Formats. The highlighted format is enclosed in square brackets ([ ]).
3. Use the UP and DOWN ARROW keys to move the bracketed format to another location in the list.
4. Select Reorder Formats again to exit move mode. The format is inserted at the new location.

Modifying a format

1. In the Format Definitions list, highlight the format you want to modify and press ENTER.
2. In the Format Definition window, modify data as necessary. See step 2 on page 3-33 for detailed information on the fields. The format name cannot be modified.

Modifying a format affects any fields selected for printing in existing ReportWriter reports, provided you have not overridden the format in ReportWriter.

3. Exit the window to save your changes.

Deleting a format

You cannot delete a structure-specific format that is used by a field in the current structure.

Repository does not maintain a list of the fields and templates that use a global format, so it is possible to delete one that is in use, but we do not recommend this. If ReportWriter tries to display a field whose format no longer exists, that field is displayed unformatted.

1. In the Format Definitions list, highlight the format you want to delete.
2. Select Format Functions > Delete Format.
3. At the prompt, select Yes to delete the format or No to cancel the deletion.

Defining Field Templates

A template is a set of field characteristics that can be assigned to one or more field definitions (up to 6,000) and one or more template definitions (up to 3,000). Templates provide consistency for your fields throughout all structures in your repository. They also provide an easy way to modify a common field type (for example, changing all date fields from **d6** to **d8**). You can assign a template when you define or modify a field or template. You can define a maximum of 9,999 templates.

To display the Template Definitions list, select Modify > Templates. For each template, the template name and description are displayed. The total number of templates in your repository is displayed at the bottom of the window. (See figure 3-10.)

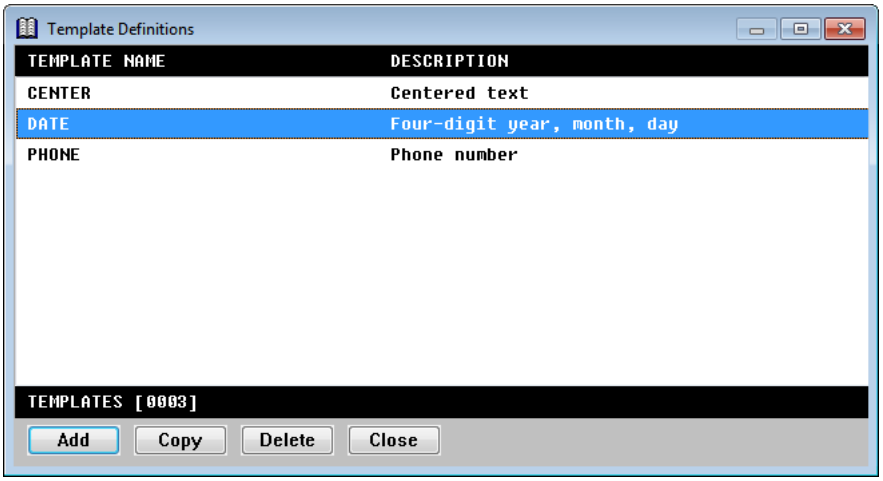


Figure 3-10. The Template Definitions list.

Defining a new template

You can define a new template from scratch or by copying and modifying an existing template.

1. From the Template Definitions list,
   - To define a template from scratch, select Template Functions > Add Template.
   - To define a template by copying, highlight the template you want to copy, and then select Template Functions > Copy Template.

The Template Definition window displays with tabs on which you can define template information. Instructions are given in this section for the Template Definition tab, but not for the other tabs, as they are the same as the tabs for fields.

- **Template Definition** tab, for defining basic template information; see below.
- **Display** tab, for defining how you want the field to display in a Toolkit input window or ReportWriter report; see "Display information" on page 3-12.
- **Input** tab, for defining how field input is handled in a Toolkit input window; see "Input information" on page 3-18.
- **Validation** tab, for defining how field input is validated in a Toolkit input window; see "Validation information" on page 3-22.
- **Method** tab, for associating methods (that are called by Toolkit) with fields; see "Method information" on page 3-26.
- **Long Desc** tab, for assigning a long description; see "Assigning a long description to a field" on page 3-29.

Defining basic template information

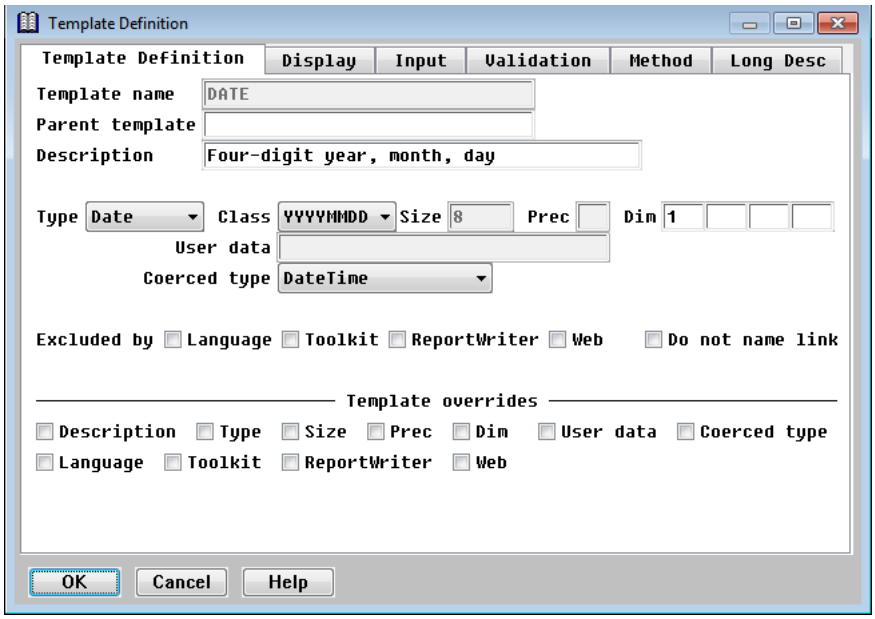


Figure 3-11. Defining basic template information.

1. Enter or modify data as instructed below.

   The example shows a date template with the format YYYYMMDD. We can assign this template to all date fields so they'll be consistent and we won't have to enter the type, size, precision, and dimensional data more than once. (See figure 3-11.)

   Template name. Enter a unique template name. The template name can have a maximum of 30 characters and must begin with a letter. The remaining characters can be letters, digits, underscores (_), or dollar signs ($).

   Parent template. If desired, enter the name of a parent template to use to create the current template. (A parent template is a template for a template.) Up to 3,000 templates can use the same parent template. To display a list of available templates, select Edit Template Functions > List Selections.

   All attributes of the parent template are copied to the current template, including display, input, validation, and method information. You can override any of the parent template attributes simply by specifying new values. If the parent template is later modified, only the attributes that have not been overridden are copied to the child template. The check boxes at the bottom of the window indicate the attributes of the parent template that have been overridden in the current template.

   Repository sets these flags when you exit the window, but you can also set them manually. You might want to do this when an attribute matches the parent and you don't want it to be changed later if the parent changes. By default, these fields are read-only. To modify them, select Edit Template Functions > Access Template Overrides. This menu entry is a toggle: select it a second time and the override fields revert to read-only. The Access Template Overrides setting remains in effect for *all* field and template definitions until you change it.

   Description. Enter a description for the template, with a maximum of 40 characters. The description displays on the Template Definitions list.

   Type. Select the type of data the field will contain:

   Alpha
   Decimal
   Integer
   Date
   Time
   User
   Binary
   Boolean
   Enum
   AutoSeq
   AutoTime

   If you select **Date**, **Time**, or **User**, the cursor moves to the Class field. If you select **Enum**, the cursor moves to the Enumeration field (see Enumeration below).

In Synergy DBL, the Binary data type is treated as an alpha.

In xfNetLink Java (when **genjava** is run with the **-c 1.5** option) and xfNetLink .NET. a Binary data type field in a structure is converted to a byte array on the client, and can be used, for example, to store an RFA. For xfNetLink Synergy clients, a Binary data type field is converted to a string.

In xfODBC, a Binary data type field is described as a binary field (SQL_BINARY). This is also true of a User type field with a class of binary (see Class below), but in this case you can use the routines for user-defined data types in xfODBC to manipulate the data read from the ISAM file and return it as a binary field to the ODBC-enabled application.

Class. If you selected **Date** or **Time** in the Type field, specify the storage format in the Class field:

- Date fields

YYMMDD	two-digit year, month, day
YYYYMMDD	four-digit year, month, day
YYJJJ	two-digit year, Julian day
YYYYJJJ	four-digit year, Julian day
YYPP	two-digit year, period
YYYYPP	four-digit year, period

- Time fields

HHMM	hour, minute
HHMMSS	hour, minute, second

If you selected **User** in the Type field, specify the user subtype in the Class field. User subtypes are used by the xfODBC user-defined processing routines and are available in the **gs_inpfld** structure within UI Toolkit's user-defined processing routines. Subtypes also affect data type mapping in xfNetLink Java and xfNetLink .NET; see "Appendix B: Data Type Mapping" in the *xfNetLink & xfServerPlus User's Guide* for details. The available subtypes are

Alpha
Numeric
Date
Binary

Additional date storage formats are supported by xfODBC. See "Appendix B: Date and Time Formats" in this manual for more information.

User data. If you selected **User** in the Type field, specify a string of up to 30 characters to identify your user-defined data type, and press ENTER to save it.

In a UI Toolkit input window, a user type field flags the runtime input processor to call the ECHKFLD_METHOD, EDSPFLD_METHOD, and EEDTDSP_METHOD subroutines for additional processing. The user data string is passed to these subroutines to be used as a control code. The user subtype (class) is available in the **gs_inpfld** structure.

User type fields also flag ReportWriter to call user-overloadable subroutines (for example, RPS_DATA_METHOD which formats the data for display). See the "Customizing ReportWriter Routines" chapter of the *ReportWriter User's Guide* for more information about the user-overloadable subroutines called by ReportWriter.

User type fields also flag xfODBC to call user-overloadable subroutines to process the data for those fields. The user data string is passed to these routines. See the "Creating Routines for User-Defined Data Types" chapter of the *xfODBC User's Guide* for more information.

Enumeration. If you selected **Enum** in the Type field, enter the enumeration name or select Edit Template Functions > List Selections and choose it from the list.

The Enum data type is not supported by UI Toolkit. To use an enumerated data field with an allow list or selection list or window, use the Enumerated field on the Validation tab. See Enumerated on page 3-25 for more information.

Coerced type. If the structure that this field belongs to is included in an xfNetLink Java JAR file or an xfNetLink .NET assembly, you can optionally specify a non-default data type for the field to be coerced to on the client side. Type coercion is available when Type is one of the following: Decimal, Integer, Date, Time, User. Note the following:

- For Decimal (with or without precision) and Integer types, select **Default** to use the default xfNetLink type mapping.
- Date types can be coerced when the format is one of the following: YYMMDD, YYYYMMDD, YYJJJ, YYYYJJJ.
- User types can be coerced only when the user subtype (i.e., the Class field) is Date and the User data field contains ^CLASS^=YYYYMMDDHHMISS or ^CLASS^=YYYYMMDDHHMISSUUUUUU (case sensitive).
- For Date, Time, and User types, the default coerced type is DateTime.

See "Appendix B: Data Type Mapping" in the *xfNetLink & xfServerPlus User's Guide* for more information on data type mapping and coercion in xfNetLink.

Size. Enter the maximum number of characters the template field can contain. Note the following:

- The maximum size of an alpha, binary, or user field is 99,999.
- The maximum size of an implied-decimal field is 28.
- Valid sizes for integer fields are 1, 2, 4 and 8.

- If the data type is date or time, the size is automatically set when you select a storage format and cannot be modified.
- If the data type is Boolean or Enum, the size is automatically set to 4 and cannot be modified.
- If the data type is AutoSeq or AutoTime, the size is automatically set to 8 and cannot be modified.

Precision. If the data type is implied-decimal, enter the number of characters to the right of the decimal point. This value must be between 1 and 28, inclusive, and must be less than or equal to the size of the field.

Dim1–4. If the template defines an array, enter the number of elements in each dimension. The maximum number of dimensions is 4. The maximum number of elements per dimension is 999. If the template doesn't define an array, the **Dim** field displays **1**.

Excluded by Language. This value determines whether a template field is available to the Synergy compiler. Select this option if you *do not* want the template field to be available to the compiler. Excluded by Language is cleared by default, which means the field *will* be included when using the .INCLUDE compiler directive to reference the structure to which this field belongs, and *will* be included in any definition files generated by the Generate Definition File utility. This feature is useful when your repository contains overlay fields defined solely for the purpose of referencing group elements from within ReportWriter.

Excluded by Toolkit. This value determines whether a template field is available to UI Toolkit. Select this option if you *do not* want to be able to reference the template field from Toolkit. Excluded by Toolkit is cleared by default, which means the field can be referenced by the Script compiler, Composer, and the IB_FIELD subroutine.

Excluded by ReportWriter. This value determines whether a template field is available in ReportWriter as a selectable field. Select this option if you *do not* want this template field to be selectable in ReportWriter. Excluded by ReportWriter is cleared by default, which means the field can be selected for inclusion in a report. This flag can also be honored when generating a system catalog in xfODBC; see "Setting catalog generation options" in the "Preliminary Steps" chapter of the *xfODBC User's Guide* for details on including and omitting fields.

Excluded by Web. This value determines how the template field is treated by xfNetLink. The Excluded by Web flag should be used *only* to control how fields in an overlay are handled. If this field is not part of a structure that contains overlays, do not select this option. Select Excluded by Web if you *do not* want this template field to be included in a Synergy JAR file assembly. Excluded by Web is cleared by default, which means that all fields are included in the Synergy component. For details on using this flag to control how overlays are handled, see "Passing Structures as Parameters" in the "Preparing Your Synergy Server Code" chapter of the *xfNetLink & xfServerPlus User's Guide*.

Do not name link. ReportWriter can use name links you establish in Repository to access related files. This field determines whether the template field is name linked to its parent (if one exists). By default, Repository will use the name of the parent to generate name links. Select this field if you want Repository to use the template field's name when generating name links. (See also "Generating a Cross-Reference File" on page 5-24.)

Template overrides. If the current template references a parent template, the Template overrides section indicates the template attributes that are overridden. If these override fields are read-only, select Edit Template Functions > Access Template Overrides to make them active. (See the description of Parent template on page 3-38 for more information about template override flags.)

2. To define display, input, validation, or method information, or to add a long description for a template, go to the desired tab or select the entry from the Edit Template Functions menu. Refer to the following sections for instructions:

   "Display information" on page 3-12
   "Input information" on page 3-18
   "Validation information" on page 3-22
   "Method information" on page 3-26
   "Assigning a long description to a field" on page 3-29 (Long Desc tab)

3. Exit the window to save the new template and return to the Template Definitions list.

Modifying a template

1. Highlight the template in the Template Definitions list and press ENTER. The Template Definition window is displayed, with the tab that you viewed last on top.

2. Modify data as desired on any of the tabs. The template name cannot be modified.

   If you modify the parent template name, all attributes of the new parent (including display, input, validation, and method information, as well as the long description) are copied to the current template, with the exception of any attributes that were overridden in the current template. See the description of Parent template on page 3-38 for more information.

   For details on completing the fields on each tab, refer to the relevant section:

   "Defining basic template information" on page 3-37 (Template Definition tab)
   "Display information" on page 3-12
   "Input information" on page 3-18
   "Validation information" on page 3-22
   "Method information" on page 3-26
   "Assigning a long description to a field" on page 3-29 (Long Desc tab)

3. Exit the window to save your changes and return to the Template Definitions list.

If the template that you're modifying is assigned to one or more fields or templates, when you save your changes you are prompted

> **Modifying template "*NAME*" will affect one or more template, field, and key definitions. Are you sure you want to save your modifications?**

The default response is No. If you press ENTER, the template modifications are ignored and you are returned to the Template Definitions list. If you select Yes, the new template information is applied to all fields and templates that refer to it. Repository also applies the changes to any fields that reference any templates that in turn reference the template being modified.

Any attribute of any field or template that overrides a template or parent template attribute is *not* modified.

Additionally, all keys that use the updated fields are updated (key size and key data type). The record size of all affected structures is updated.

Deleting a template

You cannot delete a template that is assigned to a field or to another template.

1. Highlight the template in the Template Definitions list.
2. Select Template Functions > Delete Template.
3. At the prompt, select Yes to delete the template or No to cancel the deletion.

Defining Enumerations

An enumeration is a set of related values. It has a name and one or more members (the maximum is 999) associated with it. The members may have values assigned to them, or you can let the compiler assign them. An enumeration defined in the repository can be referenced by a field or template definition, .INCLUDEd in a source file, or referenced by the Synergy Method Catalog for use with xfServerPlus and xfNetLink.

To display the Enumeration Definitions list, select Modify > Enumerations. For each enumeration, the unique enumeration name and a description are displayed. The total number of enumerations in your repository is displayed at the bottom of the list. (See figure 3-12.)

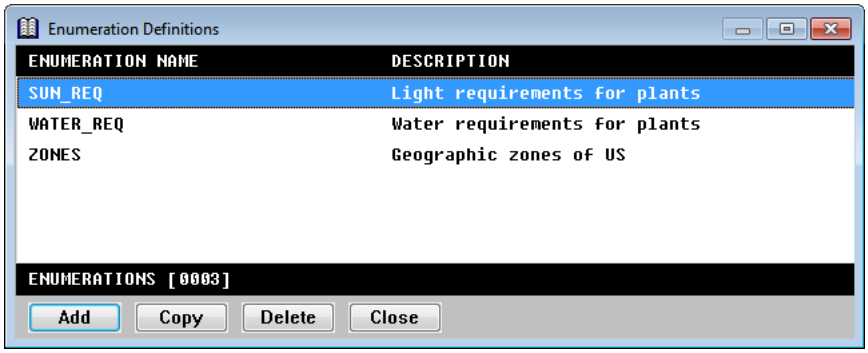


Figure 3-12. The Enumeration List

Defining a new enumeration and its members

You can define a new enumeration from scratch or by copying and modifying an existing enumeration. If enumeration definitions already exist, new definitions are inserted below the highlighted entry.

1. From the Enumeration Definitions list,
   - To define an enumeration from scratch, select Enumeration Functions > Add Enumeration.
   - To define an enumeration by copying, highlight the enumeration you want to copy, and then select Enumeration Functions > Copy Enumeration. The enumeration description and all the enumeration members and their values are copied.

   The Enumeration Definition input window is displayed. (See figure 3-13.)
2. Enter or modify data in each field as instructed below.

   Enumeration name. Enter a unique name for the enumeration. The enumeration name can have a maximum of 30 characters and must begin with a letter. The remaining characters can be letters, digits, underscores (_), or dollar signs ($).

Description. Enter a description for the enumeration, with a maximum of 40 characters. This description displays on the Enumeration Definitions list.

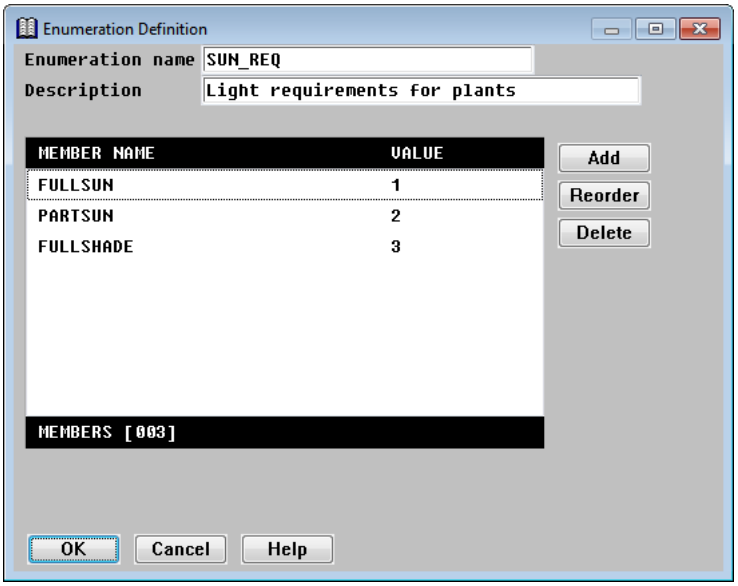


Figure 3-13. Defining an enumeration.

3. Select Member Functions > Add Member to display the Member Definition dialog. Enter or modify data in each field as instructed below:

   Member name. Enter a name for the member. The name must be unique within the enumeration. The member name can have a maximum of 30 characters and must begin with a letter. The remaining characters can be letters, digits, underscores (_), or dollar signs ($).

   Value. (optional) Enter a numeric value for the member. The value is optional; if not supplied, it will be computed at compile time following the rules documented in ENUM in the "Synergy DBL Statements" chapter of the *Synergy DBL Language Reference Manual*.

4. Exit the Member Definition window to save the member and return to the Enumeration Definition window.

5. Add more members as necessary. New members are added below the member that is selected when you choose the Add function; if you need to reorder members, see "Reordering enumeration members" on page 3-46.

6. When you're done adding members, exit the Enumeration Definition window to save your work.

Assigning a long description to an enumeration

You can assign an 1,800-character description to each enumeration. This enables you to store more detailed information about the enumeration and its use.

1. In the Enumeration Definitions list, highlight the enumeration to which you want to assign a long description.
2. Select Enumeration Functions > Edit Long Description.
3. Enter a long description.
4. Exit the window to save your changes.

Modifying an enumeration and its members

1. Highlight the enumeration in the Enumeration Definitions list and press ENTER.
2. Modify the description if desired; the enumeration name cannot be modified.
   - To modify (or add) a value for an existing enumeration member, highlight the member and press ENTER. (You cannot modify the member name; you must delete the existing member and add a new one.)
   - To reorder enumeration members, see "Reordering enumeration members", below.
   - To delete an enumeration member, highlight the member and select Member Functions > Delete Member.
3. When you are done making modifications, exit the Enumeration Definitions window to save your changes.

Reordering enumeration members

1. Highlight the enumeration member you want to move.
2. Select Member Functions > Reorder Members. The highlighted member is enclosed in square brackets ([ ]).
3. Use the UP and DOWN ARROW keys to move the bracketed member to another location in the list.
4. Select Reorder Members again to exit move mode.

Deleting an enumeration

An enumeration that is referenced by a field or template cannot be deleted.

1. Highlight the enumeration in the Enumeration Definitions list.
2. Select Enumeration Functions > Delete Enumeration.
3. At the prompt, select Yes to delete the enumeration or No to cancel the deletion.

Modifying a Field

1. From the Field Definitions list, highlight the field you want to modify and press ENTER. The Field Definition window is displayed, with the tab that you viewed last on top.
2. Modify data as desired on any of the tabs. Note that if you modify the template name, *all* attributes of the new template are copied to the field, with the exception of any attributes that were explicitly overridden within the field. See the description of Template name on page 3-4 for more information. For details on completing the fields on each tab, refer to the relevant section:

   "Basic field information" on page 3-4 (Field Definition tab)
   "Display information" on page 3-12
   "Input information" on page 3-18
   "Validation information" on page 3-22
   "Method information" on page 3-26
   "Assigning a long description to a field" on page 3-29 (Long Desc tab)
3. Exit the window to save your changes and return to the Field Definitions list.

   When you exit, Repository validates the display, input, validation, and method information. If an error exists, correct it and then exit the window again. The SIZE field below the list is updated to reflect any changes made to the structure size.

   If the field being modified has group members, but the Group field is not set, when you exit the input window, you are prompted

   Field "*NAME*" is a group. Clearing the "Group" field will delete all group members. Do you want to continue?

   If you select Yes, the group members are *not* saved with the field definition. If you select No, you are returned to the Field Definition tab.

   If the field you're modifying is used as a key segment for the current structure, when you exit the input window, the following message is displayed:

   This field is defined as a key. All affected keys in the current structure are updated when you save changes.
   Are you sure you want to make modifications?

   If the field being modified is used as an external key segment in another structure, when you exit the input window, you are prompted

   This field is defined as an external segment by another structure. All affected keys in all external structures are updated when you save changes.
   Are you sure you want to make modifications?

   Modifying a field that is a key can affect not only the keys themselves but also any relations that use those keys.

If you answer Yes to either of the above prompts, the key size and key data type of all affected keys are updated when you save your changes to the current structure. If you answer No, all of your modifications are ignored.

Modifying group members

1. Highlight the group field in the Field Definitions list.
2. Select Field Functions > Edit Group Members. (You can also press ENTER to edit the group definition, and then select Edit Field Functions > Edit Group Members while on the primary tab of the input window.)

   The Field Definitions list for the group is displayed. If this is an explicit group, you can add, modify, reorder, or delete member definitions. If this is an implicit group, the list is read-only, and you can only view member definitions. (See figure 3-14.)

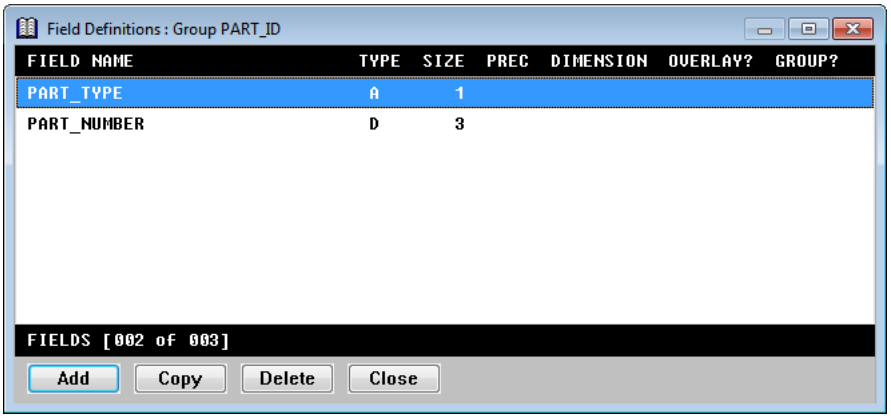


Figure 3-14. The Field Definitions list for a group.

The name of the group is displayed at the top of the list. The number of fields in the group and the total number of fields in the structure are displayed at the bottom of the list.

The following rules apply to groups:

- A group must have at least one member.
- A field and a group (or two groups) with the same immediate parent cannot have the same name.
- A field and a group (or two groups) with the same name, but different parents, is allowed.
- The size of a group must be equal to or larger than the size of its members.
- Fields within groups cannot be used as key segments, tag fields, or aliased fields. Only fields (or groups) defined at the highest (structure) level can be used for these purposes. Overlay fields that overlay group members can be defined at the highest level.

- Access to fields within groups by ReportWriter is only possible using overlay field definitions. The Excluded by ReportWriter flag can be used to prevent the inclusion of a group or field in a report.
- Access to fields within groups by Toolkit is possible if the field name is unique. A prefix may be specified for all the members of a particular group to ensure field name uniqueness. The Excluded by Toolkit flag can be used to prevent a group or field's inclusion in a window definition.
- An overlay specified on a field definition within a group must be relative to another field within that same level.
- The Generate Cross-Reference utility only supports the name linking of fields at the highest (structure) level.
- If you add, delete, or modify group members while modifying a group field and then abandon your changes, only the group field's changes are abandoned. Changes to the group members or their attributes are *not* abandoned. If, however, you add, delete, or modify group members while adding or copying a group field and then abandon your changes, *all* changes—including those to the group members—will be abandoned.

3. To return to the previous Field Definitions list or Field Definition tab, press the Exit shortcut.

When you exit, if the total size of the members exceeds the size of the group field, an error message is displayed and you are returned to the Field Definition tab of the group field. You can either increase the size of the group field or edit the group members individually and modify their sizes.

Deleting a Field

A field can be deleted only when all of the following conditions are true:

- The field is not used in a key definition for the current structure.
- The field is not used in a key definition for another structure.
- The field is not used in a tag definition for the current structure.
- The deletion does not invalidate an overlay specification.

1. Highlight the field definition in the Field Definitions list.
2. Select Field Functions > Delete Field.
3. At the prompt, select Yes to delete the field or No to cancel the deletion.

   If the field you're deleting defines a group, you are prompted

   Field "*NAME*" is a group. Deleting it will delete all group members as well. Do you want to continue?

   Enter Yes to delete the field and all its members. Enter No to cancel the deletion.

4

Working with Files

Files Overview

File definitions determine which files can be accessed through Repository and which structures can be used to access them. To use Repository with xfODBC or ReportWriter, you'll need to specify which files contain your data and assign the structures you've defined to those files. You can define up to 9,999 files; each file may have multiple structures (up to 200) assigned to it.

The File Definitions list

To display the File Definitions list, select Modify > Files. For each file, the following information is displayed:

FILE NAME . The unique file definition name.

DESCRIPTION. A descriptive identifier for the file definition.

TYPE. The file type:

DBL	DBL ISAM
ASC	ASCII
REL	relative
USE	user defined

The total number of files in your repository is displayed at the bottom of the list. (See figure 4-1.)

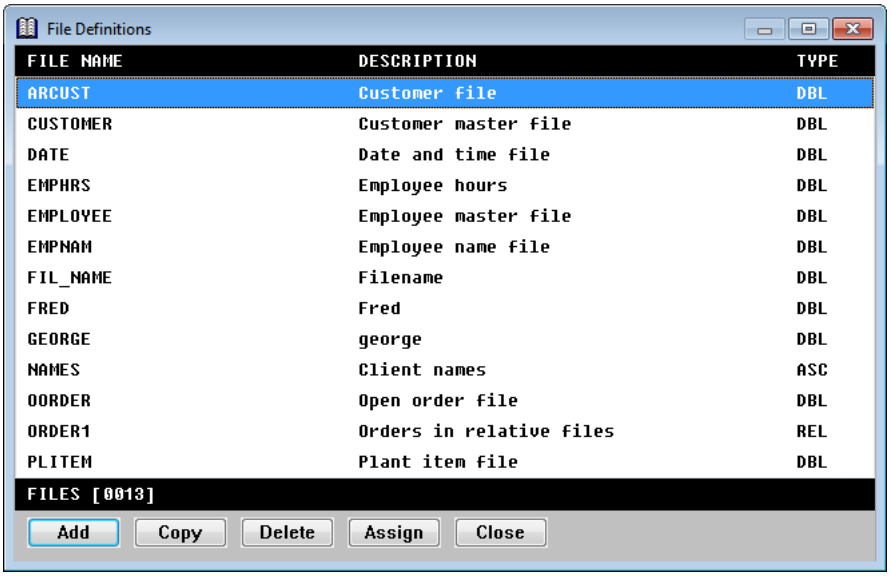


Figure 4-1. The File Definitions list.

Defining Files

You can define a new file definition from scratch or by copying and modifying an existing file.

1. From the File Definitions list,
   - To define a file from scratch, select File Functions > Add File.
   - To define a file by copying, highlight the file you want to copy, and then select File Functions > Copy File.

   The File Definition input window is displayed.

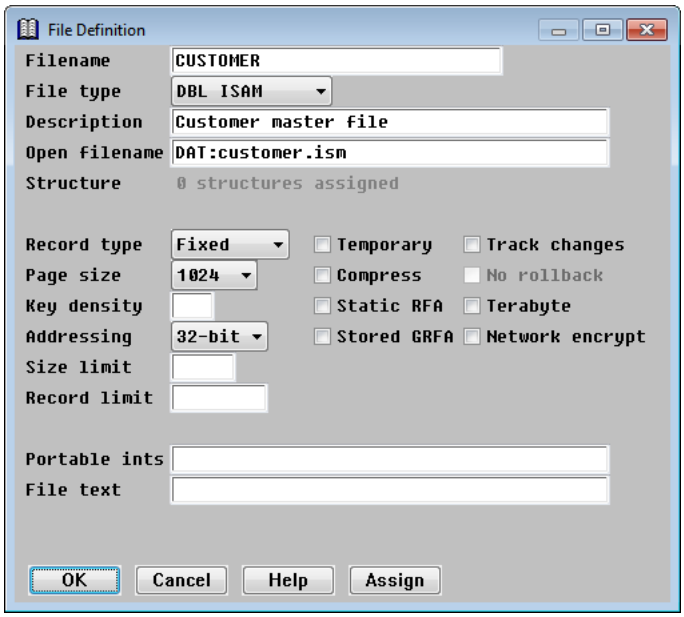


Figure 4-2. Defining a file.

2. Enter or modify data in each field as instructed below.

   To continue our customer example from the previous sections of this chapter, we define the file that contains our customer data. (See figure 4-2.) It's a DBL ISAM file located in the directory assigned to the DAT logical. The file is named **customer.ism**. *We named the file definition* **CUSTOMER**, *just like the structure we're going to assign to it in "Assigning Structures to Files" on page 4-8.*

   Filename. Enter a unique file definition name. The file name can have a maximum of 30 characters and must begin with a letter. The remaining characters can be letters, digits, underscores (_), or dollar signs ($).

File type. Select the file type you want to assign:

ASCII
DBL ISAM
RELATIVE
USER DEFINED

If a structure definition with the same name as your file exists, the file type defaults to that of the structure.

In the example, the file type defaults to **DBL ISAM***, because the structure, which has the same name, is type DBL ISAM.*

If you try to modify the file type when structures have already been assigned to the file, an error message is displayed when you try to save your changes. To modify a file definition's file type, you must first disassociate all structures from that definition. The file type of the file definition and the file type of the structures assigned to it must match.

Description. Enter a unique and meaningful description (up to 40 characters) for the file definition. This description is available when Repository displays a list of files, and ReportWriter can use it, along with the structure description, to identify your files.

Open filename. Enter the name of the actual data file, including the path specification. The filename can have a maximum of 64 characters; however, the field allows you to enter only 40 characters. To enter a longer value, select Edit File Functions > Edit Entire Text.

To edit a filename that exceeds 40 characters, you must also use the Edit Entire Text option. (If you edit it in the Open filename field, the portion that is not displayed will be lost. Similarly, deleting the 40 characters that are displayed in the field will delete the entire filename.)

Repository enables you to "map" a filename to simulate multiple copies of the same file definition with slightly different open filenames. For more information, see RPS_FILNAM_METHOD in the "Customizing ReportWriter Routines" chapter of the *ReportWriter User's Guide.*

Structure. This read-only field displays the name of the structure assigned to the file (if there's only one) or the number of assigned structures (if there are none or more than one). If this is a new file, the Structure field contains the text "0 structures assigned". It will be filled with a structure name or a number of structures after you assign one or more structures to your file, as explained in "Assigning Structures to Files" on page 4-8.

Record type. This field applies only to ISAM files. Select the format of the records in this file:

Fixed	Fixed-length records. (default)
Variable	Variable-length records.
Multiple	Multiple fixed-length records.

Page size. This field applies only to ISAM files. Select index block page size for this file:

1024 (default)
512
2048
4096
8192
16384
32768

Key density. Enter a value between 50 and 100, inclusive, to override the default key density percentage used for all keys in the file. The default density is around 50%. This field applies only to ISAM files. You can also override the density percentage for a specific key; see the description Key density on page 4-13 for more information.

Addressing. This field applies only to ISAM files. Select the file addressing for this file:

32-bit (default)
40-bit

Size limit. Enter the maximum number of megabytes that the data file (**.is1**) is allowed to reach. This field applies only to REV 6 or greater ISAM files.

Record limit. Enter the maximum number of records that the file is allowed to contain. This field applies only to REV 6 or greater ISAM files.

Temporary. Select Temporary if you *do not* want this file to be a selectable file in ReportWriter. If set, the file is considered "temporary" and you will not be able to select the file for inclusion in a report generated by ReportWriter. The "temporary" flag can optionally be honored by xfODBC to exclude tables attached to temporary files from the system catalog. For more information on how xfODBC handles temporary files, see "Setting catalog generation options" in the "Preliminary Steps" chapter of the *xfODBC User's Guide*.

Compress: Select this option if the data in this file is compressed. This field applies only to ISAM files.

Static RFA. Select this option if records in this file will retain the same RFA across WRITE operations. This field applies only to ISAM files.

Stored GRFA. Select this option if the CRC-32 portion of an RFA is to be generated and stored to each record header on each STORE or WRITE operation. This field applies only to ISAM files.

Track changes. Select this option to enable change tracking in this file. This field applies only to REV 6 or greater ISAM files. Selecting Track changes also selects Terabyte.

No rollback. If you selected Track changes, indicate whether you want to prohibit rollbacks on this file. This field applies only to REV 6 or greater ISAM files.

Terabyte. Select this option if this is a 48-bit terabyte file. This field applies only to ISAM files and is selected automatically when Track changes is selected.

Network encrypt. Select this option to indicate that clients that access this file must use encryption. This field applies only to ISAM files.

Portable ints. This field allows for the definition of one or more non-key portable integer data specifications that can be passed as an argument to the ISAMC subroutine. Non-key integer data specifications apply only to ISAM files. Use this syntax:

`I=`*pos*`:`*len*`[,I=`*pos*`:`*len*`] [,...]`

where *pos* is the starting position of non-key portable integer data and *len* is its length in bytes (1, 2, 4, or 8). No validation is performed on the contents of this field.

The maximum string length is 120 characters, but the Portable ints field allows you to enter only a 40-character string. To enter or edit a longer string, select Edit File Functions > Edit Entire Text. (**Note:** If you edit a string longer than 40 characters in the Portable ints field, the portion that is not displayed will be lost. Deleting the 40 characters that are displayed in the field will delete the entire string.)

File text. This field allows for the specification of text to be added to the header of the file and space to be allocated for user-defined text. This option applies only to REV 6 or greater ISAM files. There are three options for the syntax:

text_size`[K]`
`"`*text_string*`"`
text_size`[K]:"`*text_string*`"`

where *text_size* is the amount of space to allocate in bytes (rounded to the nearest kilobyte) for user-defined text, and *text_string* is a text string to add to the file header. No validation is performed on the contents of this field.

The maximum string length is 1800 characters, but the File text field allows you to enter only a 40-character string. To enter or edit a longer string, use the Edit Entire Text option, as described above for the Portable ints field.

You can assign structures to your file or disassociate assigned structures at this point, without exiting the File Definition input window. See "Assigning Structures to Files" on page 4-8 for more information.

3. Exit the window to save the new file definition.

Your next step is to assign a structure to your file, if you haven't done so already. Follow the instructions in "Assigning Structures to Files" on page 4-8.

Assigning a long description to a file definition

You can assign an 1,800-character description to the file definition. A long description is a place to store more detailed information about your file definition and its use.

1. From the File Definitions list, highlight the file to which you want to assign a long description.
2. Select File Functions > Edit Long Description.
3. Enter your long description.
4. Exit the window to save your changes.

Assigning a user text string to a file definition

You can associate a 60-character user-defined text string with the file definition to store additional information you want to access at runtime with the Repository subroutine library.

1. In the File Definitions list, highlight the file to which you want to assign a user-defined text string.
2. Select File Functions > Edit User Text.
3. Enter your user text string.
4. Exit the window to save your changes.

Modifying a file definition

1. From the File Definitions list, highlight the file definition you want to modify and press ENTER.
2. In the File Definition window, modify data as desired. The file definition name cannot be modified. For assistance in completing the fields, see step 2 under "Defining Files" on page 4-3.
3. Exit the window to save your changes.

You can assign structures to your file at this point (or disassociate assigned structures), without exiting the File Definition input window. See "Assigning Structures to Files" on page 4-8 for more information.

Deleting a file definition

1. Highlight the file in the File Definitions list.
2. Select File Functions > Delete File.
3. At the prompt, select Yes to delete the file definition or No to cancel the deletion.

Assigning Structures to Files

When you assign a structure to a file, you are declaring, "This data file uses this structure." You can assign more than one structure to a file; the maximum that can be assigned is 200. A given structure can be assigned to one or more files.

The file type of the structure must match the file type of the file definition that it is assigned to. The structure must have at least one field defined. In addition, the primary keys of all structures assigned to the same file must match. (The primary key is assumed to be the first key in the list and must be an access key.) Specifically, the following key information must match:

- Key size
- Sort order
- Dups allowed flag
- Key data type
- Number of segments
- Type, position, length, and order of each segment

See "Defining Keys" on page 4-10 for more information.

Assigning a structure to a file

1. While defining or modifying a file definition or while the filename is highlighted in the File Definitions list, select File Functions > Assign Structures. You can also click the Assign button in the File Definition window while defining or modifying a file definition.

   The Assigned Structures list displays the names of any structures currently assigned to that file.

2. Select Structure Functions > Add Structure.

3. Complete the fields in the Assigned Structure window as instructed below:

   Structure name. Enter the name of a structure to assign to the file. To display a list of available structures whose file type matches that of the file definition, select Structure Functions > List Selections and then select the structure you want to assign to the file.

   ODBC table name. Enter the table name to use for ODBC access when using xfODBC. If specified, the ODBC table name will be used as the name of the generated table for this particular file/structure combination. You can use this feature to distinguish table names when the same structure is assigned to more than one file, or you can use it to provide more descriptive names in any situation. The table name can have a maximum of 30 characters and must begin with a letter. The remaining characters can be letters, digits, underscores (_), or dollar signs ($).

4. Exit the window to add the structure to the list in the Assigned Structures window.

   If the primary key definition doesn't match other assigned structures, you are prompted

   Structure's primary key does not match those of other assigned structures.

   If you get this message, press ENTER, and select another structure name.

5. After you've assigned a structure, you can add other structures to the list or exit the window to return to the File Definitions list.

   If you were defining or modifying a file definition when you selected Assign Structures, when you return to that input window, the Structure field contains either the name of the assigned structure (if only one is assigned to this file) or a message stating the number of structures assigned.

Modifying an assigned structure

You can modify only the ODBC table name.

1. From the Assigned Structure list, highlight the structure you want to modify and press ENTER.
2. In the Assigned Structures window, modify the ODBC table name by typing over it and then press ENTER.
3. Exit the window to save your changes.

Disassociating a structure from a file

1. Select File Functions > Assign Structures while defining or modifying a file definition or while the filename is highlighted in the File Definitions list.
2. In the Assigned Structures list, highlight the structure you want to disassociate.
3. Select Structure Functions > Delete Structure.
4. At the prompt, select Yes to disassociate the structure and remove it from the list or No to cancel the deletion.

Defining Keys

Key definitions are associated with each structure. You can define two types of keys: *access* and *foreign*.

- Access keys represent true keys in the data file and are used to specify relationships between files.
- Foreign keys are also used to specify relationships between files, but they don't have to be true keys in the data file.

The order of your access keys determines the key of reference used by xfODBC and ReportWriter to access the file (unless you define an explicit key of reference). The maximum number of keys that can be defined within one structure is 99.

If you're using RMS indexed files, you may need to specify a key of reference greater than 99. If so, you can use the key-of-reference attribute to explicitly specify the key of reference to be used.

A relative file can have only one access key: the record number. When you create a structure whose file type is relative, Repository automatically creates an access key for you. Its name is RECORD_NUMBER, and it is ascending, allows no duplicates, and has one segment of type R. You can create additional foreign keys or delete the access key that Repository created for you; however, only one access key can exist for a relative file, and it must have the attributes described above.

The Key Definition list

To display the Key Definitions list,

1. In the Structure Definitions list, highlight the structure for which you want to define a key.
2. Select Structure Functions > Edit Attributes.
3. Select Attributes > Keys.

The Key Definitions list displays the following information for each key in the selected structure. The total number of keys for this structure is displayed at the bottom of the list. (See figure 4-3.)

KEY NAME. The unique key name.

TYPE. A for an access key or **F** for a foreign key.

ORDER. A for ascending or **D** for descending (for access keys only).

DUPS. Y if duplicates are allowed or **N** if they are not (for access keys only).

SEGMENTS. A list of segment types in the key.

SIZE. The size of the key.

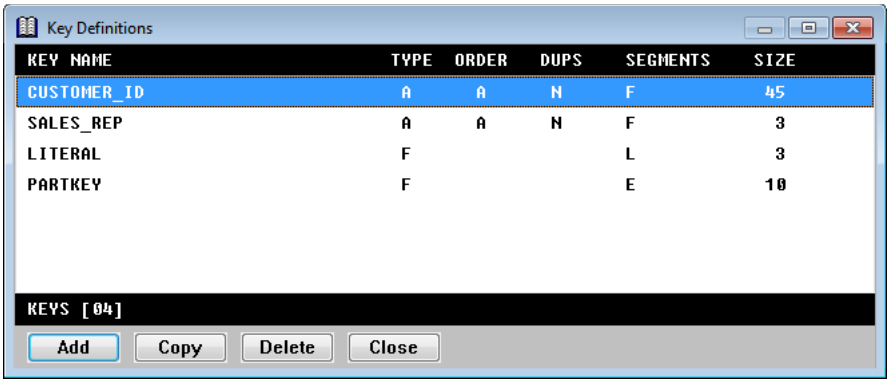


Figure 4-3. The Key Definitions list.

Reordering keys in the Key Definitions list

1. Highlight the key you want to move.
2. Select Key Functions > Reorder Keys. The highlighted key is enclosed in square brackets ([ ]).
3. Use the UP and DOWN ARROW keys to move the bracketed key to another location in the list.
4. Select Reorder Keys again to exit move mode. The key is inserted at the new location.

   Repository assumes that the first key in the list is the primary key. Access keys must all remain at the top of the list, followed by the foreign keys.

Defining a new key

You can define a new key from scratch or by copying and modifying an existing one. If key definitions already exist, new definitions are inserted below the highlighted entry.

1. From the Key Definitions list,

   - To define a key from scratch, select Key Functions > Add Key.
   - To define a key by copying, highlight the key you want to copy, and then select Key Functions > Copy Key.

   The Key Definition input window is displayed.

2. Enter or modify data in each field as instructed below.

   Key name. Enter a unique key name. The key name is used to specify the key when you define relations. The key name must be unique within the current structure. The key name can have a maximum of 30 characters and must begin with a letter. The remaining characters can be letters, digits, underscores (_), or dollar signs ($).

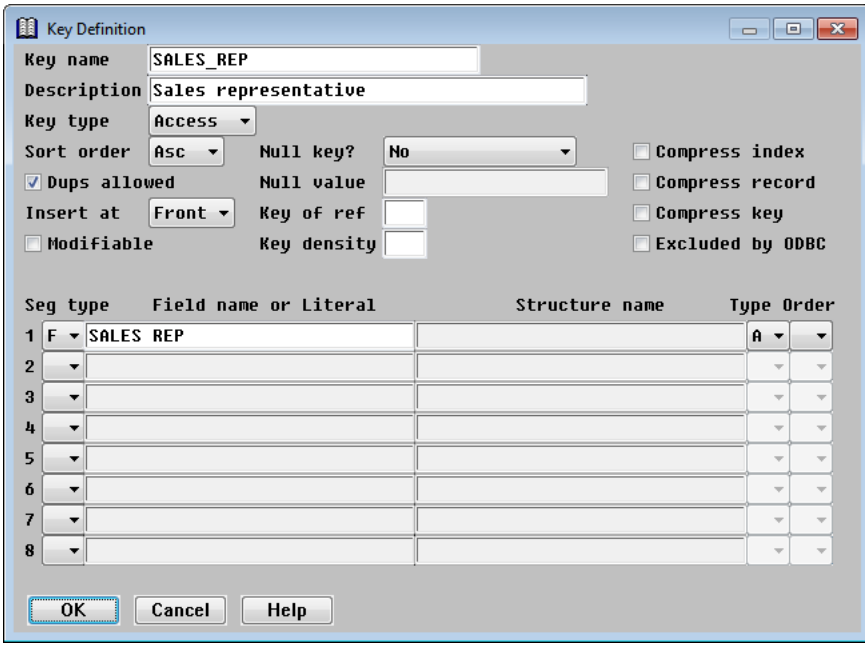


Figure 4-4. Defining a key.

Description. Enter a more descriptive identifier for the key with a maximum of 40 characters.

Key type. Select the key type:

Access	Your key is a true key in the data file. (default)
Foreign	Your key is not a true key.

If you are defining both access keys and foreign keys, the access keys *must* be defined first, followed by the foreign keys. The order of the access keys determines the key of reference used by xfODBC and ReportWriter to access the file, unless you define an explicit key of reference. (See the description of Key of ref on page 4-13.)

The following eleven fields apply to access keys only. If you select **Foreign** in the Key type field, these fields are disabled, and the cursor moves to the first segment definition.

Sort order. Choose the option that defines how the key field data is stored: **Asc** (ascending; the default) or **Desc** (descending).

Dups allowed. Set this field to indicate that the key field allows duplicates.

Insert at. If you set Dups allowed, specify Front or End to indicate where records with duplicate keys are inserted relative to other records containing the same key value. This field applies only to access keys.

Modifiable. Set this field to indicate that the key is modifiable. This field applies only to access keys other than the primary key (assumed to be the first key in the list).

Null key. This field applies only to access keys other than the primary key (assumed to be the first key in the list). Select the null key type:

No	Not a null key (default)
Replicating	Replicating null key
Non-replicating	Non-replicating null key
Short	Short null key

Null value. If you selected **Replicating** or **Non-replicating** for Null key, you can optionally specify the null key value. If the null key value contains spaces, you must enclose the string in quotation marks. The default null key value is a space.

The Null value can have a maximum length of 255 characters; however, the field allows you to enter only a 20-character string. To enter a longer string, select Edit Key Functions > Edit Entire Text.

To edit a string that exceeds 20 characters, you must also use the Edit Entire Text option. (If you edit it in the Null value field, the portion that is not displayed will be lost. Similarly, deleting the 20 characters that are displayed in the field will delete the entire string.)

Key of ref. By default, the order of the access keys determines the key of reference used by xfODBC and ReportWriter to access the file. For example, the first access key is key of reference 0, the second access key is key of reference 1, and so on. Use the Key of ref field to explicitly specify a key of reference that differs from the default.

This field can also be used to specify a key of reference greater than 99 when using RMS indexed files.

Key density. If you want to override the key density (defined at the file level) for this specific key, enter a number between 50 and 100 that represents the density percentage for the key. Density represents the percentage that each index block is filled. If unspecified, the density for this key will be the density specified for the file.

Compress index. Set this option to indicate that the key's index is compressed. Only RMS indexed files use this option. Compress index applies only to access keys.

Compress record. Set this option to indicate that the record within the data is compressed. Only RMS indexed files use this option. Compress record applies only to access keys.

Compress key. Set this option to indicate that the key within the data is compressed. Only RMS indexed files use this option. Compress key applies only to access keys.

Excluded by ODBC. This value determines whether a key is included by xfODBC in the system catalog. Set this field if you want this key to be excluded from the system catalog, which prevents xfODBC from attempting to use it for optimization. Excluded by ODBC is not set by default,

which means the key will be used for optimization, as deemed appropriate by xfODBC, when generating a system catalog. For more information on xfODBC optimization, see "Optimizing with Keys" in the "Optimizing Data Access" chapter of the *xfODBC User's Guide*.

Seg type. A key must contain at least one segment definition. A selection window with a list of the valid segment types is displayed for each of the eight Seg type fields at the bottom of the window. Select the segment type you want to use for the first key segment:

F (Field)	Defines a field in the current structure as a segment.
L (Literal)	Defines a literal as a segment, enabling you to append a constant to the beginning or end of a key or embed a constant within a key.
E (External)	Defines a field in another structure as a segment.
<blank>	Clears the segment type.

Only foreign keys can contain literal or external segment types. See "Using literal key segments" on page 4-15 and "Using external key segments" on page 4-16 for more information.

- If you select segment type **F (field)**, enter a field name from the current structure in the Field name or Literal column. Select Edit Key Functions > List Selections to display a list of available fields.
  - If the field is an arrayed field, Repository uses only the first element of that array. If you want to specify an element other than the first, define an overlay field that overlays the desired element and flag the field as excluded by ReportWriter.
  - If the field is a group, you cannot select fields within it, but you can define an overlay field to overlay the group members and then use the overlay field as the key.

  Type: This field defines whether the data type for this specific key segment overrides the data type for the key. When a key is created, Repository assigns it a default key data type. If all segments have the same data type, that type becomes the key data type. If the segments have mixed data types, the key data type is set to alpha. You can override the key data type for one or more segments by specifying a value in the Type field.

  - If the field is alpha, its segment data type can be set to **N** for no case (case-insensitive) alpha.
  - If the field is integer, its segment data type can be set to **S** for sequence, **T** for timestamp, **C** for create timestamp, or **U** for unsigned integer.
  - All fields can be set to **A** for alpha.
  - User-defined fields can be set to any type.

  Set Type to **S**, **T**, or **C** to define an autokey. Autokeys are keys that are filled in by Synergy DBMS with the appropriate values. They can contain only one segment, consisting of an 8-byte field. Autokeys cannot be null, modifiable, or allow duplicates. See "Key type" in the "Synergy DBMS" chapter of *Synergy Tools* for more information on autokeys.

Order: The default key sort order is defined in the Sort order field. To override the sort order for a specific key segment, select A for ascending or D for descending. The default key segment order is unspecified, which means it defaults to the sort order of the key.

- If you select segment type **L (literal)**, enter a literal value in the Field name or Literal column. The maximum size of a literal is 30. If you don't enter a value, Repository assumes a literal segment consisting of 30 blanks. If you want a literal segment value to have trailing blanks, enclose the literal in double or single quotation marks (" " or ' ').
- If you select segment type **E (external)**, enter a structure name in the Structure name column and a field name in the Field name or Literal column. The field name must belong to the structure, and the structure cannot be the current one. Select Edit Key Functions > List Selections to display a list of available field or structure names.

If the total size of a key's segments exceeds 255, the size is set to 255. ReportWriter uses this size when building the key for data file access.

When a key is created, Repository assigns it a data type. ReportWriter uses this key data type when accessing related data in other files. (See the description of the Type field above for more information.) When keys are used in relations, we recommend that they have the same key data type and size, but this is not required.

3. Exit the window to save the new key definition.

Using literal key segments

Literal key segments enable you to establish a relationship between two files where part of the key data is constant.

Let's assume the COMPANY_ID field in file B is a combination of a four-digit CLIENT_ID from file A and a two-digit COMPANY_CODE. Let's also assume that for a particular user, the company code will always be **01** and is therefore not stored in any file. You can create a key for file A that is composed of a literal segment whose value is **01** and a field segment using the CLIENT_ID field. You would then use this key to establish a relationship to the COMPANY_ID field in file B.

You must pad literal key segments with blanks to reach the desired length if you want exact key matches. For example, if your "from" key consists of an **a4** field and the literal "ABC," and your "to" key consists of an **a4** field and an **a6** field, you must pad the literal with three blanks ("ABC ") to do an exact match; otherwise, ReportWriter will do a partial key match on seven characters.

Using external key segments

Repository permits a special kind of relationship between files called an *external relation*. An external relation involves three or more files, where one file is accessed by a key composed of segments from the remaining files. For example, the item type from one file (file A), along with the item number from a second file (file B), can be used to access the item ID in a third file (file C).

The following example describes how to define the external relation defined above. (Assume that structure A is assigned to file A, and so forth.) We'll assume the following fields are in structures A, B, and C:

Structure A

```
A_ITMTYP      ,a2    ; Item type
A_TRANNO      ,d5    ; Transaction #
```

Structure B

```
B_ITMNO       ,d5    ; Item #
B_TRANNO      ,d5    ; Transaction #
```

Structure C

```
C_ITMID       ,a7    ; Item ID
```

Now you would do the following:

1. Establish a relationship between file A and file B. This should be a one-to-one relationship. (Each record in file A should correspond to only one record in file B.) To do this, define a key for each structure, and define the relationship in structure A as follows:

   Structure B

   Define the access key B_TRANKEY using field segment B_TRANNO.

   Structure A

   Define the access key A_TRANKEY using field segment A_TRANNO.

   Define a relation using key A_TRANKEY related to key B_TRANKEY in structure B.

2. Define a key for structure A that contains the external key segment (the external key segment refers to a field in file B).

   Structure A

   Define the foreign key A_ITMKEY using field segment A_ITMTYP and external segment B_ITMNO from structure B.

3. Define a key for structure C.

   Structure C

   Define the access key C_ITMKEY using field segment C_ITMID.

4. Define a relation (using the key containing the external segment) from file A to file C.

   Structure A

   Define a relation using key A_ITMKEY related to key C_ITMKEY in structure C.

   See "Using external key segments" in the "Miscellaneous ReportWriter Information" chapter of the *ReportWriter User's Guide* for a discussion of how this relationship appears in ReportWriter.

Modifying a key

1. From the Key Definitions list, highlight the key you want to modify and press ENTER.
2. In the Definition window, modify data as desired. For detailed information on the fields, see step 2 under "Defining a new key" on page 4-11. The key name cannot be modified.

   Before you modify segment definitions in the lower portion of the input window, you should check whether any relations use this key. Changing segment definitions might change the key data type, which will affect relations that use the key. We recommend that related keys have the same key data type and size, but to give Repository more flexibility, this is not required.
3. Exit the window to save your changes.

Deleting a key

You can delete a key only when both of the following conditions are true:

- The key is not used in a relation defined by the current structure.
- The key is not used in a relation defined by another structure.

1. Highlight the key in the Key Definitions list.
2. Select Key Functions > Delete Key.
3. At the prompt, select Yes to delete the key or No to cancel the deletion.

Defining Relations between Structures

Relations can be associated with each structure in a file. Relations enable you to link the keys of one structure with the keys of other structures. A maximum of 99 relations can be defined for a structure.

Defining relations enables xfODBC and ReportWriter to access information in related structures. For example, you might relate a customer ID key in a sales transaction file with a customer ID key in a customer file. In xfODBC, you can create SQL statements that retrieve transaction information and associated customer information. Given the same scenario, in ReportWriter the additional information about the customer is automatically accessed when you create a sales transaction report.

In order for ReportWriter to be able to cross-reference data between files, you must create relations between structures. (For information about how ReportWriter can access related files without explicit relationships being defined in the repository, see "Generating a Cross-Reference File" on page 5-24.)

The Relation Definitions list

To display the Relation Definitions list,

1. In the Structure Definitions list, highlight the structure for which you want to define a relation.
2. Select Structure Functions > Edit Attributes.
3. Select Attributes > Relations.

The Relation Definitions list displays the following information about relations for the current structure. (See figure 4-5.)

NAME. The unique relation name.

FROM KEY. The name of the key to use in the current structure.

TO STRUCTURE. The name of the structure to which the current structure is related.

TO KEY. The name of the key related to in the "to" structure.

The total number of relations for this structure is displayed at the bottom of the list.

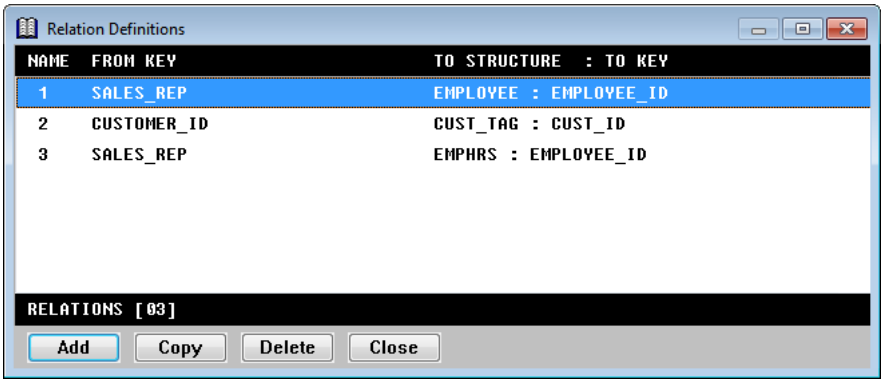


Figure 4-5. The Relation Definitions list.

Reordering relations in the Relation Definitions list

1. Highlight the relation you want to move.
2. Select Relation Functions > Reorder Relations. The highlighted relation is now enclosed in square brackets ([ ]).
3. Use the up and down arrow keys to move the bracketed relation to another location in the list.
4. Select Reorder Relations again to exit move mode. The relation is inserted at the new location.

Defining a new relation

You can define a new relation from scratch or by copying and modifying an existing relation. If relations already exist, new relations are inserted below the highlighted entry.

1. From the Relation Definitions list,
   - To define a relation from scratch, select Relation Functions > Add Relation.
   - To define a relation by copying, highlight the relation you want to copy, and then select Relation Functions > Copy Relation.

   The Relation Definition input window is displayed.

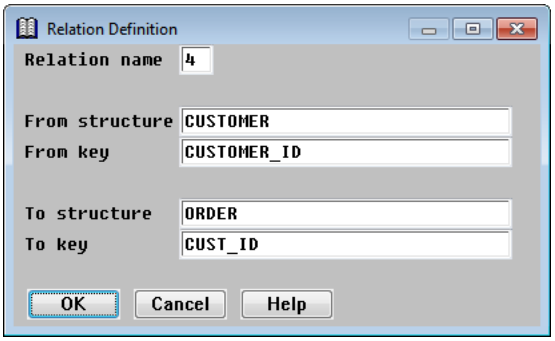


Figure 4-6. Defining a relation.

2. Enter or modify data in each field as instructed below.

   The example shown in figure 4-6 relates the customer ID key in the customer file to the customer ID key in the order file, which enables ReportWriter to cross-reference order representative data between the two files.

   Relation name. The relation name is the way the relation is identified when related files are selected in ReportWriter. A relation must have a unique name within the current structure, and it must be numeric. A default relation name is displayed; change it if desired.

   From structure. This read-only field displays the name of the current structure.

   From key. Enter the name of the key you want to relate. To display a list of the keys for the current structure, select Edit Relation Functions > List Selections. To examine a key in more detail, see "Examining a key" on page 4-21.

   To structure. Enter the name of the structure to which you want to relate the first structure. To display a list of available structures, select Edit Relation Functions > List Selections.

   To key. Enter the name of an access key to which to relate the "from" key. To display a list of access keys in the selected "to" structure, select Edit Relation Functions > List Selections. To examine a key in more detail, see "Examining a key" on page 4-21.

   We recommend that related keys have the same key data type and size, but to add more flexibility to relationships, Repository does not require this.

3. Exit the window to save the new relation.

   If you want to examine a relation in more detail, see "Examining a relation in detail" on page 4-21.

Examining a key

When you're entering a "from" or "to" key in the Relation Definition input window, you can view more detailed information about the key you're thinking of using.

1. With your cursor in the From key or To key field, select Edit Relation Functions > List Selections. The list of Available Keys is displayed.
2. Highlight the key whose segments you'd like to view.
3. Select Key Functions > Examine Key to display the key name and its segments. (See figure 4-7.)

   If the key allows duplicates, the word "Dups" is displayed to the right of the key name. The segment information includes the type and size of each key segment: F (Field), L (Literal), E (External), R (Record number).

   If the segment is a field or external segment, the field name is specified. To display the field description instead of the field name, select Examine Key > Toggle View. If the segment is a literal, the literal string is displayed.

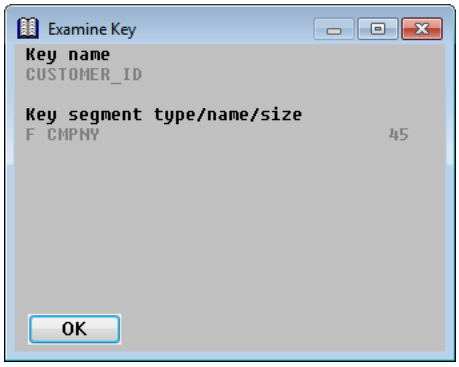


Figure 4-7. Examining a key.

Examining a relation in detail

1. From the Relation Definitions list, highlight the relation you'd like to view.

   You can also examine a relation from the Relation Definition input window while you're defining or modifying a relation.
2. Select Relation Functions > Examine Relation. This window describes the key segments that are used in the current relation, and contains the names of the "from" structure, the "from" key, the "to" structure, and the "to" key. (See figure 4-8.)

   If a key allows duplicates, "Dups" is displayed to the right of the key name. The segment information includes the type and size of each key segment: F (Field), L (Literal), E (External), R (Record number).

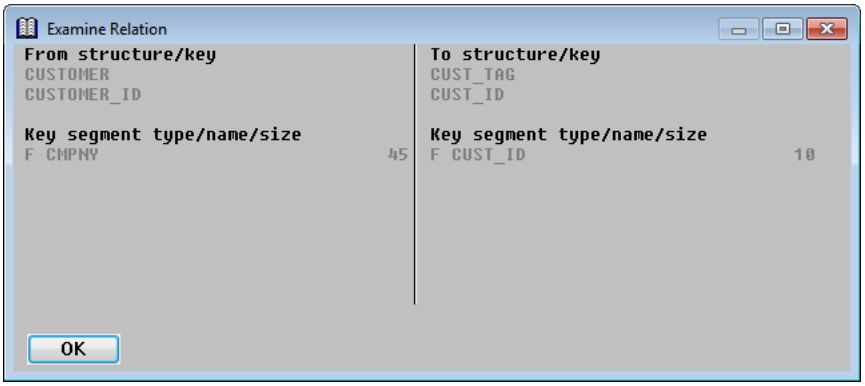


Figure 4-8. Examining a relation.

If the segment is a field or external segment, the field name is specified. To display the field description instead of the field name, select Examine Relation > Toggle View. If the segment is a literal, the literal string is displayed.

3. Exit the window.

Modifying a relation

1. From the Relation Definitions list, highlight the relation you want to modify and press ENTER.
2. In the Relation Definition window, modify data as desired. For details on the fields, see step 2 under "Defining a new relation" on page 4-19. The relation name and the "from" structure cannot be changed.
3. Exit the window to save your changes.

Deleting a relation

1. Highlight the relation in the Relation Definitions list.
2. Select Relation Functions > Delete Relation.
3. At the prompt, select Yes to delete the specified relation or No to cancel the deletion.

5

Utility Functions

Generating a Definition File

The Generate Definition File utility generates a definition file from a repository structure. This file can be .INCLUDEd in your program, and so is often referred to as an ".INCLUDE file". You can also .INCLUDE directly from the repository; see .INCLUDE in the "Preprocessor and Compiler Directives" chapter of the *Synergy DBL Language Reference Manual* for more information.

When generating a definition file, fields are generated by name, with the exception of fields that have been designated as "Excluded by Language", which are generated as unnamed fields of a given size. (Excluded overlay fields are not generated.) See Excluded by Language on page 3-10.

If any group definitions include a member prefix specification and the "Use by compiler" flag is set, the prefix is included when the members of that group are generated. See Use by compiler on page 3-10.

1. Select Utilities > Generate Definition File. The default repository filenames displayed are determined by the logic discussed in "Determining the repository files used" on page 1-18.

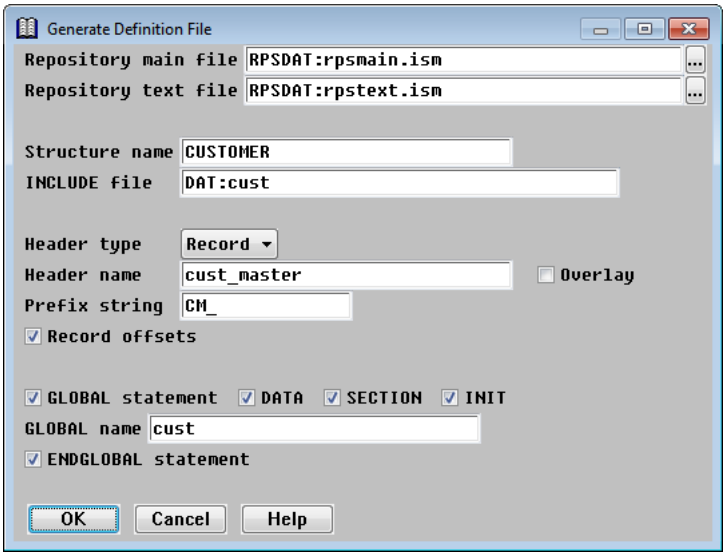


Figure 5-1. Generating a definition file.

2. Enter data in each field as instructed below.

Repository main file. Enter or select the name of the repository main file from which a definition file should be generated.

Repository text file. Enter or select the name of the repository text file from which a definition file should be generated.

Structure name. Enter the name of the structure for which you want to generate a definition. The file will contain only the definition of this structure, but you can generate definition files for other structures and append them to create a single file containing multiple definitions; see step 3 below.

INCLUDE file. Enter a name for the .INCLUDE file into which the definition will be generated. If you don't specify an extension, it defaults to **.def**. By default, the file is created in the current working directory.

Header type. Select the header type for the record definition:

Record	Definition is a local record or global data structure. (default)
Common	Definition is a shared data record.
None	Definition has no header.

If you select None, the cursor moves to the Prefix string field.

Header name. If you selected **Record** or **Common** for the header type and want the definition to be named, enter the name here.

Overlay. Set Overlay to indicate that the definition is to be a record overlay. When the definition is created, "**,X**" will follow the header. The header type must be **Record** or **Common** if you specify an overlay.

Prefix string. Enter a string that will prefix each field definition. You can use different prefix strings to distinguish fields from the same structure in different definition files (since Synergy DBL does not permit two fields to have the same name).

Record offsets. Set Record offsets to include record offsets in the field comments. By default, Record offsets is set.

GLOBAL statement. Set this option to precede the record definition with the Synergy DBL GLOBAL statement.

DATA, SECTION, INIT. If GLOBAL statement is set, you can include up to three optional keywords in the GLOBAL statement. The DATA and SECTION keywords do not affect the function of the GLOBAL statement, although you might want to include them for clarity. The INIT keyword enables you to declare initial values within the global data area.

Here's how the optional keywords will appear in the GLOBAL statement:

```
GLOBAL [DATA] [SECTION] name [,INIT]
```

GLOBAL name. If GLOBAL statement is set, enter a name to identify the global section.

ENDGLOBAL statement. Set this option to follow the record definition with the ENDGLOBAL statement. If GLOBAL statement is set, this option is selected automatically, but you can clear the check box if desired. For example, you would want to clear this field if you were appending a record definition to an existing file.

3. Exit the window to generate the definition.

   If the filename that you specified in the INCLUDE file field already exists, you are prompted

   > **Cannot create file** *filename***. File already exists. Select "No" to append output to the existing file; "Yes" to delete it.**

   Select Yes to overwrite the existing file with the new definition. Select No to append the definition to the existing file. Select Cancel to return to the INCLUDE file field and enter another filename. When processing is complete, you are returned to the Utilities menu.

Examples

The file below is an example of a definition file that was generated by Repository.

```
; Structure                : CUSTOMER
; File Type                : DBL ISAM
; Creation Date            : 24-FEB-2009, 10:00:56
; Description              : Customer master file
;    Record Size           : 146
;    # of Files            : 1
;    # of Fields           : 10
;    # of Keys             : 2
;    # of Relations        : 2

GLOBAL DATA SECTION cust

record cust_master
    CM_CUST_ID             ,D6                  ; (1,6) Customer ID
    CM_CUST_NAME           ,A30                 ; (7,36) Cust name
    CM_SALES_REP           ,D3                  ; (37,39) Sales rep ID
    CM_CUST_CONT           ,A25                 ; (40,64) Main contact
    CM_ADDRESS             ,A40                 ; (65,104) Address
    CM_CITY                ,A15                 ; (105,119) City
    CM_STATE               ,A2                  ; (120,121) State
    CM_ZIP                 ,A10                 ; (122,131) Zip code
    CM_MAILADR              ,A67  @CM_ADDRESS   ; (65,131) Mail address
    CM_PHONE               ,A15                 ; (132,146) Telephone #
ENDGLOBAL
```

All date and time fields are stored as decimal and are indicated accordingly in the generated file. (For example, a **DT8** field is generated as **D8** in the definition file.) All user type fields (for example, **U6**) are generated as alpha (for example, **A6**).

Loading fields from a definition file

In addition to generating a definition file from a defined structure, Repository also enables you to do the reverse: generate repository entries from a definition file. See "Loading Fields from a Definition File" on page 3-30.

Printing Repository Definitions

The Print Repository Definitions utility generates a listing of your repository definitions to a file.

1. Select Utilities > Print Repository Definitions. The default repository filenames displayed are determined by the logic discussed in "Determining the repository files used" on page 1-18.

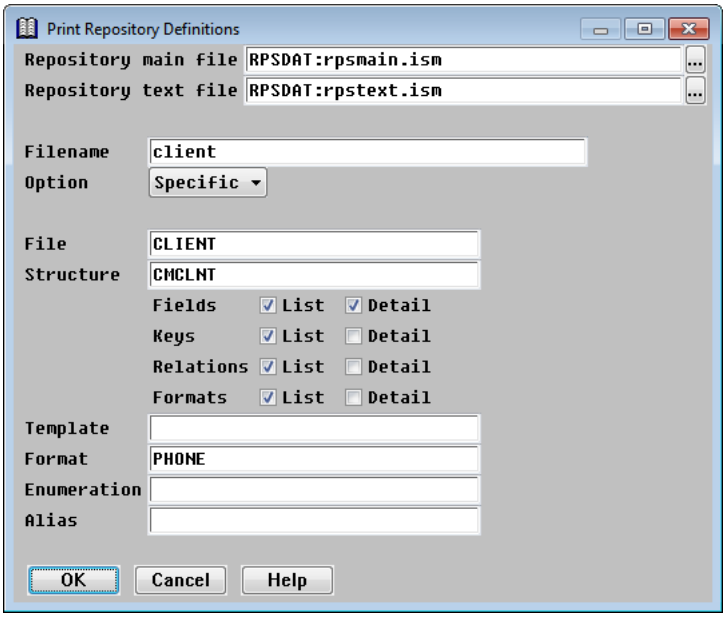


Figure 5-2. Printing your repository to a file.

2. Enter data in each field as instructed below.

   Repository main file. Enter or select the name of the repository main file to be printed.

   Repository text file. Enter or select the name of the repository text file to be printed.

   Filename. Enter the name of the file into which the contents of the repository should be printed. If you don't specify an extension, it defaults to **.ddf**. By default, the file is created in the current working directory.

Option. Indicate whether you want to print all definitions in the repository or specific (selected) definitions. Selecting Specific enables you to specify individual definitions by name. You can print a file, structure, template, format, enumeration, alias, or any combination thereof.

When selecting specific definitions to print, you can enter a partial name combined with wildcard characters (* or ?) to specify a set of definitions.

For the File, Structure, Template, Format, Enumeration, and Alias fields, you can select from a list of definitions by selecting Utility Functions > List Selections.

File. If you selected **Specific** in the Option field, enter the name of a file definition to print.

Structure. If you selected **Specific** in the Option field, enter the name of a structure definition to print.

Fields, Keys, Relations, Formats. If you selected **All** in the Option field or selected **Specific** and specified a structure name, indicate the level of detail (List or Detail) you want to print for field, key, relation, and structure-specific format definitions.

Select **List** to print definition information in a format similar to the way Repository lists data. For example:

```
        FIELD NAME  TYPE  SIZE  PREC  DIMENSION  OVERLAY?  GROUP?
        ----------  ----  ----  ----  ---------  --------  ------
1       CCKEY        A       8
2       CCCOMP       A       2                      Y
3       CCCLNT       A       6                      Y
4       CCNAME       A      40
5       CCADD1       A      40
6       CCCITY       A      25
7       CCEST        DT      8
8       CCBFLDR      D       4
9       CCUSRC       A       5          [5]
10      CCUSRI       A      40          [5]
```

Select **Detail** to print one or more lines of information for that definition type, describing each definition attribute. For example:

```
Field CCKEY   Type ALPHA   Size 8
   Description "Primary key"
```

Template. If you selected **Specific** in the Option field, enter the name of a template definition to print.

Format. If you selected **Specific** in the Option field, enter the name of a format definition to print.

Enumeration. If you selected **Specific** in the Option field, enter the name of an enumeration definition to print.

Alias. If you selected **Specific** in the Option field, enter the name of an alias definition to print.

3. Exit the window to print the repository definitions to the specified file.

   If the file already exists, you are prompted

   > **Cannot create file *filename*. File already exists. Select "No" to append output to the existing file; "Yes" to delete it.**

   Select Yes to overwrite the file with the new definitions. Select No to append the definitions to the existing file. Select Cancel to return to the Filename field and enter another filename.

   As the repository is being printed, status messages are displayed in the lower-left corner of the window. When processing is complete, a message displays the number of each type of definition printed.

4. To return to the Utilities menu, press ENTER.

Examples

This example shows sample output from the Print Repository Definitions utility.

```
;  "PRINT REPOSITORY OUTPUT"
;
;  REPOSITORY : RPSDAT:rpsmain
;             : RPSDAT:rpstext
;             : Version 9.1
;
;  GENERATED  : 06-AUG-2009, 13:20:17
;             : Version 9.1.5

File PRODUCT   DBL ISAM   "DAT:product.ism"
   Description "Product management file"
   Assigned structures : 1
   Structure PRODUCT   DBL ISAM
      Description "Product management"

Structure PRODUCT   DBL ISAM
   Description "Product management"
   Files assigned to : 1
     File PRODUCT   DBL ISAM   "DAT:product.ism"
        Description "Product management file"
   Record size [88]

       Field count    : 7

       FIELD NAME  TYPE  SIZE  PREC  DIMENSION  OVERLAY?  GROUP?
       ----------  ----  ----  ----  ---------  --------  ------
1      PRDT_ID       D     3
2      PRDT_NAME     A    25
3      PRDT_PRICE    D     8     2
4      PRDT_MNGER    D     3
```

```
5      PRDT_DATE     DT    8
6      PRDT_STATUS   A     1
7      PRDT_DESC     A    40

Field PRDT_ID   Type DECIMAL   Size 3
   Description "Product ID"
   Report Just LEFT   Prompt "ID: "   Break   Required

Field PRDT_NAME   Type ALPHA    Size 25
   Description "Product name"
   Prompt "Name: "   Uppercase

Field PRDT_PRICE   Type DECIMAL   Size 8   Precision 2
   Description "Product price"
   Report Just LEFT   Format MONEY
   Prompt "Price: "   Nodecimal   Blankifzero

Field PRDT_MNGER   Type DECIMAL   Size 3
   Description "Product manager ID"
   Report Just LEFT

Field PRDT_DATE   Type DATE   Size 8   Stored YYYYMMDD
   Description "Product available date"
   Format "#03  MM-DD-YYYY"   Prompt "Date: "
   Date Today   Date Short

Field PRDT_STATUS   Type ALPHA   Size 1
   Description "Prdt portability status"
   Prompt "Status: "   Allow "A", "D", "O", "I"

Field PRDT_DESC   Type ALPHA   Size 40
   Description "Status description"
   Prompt "Description: "

Key count      : 2

       KEY NAME            TYPE  ORDER  DUPS? SEGMENTS  SIZE
       --------            ----  -----  ----- --------  ----
1      PRODUCT_ID           A      A     N    F           3
2      PRDT MANAGER_ID      A      A     Y    F           3

Key PRODUCT_ID   ACCESS   Order ASCENDING   Dups NO
   Segment FIELD   PRDT_ID

Key PRDT_MANAGER_ID  ACCESS  Order ASCENDING  Dups YES
   Segment FIELD   PRDT_MNGER

Relation count : 2
```

```
       NAME   FROM KEY     TO STRUCTURE : TO KEY
       ----   --------     ---------------------
1        1    PRODUCT_ID   ORDER : PRODUCT_ID
2        2    PRODUCT_ID   EMPLOYEE : EMPLOYEE_ID

Relation 1  PRODUCT PRODUCT_ID  ORDER PRODUCT_ID

Relation 2  PRODUCT PRODUCT_ID  EMPLOYEE EMPLOYEE_ID

Format count   : 1

       FORMAT NAME  TYPE   FORMAT STRING
       -----------  ----   -------------
1      MONEY         N     $######.##

Format MONEY   Type NUMERIC   "$######.##"   Justify RIGHT
```

Verifying Your Repository

The Verify Repository utility verifies the integrity of your repository and attempts to repair any inconsistencies it finds. It then writes the repaired files to a new repository. You can specify the name of the repository files to check, the name of the new repository to be created, and the name of the log file where messages will be recorded. Run this utility if you suspect your repository is corrupted. Depending on the size of your repository, this utility can take a long time to run.

1. Ensure that the file **rpsload.ddf** is in the RPS directory.
2. Select Utilities > Verify Repository.

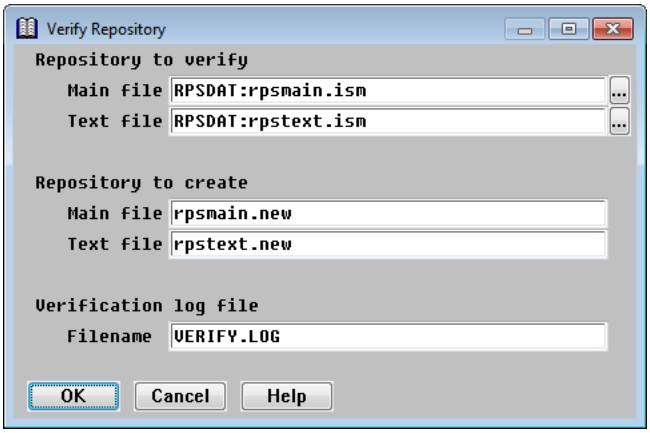


Figure 5-3. Verifying your repository.

3. Enter data in each field as instructed below.

   Repository to verify. Enter or select the names of the repository main file and the repository text file to verify. The default repository filenames displayed are determined by the logic discussed in "Determining the repository files used" on page 1-18.

   Repository to create. Enter the names of the repository main file and the repository text file to be created. If the utility finds inconsistencies in the original repository, it will attempt to repair them and then write the repaired repository to these files. If the utility finds no inconsistencies, these files are deleted.

   The default filenames for the newly created repository are **rpsmain.new** (and **rpsmain.ne1**) and **rpstext.new** (and **rpstext.ne1**). You can change these default names to whatever you like, but you can't use the same names as the repository files being verified.

   Verification log file. Enter the name of the file to which you want inconsistencies logged. If the file already exists, it is overwritten. The default log file is **VERIFY.LOG**. If the utility finds no inconsistencies, this file will contain summary information only.

4. Exit the window to verify the specified repository.

   As the utility runs, status messages are displayed in the lower-left corner of the window. When processing is complete, a message displays the name of the new repository files (if inconsistencies were found) and the name of the log file.

5. To return to the Utilities menu, press ENTER.

Examples

The example below shows the type of summary information that the log file contains if no inconsistencies are found.

```
23 structures verified
10 files verified
2 formats verified
6 templates verified
3 enumerations verified
792 logical records read
792 logical records written
```

If inconsistencies are found, they are listed before the summary information. There are two classes of inconsistencies, informational and warning, which are preceded with either “INFO” or “WARN” in the log file. Items marked with INFO were repaired by the utility. Items marked with WARN could not be repaired. Contact Synergy/DE Developer Support if your log file contains warnings.

Validating Your Repository

The Validate Repository utility validates every definition in your repository. It performs most of the same validations that would occur if you were to edit each definition within Repository. You should run this utility on every new repository that you create with the Load Repository Schema utility, especially those that have been merged.

1. Select Utilities > Validate Repository. The default repository filenames displayed are determined by the logic discussed in "Determining the repository files used" on page 1-18.

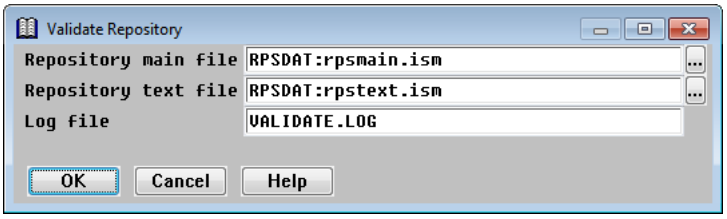


Figure 5-4. Validating repository definitions.

2. Enter data in each field as instructed below.

   Repository main file. Enter or select the name of the repository main file to validate.

   Repository text file. Enter or select the name of the repository text file to validate.

   Log file. Enter the name of the file in which you want errors to be logged. By default, the file is named **VALIDATE.LOG** and created in the current working directory. If the file already exists, it is overwritten. If the validation utility finds no errors, the file will be empty.

3. Exit the window to validate the specified repository.

   As the repository is being validated, status messages are displayed in the lower-left corner of the window. When processing is complete, a message displays the number of errors logged.

4. To return to the Utilities menu, press ENTER.

Examples

The example below shows the type of information that the log file contains if errors are found.

```
Format MONEY: Invalid justification specified.
Template DIG8MONY: Invalid format name specified.
```

Generating a Repository Schema

The Generate Repository Schema utility generates a Synergy Data Language description of your repository to a file. This description is referred to as a *schema*. See chapter 6, "Synergy Data Language," for more information about Synergy Data Language and possible uses for a repository schema file.

You can also run this utility from the command line. See "Generating and Loading Schema from the Command Line" on page 5-28.

1. Select Utilities > Generate Repository Schema. The default repository filenames displayed are determined by the logic discussed in "Determining the repository files used" on page 1-18.

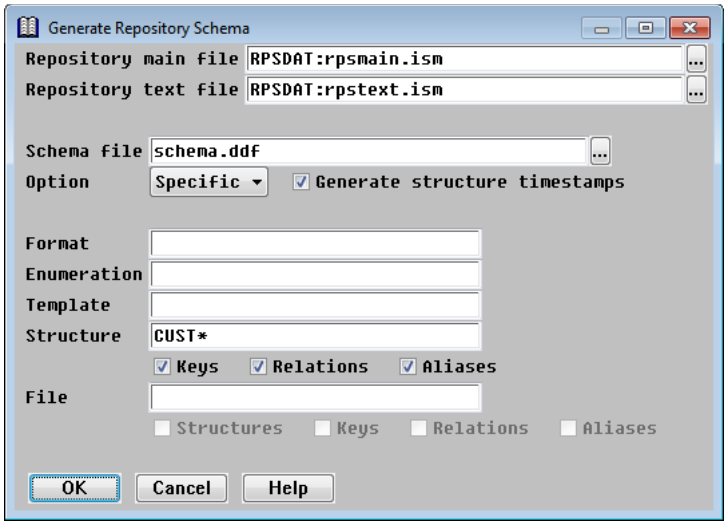


Figure 5-5. Generating a repository schema.

2. Enter data in each field as instructed below.

   Repository main file. Enter or select the name of the repository main file from which a schema should be generated.

   Repository text file. Enter or select the name of the repository text file from which a schema should be generated.

   Schema file. Enter the name of the file into which the repository schema should be generated. If you don't specify an extension, it defaults to **.ddf**. By default, the file is created in the current working directory.

Option. Indicate whether you want to generate a schema for all definitions in the repository or for specific (selected) definitions. Selecting Specific enables you to generate schema for a specific format, enumeration, template, structure, file, or any combination thereof.

If you select **All**, the generated schema file will be valid for use by the Load Repository Schema utility. The **Specific** option could result in a Synergy Data Language file that contains duplicate definitions or missing (referenced) definitions, and which therefore may not be valid for use by the Load Repository Schema utility. See "General usage rules" on page 6-3 for a description of the allowed structure of a repository schema file.

When selecting specific definitions, you can enter a partial name combined with wildcard characters (* or ?) to specify a set of definitions.

For the Format, Enumeration, Template, Structure, and File fields, you can select from a list of definitions by selecting Utility Functions > List Selections.

Generate structure timestamps. Select this option to include the date and time that each structure definition was last modified. This option adds the MODIFIED keyword to each generated structure definition. If you manually edit the schema, you should update this value for any structures that you change. See STRUCTURE on page 6-60 for more information.

Format. If you selected **Specific** in the Option field, enter the name of a format definition to generate.

Enumeration. If you selected **Specific** in the Option field, enter the name of an enumeration definition to generate.

Template. If you selected **Specific** in the Option field, enter the name of a template definition to generate.

Structure. If you selected **Specific** in the Option field, enter the name of a structure definition to generate. This can be the name of an alias structure, but you must type it in, rather than select it using the List Selections option.

Keys, Relations, Aliases. If you selected **All** in the Option field or selected **Specific** and specified a structure name, indicate whether you want to generate the keys, relations, and aliases. By default, all definitions associated with a structure are generated; clear the check boxes for those you do not want generated.

File. If you selected **Specific** in the Option field, enter the name of a file definition to generate.

Structures, Keys, Relations, Aliases. If you selected **Specific** in the Option field and specified a filename in the File field, indicate whether you want to generate the file's assigned structures as well. If you select Structures, all definitions associated with each assigned structure are generated by default; clear the check boxes for Keys, Relations, and Aliases if you do not want to generate those definitions.

To set these options when you want to generate *all* files, select Specific in the Option field and enter "*" in the File field.

3. Exit the window to generate the repository schema.

   As the repository schema is being generated, status messages are displayed in the lower-left corner of the window. When processing is complete, a message that lists the number of each type of definition generated is displayed.

4. To return to the Utilities menu, press ENTER.

Synergy Data Language file header

The output file generated by this utility includes a header that lists the name and version of the repository being generated, the date and time the utility was run, and information about the export options that were chosen. Possible export options include the following:

```
[ALL]|[FORMAT=name][ENUMERATION=name][TEMPLATE=name]
[STRUCTURE=name][FILE=name]
```

Additionally, the `[ALL]`, `[STRUCTURE]`, and `[FILE]` options may include abbreviations relative to the generated output. Possible abbreviations include the following:

-K	Exclude keys
-R	Exclude relations
-A	Exclude aliases
+S	Include assigned structures

For example, if you select **All** in the Option field and clear the default Keys setting, the information in the export options line in the output file would look like this:

```
;  EXPORT OPTIONS : [ALL-K]
```

If you select **Specific** in the Option field, specify the file definition name **CUSTOMER**, and choose to generate its assigned structures but not their aliases, the export options line would look like this:

```
;  EXPORT OPTIONS : [FILE+S-A=CUSTOMER]
```

Examples

The file below is an example of the Synergy Data Language output generated by the Generate Repository Schema utility when the **All** option is selected.

```
;  SYNERGY DATA LANGUAGE OUTPUT
;
;  REPOSITORY     : RPSDAT:rpsmain
;                 : RPSDAT:rpstext
;                 : Version 9.1
;
;  GENERATED      : 06-AUG-2009, 13:23:30
;                 : Version 9.1.5
;  EXPORT OPTIONS : [ALL]
```

```
Format STD_ID   Type NUMERIC   "ZZ-Z"

Template DIG8DATE   Type DATE   Size 8   Stored YYYYMMDD
   Description "8-digit date"
   Date Today   Date Short

Structure EMPLOYEE   DBL ISAM
   Description "Employee master file"

Field EMP_ID   Type DECIMAL   Size 3
   Description "Employee ID"
   Report Just LEFT   Format STD_ID
   Break   Required   Paint "*"

Field EMP_NAME   Type ALPHA   Size 25
   Description "Employee name"
   Info Line "Enter your full name."

Field EMP_DEPT   Type DECIMAL   Size 2
   Description "Department ID"
   Report Just LEFT

Field EMP_MNGR   Type DECIMAL   Size 3
   Description "Manager ID"
   Report Just LEFT

Field EMP_TITLE   Type ALPHA   Size 25
   Description "Title"
   Uppercase

Field EMP_DATE   Template DIG8DATE
   Description "Starting date"

Field EMP_STATUS   Type ALPHA   Size 1
   Description "Employee status"
   Selection List 2 2 3  Entries "A", "I", "V"

Key EMPLOYEE_ID   ACCESS   Order ASCENDING   Dups NO
   Segment FIELD   EMP_ID

Structure ORDER   DBL ISAM
   Description "Sales order management"

Field ORD_ID   Type DECIMAL   Size 6
   Description "Order ID"
   Report Just LEFT   Break   Noterm   Blankifzero

Field ORD_ITEM   Type DECIMAL   Size 3
   Description "Order item (product ID)"
   Report Just LEFT   Range 1 100
```

```
Field ORD_DATE   Template DIG8DATE
   Description "Order initiated date"

Field ORD_SUB   Type DECIMAL   Size 2   Dimension 2:2
   Overlay ORD_DATE:0
   Description "Order date subscripted"
   Report Just LEFT

Field ORD_STATUS   Type ALPHA   Size 1
   Description "Status"
   Allow "O", "S", "B"

Field ORD_DESC   Type ALPHA   Size 40
   Description "Status description"

Key ORDER_ID   ACCESS   Order ASCENDING   Dups NO
   Segment FIELD   ORD_ID

Key PRODUCT_ID   ACCESS   Order ASCENDING   Dups YES
   Segment FIELD   ORD_ITEM

Structure PRODUCT   DBL ISAM
   Description "Product management"

Format MONEY   Type NUMERIC   "$######.##"   Justify RIGHT

Field PRDT_ID   Type DECIMAL   Size 3
   Description "Product ID"
   Report Just LEFT   Format STD_ID
   Prompt "ID: "   Break   Required

Field PRDT_NAME   Type ALPHA   Size 25
   Description "Product name"
   Prompt "Name: "   Uppercase

Field PRDT_PRICE   Type DECIMAL   Size 8   Precision 2
   Description "Product price"
   Report Just LEFT   Format MONEY
   Prompt "Price: "   Nodecimal   Blankifzero

Field PRDT_MNGER   Type DECIMAL   Size 3
   Description "Product manager ID"
   Report Just LEFT

Field PRDT_DATE   Template DIG8DATE
   Description "Product available date"

Field PRDT_STATUS   Type ALPHA   Size 1
   Description "Prdt portability status"
   Prompt "Status: "   Allow "A", "D", "O", "I"
```

```
Field PRDT_DESC   Type ALPHA   Size 40
   Description "Status description"
   Prompt "Description: "

Key PRODUCT_ID   ACCESS   Order ASCENDING   Dups NO
   Segment FIELD   PRDT_ID

Key PRDT_MANAGER_ID   ACCESS   Order ASCENDING   Dups YES
   Segment FIELD   PRDT_MNGER

Relation  1   PRODUCT PRODUCT_ID   ORDER PRODUCT_ID

Relation  2   PRODUCT PRODUCT_ID   EMPLOYEE EMPLOYEE_ID

File EMPLOYEE   DBL ISAM   "DAT:employee.ism"
   Description "Employee master file"
   Assign EMPLOYEE

File ORDER   DBL ISAM   "DAT:order.ism"
   Description "Sales order management file"
   Assign ORDER

File PRODUCT   DBL ISAM   "DAT:product.ism"
   Description "Product management file"
   Assign PRODUCT
```

Loading a Repository Schema

The Load Repository Schema utility converts the contents of a Synergy Data Language file (i.e., a schema) into one or more repository definitions. You can use this utility to add or update definitions in an existing repository or create a new repository. See "General processing rules" on page 6-5 for specific information on how each definition type is handled when merging and for specific Synergy Data Language requirements. Depending on the size of your repository, this utility can take a long time to run.

You can also run this utility from the command line. See "Generating and Loading Schema from the Command Line" on page 5-28.

1. Ensure that the file **rpsload.ddf** is in the RPS directory.
2. Select Utilities > Load Repository Schema. The default repository filenames displayed are determined by the logic discussed in "Determining the repository files used" on page 1-18.

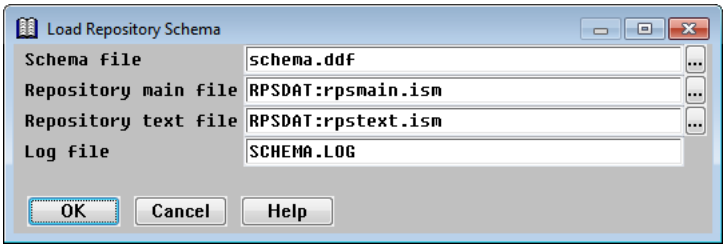


Figure 5-6. Loading a repository schema.

3. Enter data in each field as instructed below.

   Schema file. Enter or select the name of the file that contains the repository schema (the Synergy Data Language description) to be loaded.

   Repository main file. Enter or select the name of the repository main file to be created from the schema or the name of an existing repository into which the schema will be merged.

   Repository text file. Enter or select the name of the repository text file to be created or, if you are merging the schema, enter the name of the existing repository text file.

   Log file. Enter the name of the log file into which error messages generated during schema loading should be written. By default, the file is named **SCHEMA.LOG** and created in the current working directory. If the file already exists, it is overwritten.

4. Exit the window to load the repository schema.

   If the repository files that you specified already exist, you are prompted

   Repository files already exist. Do you want to merge the schema?

   Select No to specify different repository files. Select Yes to display the Merge Repository Schema window. (See figure 5-7.)

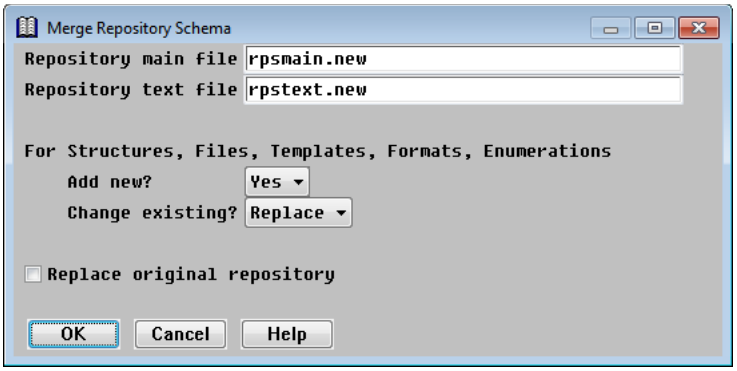


Figure 5-7. Merging a repository schema.

5. Enter data in each field as instructed below.

   Repository main file. Enter the name of the "merged" repository main file to be created. The default filename is **rpsmain.new**.

   Repository text file. Enter the name of the "merged" repository text file to be created. The default filename is **rpstext.new**.

We strongly recommend that when merging a repository, both the original files and the "merged" files be on the same drive on the same system. In some cases, failing to do this results only in slower performance. In other cases (such as when the original files are on Windows and the merged files on UNIX), the procedure fails with a rename error.

   Add new. If there are new definitions in your schema file and you want to add them to the repository, select **Yes**. If there are new definitions in the schema file that you *do not* want added to the repository, remove them from the file; selecting **No** when the file contains new definitions will generate an error. If you are only updating existing definitions and there are *no* new definitions in the schema file, this setting has no effect so you can choose either option.

   Change existing. If you want to update existing definitions in the repository based on corresponding definitions found in the schema file, select how you want the update performed. Select **Replace** if you want to delete the existing definition and replace it with the definition in the schema file. Select **Overlay** if you want to replace only a subset of attributes or add to those of an existing definition. Overlay updates existing attributes and adds new ones, but does not delete any.

If there are existing definitions in the file and you *do not* want to either replace or overlay the repository definitions, remove them from the file; selecting **No** when the file contains existing definitions will generate an error.

The Load Repository Schema utility attempts to load *all* the definitions in the schema file into the repository. If your file contains definitions you do not wish to load, you must remove them from the file. For example, if your schema file contains both new and existing definitions, and you want only to update the existing ones, you must remove the new definitions from the file before loading the schema.

If you're loading a schema file created by Toolkit's Script-to-Repository conversion program (**scridl**), specify the name of the existing repository in the Repository main file and Repository text file fields. In the Add new field, select **Yes**; in the Change existing field, select **Overlay**.

Replace original repository. When merging, the utility will create a copy of your repository. Select this field if you want the "merged" (copied) repository files to replace the original repository files if no errors occur. If you do not select this field, you can rename the merged repository files manually if desired.

6. Exit the window to load the repository schema.

   As the repository schema is being loaded, status messages are displayed in the lower-left corner of the window. The utility validates all Synergy Data Language statements. When an error occurs in a statement, the utility writes a message to the log file and ignores the entire statement.

   When processing is complete, Repository displays a message that lists the number of definitions that were loaded or the number of errors that were logged.

7. To return to the Utilities menu, press ENTER.

8. If any errors were logged, check the log file for specific definition names and error messages, correct the schema file, and reload it. The definition type and name precede the error message. For example,

   ```
   Format PHONE: Invalid format type specified.
   Field DEPT (structure EMP1): Invalid data type specified.
   ```

   If *any* error occurs while a repository is being loaded from a schema, the new (or merged) repository is not created.

9. After you have successfully loaded the schema, run both the Verify Repository and Validate Repository utilities. Certain repository integrity checks cannot be performed during schema merging and must be reported on by one of these utilities.

Creating a New Repository

The Create New Repository utility creates and initializes a new, blank repository. Every new repository contains the predefined date and time formats listed in "Appendix B: Date and Time Formats".

1. Ensure that the file **rpsload.ddf** is in the RPS directory.
2. Select Utilities > Create New Repository. The default repository filenames displayed are determined using the logic discussed in "Determining the repository files used" on page 1-18.

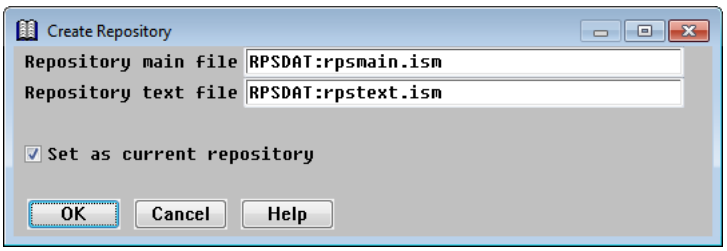


Figure 5-8. Creating a new repository.

3. Enter data in each field as instructed below.

   Repository main file. Enter or select the name of the repository main file to be created. The default extension is **.ism**.

   Repository text file. Enter or select the name of the repository text file to be created. The default extension is **.ism**.0

TIP Although you can name the repository main and text files anything you like, we recommend that you include "main" and "text" in the filenames. Not only does this help to identify the files as repository files, it also enables you to take advantage of the filename defaulting that occurs in all main and text file fields in Repository dialogs: After you enter or select the name of a repository main file and exit the field, Repository enters a default repository text filename by copying the main filename and changing the last occurrence of the characters "main" to "text".

   Set as current repository. If this field is selected, the newly created repository will become the default repository for the current session. If you don't want to change the current default repository, clear the field.

4. Exit the window to create the new repository.

   If either of the repository files already exists, the utility displays an error message and does not create the files. You must either specify new filenames or delete the old repository and run the utility again.

Setting the Current Repository

The Set Current Repository utility changes the default repository filenames for the current Repository session by temporarily setting the RPSMFIL and RPSTFIL environment variables. The default repository filenames determine the repository to open when you select an entry from the Modify or View menu, and are also used to prefill the Repository main file and Repository text file fields in the Utility functions.

You can also use this utility to check the current repository setting.

1. Select Utilities > Set Current Repository. The Set Current Repository dialog displays the filenames for the current repository.

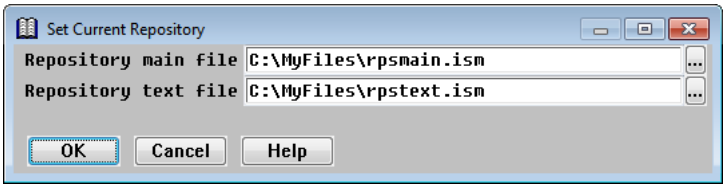


Figure 5-9. Setting the current repository.

2. Enter or select the name of the main file to use and exit the field; Repository enters a default repository text filename by copying the main filename and changing the last occurrence of the characters “main” to “text”. (You can, of course, change the text filename if necessary.) The default extension is **.ism**.
3. Exit the window to set the current repository.

Generating a Cross-Reference File

The Generate Cross-Reference utility generates a file that can be used by ReportWriter to access file relationships that are not explicitly defined in the repository. These relationships are based on name links between fields. Provided that corresponding data fields have the same name (or use the same template), this file can provide access to all potential relationships that exist within your repository. If a cross-reference file exists, you can add related files to a report even though a "relation" has not been explicitly defined.

You can also run this utility from the command line. See "Rpsxref command line syntax" on page 5-26.

This utility generates a cross-reference file for the specified repository in its current state. If you add or modify fields, templates, or keys in the repository, you must regenerate the cross-reference file.

1. Select Utilities > Generate Cross-Reference. The default repository filenames displayed are determined by the logic discussed in "Determining the repository files used" on page 1-18.

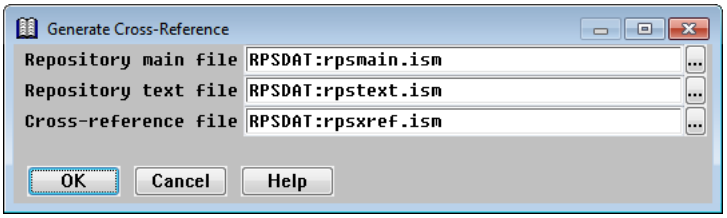


Figure 5-10. Generating a cross-reference.

2. Enter data in each field as instructed below.

   Repository main file. Enter or select the name of the repository main file to read.

   Repository text file. Enter or select the name of the repository text file to read.

   Cross-reference file. Enter or select the name of the file into which the cross-reference should be generated. The default cross-reference filename displayed is determined by the logic discussed in "Determining the cross-reference file used", below. If you enter a different filename, you must set the RPSXFIL environment variable to access this file in ReportWriter.

3. Exit the window to generate the file.

   When processing is complete, the name of the newly generated cross-reference file is displayed.

4. To return to the Utilities menu, press ENTER.

Determining the cross-reference file used

Repository and ReportWriter search for the cross-reference file as follows:

- If the RPSXFIL environment variable is defined, the cross-reference filename is the value of RPSXFIL.
- If RPSXFIL is not defined, Repository and ReportWriter attempt to open the file as **RPSDAT:rpsxref.ism**.
- If **RPSDAT:rpsxref.ism** can't be opened, Repository and ReportWriter attempt to open **rpsxref.ism** in the current directory.

How a cross-reference file is generated

Each field (and template) in the repository has a "name link" flag. This flag determines whether the field name itself or an alternate name link should be used to find matches. The alternate name link is the name of the field's template or the name of the template's parent if the name link flag is set.

Step 1: A name link is determined for each field in the repository. This name link could be the field name, its template's name, its template's parent's name, and so forth, depending on how the name link flag is set. For more information, see the "Do not name link" on page 3-42 for field templates or the "Do not name link" on page 3-11 for fields.

Step 2: For each access key (in all structures), the name link is determined for the first segment of the key. If the first segment is the tag field, the name link for the second segment is used. (The tag must be defined as a **Field** type tag and must use the EQ [equal] connector.)

Step 3: The list of name links determined in step 1 is compared to every name link determined in step 2. Each match becomes a record in the cross-reference file (assuming they're not in the same structure).

For information on how ReportWriter uses this file to access related data, see "Determining the list of available secondary files based on name links" in the "Miscellaneous ReportWriter Information" chapter of the *ReportWriter User's Guide*.

Rpsxref command line syntax

You can generate a cross-reference file by running **rpsxref.dbr** from the command line.

- On Windows and UNIX, use this syntax:

  `dbr rpsxref [-d main_file text_file] [-o output_file]`

- On OpenVMS, define this symbol:

  `rpsxref:==$RPS:rpsxref.exe`

  and then use this syntax:

  `rpsxref [-d main_file text_file] [-o output_file]`

Arguments

-d *main_file text_file*

(optional) Specify the names of the repository main and text files, overriding the defaults. The default repository filenames are determined by the logic discussed in "Determining the repository files used" on page 1-18. The *main_file* and *text_file* names can each have up to 255 characters.

-o *output_file*

Specify the name of the cross-reference file to create, overriding the default. The default cross-reference filename is determined by the logic discussed in "Determining the cross-reference file used" on page 5-25. The cross-reference filename can have up to 255 characters.

Examples

The command below generates the cross-reference file **MYRPS:rpsxref.ism** based on the repository files in **MYRPS:rpsmain.ism** and **MYRPS:rpstext.ism**.

```
dbr rpsxref -d MYRPS:rpsmain.ism MYRPS:rpstext.ism -o MYRPS:rpsxref.ism
```

Assuming that RPSMFIL, RPSTFIL, and RPSXFIL are not set, the following command generates the cross-reference file **RPSDAT:rpsxref.ism** based on the repository files **RPSDAT:rpsmain.ism** and **RPSDAT:rpstext.ism**.

```
dbr rpsxref
```

Comparing a Repository to ISAM Files

The Compare Repository to Files (**fcompare**) utility compares the definitions in the repository to the actual ISAM definitions that they represent and writes the results to a log file.

You can also run this utility from the command line. See fcompare in the "Synergy DBMS" chapter of *Synergy Tools*.

1. Select Utilities > Compare Repository to Files. The default repository filenames displayed are determined by the logic discussed in "Determining the repository files used" on page 1-18.

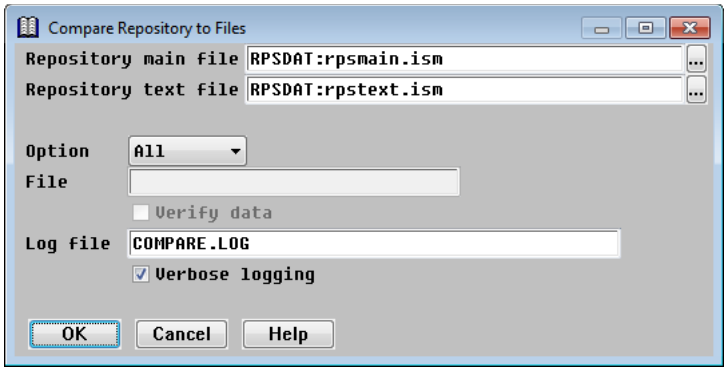


Figure 5-11. Comparing Repository and ISAM files.

2. Enter data in each field as instructed below.

   Repository main file. Enter or select the name of the repository main file to be compared.

   Repository text file. Enter or select the name of the repository text file to be compared.

   Option. To compare all repository file definitions with their associated physical files, select **All**. To compare a single repository file definition, select **Specific**.

   File. If you selected **Specific** in the Option field, enter the name of the file to compare.

   Verify data. Select this option to turn on data verification mode, which reads through all records in the ISAM file and verifies that date fields contain valid dates and decimal fields contain numbers. Verification mode is available only when Option is set to **Specific** and can significantly increase the time it takes to run a comparison.

   Log file. Specify a log file name for output. By default, the file is named **COMPARE.LOG** and created in the current working directory. If the file already exists, it is overwritten.

   Verbose logging. Select this option to log general information about the files (e.g., the number of keys and segments), warnings, and errors. Regular (non-verbose) logging logs only errors.

3. Exit the window to run the comparison.

Generating and Loading Schema from the Command Line

Instead of using the Generate Repository Schema utility and the Load Repository Schema utility, you can run the **rpsutl.dbr** program from the command line to generate (export) and load (import) Synergy Data Language files. See chapter 6, "Synergy Data Language," for more information about Synergy Data Language and possible uses for repository schema files.

To export, run **rpsutl** with the **-e** option; to import, run it with **-i**. See the sections below for the available options for these two operations. On Windows and UNIX, from the command line, run:

```
dbr rpsutl -e|-i options...
```

On OpenVMS, define this symbol:

```
rpsutl:==$RPS:rpsutl.exe
```

and then use this syntax:

```
rpsutl -e|-i options...
```

When **rpsutl** ends, if it encountered errors, it will exit with a status of D_EXIT_FAILURE (1 on Windows and UNIX; 0 on OpenVMS). Otherwise, it will exit with a status of D_EXIT_SUCCESS (0 on Windows and UNIX; 1 on OpenVMS). Most operating systems have commands that enable you to test for the exit status. See the documentation for your operating system for more information.

Exporting Synergy Data Language files

Run **rpsutl** with the **-e** option to export (generate) schema. You can export all repository definitions, export only certain types of definitions (formats, templates, structures, etc.), or export specific formats, templates, structures, etc., using the options below.

`rpsutl -e` *sdl_file* *[*`-d` *main_file text_file] [*`-em` *[format]] [*`-et` *[template]] [*`-es` *[structure[=[k][a][r]]]] [*`-ef` *[file[=[s][k][a][r]]]] [*`-ee` *{enumeration]] [*`-er`*] [*`-t`*]*

-e

Export (generate) a repository schema.

sdl_file

Specify a Synergy Data Language file to create to hold the schema. The maximum length of *sdl_file* is 255 characters. If you don't specify an extension, it defaults to .**ddf**. If the file already exists, an error is generated.

-d *main_file text_file*

(optional) Specify the names of the repository main and text files from which to generate the schema. If not specified, the defaults are used, following the logic discussed in "Determining the repository files used" on page 1-18.

-em *[format]*

(optional) Export format definitions. The default is to export all. To export a single format, specify it by name. See the Usage section below for information on using wildcards to export multiple formats.

-et *[template]*

(optional) Export template definitions. The default is to export all. To export a single template, specify it by name. See the Usage section below for information on using wildcards to export multiple templates.

-es *[structure[=[***k***][***a***][***r***]]]*

(optional) Export structure definitions. Default output includes the structure itself, fields, and tags. Additional output can include keys (**k**), aliases (**a**), and relations (**r**). *Structure* may be an alias structure. The default is to export all structures. To export a single structure definition, specify it by name. See the Usage section below for information on using wildcards to export multiple structures.

-ef *[file[=[***s***][***k***][***a***][***r***]]]*

(optional) Export file definitions. Default output includes file information only. Additional output can include structures (**s**), keys (**k**), aliases (**a**), and relations (**r**). Options **k**, **a**, and **r** are valid only if **s** is present. The default is to export all files. To export a single file definition, specify it by name. See the Usage section below for information on using wildcards to export multiple files.

-ee *[enumeration]*

(optional) Export enumeration definitions The default is to export all. To export a single enumeration, specify it by name. See the Usage section below for information on using wildcards to export multiple enumerations

-er

(optional) Export all relations. This option is equivalent to the **-es** *structure*=**r** option.

-t

(optional) Include in the schema the MODIFIED keyword and the date/time the structure was last modified. If you manually edit the schema, you should update this value for any structures that you change. See STRUCTURE on page 6-60 for more information.

-h

Display usage screen.

Usage

The variables for exporting specific elements in the repository (*format, template, structure, file, enumeration*) can include the wildcard characters * and ?. For example, to export all enumerations beginning with "color_", you would specify **-ee color_***.

If you want to use the **s**, **k**, **a**, or **r** options and export all structures or files, specify * before the equals sign. For example, to export keys, aliases, and relations for all structures, you would specify **-es *=kar**. To export assigned structures for all files, specify **-ef *=s**.

The following example exports all definitions from the repository located in **MYDICT:rpsmain.ism** and **MYDICT:rpstext.ism** into the file **mySchema.sdl**.

```
dbr rpsutl -e mySchema.sdl -d MYDICT:rpsmain.ism MYDICT:rpstext.ism
```

The following example exports the structure **CUSMAS** from the default repository into the file **custStruct.sdl**. But default, formats, fields, and tags are included; adding **=kar** specifies that keys, aliases, and relations should also be exported.

```
dbr rpsutl -e custStruct.sdl -es CUSMAS=kar
```

The following example uses wildcard characters to export selected formats and enumerations from the default repository into the file **mySchema.sdl**. Formats with names such as NUM1_FORMAT, NUM2_FORMAT, etc. will be exported, along with all enumerations beginning with "COLOR_".

```
dbr rpsutl -e mySchema.sdl -ef NUM?_FORMAT -ee COLOR_*
```

Importing Synergy Data Language files

Run **rpsutl** with the **-i** option to import (load) schema. The definitions to import may be in a single Synergy Data Language file or in multiple files. You can use this option to create a new repository, or you can merge new or updated definitions into an existing repository.

`rpsutl -i` *sdl_file […] [*`-d` *main_file text_file] [*`-ia`*] [*`-io`*]|[*`-ir`*] [*`-n` *new_main new_text] [*`-l` *[log]] [*`-r` *[sec]] [*`-s`*]*

-i

Import (load) a repository schema.

sdl_file […]

Specify one or more Synergy Data Language files to use for creating or merging a repository. The maximum number of files is 30, and the maximum length of *sdl_file* is 255 characters. If you don't specify an extension, it defaults to .**ddf**. On UNIX, wildcard specifications such as *.**sdl** are supported.

If you specify multiple Synergy Data Language files, each one is processed and validated separately. This means that all information required to validate a particular file must be contained within that file or, if you are merging definitions into an existing repository, must

already exist in the repository. If a specified file cannot be found, no further files will be processed, and, for a merge, the current repository will not be replaced.

Rpsutl attempts to load *all* the definitions in the schema file into the repository. If your file contains definitions you do not wish to load, you must remove them from the file before running **rpsutl**. For example, if your file contains both new and existing definitions, and you want only to update the existing ones, you must remove the new definitions from the file.

-d *main_file text_file*

(optional) Specify the names of the repository main and text files into which definitions will be loaded or merged. If the repository does not exist, it will be created (and will subsequently be deleted should any errors occur). If not specified, the defaults are used, following the logic discussed in "Determining the repository files used" on page 1-18.

-ia

Add new structures, files, templates, formats, and enumerations. Specify this option to create a new repository by importing the formats in *sdl_file*. The **-ia** option must be specified when the schema file contains new definitions. You can also use **-ia** with either **-io** or **-ir** to import into an existing repository and add new definitions while also overlaying or replacing existing ones.

At least one import option (**-ia**, **-io**, **-ir**) must be specified when you run **rpsutl -i**.

-io

Overlay attributes of existing structures, files, templates, formats, and enumerations. Use this option when you want to replace only a subset of attributes or add to those of an existing definition. The **-io** option updates existing attributes and adds new ones, but does not delete any. Either **-io** or **-ir** must be specified when the schema file contains existing definitions; these two options can also be used with **-ia** as described above. The **-io** option is not valid when importing into a new repository and is not valid with **-ir**.

-ir

Replace existing structures, files, templates, formats, and enumerations. Use this option when you want to delete the existing definition and replace it with the definition of the same name in the schema file. Either **-ir** or **-io** must be specified when the schema file contains existing definitions; these two options can also be used with **-ia** as described above. The **-ir** option is not valid when importing into a new repository and is not valid with **-io**.

-n *new_main new_text*

(optional) Specify the names of the temporary repository main and text files to be used when importing into an existing repository. If not specified, the names **rpsmain.new** and

rpstext.new, are used by default. See the Usage section below for more information on temporary files.

-l *[log]*

(optional) Specify that errors generated during schema loading should be logged to a file named *log*. If *log* is not specified, the default filename is **SCHEMA.LOG**, and the log file is placed in the current working directory. If the **-l** option is not specified at all, errors will be directed to standard output.

-r *[sec]*

(optional) Specify that if the repository is being modified by another user, **rpsutl** should retry the operation after waiting *sec* seconds. If *sec* is not specified, the default retry time is 10 seconds. **Rpsutl** will retry indefinitely if **-r** is specified. If **-r** is not specified, **rpsutl** does not retry, and you'll receive a "files in use" error.

-s

(optional) Suppress replacement of the repository. Specify this option if you *do not* want the temporary repository files (see Usage below) to be renamed. The **-s** option is useful when doing a test or trial import. If you are satisfied with the results, you can rename the files manually.

-h

Display usage screen.

Usage

When definitions are imported into an existing repository, they are actually imported into a copy of the repository, named **rpsmain.new** and **rpstext.new** by default. (You can specify different temporary filenames with the **-n** option.) If no errors occur during the import process and the **-s** option is *not* specified, the temporary files are renamed to the original repository filenames. (If **-s** is specified, both temporary and original files are retained.) If any errors occur, the temporary files are deleted.

On OpenVMS, you must set system option #35 in order for *all* versions of the original files to be renamed. See system option #35 in the "System Options" chapter of the *Environment Variables and System Options* manual.

We strongly recommend that when merging a repository, both the original files and the temporary (merged) files be on the same drive on the same system. In some cases, failing to do this results only in slower performance. In other cases (such as when the original files are on Windows and the temporary files on UNIX), the procedure fails with a rename error.

The following example imports the definitions from the files **dict1.sdl** and **dict2.sdl**, adding them into the repository specified by **newmain.ism** and **newtext.ism**. If the repository does not exist, it will be created. If it does exist, the definitions will be imported to a copy of the repository,

rpsmain.new and **rpstext.new**. If no errors occur, the **rpsmain.new** and **rpstext.new** files will replace the **newmain.ism** and **newtext.ism** files. If errors occur, they will be logged to the file **SCHEMA.LOG**, and the **.new** files will be deleted.

```
dbr rpsutl -i dict1.sdl dict2.sdl -ia -d newmain.ism newtext.ism -l
```

The following example imports the definitions from the file **order.sdl** into the default repository, replacing the existing definitions of the same name. Because **-s** is specified, if no errors occur, the copied repository, **rpsmain.new** and **rpstext.new**, will *not* be renamed to the default repository. If errors occur, they will be logged to standard output, and the **.new** files will be deleted.

```
dbr rpsutl -i order.sdl -ir -s
```

The example below imports the definitions in **customer.sdl** into the default repository; new definitions are added and existing ones are replaced. The temp repository files used during the import will be named **main.tmp** and **text.tmp**. If the import is successful, they will be renamed to the default filenames. If errors occur, they will be logged to standard output, and the **.tmp** files will be deleted.

```
dbr rpsutl -i customer.sdl -ia -ir -n main.tmp text.tmp
```

6

Synergy Data Language

Introduction to the Synergy Data Language

The Synergy Data Language describes the contents of a Synergy/DE repository. It was designed as a tool for documenting and analyzing repositories, converting foreign repositories, and modifying repositories.

The Generate Repository Schema and Load Repository Schema utilities generate and interpret the Synergy Data Language. (Optionally, you can use the command line utility, **rpsutl**.) The Generate Repository Schema utility converts specified repository definitions into the Synergy Data Language; this is referred to as a schema file. Conversely, the Load Repository Schema utility converts the contents of a Synergy Data Language file into a new repository or merges the contents into an existing repository. See the following for information on using these utilities:

- "Generating a Repository Schema" on page 5-13
- "Loading a Repository Schema" on page 5-19
- "Generating and Loading Schema from the Command Line" on page 5-28

The Synergy Data Language is also used as the basis for the Print Repository Definitions utility, which prints the structure, file, template, format, enumeration, and detail definitions in your repository to a file.

Because the schema file is simply a text file, it can be used in a number of ways:

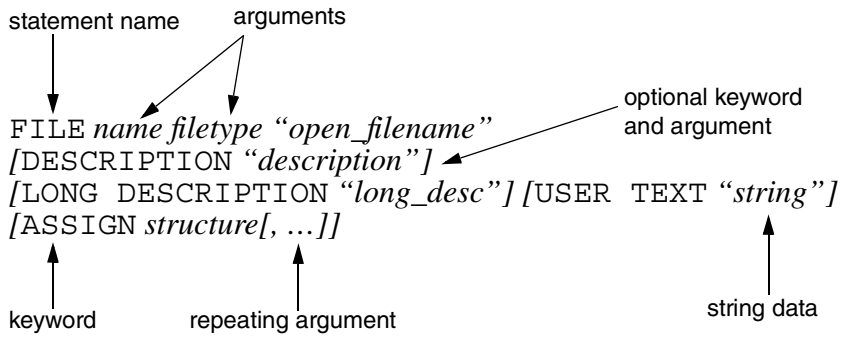
- Aid in converting foreign repositories to Repository format. By writing a program to convert a foreign repository to the Synergy Data Language, you can easily convert your existing non-Repository definitions into Repository format.
- Make mass repository modifications. To do this, use the Generate Repository Schema utility to generate a schema file for your repository. Then modify the file and reload it with the Load Repository Schema utility.
- Move repository files to a different operating system. To do this, use the Generate Repository Schema utility to generate a schema file for your repository. Copy this file to another system, and then run the Load Repository Schema utility to convert the contents of the schema into a new repository. This method of moving repository files works between any two systems, regardless of operating system.

Using Synergy Data Language Statements

We used the following conventions in documenting the syntax of the Synergy Data Language statements:

- Statement names (such as FIELD) and keywords (such as TYPE) are shown in UPPERCASE.
- Arguments that you should replace with actual data are shown in *lowercase italics*.
- Optional items are enclosed in *[*italic square brackets*]*.
- String data is enclosed in "quotation marks".
- Arguments that may be repeated more than once are followed by an ellipsis (…).

Here's an example:



General usage rules

- Names, keywords, and arguments can be specified in either uppercase or lowercase, except for quoted definition names, which must be uppercase. All non-quoted data is converted to uppercase on input.
- If required data is not supplied, or if it is invalid, the entire statement is ignored.
- Statements must be specified in the proper order. See "Recommended statement order" on page 6-4.
- Keywords can be specified in any order, unless otherwise noted. In the statement syntax, we've listed keywords in the order in which they are generated by the Generate Repository Schema utility.
- A keyword that is longer than a single physical word cannot span multiple lines. For example, the following is *invalid*, because the LONG DESCRIPTION keyword is split over two lines:

```
FIELD ccname TYPE date SIZE 8 STORED YYYYMMDD
DESCRIPTION "Opened account" LONG
DESCRIPTION "Opened account (YYYYMMDD)"
```

- For a few keywords, the keyword and associated data must be kept together on the same line. Where this is the case, it is documented with the keyword.
- Keyword data that contains colons (:) (for example, the arguments for the OVERLAY and DIMENSION keywords) cannot contain embedded spaces.
- Negative values for numeric keyword arguments are not permitted, except where noted. When a negative value is used, it must be immediately preceded by a minus sign (for example, -3 or -10).
- String data must be enclosed in a set of matching double or single quotation marks (“ ” or ‘ ’) and cannot span multiple lines.
- String data can contain a quotation character if that character is different from the character that encloses the data. For example, both of the following strings are valid:

```
"Type 'Return' to continue"
'Type "Return" to continue'
```

- Data that exceeds the maximum size allowed is truncated.
- Comments are indicated by placing a semicolon as the first character (in the first column) on the line. Comments are not permitted within a Synergy Data Language statement.

Recommended statement order

Statements must be specified in the proper order. If one definition name is referenced by another definition, the first definition must precede the referencing definition. For example, if you are defining a field that references a template, you must define the template before you reference its name in the field definition.

To avoid problems, we recommend that you define elements in the following order:

1. Define all global formats.

   Global formats must be defined before any templates (if any templates reference them) and before any structures.

2. Define all enumerations.

   Enumerations must be defined before any fields or templates that reference them.

3. Define all templates in reference order.

   All templates must be grouped together, and they must be specified in reference order. That is, all parent templates must be defined before their child templates. The Generate Repository Schema utility generates templates in reference order (when *all* templates are being output).

4. Define each structure in reference order, along with all of its formats, fields, keys, relations, and aliases, in that order.

Structures must be defined before any files that reference them. A structure referenced within an implicit group specification or as a Struct data type must be defined before the structure that references it.

Each structure must be followed by its formats, fields, keys, aliases, relations, and tag. The formats must occur before any fields. The fields must be grouped together and must occur before any keys or aliases that reference them. Relations can occur anywhere within the structure, except between fields. (Relations can actually be defined anywhere within the schema file, since they aren't validated until the end of the schema loading process.) A structure's tag must be defined before the group of fields or after the entire group.

5. Define all files.

General processing rules

A Synergy Data Language file (schema file) can be used to create a new repository, or it can be used to update an existing repository.

- When a schema file is used to create a new repository, you simply specify the name of the new repository, and it is created for you by the Load Repository Schema utility. If *any* errors occur, the new repository is not created; you must correct the schema file and reload it.
- When a schema file is used to update (merge) an existing repository, the Load Repository Schema utility makes a copy of the specified repository, and then updates the copy. If *any* errors occur, the copied repository is deleted; you must correct the schema file and reload it. If no errors occur, the copied repository either replaces the original or not, depending on the options you specified.

When a repository has been successfully loaded or updated, you should run both the Verify Repository utility (see "Verifying Your Repository" on page 5-10) and the Validate Repository utility (see "Validating Your Repository" on page 5-12) on the new (or copied) repository.

The Load Repository Schema utility is able to process the specification of both new and existing repository definitions. The way each is handled depends on the options you set when running the utility.

- If you are loading a schema into a new repository, all definitions specified in the schema file are added. If a duplicate definition exists in the schema file, an error is logged in the log file.
- Adding new structures also adds any fields, keys, relations, formats, and tags specified for the structure.
- If you are merging a schema into an existing repository, all new definitions are added. You can choose to replace or overlay existing definitions.
  - **Replace** deletes the current definition and stores the new one from the schema file. Replacing an existing structure essentially deletes all fields, keys, etc., for the current structure, and adds back only those specified in the schema file.

- **Overlay** updates existing fields, etc., with those in the schema file and adds new fields, etc. It does not delete any fields, etc. Overlay is typically used to add to the attributes of existing definitions.

The Load Repository Schema utility attempts to load *all* the definitions in the schema file into the repository. If your file contains definitions you do not wish to load, you must remove them from the file. For example, if your schema file contains both new and existing definitions, and you want only to update the existing ones, you must remove the new definitions from the file before loading the schema.

The table below summarizes the rules that your schema must follow to use the Load Repository Schema utility to update (merge) an existing repository. Carefully review this table before merging schemas.

Rules for Schema Files		
Option	**Definition type**	**Rules**
Add	<All>	All required keywords and data must be specified. Non-specified, non-required attributes are set to default values.
	Global formats	Must be defined before any templates or structures.
	Enumerations	Must be defined before any templates or structures.
	Templates	Must be defined before any structures.
	Structures	Must be defined before any files.
	Formats	Must be defined before any fields.
	Fields	Must be defined before any keys.
	Keys	Must be defined before any relations.
	Tags	Must be defined before any fields or after all fields for the current structure.
	Files	No additional rules.

Rules for Schema Files (Continued)		
Option	**Definition type**	**Rules**
Replace	<All>	(See “Add” above.)
	Global formats	No additional rules.
	Enumerations	No additional rules.
	Templates	No additional rules.
	Structures	No additional rules. **Notes:** New fields, keys, etc., are added. Existing fields, keys, etc., are replaced. Non-specified existing fields, keys, etc., are deleted. New alias structures and alias fields are added. All alias fields in existing alias structures are replaced by specified alias fields. Non-specified alias structures (and their fields) are unaffected.
	Formats	No additional rules.
	Fields	No additional rules.
	Keys	No additional rules.
	Relations	No additional rules.
	Tags	No additional rules.
	Files	To disassociate structures from a file, specify only the ones you want to keep. Specifying no assigned structures clears all current assigned structures.

Rules for Schema Files (Continued)		
Option	**Definition type**	**Rules**
Overlay	<All>	The definition name must be specified, followed by the attributes to be overlaid or added to the definition. (Exceptions to this rule are listed below.) All other attributes remain unchanged.
	Global formats	All keywords and data except JUSTIFY must be respecified.
	Enumerations	All members must be respecified when you overlay one or more members.
	Templates	No additional rules. **Notes:** When a parent is added to an existing template, both the existing override flags and the template attributes that are explicitly specified in the schema file are retained as parent overrides.
	Structures	No additional rules. **Notes:** New fields, keys, etc., are added. Existing fields, keys, etc., are updated. Non-specified existing fields, keys, etc., remain unchanged. New alias structures and alias fields are added. All alias fields in existing alias structures are replaced by specified alias fields.
	Formats	All keywords and data except JUSTIFY must be respecified.
	Fields	No additional rules. **Notes:** When a template is added to an existing field, only the existing override flags and the field attributes that are explicitly specified in the schema file are retained as template overrides.
	Keys	All key segments must be respecified when you overlay one or more segments. (To clear all segments, use **Replace**.) All compression options must be respecified when you overlay one or more options.
	Relations	All attributes must be respecified.
	Tags	All attributes must be respecified.
	Files	All assigned structures must be respecified (along with any ODBC table names) when changing one or more.

ALIAS – Describe an alias for a structure or field

`ALIAS` *alias type name*

Arguments

alias

The name of a new or existing alias. This name can have a maximum of 30 characters.

type

One of the following alias types:

STRUCTURE
FIELD

name

The name of the aliased structure or field. This name can have a maximum of 30 characters.

Discussion

The ALIAS statement describes an alternate name—an alias—for a structure or field.

Aliases enable you to associate a different name (or a name link) with a structure or field. For example, aliases are useful when you're converting an application to use the Repository. If your application uses short, cryptic identifier names, but you would like to use longer names in the repository, you can use aliases to simplify updating your Synergy code. Aliases can also be useful when you want to define structfields, but still need that repository structure to be included as a record in your Synergy code. You can create an alias and reference that in your structfield definition.

When you use .INCLUDE to reference the repository from your Synergy code, you can use either the real or the alias name. The compiler first searches for a structure or field that has the name specified in the .INCLUDE statement. If it can't find one, it searches for an alias with that name. Therefore, *all* structure names, whether real or alias, must be unique. Likewise, all field names—real or alias—must be unique within a given structure.

Within your Synergy Data Language file, an alias must be defined within the structure that it references (referred to as the *aliased structure*). You can link more than one alias to the same structure. Within an aliased structure, you can link more than one alias to the same field.

An aliased field is associated with the most recently defined aliased structure. If you haven't defined an aliased structure yet, the aliased field is ignored. You cannot alias fields that are defined as members of a group.

Aliases for fields are supported only through the Synergy Data Language. Aliases for structures can also be defined through the Repository user interface; see "Defining Aliases" on page 2-9.

Adding new definitions

The order in which you define aliased fields determines the order in which they exist within the aliased structure. The maximum number of alias fields you can define within one aliased structure is 650.

Replacing existing definitions

All required keywords and data must be specified. The existing aliased structure is cleared and all of its aliased fields are deleted. The aliased structure is set to the specified attributes, and the specified aliased fields are added. (Note that when you replace a structure, any aliased structures that are not explicitly specified in the schema are unaffected.)

Overlaying existing definitions

All required keywords and data must be respecified. All of the aliased fields are deleted. The current aliased structure attributes are overwritten by the attributes specified, and the specified aliased fields are added.

Deleting aliases

To delete aliases that reference a structure or field that no longer exists, run the Verify Repository utility. See "Verifying Your Repository" on page 5-10.

Examples

```
ALIAS cusmas STRUCTURE customer_master
   ALIAS cusnam FIELD customer_name
   ALIAS cusid FIELD customer_id
   ALIAS cusadd FIELD customer_address
```

ENDGROUP – End a group definition

```
ENDGROUP
```

Discussion

The ENDGROUP statement ends the current group definition. It must follow the last FIELD member or GROUP definition for the current group.

Examples

See GROUP on page 6-49 for an example.

ENUMERATION – Describe an enumeration definition

```
ENUMERATION name [DESCRIPTION "description"] [LONG DESCRIPTION "long_desc"]
MEMBERS member [value] [, member [value]] [, ...]
```

Arguments

name

The name of a new or existing enumeration. This name can have a maximum of 30 characters.

DESCRIPTION

(optional) Indicates that an enumeration definition description follows.

description

A description of the enumeration. It can have a maximum of 40 characters and must be enclosed in double or single quotation marks (“ ” or ‘ ’).

LONG DESCRIPTION

(optional) Indicates that a long description for the enumeration definition follows.

long_desc

A more detailed description of the enumeration and its use. It can contain 30 lines of up to 60 characters each. Each line must be enclosed in double or single quotation marks (“ ” or ‘ ’).

MEMBERS

Indicates that an enumeration member definition follows. An enumeration must have at least one member.

member

The name of the enumeration member. This name can have a maximum of 30 characters.

value

(optional) Specifies the member value, which must be a number. If *value* is specified, it must be on the same line as *member*.

Discussion

The ENUMERATION statement is used to describe an enumeration definition. An enumeration is a set of related values, which has a name and one or more members associated with it.

When specifying members and values, the pair cannot be split over two lines. See the examples in “General usage rules” on page 6-3.

Adding new definitions

The maximum number of enumerations that can be defined is 9,999. The maximum number of members that can be defined for an enumeration is 100. An enumeration must have at least one member. The order in which you specify members determines their order in the enumeration.

Replacing existing definitions

All required keywords and data must be specified. The existing enumeration and all its members are cleared and replaced by the specified enumeration and members.

Overlaying existing definitions

All members must be respecified when you overlay one or more members.

Examples

```
ENUMERATION colors DESCRIPTION "Colors of the rainbow"
MEMBERS red 1, orange 2, yellow 3, green 4, blue 5, indigo 6, violet 7
```

FIELD – Describe a field definition

FIELD *name* [TEMPLATE *template*] TYPE *type* SIZE *size* [STORED *store_format*]
[ENUM *name* | STRUCT *name*] [NODATE] [NOTIME]
[USER TYPE *"user_type"*] [NOUSER TYPE] [PRECISION *dec_places*]
[DIMENSION *#elements*[:*#elements* ...]] [LANGUAGE VIEW] [LANGUAGE NOVIEW]
[SCRIPT VIEW] [SCRIPT NOVIEW] [REPORT VIEW] [REPORT NOVIEW]
[WEB VIEW] [WEB NOVIEW] [COERCED TYPE *type*] [NOCOERCED TYPE]
[OVERLAY *field*[:*offset*]] [NONAMELINK] [DESCRIPTION *"description"*] [NODESC]
[LONG DESCRIPTION *"long_desc"*] [NOLONGDESC]
[POSITION [*pos_type*] *row column*] [NOPOSITION]
[FPOSITION [*fpos_type*] *frow fcolumn*] [NOFPOSITION] [PROMPT *"prompt"*] [NOPROMPT]
[HELP *"help"*] [NOHELP] [INFO LINE *"info_line"*] [NOINFO]
[USER TEXT *"user_text"*] [NOUSER TEXT] [FORMAT *format*] [NOFORMAT]
[REPORT HEADING *"heading"*] [NOHEADING] [ALTERNATE NAME *alt_name*]
[NOALTERNATE NAME] [REPORT JUST *just*] [INPUT JUST *ijust*] [BLANKIFZERO]
[NOBLANKIFZERO] [PAINT *"paint_char"*] [NOPAINT] [RADIO | CHECKBOX] [NORADIO]
[NOCHECKBOX] [FONT *font*] [NOFONT] [PROMPTFONT *prompt_font*] [NOPROMPTFONT]
[READONLY] [NOREADONLY] [DISABLED] [NODISABLED|ENABLED]
[COLOR *palette#*] [NOCOLOR] [HIGHLIGHT] [NOHIGHLIGHT] [REVERSE] [NOREVERSE]
[BLINK] [NOBLINK] [UNDERLINE] [NOUNDERLINE] [NOATTRIBUTES]
[DISPLAY LENGTH *length*] [NODISPLAY LENGTH]
[VIEW LENGTH *length*] [NOVIEW LENGTH] [UPPERCASE] [NOUPPERCASE]
[NODECIMAL] [DECIMAL_REQUIRED] [NOTERM] [TERM]
[RETAIN POSITION] [NORETAIN POSITION]
[DEFAULT *default* | COPY | INCREMENT | DECREMENT] [NODEFAULT]
[AUTOMATIC] [NOAUTOMATIC] [NOECHO] [NOECHOCHR *"display_char"*] [ECHO]
[DATE TODAY] [DATE NOTODAY] [DATE SHORT] [DATE NOSHORT]
[TIME NOW] [TIME NONOW] [TIME AMPM] [TIME NOAMPM]
[WAIT *"time"* | WAIT IMMEDIATE | WAIT GLOBAL | WAIT FOREVER] [NOWAIT]
[INPUT LENGTH *length*] [NOINPUT LENGTH] [BREAK [*break_type*]] [NOBREAK]
[REQUIRED] [NOREQUIRED] [NEGATIVE [ONLY | ORZERO]] [NONEGATIVE]
[NULL ALLOWED | NULL DEFAULT | NONULL] [ALLOW *entry*[, ...]] [NOALLOW]
[MATCH CASE] [MATCH NOCASE] [MATCH EXACT] [MATCH NOEXACT]
[SELECTION LIST *sl_row sl_column sl_height* ENTRIES *sl_entry*[, ...]]
[SELECTION WINDOW *sw_row sw_column sw_name*] [NOSELECT]
[ENUMERATED *length base step*] [NOENUMERATED] [RANGE *min max*] [NORANGE]
[ARRIVE METHOD *arrive_meth*] [NOARRIVE METHOD]
[LEAVE METHOD *leave_meth*] [NOLEAVE METHOD]
[DRILL METHOD *drill_meth*] [NODRILL METHOD]
[HYPERLINK METHOD *hyperlink_meth*] [NOHYPERLINK METHOD]
[CHANGE METHOD *change_meth*] [NOCHANGE METHOD]
[DISPLAY METHOD *display_meth*] [NODISPLAY METHOD]
[EDITFMT METHOD *editfmt_meth*] [NOEDITFMT METHOD]

Arguments

name

The name of a new or existing field. This name can have a maximum of 30 characters.

TEMPLATE

(optional) Assigns a template to the field.

template

The name of a template to assign to this field. All field attributes, including type and size, are obtained from the specified template. The maximum size of the template name is 30 characters. The specified template must already be defined. If it is not defined, the field will be logged in error. If a template is assigned to a field and none of the template's attributes are overridden in the field, no additional keywords are required. The Synergy Data Language assumes that any keywords specified in addition to the template name are overrides to the template.

TYPE

Indicates that the data type for the field follows. This keyword and the data type are optional if you're overlaying an existing field or if a template is assigned to this field.

type

One of the following data types for this field:

ALPHA
DECIMAL
INTEGER
DATE
TIME
USER
BOOLEAN
ENUM
STRUCT
AUTOSEQ
AUTOTIME

If you specify DATE, a default storage format of YYMMDD is assigned. If you specify TIME, a default storage format of HHMM is assigned. If you specify USER, a default subtype of ALPHA is assigned. You can override these defaults with the STORED keyword.

SIZE

Indicates that the field size follows. This keyword and the size are optional if you're overlaying an existing field or if a template is assigned to this field.

size

The maximum number of characters the field can contain. See the table below for valid sizes for each data type.

Data type	Valid sizes
alpha	1–99,999
decimal	28
integer	1, 2, 4, 8
date time	The size of the specified format. E.g., 6 for YYMMDD; 4 for HHMM.
user	1–99,999
boolean	4
enum	4
struct	The size of the referenced structure
autoseq	8
autotime	8

STORED

(optional) Indicates that the storage format for a date or time field, or the subtype for a user field follows. If TYPE is specified, it must precede STORED.

store_format

Specifies the storage format if the data type is date or time. Specifies the subtype if the data type is user. It can also be used to specify that an alpha field contains binary data.

If this keyword is not specified for a date field, the default format is YYMMDD. If not specified for a time field, the default format is HHMM. If not specified for a user field, the default subtype is ALPHA. If the data type is not date, time, user, or alpha, this value is ignored.

If the type is **date**, the format must be one of the following:

YYMMDD
YYYYMMDD
YYJJJ
YYYYJJJ
YYPP
YYYYPP

If the type is **time**, the format must be one of the following:

HHMM
HHMMSS

If the type is **user**, the format must be one of the following:

ALPHA
NUMERIC
DATE
BINARY

To specify that an alpha field contains binary data, the format must be

BINARY

In xfNetLink Java (when **genjava** is run with the **-c 1.5** option) and xfNetLink .NET, an alpha type with a binary format is converted to a byte array on the client and can be used, for example, to store an RFA.

In xfODBC, an alpha type with a binary format is described as a binary field (SQL_BINARY). This is also true of a user type with binary format, but in this case you can use the routines for user-defined data types in xfODBC to manipulate the data read from the ISAM file and return it as a binary field to the ODBC-enabled application.

ENUM

(optional) Indicates that the enumeration name follows. Required when TYPE is ENUM.

name

The name of the enumeration. The specified enumeration must already be defined. If it is not, the field will be logged in error.

STRUCT

(optional) Indicates that the structure name follows. Required when TYPE is STRUCT.

name

The name of the structure or its alias. The specified structure must already be defined. If it is not, the field will be logged in error.

NODATE

(optional) The default state if a date data type has not been specified. If present, NODATE resets the date field to a normal decimal field. Specifying NODATE also clears any date storage format and any DATE TODAY or DATE SHORT keywords.

NOTIME

(optional) The default state if a time data type has not been specified. If present, NOTIME resets the time field to a normal decimal field. Specifying NOTIME also clears any time storage format and any TIME NOW or TIME AMPM keywords.

USER TYPE

(optional) Indicates that a user data type string follows.

user_type

A string that more uniquely defines a user data type. It can have a maximum of 30 characters and must be enclosed in double or single quotation marks (“ ” or ‘ ’). USER TYPE is ignored for any data type other than the user data type.

NOUSER TYPE

(optional) Cancels any user data type string that has been specified.

PRECISION

(optional) Indicates that the number of decimal places in implied-decimal fields follows. PRECISION is only valid when the field’s data type is decimal or user.

dec_places

The number of characters to the right of the decimal point in an implied-decimal field. If this attribute is present, it must be between 1 and 28, inclusive. It also must be less than or equal to the size of the field.

DIMENSION

(optional) Indicates that this field is arrayed.

#elements

The number of elements in each dimension if this field defines an array. The maximum number of dimensions is four. Additional dimensions are ignored. The maximum number of elements per dimension is 999. If more than one dimension is specified, the dimensions must be separated by a colon and can contain no embedded spaces.

LANGUAGE VIEW

(optional) Indicates that the field will be available to the Synergy compiler when the structure is .INCLUDEd into a source file. LANGUAGE VIEW is the default.

LANGUAGE NOVIEW

(optional) Indicates that the field will *not* be available to the Synergy compiler when the structure is .INCLUDEd into a source file.

SCRIPT VIEW

(optional) Indicates that the field will be available to UI Toolkit when defining an input window. SCRIPT VIEW is the default.

SCRIPT NOVIEW

(optional) Indicates that the field will *not* be available to the UI Toolkit when defining an input window.

REPORT VIEW

(optional) Indicates that the field will be available as a selectable field in xfODBC and ReportWriter. REPORT VIEW is the default.

REPORT NOVIEW

(optional) Indicates that the field will *not* be available as a selectable field in xfODBC and ReportWriter.

WEB VIEW

(optional) Indicates that the field will be included by xfNetLink when creating Synergy components (JAR file or assembly). WEB VIEW is the default.

WEB NOVIEW

(optional) Indicates that the field will not be included by xfNetLink when creating Synergy components.

COERCED TYPE

(optional) Indicates that the coerced type for use by xfNetLink follows.

type

The data type to which this field should be coerced on the xfNetLink Java or xfNetLink .NET client. Valid values depend on the value of TYPE.

If type is **decimal** (without precision) the coerced type may be one of the following:

DEFAULT
BYTE
SHORT
INT
LONG
SBYTE
USHORT
UINT
ULONG
BOOLEAN
DECIMAL
NULLABLE DECIMAL

If the type is **decimal** (with precision), the coerced type may be one of the following:

DEFAULT
DOUBLE
FLOAT
NULLABLE DECIMAL
DECIMAL

If type is **integer**, the coerced type may be one of the following:

DEFAULT
BYTE
SHORT
INT
LONG
SBYTE
USHORT
UINT
ULONG
BOOLEAN

If the type is **date** (with a format of YYMMDD, YYYYMMDD, YYJJJ, or YYYYJJJ), **time**, or **user** (with a subtype of DATE and user data string of ^CLASS^=YYYYMMDDHHMISS or ^CLASS^=YYYYMMDDHHMISSUUUUUU), the coerced type may be one of the following:

DATETIME
NULLABLE_DATETIME

If this keyword is not specified for a decimal or integer field, the default is DEFAULT. If not specified for a date field (with one of the formats mentioned above) or for a time field or for a user field (with the restrictions mentioned above), the default is DATETIME. See "Appendix B: Data Type Mapping" in the *xfNetLink & xfServerPlus User's Guide* for more information on data type mapping and coercion in xfNetLink.

NOCOERCED TYPE

(optional) The default state if a coerced type has not been specified. If present, NOCOERCED TYPE cancels any coerced type for the field.

OVERLAY

(optional) Defines this field as an overlay to another field. For example, the year, month, and day might be overlays for a date field.

field

The name of the field to overlay. The overlaid field must be a field that precedes the current field and can be a maximum of 30 characters.

offset

(optional) Represents an offset to add to the starting position of the overlaid field. The default offset is zero. No embedded spaces are allowed.

NONAMELINK

(optional) Indicates that the field name itself is to be used for name-linking purposes. See Do not name link on page 3-11 for more information about name links.

DESCRIPTION

(optional) Indicates that a field definition description follows.

description

A description of the field definition. It can have a maximum of 40 characters and must be enclosed in double or single quotation marks (“ ” or ‘ ’). This description appears as the comment for the field when a definition file is generated by the Generate Definition File utility, and it can be used as a way to identify fields in ReportWriter and Repository. In addition, if the structure that this field belongs to is included in an xfNetLink Java JAR file or xfNetLink .NET assembly, this description is included in the generated source code as a comment for the property or field.

NODESC

(optional) Cancels any field description that has been specified.

LONG DESCRIPTION

(optional) Indicates that a long description for the field definition follows.

long_desc

A more detailed description of the field definition and its use. It can contain 30 lines of up to 60 characters each. Each line must be enclosed in double or single quotation marks (“ ” or ‘ ’).

NOLONGDESC

(optional) Cancels any field long description that has been specified.

POSITION

(optional) Provides position information for this field.

pos_type

Specifies the position type associated with this field. If *pos_type* is specified, it must contain one of the following values:

ABSOLUTE (default)
RELATIVE

row

Specifies the row position to be associated with this input window field. If no prompt is defined, the input field begins at the specified position. If a prompt is defined, the prompt begins at the specified position. If the RELATIVE keyword precedes *row*, this value specifies the number of rows that the current position will change; otherwise, it designates absolute coordinates relative to the input window. Relative values can be negative.

column

Specifies the column position to be associated with this input window field. If no prompt is defined, the input field begins at the specified position. If a prompt is defined, the prompt begins at the specified position. If the RELATIVE keyword precedes the *row* argument, the

column value specifies the number of columns that the current position will change; otherwise, it designates absolute coordinates relative to the input window. Relative values can be negative.

NOPOSITION

(optional) Resets the position of the prompt to the default next character available, rather than the position specified by the POSITION keyword. NOPOSITION is the default.

FPOSITION

(optional) Provides position information for this field, independent of its prompt.

fpos_type

Specifies the position type associated with this field. If present, *fpos_type* must contain one of the following values:

ABSOLUTE (default)
RELATIVE

frow

Specifies the row position to be associated with this input window field, independent of its prompt. If the RELATIVE keyword precedes the *frow* argument, this value specifies the number of rows that the current position will change; otherwise, it designates absolute coordinates relative to the input window. Relative values can be negative.

fcolumn

Specifies the column position to be associated with this input window field, independent of its prompt. If the RELATIVE keyword precedes the *frow* argument, the *fcolumn* value specifies the number of columns that the current position will change; otherwise, it designates absolute coordinates relative to the input window. Relative values can be negative.

NOFPOSITION

(optional) Resets the position of the input window field to the default next character after the prompt, rather than the position specified by the FPOSITION keyword. NOFPOSITION is the default.

PROMPT

(optional) Indicates that the user prompt for this field follows.

prompt

Either a fixed or a variable prompt. A fixed prompt is a string that is displayed in the input window to prompt the user for input. The prompt string must be enclosed in double or single quotation marks (“ ” or ‘ ’), and it must include any spacing that you want to display between the prompt and the input field. The maximum length of a prompt string is 80 characters. Fixed prompts may be modified at runtime with the UI Toolkit I_PROMPT subroutine. To define a variable prompt, enter a numeric value. This value will be used by I_PROMPT to set the variable prompt length. The quotation marks around variable prompts are optional. See also I_PROMPT in the “Input Routines” chapter of the *UI Toolkit Reference Manual*.

NOPROMPT

(optional) The default state if a user prompt has not been specified. If present, NOPROMPT cancels any prompt string.

HELP

(optional) Indicates that a help identifier for this field follows.

help

Specifies a help identifier. This help identifier is passed as an argument to the UI Toolkit EHELP_METHOD subroutine. The help identifier string must be enclosed in double or single quotation marks (“ ” or ‘ ’). The maximum length of a help identifier is 80 characters. See EHELP_METHOD in the “Environment Routines” chapter of the *UI Toolkit Reference Manual* for more information.

NOHELP

(optional) The default state if a help identifier has not been specified. If present, NOHELP cancels any help identifier.

INFO LINE

(optional) Indicates that the text to appear on the information line follows.

info_line

Specifies a text string that is displayed on the information line when input is being processed for this field. The *info_line* string must be enclosed in double or single quotation marks (“ ” or ‘ ’). The maximum length of an *info_line* string is 80 characters.

NOINFO

(optional) The default state if an *info_line* string has not been specified. If present, NOINFO cancels any *info_line* string.

USER TEXT

(optional) Indicates that a user-defined text string follows.

user_text

A user-defined text string associated with this input field. The *user_text* string must be enclosed in double or single quotation marks (“ ” or ‘ ’). The maximum length of the *user_text* string is 80 characters.

NOUSER TEXT

(optional) The default state if a user text string has not been specified. If present, NOUSER TEXT cancels any user text string.

FORMAT

(optional) Indicates that the display format for this field follows.

format

The name of a global or structure-specific format to use with this field if it is used in a UI Toolkit input window or is selected in a ReportWriter report. The maximum size of the format name is 30 characters. The specified format must already be defined. If it is not defined, the format name is ignored.

If the field is a date or time field, we recommend you use one of the formats listed in "Appendix B: Date and Time Formats". (ReportWriter treats these formats differently and actually re-orders the data being displayed if necessary. For example, a date stored as YYMMDD can be displayed as MM/DD/YY.)

NOFORMAT

(optional) The default state if a format has not been specified. If present, NOFORMAT cancels any format for the field.

REPORT HEADING

(optional) Indicates that the report column heading for this field follows.

heading

The default column heading this field will have when it is used in a ReportWriter report. This value is also used by xfNetLink .NET for the column caption if the structure that contains this field is included in a DataTable class. This string can have a maximum of 40 characters and must be enclosed in double or single quotation marks (" " or ' ').

NOHEADING

(optional) The default state if a report heading has not been specified. If present, NOHEADING cancels any report heading for the field.

ALTERNATE NAME

(optional) Indicates that the alternate name for this field follows.

alt_name

The alternate name for the field within the xfODBC system catalog, the xfNetLink Java JAR file, or the xfNetLink .NET assembly. The maximum size of the name is 30 characters.

NOALTERNATE NAME

(optional) The default state if an alternate name has not been specified. If specified, NOALTERNATE NAME cancels any alternate name for the field.

REPORT JUST

(optional) Indicates how the field's data will be justified in a ReportWriter report.

just

The justification of the data within the column when this field is used in a ReportWriter report. Valid values are

LEFT
RIGHT
CENTER

The default is LEFT for alpha, user, date, and time fields and RIGHT for decimal, implied-decimal, and integer fields. CENTER is allowed only for alpha and user fields.

INPUT JUST

(optional) Indicates that a text justification argument follows.

ijust

Designates how the text is justified within the input field when this field is used in a Toolkit input window. Valid values are

LEFT
RIGHT
CENTER

The default is LEFT for alpha, date, time, and user type fields, and RIGHT for decimal, implied-decimal, and integer fields. CENTER alignment is not allowed for numeric fields; neither RIGHT nor CENTER alignment is allowed for text fields (multi-dimensional alpha fields).

BLANKIFZERO

(optional) Indicates that a decimal, implied-decimal, or integer field will be left blank if the user enters a value of zero.

NOBLANKIFZERO

(optional) The default state if BLANKIFZERO has not been specified. If present, NOBLANKIFZERO indicates that a zero will be displayed if the user enters a value of zero.

PAINT

(optional) Indicates that the field is filled with the specified paint character until the user enters input.

paint_char

Used to "paint" the empty field to indicate where the user is supposed to type input. You must enclose the paint character in double or single quotation marks (" " or ' ').

NOPAINT

(optional) The default state if no paint character has been specified. If present, NOPAINT cancels any paint character for the field and resets the field to use the default paint character.

RADIO

(optional) Indicates that this field is to be displayed as a set of radio buttons on Windows. RADIO is only valid when SELECTION LIST or SELECTION WINDOW has been specified.

CHECKBOX

(optional) Indicates that this field is to be displayed as a check box on Windows. On non-Windows environments, it will be displayed as a one-character field ("X" if non-zero, or space if zero). You can only specify the CHECKBOX keyword when the field's data type is decimal, implied-decimal, or integer. Since a check box is implicitly an enumerated field, you cannot specify the CHECKBOX keyword in conjunction with ENUMERATED.

NORADIO

(optional) The default state if RADIO has not been specified. If present, NORADIO indicates that the field will not be displayed as a set of radio buttons.

NOCHECKBOX

(optional) The default state if CHECKBOX has not been specified. If present, NOCHECKBOX indicates that the field will not be displayed as a check box.

FONT

(optional) Indicates that the font for this field follows.

font

The name of a font to use when displaying the contents of the input field on Windows. This name is assumed to be defined in the font palette (specified in the [FONTS] section of the **synergy.ini** or **synuser.ini** file).

NOFONT

(optional) The default state if a font has not been specified. If specified, NOFONT cancels any font for the field.

PROMPTFONT

(optional) Indicates that the font for the field's prompt follows.

prompt_font

The name of a font to use for displaying the input field's prompt on Windows. This name must also be defined in the font palette (specified in the [FONTS] section of the **synergy.ini** or **synuser.ini** file).

NOPROMPTFONT

(optional) The default state if a prompt font has not been specified. If specified, NOPROMPTFONT cancels any font for the field's prompt.

READONLY

(optional) Indicates that this field is read-only. The READONLY keyword cannot be specified in conjunction with RADIO, CHECKBOX, SELECTION LIST, or SELECTION WINDOW. For data types AUTOSEQ and AUTOTIME, READONLY is required. This flag is honored by xfNetLink Java if this field is in a structure that is included in a JAR file and **genjava** is run with the **-ro** option. This flag is honored by xfNetLink .NET if this field is in a structure that is included in an assembly, and structure members are generated as properties, rather than fields.

NOREADONLY

(optional) The default state if READONLY has not been specified. If present, NOREADONLY allows modification of the field.

DISABLED

(optional) Indicates that this field is disabled.

NODISABLED | ENABLED

(optional) The default state if DISABLED has not been specified. If present, NODISABLED (or ENABLED) allows modification and focus of the field.

COLOR

(optional) Indicates that a color palette number follows.

palette#

Specifies the color palette for the field. If this attribute is present, it must be between 1 and 16, inclusive.

NOCOLOR

(optional) The default state if a color palette number has not been specified. If present, NOCOLOR cancels any color for the field. Specifying NOCOLOR is the same as COLOR 0.

Specifying any of the following eight keywords indicates that field-level attributes are in effect for this field, and the input window attributes are overridden. For example, if the input window defines blink and underline, specifying HIGHLIGHT for the field will turn on highlighting and turn off blink and underline.

HIGHLIGHT

(optional) Indicates that the field is highlighted.

NOHIGHLIGHT

(optional) The default state if HIGHLIGHT has not been specified. If present, NOHIGHLIGHT indicates that the field will not be highlighted.

REVERSE

(optional) Indicates that the field is in reverse video.

NOREVERSE

(optional) The default state if REVERSE has not been specified. If present, NOREVERSE indicates that the field will not be in reverse video.

BLINK

(optional) Indicates that the field is blinking. (On Windows, the field is displayed in italic typeface.)

NOBLINK

(optional) The default state if BLINK has not been specified. If present, NOBLINK indicates that the field will not blink.

UNDERLINE

(optional) Indicates that the field is underlined.

NOUNDERLINE

(optional) The default state if UNDERLINE has not been specified. If present, NOUNDERLINE indicates that the field will not be underlined.

NOATTRIBUTES

(optional) Indicates that all attributes specified for the field should be cleared, and the input window attributes used instead. This is the default state if none of the above eight keywords has been specified.

DISPLAY LENGTH

(optional) Indicates that the display length follows.

length

The maximum number of characters that you want to be displayed in the field. Valid values are 0 through 65,535. Display length cannot be used in conjunction with RADIO, CHECKBOX, SELECTION LIST, SELECTION WINDOW, or DIMENSION.

NODISPLAY LENGTH

(optional) The default state if DISPLAY LENGTH is not specified. If present, NODISPLAY LENGTH indicates that Toolkit's default computation for display length will be used.

VIEW LENGTH

(optional) Indicates that the view length follows.

length

The number of characters that you want to use to determine the width of the field on the screen (i.e., the width of the area on the screen that will display data for the field). Valid values are 0 through 9,999. View length cannot be used in conjunction with RADIO or CHECKBOX.

On Windows, this value is multiplied by the width of the sizing character for the current font to determine the field width. If view length is less than display length, the field will be scrollable

up to the display length. On UNIX and OpenVMS, the width of the field is set to the number of characters specified. If view length is less than display length, the field will be truncated to fit in the view length.

NOVIEW LENGTH

(optional) The default state if VIEW LENGTH is not specified. If present, NOVIEW LENGTH indicates that Toolkit's default computation for determining the width of the field will be used.

UPPERCASE

(optional) Converts lowercase input characters to uppercase. UPPERCASE is valid only when the field's data type is alpha or user.

NOUPPERCASE

(optional) The default state if UPPERCASE has not been specified. If present, NOUPPERCASE allows lowercase input for the field.

NODECIMAL

(optional) Indicates that the user does not need to type a decimal point when typing input in a numeric field. NODECIMAL is valid only when the field's data type is decimal, implied-decimal, or integer.

DECIMAL_REQUIRED

(optional) The default state if NODECIMAL has not been specified. If present, DECIMAL_REQUIRED aligns the numeric value based on an entered decimal point only.

NOTERM

(optional) Terminates the field automatically when the field is filled.

TERM

(optional) The default state if NOTERM has not been specified. If present, TERM indicates that the user must press ENTER to terminate field input.

RETAIN POSITION

(optional) Indicates that the position within a text field (multi-dimensional alpha field) will be retained on subsequent re-entry to the field, until the field is reinitialized or redisplayed. You can specify the RETAIN POSITION keyword only for text fields.

NORETAIN POSITION

(optional) The default state if RETAIN POSITION has not been specified. If present, NORETAIN POSITION cancels position retention in the text field.

DEFAULT

(optional) Indicates that the specified default value should be displayed in the field. DEFAULT overrides any COPY, INCREMENT, or DECREMENT value that was previously specified.

default

The default value to be displayed in the input field. This value must be appropriate for the field's data type. For date and time fields, the default value must be specified in storage form (STORED), rather than input or display form. If the default value contains spaces or is case sensitive, you must enclose the entire value in double or single quotation marks (" " or ' ').

COPY

(optional) Copies the value from the data area that corresponds to this field if the field is empty. COPY overrides any DEFAULT, INCREMENT, or DECREMENT value that was previously specified.

INCREMENT

(optional) Designates that if the user does not enter a value the first time a numeric field is processed, the last value in the field plus one will be used. INCREMENT is only valid when the field's data type is decimal, implied-decimal, or integer. INCREMENT overrides any DEFAULT, COPY, or DECREMENT value that was previously specified.

DECREMENT

(optional) Designates that if the user does not enter a value the first time a numeric field is processed, the last value in the field minus one will be used. DECREMENT is only valid when the field's data type is decimal, implied-decimal, or integer. DECREMENT overrides any DEFAULT, COPY, or INCREMENT value that was previously specified.

NODEFAULT

(optional) The default state when DEFAULT, COPY, INCREMENT, and DECREMENT have not been specified. If present, NODEFAULT cancels any default value that is specified. It also cancels any COPY, INCREMENT, or DECREMENT keyword.

AUTOMATIC

(optional) Specifies that when an empty field is processed, the specified default action—default, copy, increment, or decrement—occurs automatically, as if the user had pressed ENTER without entering any input. AUTOMATIC is only valid when one of the default actions listed above has already been specified. If neither DEFAULT, COPY, INCREMENT, nor DECREMENT has been specified, the AUTOMATIC keyword is ignored.

NOAUTOMATIC

(optional) The default state if AUTOMATIC has not been specified. If present, NOAUTOMATIC cancels automatic entry of data in the field.

NOECHO

(optional) Prevents the text entered by the user from being displayed in the input field. You can optionally specify a character to be displayed instead of each typed character; see the *display_char* argument, below. You can only specify the NOECHO keyword when the field's data type is alpha or user.

NOECHOCHR

(optional) Indicates that a character to be displayed in place of user input follows.

display_char

The character to be displayed for every character that the user types when entering field input. You can only specify the display character when the field's data type is alpha or user. If you specify a display character without specifying NOECHO, NOECHO is set automatically. The display character must be enclosed in double or single quotation marks (" " or ' ').

ECHO

(optional) The default state if NOECHO or NOECHOCHR has not been specified. If present, ECHO leaves echo on for field input. It also clears any specified *display_char*.

DATE TODAY

(optional) Defaults the date to today's date if the user presses ENTER without entering anything in a blank date field.

DATE NOTODAY

(optional) The default state if DATE TODAY has not been specified. If DATE NOTODAY is present, the date field no longer defaults to today's date.

DATE SHORT

(optional) Displays a date type field in fewer than the normal 11 characters. You can only specify the DATE SHORT keyword when the field's data type is date.

DATE NOSHORT

(optional) The default state if DATE SHORT has not been specified. If present, DATE NOSHORT cancels the short date option for the field.

TIME NOW

(optional) Defaults the time to the current system time if the user presses ENTER without entering any data in a blank time field.

TIME NONOW

(optional) The default state if TIME NOW has not been specified. If TIME NONOW is present, the time field no longer defaults to the current system time.

TIME AMPM

(optional) Specifies that the display format of a time field is 12-hour time, followed by an AM or PM indicator. This keyword is valid only when the field's data type is time.

TIME NOAMPM

(optional) The default state if TIME AMPM has not been specified. If present, TIME NOAMPM resets the display format to 24-hour time.

WAIT

(optional) Specifies an input time-out for this field, overriding the value in the UI Toolkit **g_wait_time** field.

time

(optional) Specifies the number of seconds to wait for input processing to be complete. This value may optionally be enclosed in double or single quotation marks (" " or ' ').

WAIT IMMEDIATE

(optional) Designates that immediate user response is required; do not wait.

WAIT GLOBAL

(optional) Designates that the global wait time (defined by **g_wait_time**) should be used.

WAIT FOREVER

(optional) Designates that UI Toolkit should wait until input processing is complete.

NOWAIT

(optional) The default state if WAIT has not been specified. If present, NOWAIT indicates that the UI Toolkit **g_wait_time** field defines the global wait time.

INPUT LENGTH

(optional) Indicates that the input length follows.

length

The maximum number of characters the user is permitted to enter in the field. Valid values are 0 through 65,535. Input length cannot be used in conjunction with RADIO, CHECKBOX, SELECTION LIST, SELECTION WINDOW, or DIMENSION.

NOINPUT LENGTH

(optional) The default state if INPUT LENGTH is not specified. If present, NOINPUT LENGTH indicates that Toolkit's default computation for input length will be used.

BREAK

(optional) Triggers a break in input set processing on a field. If the BREAK keyword is present, a break occurs after input to this field has been processed.

break_type

Specifies a different type of break processing. If *break_type* is specified, it must be on the same line as BREAK. Valid values are

ALWAYS	A break occurs whenever the field is accessed.
RETURN	A break occurs only when you press ENTER for that field.

NOBREAK

(optional) The default state if a break type has not been specified. If present, NOBREAK cancels break processing on the field.

REQUIRED

(optional) Specifies that a non-blank alpha or non-zero numeric entry is required in the field.

NOREQUIRED

(optional) The default state if REQUIRED has not been specified. If present, NOREQUIRED makes input in the field optional.

NEGATIVE

(optional) Allows negative values on decimal, implied-decimal, and integer fields.

ONLY

(optional) Specifies that *only* negative numbers are allowed as input for this field. If ONLY is specified, it must be on the same line as NEGATIVE.

ORZERO

(optional) Specifies that *only* negative numbers *or* zero are allowed as input for this field. If ORZERO is specified, it must be on the same line as NEGATIVE.

NONEGATIVE

(optional) The default state when NEGATIVE has not been specified. If present, NONEGATIVE indicates that negative values cannot be entered as input in the field.

NULL ALLOWED

(optional) Indicates that null is permitted for the field. This keyword is used by xfODBC to determine the null property for the column in the system catalog. Valid only for data types alpha (except when the storage format is binary), decimal, date, and time.

NULL DEFAULT

(optional) The default state when neither NULL ALLOWED nor NONULL is specified. If present, NULL DEFAULT indicates that the default behavior will be used in xfODBC to determine the null property for the column.

NONULL

(optional) Indicates that null is not permitted for the field. This keyword is used by xfODBC to determine the null property for the column in the system catalog. Valid for all data types.

ALLOW

(optional) Indicates that a list of valid entries for the field follows.

entry

An entry in the list of allowable entries for the field. If the entry contains spaces or is case sensitive, you must enclose it in double or single quotation marks (“ ” or ‘ ’). Blank entries must be stored in quotation marks; therefore, to specify them in your Synergy Data Language file, you must enclose them in a second set of different quotation marks. (For example, “‘ ’” or “‘ ’”.) You can specify up to 99 entries, each with a maximum length of 80 characters.

NOALLOW

(optional) The default state if no ALLOW entries have been specified. If present, NOALLOW cancels any ALLOW list for the field.

MATCH CASE

(optional) Specifies that the alpha or user field input must match the case of a specified allowable entry. (See the ALLOW keyword, above.) This keyword is valid only when the field’s data type is alpha or user, and an ALLOW list has already been specified.

MATCH NOCASE

(optional) The default state if MATCH CASE has not been specified. If present, MATCH NOCASE cancels case-sensitive matching for the field.

MATCH EXACT

(optional) Specifies that the alpha or user field input must match all characters in the specified allowable entry. (See the ALLOW keyword, above.) You can only specify the MATCH EXACT keyword when the field’s data type is alpha or user. MATCH EXACT is only valid when an ALLOW list has already been specified.

MATCH NOEXACT

(optional) The default state if MATCH EXACT has not been specified. If present, MATCH NOEXACT cancels full-length matching for the field.

SELECTION LIST

(optional) Designates that when this input field is processed, a selection window will be created, and it will contain the specified entries. The window should be placed at the given location, and it should have the specified height.

sl_row

Specifies the screen row at which the upper-left corner of the selection window will be placed, relative to the beginning of the data field. This value can be negative.

sl_column

Specifies the screen column at which the upper-left corner of the selection window will be placed, relative to the beginning of the data field. This value can be negative.

sl_height

The maximum number of rows in the selection window.

ENTRIES

(optional) Indicates that a list of entries to appear in the selection window follows.

sl_entry

An entry in the selection list. If the entry contains spaces or is case sensitive, it must be enclosed in double or single quotation marks (“ ” or ‘ ’). Blank entries must be stored in quotation marks; therefore, to specify them in your Synergy Data Language file, you must enclose them in a second set of different quotation marks (for example, “‘ ’” or ‘“ ”’). You can specify up to 99 entries, each with a maximum length of 80 characters.

SELECTION WINDOW

(optional) Designates that when this input field is processed, the given selection window will be placed on the screen at the specified location.

sw_row

Specifies the screen row at which the upper-left corner of the selection window will be placed, relative to the beginning of the data field. This value can be negative.

sw_column

Specifies the screen column at which the upper-left corner of the selection window will be placed, relative to the beginning of the data field. This value can be negative.

sw_name

The name of an existing selection window to associate with this field. This name can have a maximum of 15 characters and can optionally be enclosed in double or single quotation marks (“ ” or ‘ ’).

NOSELECT

(optional) The default state if SELECTION LIST or SELECTION WINDOW has not been specified. If present, NOSELECT cancels any selection list or selection window associated with the field.

ENUMERATED

(optional) Specifies that this field returns a decimal value for a displayed text entry. You can only specify the ENUMERATED keyword when the field’s data type is decimal, implied-decimal, or integer. You can only use the ENUMERATED keyword in conjunction with an allow list or a selection list or window (including both existing windows and windows built on the fly with the UI Toolkit S_SELBLD subroutine).

length

The length of the displayed field, if the field is enumerated. (Note that the length of the displayed field and the length of the actual input field are not necessarily the same.)

base

The return value of the first entry in an enumerated field. This value can be negative.

step

The value added to each successive entry in an enumerated field. This value can be negative.

NOENUMERATED

(optional) The default state if ENUMERATED has not been specified. If present, NOENUMERATED resets the field to a non-enumerated field.

RANGE

(optional) Indicates that a range of allowable values for a decimal, implied-decimal, integer, date, or time field follows. For date and time fields, the range values must be specified in storage form (STORED), rather than input or display form. You cannot specify the RANGE keyword in conjunction with an allow list or a selection list or window.

min

Defines the minimum value for a decimal, implied-decimal, integer, date, or time field. This value can be negative. *Min* must be less than or equal to the range maximum (as specified by the *max* argument). For date and time fields, *min* must be specified in storage form.

max

Defines the maximum value for a decimal, implied-decimal, integer, date, or time field. This value can be negative. *Max* must be greater than or equal to the range minimum (as specified by the *min* argument). For date and time fields, *max* must be specified in storage form.

NORANGE

(optional) The default state if RANGE has not been specified. If present, NORANGE cancels range-checking for the field.

ARRIVE METHOD

(optional) Indicates that the arrive method for this field follows.

arrive_method

The name of the subroutine (method) to be called before the field is processed by the UI Toolkit I_INPUT subroutine. The maximum size of the method name is 30 characters.

NOARRIVE METHOD

The default state if an arrive method has not been specified. If specified, NOARRIVE METHOD cancels any arrive method for the field.

LEAVE METHOD

(optional) Indicates that the leave method for this field follows.

leave_method

The name of the subroutine (method) to be called after this field is processed by the UI Toolkit I_INPUT subroutine. The maximum size of the method name is 30 characters.

NOLEAVE METHOD

The default state if a leave method has not been specified. If specified, NOLEAVE METHOD cancels any leave method for the field.

DRILL METHOD

(optional) Indicates that the drill method for this field follows. On Windows, if a drill method is specified, a drilldown button will be placed to the right of the input field. If the user clicks the button, the drill method is invoked. On both Windows and non-Windows environments, an I_DRILL menu entry will invoke the drill method.

drill_method

The name of the subroutine (method) to be called when this field's drill button is clicked or the I_DRILL menu entry is selected. The maximum size of the method name is 30 characters.

NODRILL METHOD

The default state if a drill method has not been specified. If specified, NODRILL METHOD cancels any drill method for the field.

HYPERLINK METHOD

(optional) Indicates that the hyperlink prompt method for this field follows. On Windows, if a hyperlink method is specified, when the field is a member of an input set being processed by I_INPUT, any prompt text associated with the field will be highlighted. If the user clicks on the highlighted text, the hyperlink method is invoked. On both Windows and non-Windows environments, an I_HYPER menu entry will invoke the hyperlink method.

hyperlink_method

The name of the subroutine (method) to be called when either this field's prompt text is clicked or the I_HYPER menu entry is selected. The maximum size of the method name is 30 characters.

NOHYPERLINK METHOD

The default state if a hyperlink method has not been specified. If specified, NOHYPERLINK METHOD cancels any hyperlink method for the field.

CHANGE METHOD

(optional) Indicates that the change method for this field follows.

change_method

The name of the subroutine (method) to be called after this field is validated by the UI Toolkit I_INPUT subroutine. The maximum size of the method name is 30 characters.

NOCHANGE METHOD

The default state if a change method has not been specified. If specified, NOCHANGE METHOD cancels any change method for the field.

DISPLAY METHOD

(optional) Indicates that the display method for this field follows.

display_method

The name of the subroutine (method) to be called whenever the field is about to be displayed by UI Toolkit. It is called after UI Toolkit has formatted the display according to its own rules. DISPLAY METHOD cannot be specified when the field's view is set to radio buttons or check box, or when a selection list or window has been specified.

NODISPLAY METHOD

The default state if a display method has not been specified. If specified, NODISPLAY METHOD cancels any display method for the field.

EDITFMT METHOD

(optional) Indicates that the edit format method for this field follows.

editfmt_method

The name of the subroutine (method) to be called by UI Toolkit whenever text in the field is being formatted for editing purposes. It is called after UI Toolkit has formatted the display according to its own rules. EDITFMT METHOD cannot be specified when a field's view is set to radio buttons, or check box, or when a selection list or window has been specified.

NOEDITFMT METHOD

The default state if an edit format method has not been specified. If specified, NOEDITFMT METHOD cancels any edit format method for the field.

Discussion

The FIELD statement describes a field definition. This field is associated with the most recently defined structure. If no structure has been defined, the field is ignored.

Adding new definitions

The order in which you specify the fields determines the order in which they will exist in the structure. The maximum number of fields that can be defined in one structure is 999.

Replacing existing definitions

All required keywords and data must be specified. The existing field is deleted and replaced by the specified one. (Note that when you're replacing a structure, any fields that are not explicitly specified in the schema are deleted.)

Overlaying existing definitions

Name must be respecified along with the desired attributes. The current field attributes are overwritten with any new attributes specified. When a *template* is added to an existing field, both the existing override flags and the field attributes explicitly specified in the schema are retained as template overrides.

Keep in mind that items such as allow list or selection list entries are specified by a single keyword. Hence, to change one allow list entry, you must respecify all allow list entries.

Examples

```
FIELD ccname TYPE alpha SIZE 40 DESCRIPTION "Name"
REPORT HEADING "Name" PROMPT "Name : " HELP "h_namehlp"
INFO LINE "Please enter your full name." BREAK ALWAYS

FIELD ccopen TYPE date SIZE 8 STORED YYYYMMDD
DESCRIPTION "Opened account"
LONG DESCRIPTION "Opened account (YYYYMMDD)"
REPORT JUST right REPORT HEADING "Opened account"
FORMAT "#04  MM-DD-YY" POSITION relative -3 3

FIELD transdate TEMPLATE date8
DESCRIPTION "Transaction date"
NONAMELINK
```

FILE – Describe a file definition

FILE *name filetype "open_filename"* [DESCRIPTION *"description"*]
[LONG DESCRIPTION *"long_desc"*] [USER TEXT *"string"*] [RECTYPE *rectype*]
[PAGE SIZE *page_size*] [DENSITY *percentage*] [NODENSITY] [ADDRESSING *addressing*]
[SIZE LIMIT *size_limit*] [NOSIZE LIMIT] [RECORD LIMIT *record_limit*]
[NORECORD LIMIT] [TEMPORARY] [NOTEMPORARY] [COMPRESS] [NOCOMPRESS]
[STATIC RFA] [NOSTATIC RFA] [TRACK CHANGES] [NOTRACK CHANGES]
[TERABYTE] [NOTERABYTE] [STORED GRFA] [NOSTORED GRFA]
[NOROLLBACK] [ROLLBACK] [NETWORK ENCRYPT] [NONETWORK ENCRYPT]
[PORTABLE *"integer_specs"*] [NOPORTABLE] [FILE TEXT *"file_text"*] [NOFILE TEXT]
[ASSIGN *structure* [ODBC NAME *name*] [, *structure* [ODBC NAME *name*]] [, …]]

Arguments

name

The name of a new or existing file definition. This name can have a maximum of 30 characters.

filetype

The type of file this definition describes. Valid values are

ASCII
DBL ISAM
RELATIVE
USER DEFINED

open_filename

The name of the actual data file, including the path specification. It can have a maximum of 64 characters. This string must be enclosed in double or single quotation marks (“ ” or ‘ ’).

DESCRIPTION

(optional) Indicates that a file definition description follows.

description

A description of the file definition. It can have a maximum of 40 characters and must be enclosed in double or single quotation marks (“ ” or ‘ ’). This description is available when Repository displays a list of files.

LONG DESCRIPTION

(optional) Indicates that a long description for the file follows.

long_desc

A more detailed description of the file definition and its use. It can contain 30 lines of up to 60 characters each. Each line must be enclosed in double or single quotation marks (“ ” or ‘ ’).

USER TEXT

(optional) Indicates that a user-defined text string follows.

string

A user-defined text string. It can contain a maximum of 60 characters and must be enclosed in double or single quotation marks (“ ” or ‘ ’).

RECTYPE

(optional) Indicates that the record type follows.

rectype

Specifies the record type. This value is ignored if specified for a *filetype* other than DBL ISAM. Valid values are

FIXED (default)
VARIABLE
MULTIPLE

PAGE SIZE

(optional) Indicates that the page size follows.

page_size

Specifies the index block page size. This value is ignored if specified for a *filetype* other than DBL ISAM. Valid values are

512
1024 (default)
2048
4096
8192
16384
32768

DENSITY

(optional) Indicates that the key density percentage follows.

percentage

Specifies the key density percentage used for all keys in the file. Used for DBL ISAM files only. *Percentage* must be between 50 and 100, inclusive. The default density is around 50%.

NODENSITY

(optional) Specifies that the default key density (around 50%) is to be used for all keys in the file.

ADDRESSING

(optional) Indicates that the file addressing follows.

addressing

Specifies the index block page size. This value is ignored if specified for a *filetype* other than DBL ISAM. Valid values are

32BIT (default)
40BIT

SIZE LIMIT

(optional) Indicates that a size limit for the data file follows.

size_limit

Specifies the maximum number of megabytes that the data file (**.is1**) is allowed to reach. This value applies only to REV 6 or greater ISAM files and is ignored if specified for a *filetype* other than DBL ISAM.

NOSIZE LIMIT

(optional) Indicates that there is no size limit for this file.

RECORD LIMIT

(optional) Indicates that a record limit for the file follows.

record_limit

Specifies the maximum number of records that the file is allowed to contain. This value applies only to REV 6 or greater ISAM files and is ignored if specified for a *filetype* other than DBL ISAM.

NORECORD LIMIT

(optional) Indicates that the file has no record limit.

TEMPORARY

(optional) Specifies that the file definition is a temporary one and will be excluded from the list of available files if used in ReportWriter or xfODBC.

NOTEMPORARY

(optional) Specifies that the file definition is not a temporary one and will be included in the list of available files if used in ReportWriter or xfODBC.

COMPRESS

(optional) Specifies that the data in the file is compressed. This value is ignored if specified for a *filetype* other than DBL ISAM.

NOCOMPRESS

(optional) Specifies that the data in the file is not compressed.

STATIC RFA

(optional) Used for DBL ISAM files only and specifies that the records in this file will retain the same RFA across WRITE operations. This value is ignored if specified for a *filetype* other than DBL ISAM.

NOSTATIC RFA

(optional) Specifies that the records in this file will not retain the same RFA across WRITE operations.

TRACK CHANGES

(optional) Specifies that change tracking is enabled in this file. This value applies only to REV 6 or greater ISAM files and is ignored if specified for a *filetype* other than DBL ISAM.

NOTRACK CHANGES

(optional) Specifies that change tracking is not enabled in this file.

TERABYTE

(optional) Specifies that this is a 48-bit terabyte file. This value is ignored if specified for a *filetype* other than DBL ISAM.

NOTERABYTE

(optional) Specifies that this is not a 48-bit terabyte file.

STORED GRFA

(optional) Specifies that the CRC-32 portion of an RFA is to be generated and stored to each record header on each STORE or WRITE operation. This value is ignored if specified for a *filetype* other than DBL ISAM.

NOSTORED GRFA

(optional) Specifies that the CRC-32 portion of an RFA is not to be generated and stored to each record header on each STORE or WRITE operation.

NOROLLBACK

(optional) Specifies that change tracking rollbacks to the file are prohibited. This value applies only to REV 6 or greater ISAM files and is ignored if specified for a *filetype* other than DBL ISAM.

ROLLBACK

(optional) Specifies that change tracking rollbacks to the file are permitted.

NETWORK ENCRYPT

(optional) Specifies that clients accessing this file must use encryption. This value is ignored if specified for a *filetype* other than DBL ISAM.

NONETWORK ENCRYPT

(optional) Specifies that clients accessing this file must not use encryption.

PORTABLE

(optional) Indicates that non-key portable integer data specifications follow. Used for DBL ISAM files only.

integer_specs

One or more non-key portable integer data specifications that can be passed as arguments to the ISAMC subroutine. Non-key integer data specifications have the following syntax:

`I=`*pos*`:`*len*`[,I=`*pos*`:`*len*`] [,...]`

where *pos* is the starting position of non-key portable integer data and *len* is its length in bytes (1, 2, 4, or 8). No validation is performed on this string.

NOPORTABLE

(optional) Specifies that no non-key portable integer data specifications are defined for the file.

FILE TEXT

(optional) Indicates that a file text specification follows.

file_text

A specification for text to be added to the header of the file and/or space to be allocated for user-defined text. This value applies only to REV 6 or greater ISAM files and is ignored if specified for a *filetype* other than DBL ISAM. The syntax of the specification must be one of the following:

text_size`[K]`
`"`*text_string*`"`
text_size`[K]:"`*text_string*`"`

where *text_size* is the amount of space to allocate in bytes (rounded to the nearest kilobyte) for user-defined text, and *text_string* is a text string to add to the file header. No validation is performed on this string.

NOFILE TEXT

(optional) Specifies that no file text specification is defined for the file.

ASSIGN

(optional) Indicates that the name of one or more structures to be assigned to this file follows.

structure

The name of a structure to assign to the file. The specified structure must already be defined. The maximum number of structures that can be assigned to the file is 200. The maximum size of a structure name is 30 characters.

ODBC NAME

(optional) Indicates that the table name to use for ODBC access follows.

name

The table name to use for ODBC access. The maximum size of a table name is 30 characters.

Discussion

The FILE statement is used to describe a file definition. File definitions determine which files can be accessed through Repository and which structures can be used to access them.

Only structures whose file type matches that of the file definition can be assigned to a definition. Also, the structure must have at least one field defined. When you assign the second or a subsequent structure to a file, the primary key definition must match those of already assigned structures. (The primary key is assumed to be the first key defined and must be an access key.) Specifically, the following key information must match:

- Key size
- Sort order
- Dups allowed flag
- Key data type
- Number of segments
- Type, position, length, and order of each segment

Adding new definitions

The maximum number of files that can be defined is 9,999.

Replacing existing definitions

All required keywords and data must be specified. The existing file and its list of assigned structures are cleared and set to the specified attributes. To disassociate structures from a file, specify only the ones you want to keep. Omitting the ASSIGN keyword will disassociate all assigned structures.

Overlaying existing definitions

Name, *filetype*, and *open_filename* must be specified, because they are position-dependent. The current file attributes are overwritten with any new attributes specified. All assigned structure names (and corresponding ODBC table names) must be respecified when changing one or more. To clear all assigned structures, use the **Replace** option in the Load Repository Schema utility.

Examples

```
FILE cmclnt dbl isam "FIL:cmclnt"
DESCRIPTION "CM Clients"
ASSIGN client

FILE cusmas dbl isam "FIL:cusmas"
DESCRIPTION "Customer Master"
RECTYPE variable DENSITY 75
PORTABLE "I=10:8,I=20:4"
ASSIGN cusmas1,cusmas2
```

FORMAT – Describe a global or structure-specific format

FORMAT *name* TYPE *type "string"* [JUSTIFY *just]*

Arguments

name

The name of a new or existing format. This name can have a maximum of 30 characters.

TYPE

Indicates that the format type follows.

type

The format type. Valid values are

ALPHA
NUMERIC

string

The format string, which must be enclosed in double or single quotation marks (" " or ' '). For alpha formats, an "at" sign (@) stands for an alphanumeric character. For numeric formats, Synergy DBL data formatting characters are used to represent the data. Refer to Appendix D for a list of valid formatting characters. Any other characters (such as dashes, backslashes, and so forth) appear wherever they are placed in the format. The maximum length of a format string is 30 characters.

JUSTIFY

(optional) Indicates that the format justification follows.

just

The justification type. This affects how the format is truncated before being applied to a field. Valid values are

NONE (default)
LEFT
RIGHT

Discussion

The FORMAT statement is used to describe a global or structure-specific format. If no structures have previously been defined, this format is stored as a global format. Otherwise, it is associated with the most recently defined structure.

Global formats must be defined before any templates or structures that reference them. Structure-specific formats must be defined after the STRUCTURE statement and before any fields.

Adding new definitions

If *name* is a global format, the name must be unique within the entire repository. If *name* is a structure-specific format, the name must be unique for the current structure.

The maximum number of global formats that can be defined is 9,999. The maximum number of structure-specific (local) formats that can be defined in one structure is 250.

Replacing existing definitions

All required keywords and data must be specified. The existing format is cleared and set to the specified attributes.

Overlaying existing definitions

All required keywords and data must be respecified because *"string"* is position-dependent. The current format attributes are overwritten with any new attributes that are specified.

Examples

```
FORMAT dig3num TYPE numeric "ZZZ"

FORMAT dig8mony TYPE numeric "ZZZ,ZZZ.ZZZ" JUSTIFY right

FORMAT license_num TYPE alpha "@@@@-@@"
```

GROUP – Begin a group definition

GROUP *name* [REFERENCE *structure*] [PREFIX *prefix*] [COMPILE PREFIX]
[NOCOMPILE PREFIX] [TEMPLATE *template*] TYPE *type* [SIZE *size*] [NOSIZE]
[STORED *store_format*] [NODATE] [NOTIME] [USER TYPE *"user_type"*] [NOUSER TYPE]
[PRECISION *dec_places*] [DIMENSION *#elements[:#elements ...]*]
[COERCED TYPE *type*] [NOCOERCED TYPE] [OVERLAY]
[LANGUAGE VIEW] [LANGUAGE NOVIEW] [SCRIPT VIEW] [SCRIPT NOVIEW]
[REPORT VIEW] [REPORT NOVIEW] [WEB VIEW] [WEB NOVIEW] [NONAMELINK]
[DESCRIPTION *"description"*] [NODESC] [LONG DESCRIPTION *"long_desc"*]
[NOLONGDESC] [POSITION *[pos_type] row column*] [NOPOSITION]
[FPOSITION *[fpos_type] frow fcolumn*] [NOFPOSITION] [PROMPT *"prompt"*] [NOPROMPT]
[HELP *"help"*] [NOHELP] [INFO LINE *"info_line"*] [NOINFO] [USER TEXT *"user_text"*]
[NOUSER TEXT] [FORMAT *format*] [NOFORMAT] [REPORT HEADING *"heading"*]
[NOHEADING] [ALTERNATE NAME *alt_name*] [NOALTERNATE NAME]
[REPORT JUST *just*] [INPUT JUST *ijust*] [BLANKIFZERO] [NOBLANKIFZERO]
[PAINT *"paint_char"*] [NOPAINT] [RADIO|CHECKBOX] [NORADIO] [NOCHECKBOX]
[FONT *font*] [NOFONT] [PROMPTFONT *prompt_font*] [NOPROMPTFONT]
[READONLY] [NOREADONLY] [DISABLED] [NODISABLED|ENABLED] [COLOR *palette#*]
[NOCOLOR] [HIGHLIGHT] [NOHIGHLIGHT] [REVERSE] [NOREVERSE]
[BLINK] [NOBLINK] [UNDERLINE] [NOUNDERLINE] [NOATTRIBUTES]
[DISPLAY LENGTH *length*] [NODISPLAY LENGTH]
[VIEW LENGTH *length*] [NOVIEW LENGTH] [UPPERCASE] [NOUPPERCASE]
[NODECIMAL] [DECIMAL_REQUIRED] [NOTERM] [TERM] [RETAIN POSITION]
[NORETAIN POSITION] [DEFAULT *default*|COPY|INCREMENT|DECREMENT]
[NODEFAULT] [AUTOMATIC] [NOAUTOMATIC] [NOECHO] [NOECHOCHR *"display_char"*]
[ECHO] [DATE TODAY] [DATE NOTODAY] [DATE SHORT] [DATE NOSHORT]
[TIME NOW] [TIME NONOW] [TIME AMPM] [TIME NOAMPM]
[WAIT *"time"*|WAIT IMMEDIATE|WAIT GLOBAL|WAIT FOREVER] [NOWAIT]
[INPUT LENGTH *length*] [NOINPUT LENGTH] [BREAK *[break_type]*] [NOBREAK]
[REQUIRED] [NOREQUIRED] [NEGATIVE [ONLY|ORZERO]] [NONEGATIVE]
[NULL ALLOWED|NULL DEFAULT|NONULL] [ALLOW *entry[, ...]*] [NOALLOW]
[MATCH CASE] [MATCH NOCASE] [MATCH EXACT] [MATCH NOEXACT]
[SELECTION LIST *sl_row sl_column sl_height* ENTRIES *sl_entry[, ...]*]
[SELECTION WINDOW *sw_row sw_column sw_name*] [NOSELECT]
[ENUMERATED *length base step*] [NOENUMERATED] [RANGE *min max*] [NORANGE]
[ARRIVE METHOD *arrive_meth*] [NOARRIVE METHOD]
[LEAVE METHOD *leave_meth*] [NOLEAVE METHOD]
[DRILL METHOD *drill_meth*] [NODRILL METHOD]
[HYPERLINK METHOD *hyperlink_meth*] [NOHYPERLINK METHOD]
[CHANGE METHOD *change_meth*] [NOCHANGE METHOD]
[DISPLAY METHOD *display_meth*] [NODISPLAY METHOD]
[EDITFMT METHOD *editfmt_meth*] [NOEDITFMT METHOD]

Arguments

See FIELD on page 6-14 for a description of the arguments not listed here.

name

The name of a new or existing group (field). This name can have a maximum of 30 characters.

REFERENCE

Indicates that this is an implicit group, and that the name of the referenced structure follows.

structure

The name of the structure that defines the members of the group.

PREFIX

Indicates that the group member prefix follows.

prefix

The prefix added to group member names when accessed by Synergy DBL, UI Toolkit, and xfODBC. This prefix can have a maximum of 30 characters.

COMPILE PREFIX

(optional) Indicates that any group member prefix specified will be added to all group member fields when referenced by the Synergy compiler.

NOCOMPILE PREFIX

(optional) Indicates that any group member prefix specified will not be added to group member fields when referenced by the Synergy compiler. NOCOMPILE PREFIX is the default.

SIZE

(optional) Indicates that the group size follows. If SIZE is not specified, the size of the group is determined by the size of its members.

size

The maximum number of characters the group (field) can contain. The maximum size is 99,999 for all fields except implied-decimal fields, where the maximum size is 28. If the type is date or time, the size must be appropriate for the selected storage format. (See store_format on page 6-16.) For example, the size must be 6 for the format YYMMDD, 8 for the format YYYYMMDD, and so forth.

NOSIZE

(optional) Indicates that the size of the group is unspecified and is determined by the size of its members. NOSIZE must be explicitly specified if the group references a template, and you want to use the size of the group members rather than the template's size.

OVERLAY

(optional) Defines this group as an overlay to the previous non-overlay field or group.

Discussion

The GROUP statement describes a group (field) definition. This group will be associated with the most recently defined structure. If no structure has been defined yet, the group is ignored.

Adding new definitions

The order in which you specify group and non-group fields determines the order in which they will exist in the structure. The maximum number of group and non-group fields that can be defined in one structure is 999.

Replacing existing definitions

All required keywords and data must be specified. The existing group is deleted and replaced by the specified one. When replacing a group you must respecify its ancestry. In other words, you must first specify any groups to which it belongs (parent, grandparent, and so forth). This is required because group names are only required to be unique within their own level. (Note that when you're replacing a structure, any group or non-group fields that are not explicitly specified in the schema are deleted.)

Overlaying existing definitions

Name must be respecified along with the desired attributes. The current group attributes are overwritten with any new attributes specified. When overlaying a group you must respecify its ancestry. In other words, you must first specify any groups to which it belongs (parent, grandparent, and so on). This is required because group names are required to be unique only within their own level. Keep in mind that items such as allow list or selection list entries are specified by a single keyword. Hence, to change one allow list entry, you must respecify all allow list entries.

Examples

```
STRUCTURE info   DBL ISAM
GROUP customer   TYPE alpha
   FIELD name   TYPE alpha   SIZE 40
   GROUP office   TYPE alpha   SIZE 70
      FIELD bldg   TYPE alpha   SIZE 20
      GROUP address   TYPE alpha   SIZE 50
         FIELD street   TYPE alpha   SIZE 40
         FIELD zip   TYPE decimal   SIZE 10
      ENDGROUP
   ENDGROUP
   GROUP contact   TYPE alpha   SIZE 90
      FIELD name   TYPE alpha   SIZE 40
      GROUP address   TYPE alpha   SIZE 50
         FIELD street   TYPE alpha   SIZE 40
         FIELD zip   TYPE decimal   SIZE 10
      ENDGROUP
   ENDGROUP
ENDGROUP
```

KEY – Describe a key definition

KEY *name type* [ORDER *order*] [DUPS *dups*] [INSERT *location*] [MODIFIABLE *modifiable*] [NONULL|NULL REPLICATING|NULL NONREPLICATING|NULL SHORT [VALUE *value*]] [DENSITY *percentage*] [NODENSITY] [COMPRESS [INDEX] [RECORD] [KEY]] [NOCOMPRESS] [ODBC VIEW] [ODBC NOVIEW] [KRF *krf*] [DESCRIPTION *"description"*] SEGMENT *segtype data* [SEGMENT *segtype data*] […]

Arguments

name

The name of a new or existing key. This name can have a maximum of 30 characters.

type

The key type. Valid values are

ACCESS
FOREIGN

Access keys represent true keys in the data file and are used to specify relationships between files. Foreign keys are also used to specify relationships between files, but they don't have to be true keys in the data file.

ORDER

(optional) Indicates that the data order follows.

order

Specifies how the key data for an access key is stored. This value is ignored if it is specified for a foreign key. If the current structure's file type is relative, the order must be ASCENDING. Valid values are

ASCENDING (default)
DESCENDING

DUPS

(optional) Indicates that the duplicates value follows.

dups

Specifies whether an access key allows duplicates. This value is ignored if it is specified for a foreign key. If the current structure's file type is relative, the DUPS value must be NO. Valid values are

YES (default for all keys except the primary key [the first key defined])
NO

INSERT

(optional) Indicates that the insertion value follows.

location

Used for access keys only and specifies where records with duplicate keys are inserted relative to other records containing the same key value. This value is ignored if specified for a foreign key. Valid values are

FRONT (default)
END

MODIFIABLE

(optional) Indicates that the modifiable value follows.

modifiable

Specifies whether an access key is modifiable. Used only for access keys other than the primary key (the first one defined). This value is ignored if specified for a foreign key. Valid values are

YES
NO (default)

The following five keywords are used only for access keys other than the primary key (the first one defined). They are ignored if specified for a foreign key.

NONULL

(optional) Specifies that the key is not a null key. The default null key value is NONULL. This keyword is used to modify an existing null key.

NULL REPLICATING

(optional) Specifies that the key is a replicating null key. The null key *value* is a single character and must match each byte of the specified key.

NULL NONREPLICATING

(optional) Specifies that the key is a non-replicating null key. The null key *value* is a string that must match the key, from the beginning of the key and for the length of the key.

NULL SHORT

(optional) Specifies that the key is a short null key.

VALUE

(optional) Indicates that the null key value follows.

value

The replicating or non-replicating null key value. For a replicating null key, *value* is a single character. For a non-replicating null key, *value* is a string. The default null *value* is a space.

DENSITY

(optional) Indicates that the key density value follows.

percentage

Used for access keys only and specifies a number between 50 and 100 representing the density percentage for this key. If specified, *percentage* overrides the density of the file to which this key belongs. The default key density is unspecified.

NODENSITY

(optional) Indicates that no file density override is specified, and that the key density used will be that of the file to which the key belongs. This keyword is used to modify an existing key.

COMPRESS

(optional) Indicates that up to three key compression options follow. These options are used only by RMS indexed files. The compression options are treated as a set. Therefore, to overlay one or more of them, you must respecify all of them. All compression options must be specified on the same line together with COMPRESS. No compression options are set by default.

INDEX

Used for access keys only and specifies that the key's index is compressed.

RECORD

Used for access keys only and specifies that the record within the data is compressed.

KEY

Used for access keys only and specifies that the key within the data is compressed.

NOCOMPRESS

(optional) Indicates that no index, record, or key compression is specified. This keyword is used to modify an existing key.

ODBC VIEW

(optional) Indicates that the key will be available to xfODBC as a described index in the system catalog. ODBC VIEW is the default.

ODBC NOVIEW

(optional) Indicates that the key will *not* be available to xfODBC as a described index in the system catalog.

KRF

(optional) Indicates that the key of reference follows.

krf

Used by ReportWriter and xfODBC to access the file. The key of reference is 0-based. For example, the third key defined in your file is key of reference 2. You only need to specify the key of reference when you use RMS indexed files (which allow more than 32 keys) and you want to access your file with a key other than one of the first 32.

DESCRIPTION

(optional) Indicates that a key definition description follows.

description

A description of the key definition. It can have a maximum of 40 characters and must be enclosed in double or single quotation marks (“ ” or ‘ ’). This description is available when Repository displays a list of keys.

SEGMENT

Indicates that the data for a key segment follows. The segment type and data must be specified on the same line together with SEGMENT.

segtype

The segment type for one segment of the key. A key must contain at least one segment definition and may contain a maximum of eight segment definitions. Valid values are

FIELD	Defines a field in the current structure as a segment
LITERAL	Defines a literal as a segment, enabling you to preface, append, or embed a constant within a key
EXTERNAL	Defines a field in another structure as a segment
RECORD NUMBER	Defines the one access key for a relative file

data

The value that helps define each segment of the key. If you specify a *segtype* of FIELD, *data* must be in the following form:

field_name [`SEGTYPE` *segdtype] [*`SEGORDER` *segorder] [*`NOSEGORDER`*]*

field_name

The name of a field in the current structure. The maximum size is 30 characters.

SEGTYPE

(optional) Indicates that the key segment data type follows.

segdtype

Used for access keys only and specifies the data type for this specific segment which overrides the default data type of the key. The default segment data type override is unspecified. Valid values are

ALPHA
NOCASE
DECIMAL
INTEGER
UNSIGNED
SEQUENCE
TIMESTAMP
CTIMESTAMP

SEGORDER

(optional) Indicates that the key segment order follows.

segorder

Used for access keys only. Specifies an override to the key's sort order. The default key segment order override is unspecified. Valid values are

ASCENDING
DESCENDING

NOSEGORDER

(optional) Used for access keys only and specifies that the key sort order (rather than key segment sort order) is used for the key. This keyword is used to modify an existing key.

If you specify a *segtype* of LITERAL, *data* must be a literal value. If the literal value contains spaces, the entire value must be enclosed in double or single quotation marks (" " or ' '). The maximum size of a literal value is 30 characters.

If you specify a *segtype* of EXTERNAL, *data* must be the name of another structure, followed by the name of a field within that structure. The maximum size of a structure name is 30 characters.

If you specify a *segtype* of RECORD NUMBER, *data* is not required.

If a specified field or structure has not yet been defined, the segment is ignored.

Discussion

The KEY statement is used to describe a key definition. This key is associated with the most recently defined structure. If no structure has been defined yet, the key is ignored.

Adding new definitions

The first key defined for a structure is assumed to be the primary key. Access keys must be defined first, followed by any foreign keys. The order of your access keys determines the key of reference used by ReportWriter and xfODBC. You can define a maximum of 99 keys in one structure.

Replacing existing definitions

All required keywords and data must be specified. The existing key is deleted and replaced by the specified one. (Note that when you replace a structure, any keys that are not explicitly specified in the schema will be deleted.)

Overlaying existing definitions

Name and *type* must be respecified because *type* is position-dependent. The current key attributes are overwritten with any new attributes that are specified. Keep in mind that because the key segments are not numbered, to overlay one segment, you must respecify all of them.

For structures whose file type is relative, only one access key can be defined, and it must be ascending, allow no duplicates, and have one segment of type record number. Any additional access keys are ignored.

Examples

```
KEY cckey access ORDER ascending DUPS no
SEGMENT field cccomp SEGMENT field ccclnt

KEY prim_cont foreign SEGMENT field cccomp
DESCRIPTION "Primary contact key"
SEGMENT literal "01" SEGMENT external cccont cnname
```

RELATION – Describe a relation definition

```
RELATION name from_structure from_key to_structure to_key
```

Arguments

name

The name of a new or existing relation. This name can have a maximum of two characters and must be a numeric value between 1 and 99.

from_structure

The name of the structure defining the relation. The maximum size of the name is 30 characters. The specified structure must already be defined.

from_key

The name of the key to use in the *from_structure*. This may be an access or foreign key. The maximum size of the key name is 30 characters.

to_structure

The name of the structure to which you want to relate the *from_structure*. The maximum size of the name is 30 characters.

to_key

The name of an access key in the *to_structure*, from which to relate the *from_key.* The maximum size of the key name is 30 characters.

Discussion

The RELATION statement is used to describe a relation definition.

Relations enable you to link the keys of one structure with the keys of other structures. Relations can be defined within their defining structure (*from_structure*, after all keys for that structure) or after all structures. (The Generate Repository Schema utility will generate them along with their defining structure.) At the end of the "load" process, all relations are validated. If either structure or either key specified in a relation is undefined, an error is logged and the relation is not loaded into the repository.

Adding new definitions

You can define a maximum of 99 relations for any one structure (*from_structure*).

Replacing existing definitions

All required keywords and data must be specified. The existing relation is deleted and replaced by the specified one. (Note that when you replace a structure, any relations that are not explicitly specified in the schema are deleted.)

Overlaying existing definitions

All required keywords and data must be respecified. The current relation attributes are overwritten with any new attributes that are specified.

Examples

```
RELATION 1 client prim_cont cmcont cnkey
```

STRUCTURE – Describe a structure definition

```
STRUCTURE name filetype [MODIFIED date] [DESCRIPTION "description"]
[LONG DESCRIPTION "long_desc"] [USER TEXT "string"]
```

Arguments

name

The name of a new or existing structure. This name can have a maximum of 30 characters.

filetype

The type of file to which this structure will be assigned. Valid values are

ASCII
DBL ISAM
RELATIVE
USER DEFINED

MODIFIED

(optional) Indicates that a modification date follows. This keyword displays in a generated structure if the "Generate structure timestamps" option (**-t**) was selected when generating the schema. When editing a structure via schema, you must modify the value of this keyword. When the schema is reloaded, this value will be used as the timestamp value for the structure. If it is not present, the current date and time will be used.

date

The date and time that the structure was last modified in the format YYYYMMDDHHMMSS.

DESCRIPTION

(optional) Indicates that a structure definition description follows.

description

A description of the structure definition. It can have a maximum of 40 characters and must be enclosed in double or single quotation marks (" " or ' '). ReportWriter can use it, along with the file description, to identify your file. This description is available when Repository displays a list of structures.

LONG DESCRIPTION

(optional) Indicates that a long description for the structure follows.

long_desc

A more detailed description of the structure definition and its use. It can contain 30 lines of up to 60 characters each. Each line must be enclosed in double or single quotation marks (" " or ' ').

USER TEXT

(optional) Indicates that a user-defined text string follows.

string

A user-defined text string. It can contain a maximum of 60 characters and must be enclosed in double or single quotation marks (“ ” or ‘ ’).

Discussion

The STRUCTURE statement is used to describe a structure definition.

A structure is a record definition or compilation of field and key characteristics for a particular file or files. Structures must be defined after all global formats and templates.

If a structure is invalid for any reason, it will not be loaded into the repository, nor will its fields, keys, formats, relations, aliases, or tags be loaded.

Adding new definitions

The maximum number of structures that can be defined is 9,999.

Replacing existing definitions

All required keywords and data must be specified. The existing structure is cleared. All existing fields, keys, relations, local formats, and tags are deleted, and only the ones specified in the schema are added.

If aliases are specified with the structure, new aliased structures and their fields are added. All alias fields in existing specified alias structures are replaced by specified alias fields. Existing alias structures (and their fields) that are not specified are unaffected.

Overlaying existing definitions

Name and *filetype* must be respecified, because they are position-dependent. The current structure attributes are overwritten with any new attributes specified. All specified fields, keys, relations, local formats, and tags are updated with any new attributes as well. Any new fields, keys, and so forth are added. Existing fields, keys, and so forth that are not specified are unaffected.

If aliases are specified with the structure, the same rules apply as when replacing existing definitions. (See “Replacing existing definitions” above.)

Examples

```
STRUCTURE client dbl isam DESCRIPTION "CM Clients"
```

TAG – Describe a structure tag definition

TAG *type [field_1 op_1 value_1 [connect field_2 op_2 value_2] [… connect field_10 op_10 value_10]]*

Arguments

type

The tag type. Valid values are

FIELD
SIZE
NONE

If you specify FIELD, you must also specify the *field_1*, *op_1*, and *value_1* arguments. If you specify SIZE or NONE, the remaining arguments are not applicable and should not be specified.

field_1 to *field_10*

The name of a tag field. The maximum size of the field name is 30 characters. *Field_n* must be the name of a field in the current structure.

op_1 to *op_10*

The operator used to compare *field_n* with *value_n*. Valid values are

EQ	Equal to
NE	Not equal to
LE	Less than or equal to
LT	Less than
GE	Greater than or equal to
GT	Greater than

value_1 to *value_10*

The first comparison value for *field_n*. The maximum size of this value is 15 characters. If the value contains spaces or is case sensitive, it must be enclosed in double or single quotation marks (“ ” or ‘ ’).

connect

The connector used if you’re specifying more than one comparison criterion. If you specify *connect*, you must also specify corresponding *field_n*, *op_n*, and *value_n* arguments. Valid values are

AND
OR

Discussion

The TAG statement is used to describe a structure tag definition. A tag definition consists of one or more comparison criteria.

Tags are associated with a structure and are used when multiple structures are assigned to one file. The tag information is the information that uniquely identifies one structure (or record type) from another. Tag criteria can be used during I/O operations to filter records.

The tag is associated with the most recently defined structure. It must be defined before the first field or after all fields for the current structure. If no structure has been defined yet, the tag is ignored.

Although you can specify multiple comparison criteria and reference multiple fields, ReportWriter only supports a maximum of two criteria, and they must reference the same field.

Adding new definitions

You can define a maximum of 10 tags although ReportWriter only supports a maximum of two. See note above.

Replacing existing definitions

All required keywords and data must be specified. The existing tag is cleared in the current structure and set to the specified attributes.

Overlaying existing definitions

All required keywords and data must be respecified. The current attributes are overwritten with any new attributes that are specified.

Examples

```
TAG SIZE

TAG FIELD transtype EQ "C"

TAG FIELD cm_code GE 10 AND cm_code LE 15

TAG FIELD amount GE 1000 AND amount LT 5000 AND cus_type EQ "VAR"
```

TEMPLATE – Describe a template definition

TEMPLATE *name* [PARENT *template*] TYPE *type* SIZE *size* [STORED *store_format*]
[ENUM *name*] [NODATE] [NOTIME] [USER TYPE *"user_type"*] [NOUSER TYPE]
[PRECISION *dec_places*] [DIMENSION *#elements*[:*#elements* ...]]
[LANGUAGE VIEW] [LANGUAGE NOVIEW] [SCRIPT VIEW] [SCRIPT NOVIEW]
[REPORT VIEW] [REPORT NOVIEW] [WEB VIEW] [WEB NOVIEW]
[COERCED TYPE *type*] [NOCOERCED TYPE] [OVERLAY *field*[:*offset*]] [NONAMELINK]
[DESCRIPTION *"description"*] [NODESC] [LONG DESCRIPTION *"long_desc"*]
[NOLONGDESC] [POSITION [*pos_type*] *row column*] [NOPOSITION]
[FPOSITION [*fpos_type*] *frow fcolumn*] [NOFPOSITION] [PROMPT *"prompt"*] [NOPROMPT]
[HELP *"help"*] [NOHELP] [INFO LINE *"info_line"*] [NOINFO] [USER TEXT *"user_text"*]
[NOUSER TEXT] [FORMAT *format*] [NOFORMAT] [REPORT HEADING *"heading"*]
[NOHEADING] [ALTERNATE NAME *alt_name*] [NOALTERNATE NAME]
[REPORT JUST *just*] [INPUT JUST *ijust*] [BLANKIFZERO] [NOBLANKIFZERO]
[PAINT *"paint_char"*] [NOPAINT] [RADIO|CHECKBOX] [NORADIO] [NOCHECKBOX]
[FONT *font*] [NOFONT] [PROMPTFONT *prompt_font*] [NOPROMPTFONT]
[READONLY] [NOREADONLY] [DISABLED] [NODISABLED|ENABLED]
[COLOR *palette#*] [NOCOLOR] [HIGHLIGHT] [NOHIGHLIGHT] [REVERSE] [NOREVERSE]
[BLINK] [NOBLINK] [UNDERLINE] [NOUNDERLINE] [NOATTRIBUTES]
[DISPLAY LENGTH *length*] [NODISPLAY LENGTH]
[VIEW LENGTH *length*] [NOVIEW LENGTH] [UPPERCASE] [NOUPPERCASE]
[NODECIMAL] [DECIMAL_REQUIRED] [NOTERM] [TERM]
[RETAIN POSITION] [NORETAIN POSITION]
[DEFAULT *default*|COPY|INCREMENT|DECREMENT] [NODEFAULT]
[AUTOMATIC] [NOAUTOMATIC] [NOECHO] [NOECHOCHR *"display_char"*] [ECHO]
[DATE TODAY] [DATE NOTODAY] [DATE SHORT] [DATE NOSHORT]
[TIME NOW] [TIME NONOW] [TIME AMPM] [TIME NOAMPM]
[WAIT *"time"*|WAIT IMMEDIATE|WAIT GLOBAL|WAIT FOREVER] [NOWAIT]
[INPUT LENGTH *length*] [NOINPUT LENGTH] [BREAK [*break_type*]] [NOBREAK]
[REQUIRED] [NOREQUIRED] [NEGATIVE [ONLY|ORZERO]] [NONEGATIVE]
[NULL ALLOWED|NULL DEFAULT|NONULL] [ALLOW *entry*[, ...]] [NOALLOW]
[MATCH CASE] [MATCH NOCASE] [MATCH EXACT] [MATCH NOEXACT]
[SELECTION LIST *sl_row sl_column sl_height* ENTRIES *sl_entry*[, ...]]
[SELECTION WINDOW *sw_row sw_column sw_name*] [NOSELECT]
[ENUMERATED *length base step*] [NOENUMERATED] [RANGE *min max*] [NORANGE]
[ARRIVE METHOD *arrive_method*] [NOARRIVE METHOD]
[LEAVE METHOD *leave_method*] [NOLEAVE METHOD]
[DRILL METHOD *drill_method*] [NODRILL METHOD]
[HYPERLINK METHOD *hyperlink_method*] [NOHYPERLINK METHOD]
[CHANGE METHOD *change_method*] [NOCHANGE METHOD]
[DISPLAY METHOD *display_method*] [NODISPLAY METHOD]
[EDITFMT METHOD *editfmt_method*] [NOEDITFMT METHOD]

Arguments

See FIELD on page 6-14 for a description of the arguments not listed here, but note that Struct is not a supported data type for templates.

name

The name of a new or existing template. This name can have a maximum of 30 characters.

PARENT

(optional) Assigns a parent template to the template. (A parent is to a template what a template is to a field.)

template

The name of a different template, or parent, to assign to the current template. All template attributes, including type and size, are obtained from the specified parent template. The maximum size of the parent name is 30 characters. The specified parent template must already exist in the repository or be defined previously within the schema. If it is not, an error will be logged. If a parent is assigned to a template and none of the parent's attributes are overridden in the template, no additional keywords are required. The Synergy Data Language assumes that any keywords specified in addition to the parent name are overrides to the parent.

Discussion

The TEMPLATE statement is used to describe a template. A template is a set of field characteristics that can be assigned to one or more field or template definitions. Templates *must* be defined after any global formats and before any structure definitions. Parent templates must be defined before child templates that reference them. The maximum number of fields that can use the same template is 6,000. The maximum number of templates that can use the same parent template is 3,000.

Adding new definitions

You can define a maximum of 9,999 templates.

Replacing existing definitions

All required keywords and data must be specified. The existing template is cleared and set to the specified attributes.

Overlaying existing definitions

Name must be respecified along with the desired attributes. The current template attributes are overwritten with any new attributes that are specified. When a *parent* is added to an existing template, both the existing override flags and the template attributes that are explicitly specified in the schema are retained as parent overrides.

Keep in mind that items such as allow list or selection list entries are specified by a single keyword. Hence, to change one allow list entry, you must respecify all allow list entries.

Examples

```
TEMPLATE amount TYPE decimal SIZE 6 PRECISION 2
DESCRIPTION "Implied-decimal amount"

TEMPLATE date8 TYPE date SIZE 8 STORED YYYYMMDD
DESCRIPTION "Four-digit year"
LONG DESCRIPTION "Four-digit year (YYYYMMDD)"
FORMAT "#03  MM-DD-YYYY" DATE TODAY

TEMPLATE color TYPE alpha SIZE 6
DESCRIPTION "Order item color"
POSITION relative 4 4 PROMPT "Enter desired color: "
ALLOW red,blue,yellow,green

TEMPLATE date_mmddyyyy PARENT date8
DESCRIPTION "Date formatted MM-DD-YYYY"
```

7

Subroutine Library

Using the Repository Subroutine Library

The subroutines in the Repository subroutine library provide read-only access to Repository information about structures, enumerations, files, templates, fields, keys, relations, formats, tags, and aliases. Most of these subroutines have more than one function, which enables them to obtain a variety of data.

These routines are useful any time you need information about repository data at runtime. They are most often used during development, but they may also be used, for example, to construct UI input windows at runtime. See "Sample programs" on page 7-36 for some examples.

To use the Repository subroutine library you must do the following:

- Include the file **RPSLIB:ddinfo.def** in your calling program using the .INCLUDE statement. (See below for more information on using **ddinfo.def**.)
- Call the DD_INIT subroutine before other Repository subroutines, and call DD_EXIT after the other Repository subroutines.
- On Windows and UNIX, link the file **RPSLIB:ddlib.elb** with your program. On OpenVMS, link the file **TKLIB_SH.EXE** with your program. See the "Building and Running Synergy Applications" chapter of *Synergy Tools* for the link commands for your operating system.
- On Synergy .NET, reference **Synergex.SynergyDE.ddlib.dll** in your Visual Studio project.

The ddinfo.def file

You can use the **ddinfo.def** file for informational purposes, but you should not change its contents. This file is distributed in your rps\lib directory. See "Definition file" on page 7-45 for a listing of **ddinfo.def**.

Your calling program and the Repository subroutines communicate with each other through **ddinfo.def**, which contains definitions of the function codes, flags, and record layouts used by the subroutine library. One of these record layouts is the repository control structure (**dcs**), which is passed to all Repository subroutines as the first argument. The first two bytes of the control structure hold the error code that is set by each subroutine. You should check the error code after each subroutine returns. It will be zero if no error has occurred.

You can include and exclude parts of **ddinfo.def** by using DDINFO_DEFINES_ONLY and DDINFO_INGLOBAL:

- To include only the Repository definitions (not the record layouts), define DDINFO_DEFINES_ONLY before including **ddinfo.def**.
- To exclude STACK records and STRUCTUREs, which are not allowed when including **ddinfo.def** within a global data section, define DDINFO_INGLOBAL before including **ddinfo.def**.

You can change the way data structures are included by using DDINFO_STRUCTURE:

- To define all data structures as STRUCTUREs, define DDINFO_STRUCTURE before including **ddinfo.def**. The elements of each data structure will be enclosed in a STRUCTURE–ENDSTRUCTURE statement. (See STRUCTURE–ENDSTRUCTURE in the "Synergy DBL Statements" chapter of the *Synergy DBL Language Reference Manual*.)
- To define all data structures as RECORDs, leave DDINFO_STRUCTURE undefined, or undefine it if it was previously defined. The elements of each data structure will be enclosed in a RECORD–ENDRECORD statement. (See RECORD–ENDRECORD in the "Synergy DBL Statements" chapter of the *Synergy DBL Language Reference Manual*.)

Defining DDINFO_STRUCTURE also affects how arrays within data structures are handled. When DDINFO_STRUCTURE is defined, arrays within data structures are defined as real arrays, because pseudo arrays are not allowed within a STRUCTURE statement. When DDINFO_STRUCTURE is not defined, arrays within data structures are defined as pseudo arrays for compatibility with existing code.

DD_ALIAS – Retrieve alias information

WT	WN	U	V

`xcall DD_ALIAS(`*dcs*`, DDA_INFO,` *name, a_info*`)`

or

`xcall DD_ALIAS(`*dcs*`, DDA_SLIST,` *names_req, array, [array2], [start][, #names]*`)`

Arguments

dcs

The repository control structure.

DDA_INFO

Returns general alias information and sets the current alias.

name

The unique alias name. (**a30**)

a_info

Returned with the alias data. See the **a_info** record definition in the **ddinfo.def** file.

DDA_SLIST

Returns the current alias structure's alias and aliased field names in sequence order.

names_req

The number of alias names requested. (**d3**)

array

Returned with the array of alias field names. ((*)**a30**)

array2

(optional) Returned with the array of aliased field names. ((*)**a30**)

start

(optional) Contains the alias field name at which to start. (**a30**)

#names

(optional) Returned with the number of alias field names. (**d3**)

Discussion

The DD_ALIAS subroutine returns information about aliases. There are two ways to call DD_ALIAS:

- DDA_INFO enables you to retrieve general information about an alias.
- DDA_SLIST enables you to retrieve a list of alias and aliased fields for an alias structure.

DDA_INFO

If you pass DDA_INFO, the DD_ALIAS subroutine reads the specified alias. If that alias is not found, the relevant error code is returned in the control structure. If the alias is found, information about the alias is recorded in the control structure, and general information is returned in *a_info*.

Once you set a current alias with the DDA_INFO function, the DDA_SLIST function becomes valid.

DDA_SLIST

Once an alias has been selected, the DDA_SLIST function becomes valid. The DDA_SLIST function returns an array of alias field names for the current alias structure. The names are returned in sequence order (the order defined within the alias structure), starting with either the first name or the specified name. If you pass *array2*, DD_ALIAS also returns a corresponding list of the aliased field names. You can then use DD_FIELD to obtain general information about the aliased fields.

DD_ALIAS returns as many field names as are found or as are requested, whichever is smaller. The actual number of names in the array can be returned in *#names*. You must make sure that the buffer passed is large enough to hold the number of names that you are requesting.

DD_CONTROL – Retrieve control record information

WT	WN	U	V

```
xcall DD_CONTROL(dcs, DDC_INFO, c_info)
```

Arguments

dcs

The repository control structure.

DDC_INFO

Returns general repository control record information.

c_info

Returned with the control record data. See the **c_info** record definition in the **ddinfo.def** file.

Discussion

The DD_CONTROL subroutine returns control record information about the current repository. This information includes the repository version and the date and time of the last repository modification.

DD_ENUM – Retrieve enumeration information

WT	WN	U	V

`xcall DD_ENUM(`*dcs*`, DDE_INFO,` *name, e_info*`)`

or

`xcall DD_ENUM(`*dcs*`, DDE_TEXT,` *field, data*`)`

or

`xcall DD_ENUM(`*dcs*`, DDE_MBRS,` *names_req, array, array2, [start] [,#names]*`)`

Arguments

dcs

The repository control structure.

DDE_INFO

Returns general enumeration information and sets the current enumeration.

name

The unique enumeration definition name. (**a30**)

e_info

Returned with the enumeration data (including number of members). See the **e_info** record definition in the **ddinfo.def** file.

DDE_TEXT

Returns textual information about the current enumeration.

field

A field in the **e_info** record that indicates what type of textual information should be returned in *data* (if the field is non-zero):

ei_desc	Short description. (**a40**)
ei_ldesc	Long description. (**a1800**)

data

Returned with the requested textual data.

DDE_MBRS

Returns an enumeration's member names and values.

names_req

The number of member names requested. (**d3**)

array

Returned with an array of member names. (**(*)a30**)

array2

Returned with an array of corresponding member values. (**(*)a11**)

start

(optional) Contains the member name at which to start. (**a30**)

#names

(optional) Returned with the number of member names. (**d3**)

Discussion

The DD_ENUM subroutine returns information about enumerations. There are three ways to call DD_ENUM:

- DDE_INFO enables you to retrieve general information about an enumeration.
- DDE_TEXT enables you to retrieve textual information about an enumeration.
- DDE_MBRS enables you to retrieve the enumeration's member names and values.

DDE_INFO

If you pass DDE_INFO, the DD_ENUM subroutine reads the specified enumeration. If that enumeration is not found, the relevant error code is returned in the control structure. If it is found, the enumeration name is recorded in the control structure and general information is returned in *e_info*.

DDE_TEXT

Once an enumeration has been selected, the DDE_TEXT function is valid. DDE_TEXT is used to obtain textual information about the enumeration. For each type of textual information, a corresponding field in the **e_info** record is non-zero. For example, if the **ei_desc** field in the **e_info** record is non-zero, a short description exists for the enumeration. If you pass DDE_TEXT along with the non-zero field, the corresponding textual information is returned.

DDE_MBRS

Once an enumeration has been selected, the DDE_MBRS function is valid. DDE_MBRS returns two arrays: the first is an array of member names defined for this enumeration and the second is a list of corresponding values. The names are returned in the order defined in the enumeration, starting with either the first name or the name specified with *start*. DD_ENUM returns all the member names found or the number requested, whichever is smaller. The count of member names in the array can be returned in *#names*. You must ensure that the buffer passed is large enough to hold the number of names that you are requesting.

DD_EXIT – Terminate an information session

WT	WN	U	V

```
xcall DD_EXIT(dcs)
```

Arguments

dcs

The repository control structure.

Discussion

DD_EXIT closes the repository files and terminates the information session. It must be the last subroutine called when using the Repository information subroutines.

This subroutine closes the open file channels and clears the control structure.

DD_FIELD – Retrieve field information

WT	WN	U	V

`xcall DD_FIELD(`*dcs*`, DDF_LIST,` *names_req, array, [start][, #names]*`)`

or

`xcall DD_FIELD(`*dcs*`, DDF_SLIST,` *names_req, array, [start][, #names]*`)`

or

`xcall DD_FIELD(`*dcs*`, DDF_INFO,` *name, f_info*`)`

or

`xcall DD_FIELD(`*dcs*`, DDF_TEXT,` *field, data*`)`

or

`xcall DD_FIELD(`*dcs*`, DDF_GROUP,` *name*`)`

or

`xcall DD_FIELD(`*dcs*`, DDF_ENDGROUP)`

Arguments

dcs

The repository control structure.

DDF_LIST

Returns the current structure's field names in alphabetical order.

DDF_SLIST

Returns the current structure's field names in sequence order.

names_req

The number of field names requested. (**d3**)

array

Returned with the array of field names. ((*)**a30**)

start

(optional) Contains the field name at which to start. (**a30**)

#names

(optional) Returned with the number of field names. (**d3**)

DDF_INFO

Returns general field information and sets the current field.

name

The unique field name. (**a30**)

f_info

Returned with the field data. See the **f_info** record definition in the **ddinfo.def** file.

DDF_TEXT

Returns textual information about the current field.

field

A field in the **f_info** record that indicates what type of textual information should be returned in *data* (if the field is non-zero):

fi_struct	Referenced structure for implicit group. (**a30**)
fi_prefix	Group member prefix. (**a30**)
fi_desc	Short description. (**a40**)
fi_ldesc	Long description. (**a1800**)
fi_usrtyp	User data type string. (**a30**)
fi_enmfld	Enumeration name for Enum field. (**a30**)
fi_strfld	Structure (or alias) name for Struct field. (**a30**)
fi_heading	Column heading. (**a40**)
fi_prompt	Prompt text. (**a80**)
fi_help	Help identifier. (**a80**)
fi_infoln	Information string. (**a80**)
fi_utext	User text string. (**a80**)
fi_altnm	Alternate field name. (**a30**)
fi_font	Font. (**a30**)
fi_prmptfont	Prompt font. (**a30**)
fi_def	Default value. (**a80**)
fi_alwlst	Allow list entries. (See **fti_entlst** in **ddinfo.def**.)
fi_range	Range values. (See **fti_range** in **ddinfo.def**.)
fi_enum	Enumerated field data. (See **fti_enum** in **ddinfo.def**.)
fi_sellist	Selection list entries. (See **fti_entlst** in **ddinfo.def**.)

fi_arrivemeth	Arrive method. (**a30**)
fi_leavemeth	Leave method. (**a30**)
fi_drillmeth	Drill method. (**a30**)
fi_hypermeth	Hyperlink method. (**a30**)
fi_changemeth	Change method. (**a30**)
fi_dispmeth	Display method. (**a30**)
fi_editfmtmeth	Edit format method. (**a30**)

data

Returned with the requested textual data.

DDF_GROUP

Sets field context to the first member of the group *name*.

DDF_ENDGROUP

Sets field context back to the group member's parent field.

Discussion

The DD_FIELD subroutine returns information about fields for the current structure. There are six ways to call DD_FIELD:

- DDF_LIST and DDF_SLIST enable you to retrieve the structure's field names.
- DDF_INFO enables you to retrieve general information about a field.
- DDF_TEXT enables you to retrieve textual information about a field.
- DDF_GROUP and DDF_ENDGROUP enable you to establish group context.

You must have previously set the current structure with the DD_STRUCT subroutine. The same DD_STRUCT call should also have told you the number of fields that exist.

DDF_LIST and DDF_SLIST

If you pass DDF_LIST or DDF_SLIST, the DD_FIELD subroutine returns an array of field names for the current structure. If you pass DDF_LIST, the names are returned in alphabetical order, starting with either the first name found or the specified name. If you pass DDF_SLIST, the names are returned in sequence order (the order defined within the structure), starting with either the first name defined or the specified name.

DD_FIELD returns as many field names as are found or as are requested, whichever is smaller. The actual number of names in the array can be returned in *#names*. You must make sure that the buffer passed is large enough to hold the number of names that you are requesting.

To obtain a list of fields for an alias structure, use the DD_ALIAS subroutine.

When you pass DDF_LIST or DDF_SLIST, the DD_FIELD subroutine returns field names from the structure level only. To access the fields that are members of a group, you must first use DDF_INFO on each field and test the group flag (**fi_group**). If the group flag is set, then the field is a group. You must next determine if it is an explicit group or an implicit group (**fi_struct** is non-blank for implicit groups).

If the field is an explicit group, you must use DDF_GROUP to establish context for that group, and then use DDF_LIST or DDF_SLIST to access its members. This logic must be programmed in a recursive manner, as there is no limit to the number of nested groups.

If the field is an implicit group, you must copy the data from the current **dcs** into another control structure, and then pass it and **fi_struct** to the DD_STRUCT subroutine to establish context for obtaining the implicit group members. You would then use DD_FIELD to access its members, just as you did for the original structure. Remember that this structure can have implicit and explicit groups as well.

DDF_INFO

If you pass DDF_INFO, the DD_FIELD subroutine reads the specified field. If that field is not found, the relevant error code is returned in the control structure. If it is found, the field name is recorded in the control structure and general information is returned in *f_info*.

DDF_TEXT

Once a field has been selected, the DDF_TEXT function is valid. The DDF_TEXT function is used to obtain textual or variable-length information about the field. For each type of textual information, a corresponding field in the **f_info** record is non-zero. For example, if the **fi_desc** field in the **f_info** record is non-zero, a short description exists for the field. If you pass DDF_TEXT along with the non-zero field, the corresponding textual information is returned.

DDF_GROUP and DDF_ENDGROUP

If you pass DDF_GROUP, the DD_FIELD subroutine sets the context to group level *name* if *name* is an explicit group within the current level. (An explicit group is a field definition which has the group flag set, but does not reference another structure.) All subsequent calls to DD_FIELD will access fields at that group level until DD_FIELD is called with the DDF_ENDGROUP function.

If *name* is not found at the current level, the relevant error code is returned in the control structure. If it is found, but is not an explicit group, context will not be changed.

If you pass DDF_ENDGROUP, the DD_FIELD subroutine resets the context to the current level's parent. If the current level is the structure, the context is not changed.

DD_FILE – Retrieve file information

WT	WN	U	V

`xcall DD_FILE(`*dcs*`, DDL_INFO,` *name, fl_info*`)`

or

`xcall DD_FILE(`*dcs*`, DDL_TEXT,` *field, data*`)`

or

`xcall DD_FILE(`*dcs*`, DDL_STRS,` *names_req, array, [start], [#names][, array2]*`)`

Arguments

dcs

The repository control structure.

DDL_INFO

Returns general file information and sets the current file.

name

The unique file definition name. (**a30**)

fl_info

Returned with the file data (including file type, open filename, number of assigned structures, and name of the first assigned structure). See the **fl_info** record definition in the **ddinfo.def** file.

DDL_TEXT

Returns textual information about the current file.

field

A field in the **fl_info** record that indicates what type of textual information should be returned in *data* (if the field is non-zero):

fli_desc	Short description. (**a40**)
fli_ldesc	Long description. (**a1800**)
fli_utext	User-defined text string. (**a60**)
fli_portable	Portable integer specifications. (**a120**)
fli_filetext	File text. (**a1800**)

data

Returned with the requested textual data.

DDL_STRS

Returns a file's assigned structure names.

names_req

The number of assigned structure names requested. (**d3**)

array

Returned with the array of assigned structure names. ((*)**a30**)

start

(optional) Contains the assigned structure name at which to start. (**a30**)

#names

(optional) Returned with the number of assigned structure names. (**d3**)

array2

(optional) Returned with the array of corresponding ODBC table names. ((*)**a30**)

Discussion

The DD_FILE subroutine returns information about file definitions. There are three ways to call DD_FILE:

- DDL_INFO enables you to retrieve general information about a file.
- DDL_TEXT enables you to retrieve textual information about a file.
- DDL_STRS enables you to retrieve the file's assigned structure names.

DDL_INFO

If you pass DDL_INFO, the DD_FILE subroutine reads the specified file. If that file is not found, the relevant error code is returned in the control structure. If it is found, the file name is recorded in the control structure and general information is returned in *fl_info*.

Once a file has been selected, the other functions are valid.

DDL_TEXT

The DDL_TEXT function is used to obtain textual or variable-length information about the file. For each type of textual information, a corresponding field in the **fl_info** record is non-zero. For example, if the **fli_desc** field in the **fl_info** record is non-zero, a short description exists for the file. If you pass DDL_TEXT along with the non-zero field, the corresponding textual information is returned.

DDL_STRS

If you pass DDL_STRS, this subroutine returns an array of structure names assigned to this file. If only one structure is assigned, that name is returned in *fl_info* from the DDL_INFO function. The names are returned either in the order in which they were assigned, starting with the first one, or

starting with the specified name. DD_FILE returns as many structure names as are found or as are requested, whichever is smaller. The actual number of names in the array can be returned in *#names*. You must ensure that the buffer passed is large enough to hold the number of names that you are requesting.

DD_FILESPEC – Retrieve file specifications

WT	WN	U	V

`xcall DD_FILESPEC(`*dcs, name, [structure], fls_info, k_info[, …]*`)`

Arguments

dcs

The repository control structure.

name

The unique file definition name. (**a30**)

structure

(optional) Contains the name of a structure assigned to the file. (**a30**)

fls_info

Returned with the file specification data. See the **fls_info** record in the **ddinfo.def** file.

k_info

(optional) Returned with the key information for a maximum of 99 access keys. See the **k_info** record in the **ddinfo.def** file. (This can be a dimensioned argument. See below.)

Discussion

The DD_FILESPEC subroutine returns information related to the given file definition, with which the calling routine can create the file. If DD_FILESPEC can't find the specified file definition, the relevant error code is returned in the control structure.

If you don't specify the structure name, the name of the first structure assigned to the file is used. If no structures are assigned or if the specified structure is not assigned to the file, the relevant error code is returned in the control structure.

If both the file definition name and the assigned structure are found, the file specification information is returned in *fls_info*. This information includes the actual open filename, the file type, the structure name, the record size, and the number of keys.

The DD_FILESPEC subroutine returns information for a maximum of 99 access keys. The information for each key is returned in *k_info*. Optionally, you can pass one *k_info* argument which is declared as a real (bracketed) array. You must pass it to DD_FILESPEC without the brackets. To obtain textual information about each key, such as a description or null key value, use the DD_KEY subroutine.

DD_FILESPEC returns the keys in sequence number order and assumes that all access keys are defined before any foreign keys. DD_FILESPEC returns either as many keys as the number of *k_info* arguments passed to it (or the number of elements in the *k_info* array) or as many keys as it finds, depending on which number is smaller. The **fls_info** record contains the total number of access keys that are defined for the given file.

DD_FORMAT – Retrieve format information

WT	WN	U	V

`xcall DD_FORMAT(`*dcs*`, DDM_INFO,` *name, type, format*`)`

or

`xcall DD_FORMAT(`*dcs*`, DDM_LIST,` *names_req, array, [start][, #names]*`)`

or

`xcall DD_FORMAT(`*dcs*`, DDM_SINFO,` *sname, stype, sformat*`)`

Arguments

dcs

The repository control structure.

DDM_INFO

Returns general global or predefined format information.

name

The unique global or predefined format name. (**a30**)

type

Returned with the global format type. (**a1**)

format

Returned with the global format string. (**a30**)

DDM_LIST

Returns the current structure's format names.

names_req

The number of structure-specific format names requested. (**d4**)

array

Returned with the array of structure-specific format names. (**(*)a30**)

start

(optional) Contains the structure-specific format name at which to start. (**a30**)

#names

(optional) Returned with the number of structure-specific format names. (**d4**)

DDM_SINFO

Returns general structure-specific format information.

sname

The structure-specific format name. (**a30**)

stype

Returned with the structure-specific format type. (**a1**)

sformat

Returned with the structure-specific format string. (**a30**)

Discussion

The DD_FORMAT subroutine returns information about global, predefined, and structure-specific format definitions. (Predefined formats are the date and time formats.) There are three ways to call DD_FORMAT:

- DDM_INFO enables you to retrieve global or predefined information about a format.
- DDM_LIST enables you to retrieve a structure's format names.
- DDM_SINFO enables you to retrieve structure-specific format information.

You must have previously set the current structure with the DD_STRUCT subroutine to access information about structure-specific formats. You should know the number of structure-specific formats that exist from that same DD_STRUCT call.

DDM_INFO

If you pass DDM_INFO, this subroutine reads the specified global or predefined date or time format. (It searches for a predefined format first.) If that format is not found, the relevant error code is returned in the control structure. If it is found, information about the format is returned.

DDM_LIST

If you pass DDM_LIST, this subroutine returns an array of format names for the current structure. The names are returned in alphabetical order, starting with either the first name found or the specified name. DD_FORMAT returns either as many format names as it finds or as you request, depending on which is smaller. The actual number of names in the array can be returned in *#names*.

DDM_SINFO

If you pass DDM_SINFO, this subroutine reads the specified structure-specific format. If that format is not found, the relevant error code is returned in the control structure. If the format is found, information about the format is returned.

DD_INIT – Initialize an information session

WT	WN	U	V

`xcall DD_INIT(`*dcs, [main_file], [text_file], [main_open][, text_open]*`)`

Arguments

dcs

The repository control structure.

main_file

(optional) Contains the name of the repository main file to open. (**a255**)

text_file

(optional) Contains the name of the repository text file to open. (**a255**)

main_open

(optional) Returned with the name of the repository main file opened. (**a255**)

text_open

(optional) Returned with the name of the repository text file opened. (**a255**)

Discussion

The DD_INIT subroutine initializes an information session for a particular repository. It must be the first subroutine called when using the Repository subroutines. It initializes the repository control structure that is passed to it.

You can optionally specify the name of the repository main and text files to use. If the *main_file* and *text_file* arguments are not passed or are blank, DD_INIT opens the repository main and text files specified by the environment variables RPSMFIL and RPSTFIL. If these environment variables are not set, DD_INIT opens the repository files found in the RPSDAT directory (**rpsmain.ism** and **rpstext.ism**).

Once the files are found, their channel numbers are stored in the control structure. If *main_open* and *text_open* are passed, they are returned with the names of the opened files. If no files are found or if the specified files can't be opened, an error is returned in the control structure. One of three error codes (defined in **ddinfo.def**) is returned:

E_OPNERRM
E_OPNERRT
E_BADVERS

If the error code in *dcs* is non-zero after calling DD_INIT, the contents of *main_open* and *text_open* are undefined.

DD_KEY – Retrieve key information

WT	WN	U	V

`xcall DD_KEY(`*dcs*`, DDK_LIST, `*names_req, array, [start][, #names]*`)`

or

`xcall DD_KEY(`*dcs*`, DDK_SLIST, `*names_req, array, [start][, #names]*`)`

or

`xcall DD_KEY(`*dcs*`, DDK_INFO, `*name, k_info*`)`

or

`xcall DD_KEY(`*dcs*`, DDK_TEXT, `*field, data*`)`

Arguments

dcs

The repository control structure.

DDK_LIST

Returns the current structure's key names in alphabetical order.

DDK_SLIST

Returns the current structure's key names in sequence order.

names_req

The number of key names requested. (**d2**)

array

Returned with the array of key names. (**(*)a30**)

start

(optional) Contains the key name at which to start. (**a30**)

#names

(optional) Returned with the number of key names. (**d2**)

DDK_INFO

Returns general key information.

name

The unique key name. (**a30**)

k_info

Returned with the key data. See the **k_info** record definition in the **ddinfo.def** file.

DDK_TEXT

Returns textual information about the current key.

field

A field in the **k_info** record that indicates what type of textual information should be returned in *data* (if the field is non-zero):

ki_desc	Short description. (**a40**)
ki_nullval	Null key value. (**a255**)

data

Returned with the requested textual data.

Discussion

The DD_KEY subroutine returns information about keys for the current structure. There are four ways to call DD_KEY:

- DDK_LIST and DDK_SLIST enable you to retrieve the structure's key names.
- DDK_INFO enables you to retrieve general information about a key.
- DDK_TEXT enables you to retrieve textual information about a key.

You must have previously set the current structure with the DD_STRUCT subroutine. The same DD_STRUCT call should also have told you the number of keys that exist.

DDK_LIST and DDK_SLIST

If you pass DDK_LIST or DDK_SLIST, the DD_KEY subroutine returns an array of key names for the current structure. If you pass DDK_LIST, the names are returned in alphabetical order, starting with either the first name found or the specified name. If you pass DDK_SLIST, the names are returned in sequence order (the order defined in the structure), starting with either the first name found or the specified name. DD_KEY returns as many key names as are found or as are requested, whichever is smaller. The actual number of names in the array can be returned in *#names*.

You must make sure that the buffer passed is large enough to hold the number of names that you are requesting.

DDK_INFO

If you pass DDK_INFO, this subroutine reads the specified key. If that key is not found, the relevant error code is returned in the control structure. If it is found, general key information is returned in *k_info*.

DDK_TEXT

Once a key has been selected, the DDK_TEXT function is then valid. The DDK_TEXT function is used to obtain textual or variable-length information about the key. For each type of textual information, a corresponding field in the **k_info** record is non-zero. For example, if the **ki_desc** field in the **k_info** record is non-zero, a short description exists for the key. If you pass DDK_TEXT along with this non-zero field, the corresponding textual information is returned.

DD_NAME – Retrieve a list of definition names

WT	WN	U	V

`xcall DD_NAME` (*dcs*, `DDN_COUNT`, *c_id, count*)

or

`xcall DD_NAME` (*dcs*, `DDN_LIST`, *l_id, names_req, array, [start][, #names]*)

Arguments

dcs

The repository control structure.

DDN_COUNT

Returns the count for a given definition type.

c_id

One of the following functions:

DDN_STRUCT	Returns the count of structure definitions.
DDN_FILE	Returns the count of file definitions.
DDN_TEMPLATE	Returns the count of template definitions.
DDN_ENUM	Returns the count of enumeration definitions.
DDN_FMT	Returns the count of global format definitions.
DDN_DFMT	Returns the count of predefined date formats.
DDN_TFMT	Returns the count of predefined time formats.

count

Returned with the item count. (**d4**)

DDN_LIST

Returns a list of definition names.

l_id

One of the following functions:

DDN_STRUCT	Returns structure definition names.
DDN_FILE	Returns file definition names.
DDN_TEMPLATE	Returns template definition names.
DDN_ENUM	Returns enumeration definition names.

DDN_FMT Returns global format definition names.

DDN_DFMT Returns predefined date format definition names.

DDN_TFMT Returns predefined time format definition names.

names_req

The number of names requested. (**d4**)

array

Returned with the array of names. (**(*)a30**)

start

(optional) Contains the name at which to start. (**a30**)

#names

(optional) Returned with the number of names. (**d4**)

Discussion

The DD_NAME subroutine enables you to find out how many definitions of a given definition type exist, and then ask for a list of their names. There are two ways to call DD_NAME:

- DDN_COUNT enables you to retrieve the count for a definition type.
- DDN_LIST enables you to retrieve a list of definition names.

DDN_COUNT

If you pass DDN_COUNT, the DD_NAME subroutine returns the number of definitions that exist for the specified type.

DDN_LIST

If you pass DDN_LIST, DD_NAME returns an array of definition names for the requested type. You can retrieve as many names as you want, and you can specify the name at which to start. These names are returned in alphabetical order, starting with either the first name found or the specified name. DD_NAME returns as many names as are found or as are requested, whichever is smaller. The actual number of names in the array can be returned in *#names*.

You must make sure that the buffer passed is large enough to hold the number of names that you are requesting.

DD_RELATION – Retrieve relation information

WT	WN	U	V

`xcall DD_RELATION(`*dcs*`, DDR_LIST,` *names_req, array, [start][, #names]*`)`

or

`xcall DD_RELATION(`*dcs*`, DDR_INFO,` *name, from_key, to_struct, to_key*`)`

Arguments

dcs

The repository control structure.

DDR_LIST

Returns the current structure's relation names.

names_req

The number of relation names requested. (**d2**)

array

Returned with the array of relation names. (**(*)a30**)

start

(optional) Contains the relation name at which to start. (**a30**)

#names

(optional) Returned with the number of relation names. (**d2**)

DDR_INFO

Returns general relation information.

name

The unique relation name. (**a30**)

from_key

Returned with the name of the relation's "from" key. (**a30**)

to_struct

Returned with the name of the relation's "to" structure. (**a30**)

to_key

Returned with the name of the relation's "to" key. (**a30**)

Discussion

The DD_RELATION subroutine returns information about relations for the current structure. There are two ways to call DD_RELATION:

- DDR_LIST enables you to retrieve the structure's relation names.
- DDR_INFO enables you to retrieve general information about a relation.

You must have previously set the current structure with the DD_STRUCT subroutine. The same DD_STRUCT call should also have told you the number of relations that exist.

DDR_LIST

If you pass DDR_LIST, the DD_RELATION subroutine returns an array of relation names for the current structure. The names are returned in alphabetical order, starting with either the first name found or the specified name. DD_RELATION returns as many relation names as are found or as are requested, whichever is smaller. The actual number of names in the array can be returned in *#names*.

You must ensure that the buffer passed is large enough to hold the number of names you are requesting.

DDR_INFO

If you pass DDR_INFO, this subroutine reads the specified relation. If that relation is not found, the relevant error code is returned in the control structure. If it is found, general relation information is returned.

DD_STRUCT – Retrieve structure information

WT	WN	U	V

`xcall DD_STRUCT(`*dcs,* `DDS_INFO,` *name, s_info[, s_name]*`)`

or

`xcall DD_STRUCT(`*dcs,* `DDS_TEXT,` *field, data*`)`

or

`xcall DD_STRUCT(`*dcs,* `DDS_FILS,` *names_req, array, [start][, #names]*`)`

Arguments

dcs

The repository control structure.

DDS_INFO

Returns general structure information and sets the current structure.

name

The unique structure name. It can be the name of an alias structure. (**a30**)

s_info

Returned with the structure data (including file type, record size, number of fields, keys, relations, formats, number of files to which it is assigned, name of the first file to which it is assigned, and structure tag information). See the **s_info** record definition in **ddinfo.def**.

s_name

(optional) Returned with the name of the aliased structure, if *name* is an alias structure name.

DDS_TEXT

Returns textual information about the current structure.

field

A field in the **s_info** record that indicates what type of textual information should be returned in *data* (if the field is non-zero):

si_desc	Short description. (**a40**)
si_ldesc	Long description. (**a1800**)
si_utext	User-defined text string. (**a60**)

data

Returned with the requested textual data.

DDS_FILS

Returns the list of files to which a structure is assigned.

names_req

The number of assigned file definition names requested. (**d3**)

array

Returned with the array of assigned file definition names. (**(*)a30**)

start

(optional) Contains the assigned file definition name at which to start. (**a30**)

#names

(optional) Returned with the number of assigned file definition names. (**d3**)

Discussion

The DD_STRUCT subroutine returns information about structures. It also sets the current structure. There are three ways to call DD_STRUCT:

- DDS_INFO enables you to retrieve general information about a structure.
- DDS_TEXT enables you to retrieve textual information about a structure.
- DDS_FILS enables you to retrieve a structure's list of assigned files.

DDS_INFO

If you pass DDS_INFO, the DD_STRUCT subroutine reads the specified structure. If that structure is not found, an error code is returned in the control structure. If it is found, information about the structure is recorded in the control structure and general information is returned in *s_info*.

The s_info record returns the total number of fields in the structure (including any groups) in si_nmflds. However, because si_nmflds is only a **d3**, if the field count is equal to or greater than 999, si_nmflds is set to 999, and your application must iterate through the structure and its groups to calculate the total field count using si_childct and fi_childct (which are what Repository uses internally to calculate the total).

The structure can be an alias structure, in which case the name of the aliased structure can be returned in *s_name*. Also, the structure name recorded in the control structure (the current structure) will be the name of the aliased structure.

Once you set a current structure with the DDS_INFO function, you can then use other subroutines to access information about the fields, keys, relations, formats, and tag associated with that structure. The DDS_TEXT and DDS_FILS functions also become valid.

DDS_TEXT

The DDS_TEXT function is used to obtain textual or variable-length information about the structure. For each type of textual information, a corresponding field in the **s_info** record is non-zero. For example, if the **si_desc** field in the **s_info** record is non-zero, a short description exists for the structure. If you pass DDS_TEXT along with the non-zero field, the corresponding textual information is returned.

DDS_FILS

If you pass DDS_FILS, this subroutine returns an array of file definition names to which this structure is assigned. If this structure is assigned to only one file, that file definition name is returned in *s_info*. The names are returned in the order in which the structure was assigned to them, starting with either the first one or the specified name. DD_STRUCT returns either as many file definition names as it finds or as you request, whichever is smaller. The actual number of names in the array can be returned in *#names*. You must ensure that the buffer passed is large enough to hold the number of names you are requesting.

DD_TAG – Retrieve tag information

WT	WN	U	V

`xcall DD_TAG(`*dcs*`, DDTG_LIST,` *names_req, array, [start][, #names]*`)`

or

`xcall DD_TAG(`*dcs*`, DDTG_INFO,` *name, tg_info*`)`

Arguments

dcs

The repository control structure.

DDTG_LIST

Returns the current structure's tag names.

names_req

The number of tag names requested. (**d2**)

array

Returned with the array of tag names. ((*****)**a30**)

start

(optional) Contains the tag name at which to start. (**a30**)

#names

(optional) Returned with the number of tag names. (**d2**)

DDTG_INFO

Returns general tag information.

name

The unique tag name. (**a30**)

tg_info

Returned with the tag data. See the **tg_info** record definition in the **ddinfo.def** file.

Discussion

The DD_TAG subroutine returns information about tags for the current structure. There are two ways to call DD_TAG:

- DDTG_LIST enables you to retrieve the structure's tag names.
- DDTG_INFO enables you to retrieve general information about a tag.

You must have previously set the current structure with the DD_STRUCT subroutine. The same DD_STRUCT call should also have told you the number of tags that exist.

DDTG_LIST

If you pass DDTG_LIST, the DD_TAG subroutine returns an array of tag names for the current structure. The names are returned in alphabetical order, starting with either the first name found or the specified name. DD_TAG returns as many tag names as are found or as are requested, whichever is smaller. The actual number of names in the array can be returned in *#names*.

You must ensure that the buffer passed is large enough to hold the number of names you are requesting.

DDTG_INFO

If you pass DDTG_INFO, this subroutine reads the specified tag. If that tag is not found, the relevant error code is returned in the control structure. If it is found, general tag information is returned in *tg_info*.

DD_TEMPLATE – Retrieve template information

WT	WN	U	V

`xcall DD_TEMPLATE(`*dcs*`, DDT_INFO,` *name, t_info*`)`

or

`xcall DD_TEMPLATE(`*dcs*`, DDT_TEXT,` *field, data*`)`

Arguments

dcs

The repository control structure.

DDT_INFO

Returns general template information and sets the current template.

name

The unique template name. (**a30**)

t_info

Returned with the template data. See the **t_info** record definition in **ddinfo.def**.

DDT_TEXT

Returns textual information about the current template.

field

A field in the **t_info** record that indicates what type of textual information should be returned in *data* (if the field is non-zero):

ti_desc	Short description. (**a40**)
ti_ldesc	Long description. (**a1800**)
ti_usrtyp	User data type string. (**a30**)
ti_heading	Column heading. (**a40**)
ti_prompt	Prompt text. (**a80**)
ti_help	Help identifier. (**a80**)
ti_infoln	Information string. (**a80**)
ti_utext	User text string. (**a80**)
ti_altnm	Alternate field name. (**a30**)
ti_font	Font. (**a30)**

ti_prmptfont	Prompt font. (**a30)**
ti_def	Default value. (**a80**)
ti_alwlst	Allow list entries. (See **fti_entlst** in **ddinfo.def**.)
ti_range	Range values. (See **fti_range** in **ddinfo.def**.)
ti_enum	Enumerated template data. (See **fti_enum** in **ddinfo.def**.)
ti_sellist	Selection list entries. (See **fti_entlst** in **ddinfo.def**.)
ti_arrivemeth	Arrive method. (**a30**)
ti_leavemeth	Leave method. (**a30**)
ti_drillmeth	Drill method. (**a30**)
ti_hypermeth	Hyperlink method. (**a30**)
ti_changemeth	Change method. (**a30**)
ti_dispmeth	Display method. (**a30**)
ti_editfmtmeth	Edit format method. (**a30**)

data

Returned with the requested textual data.

Discussion

The DD_TEMPLATE subroutine returns information about template definitions. There are two ways to call DD_TEMPLATE:

- DDT_INFO enables you to retrieve general information about a template.
- DDT_TEXT enables you to retrieve textual information about a template.

DDT_INFO

If you pass DDT_INFO, the DD_TEMPLATE subroutine reads the specified template. If that template is not found, the relevant error code is returned in the control structure. If it is found, the template name is recorded in the control structure and general information is returned in *t_info*.

DDT_TEXT

Once a template has been selected, the DDT_TEXT function is then valid. The DDT_TEXT function is used to obtain textual or variable-length information about the template. For each type of textual information, a corresponding field in the **t_info** record is non-zero. For example, if the **ti_desc** field in the **t_info** record is non-zero, a short description exists for the template. If you pass DDT_TEXT along with the non-zero field, the corresponding textual information is returned.

Sample programs

The program below uses the Repository subroutine library to display a selection window. Program 2 on page 7-43 illustrates how to traverse fields and groups in a structure.

Program 1

```
; This program displays a selection window containing a list of all
; structures in a repository. From that window, the user can then select
; a structure and a second selection window displays, containing a list
; of all fields in that structure.

; Script for Repository information subroutine demo

.column c_select,          "Select"
.entry o_exit,             "Exit",             key(f4)
.entry o_nxtpg,            "Next page",        key(next)
.entry o_prvpg,            "Previous page",    key(prev)
.entry s_up,               "Move up",          key(up)
.entry s_down,             "Move down",        key(down)
.end

; dddemo.dbl - Demo Repository information subroutines

.include "RPSLIB:ddinfo.def"     ;.defines and data structures
.include "WND:tools.def"
.include "WND:windows.def"
.define SELWND_SIZE ,10          ;# rows in selection window
.define MAX_PAGE    ,98          ;Max pages in selection window

record
  struct            ,a30                ;A structure name
  structs           ,SELWND_SIZE a30    ;Structure names
  sinfo             ,SELWND_SIZE a62    ;Selection window items
  st_base           ,MAX_PAGE a30       ;Base struct for selection page
  field             ,a30                ;A field name
  fields            ,SELWND_SIZE a30    ;Field names
  finfo             ,SELWND_SIZE a49    ;Selection window items
  fld_base          ,MAX_PAGE a30       ;Base field for selection page
  colid             ,i4                 ;Selection window column ID
  st_id             ,i4                 ;Structure selection window ID
  fld_id            ,i4                 ;Field selection window id
                                        ;Selection window titles
  nmstrcts          ,d5                 ;Number of structures
  nmflds            ,d5                 ;Number of fields
  ret               ,d2                 ;Number of items retrieved
  sx                ,d3                 ;Structure index
  fx                ,d3                 ;Field index
```

```
record st_title
                    ,a*,' Structures - Page '
  st_page           ,d2
                    ,a*,' of '
  st_last           ,d2

record st_info
  st_name           ,a30          ;Structure name
                    ,a4
  st_filtyp         ,a15          ;File type
                    ,a3
  st_desc           ,a10          ;First 10 chars of short description

record fld_title
                    ,a*,' Fields in '
  fld_sname         ,a30
                    ,a*,' - Page '
  fld_page          ,d2
                    ,a*,' of '
  fld_last          ,d2

record fld_info
  fld_name          ,a30          ;Field name
                    ,a2
  fld_type          ,a1           ;Field type
  fld_size          ,a4           ;Field size
                    ,a2
  fld_desc          ,a10          ;First 10 chars of short description

proc
  xcall u_start("dddemo")         ;Start UI Toolkit
  xcall m_ldcol(colid, g_utlib, "c_select")
                                  ;Load selection column
  xcall dd_init(dcs)              ;Start repository routines
  if (error)                      ;Check error state in dcs
    call error
                                  ;Get count of structures
  xcall dd_name(dcs, DDN_COUNT, DDN_STRUCT, nmstrcts)
  if (error)
    call error
  st_page = 1                    ;Start at the beginning (novel)
  st_last = nmstrcts/SELWND_SIZE ;Compute last page
  if (st_last*SELWND_SIZE .lt. nmstrcts)
    incr st_last
  call load_structs               ;Load a selection page
  do
    call process_structs          ;Process selection window
```

```
  until (g_select)                ;Unsatisfied menu entry = Exit
  xcall dd_exit(dcs)              ;Shut down repository routines
  xcall u_finish                  ;Shut down UI Toolkit
  stop
;
; Description: Load the selection window for structures
;
load_structs,
                                  ;Get a page full of names
  xcall dd_name(dcs, DDN_LIST, DDN_STRUCT, SELWND_SIZE,
  &             structs, st_base(st_page), ret)
  if (error)                      ;Check error state
    call error
  for sx from 1 thru ret          ;Get info about each structure
    begin
      st_name = structs(sx)
      xcall dd_struct(dcs, DDS_INFO, st_name, s_info)
      if (error)
        call error
      st_filtyp = si_filtyp       ;Load the file type
      if (si_desc) then           ;Is there a short description?
        begin
          xcall dd_struct(dcs, DDS_TEXT, si_desc, st_desc)
          if (error)
            call error
        end
      else                        ;No short description,
        if (si_ldesc) then        ;Is there a long description?
          begin
            xcall dd_struct(dcs, DDS_TEXT, si_ldesc, st_desc)
            if (error)
              call error
          end
        else                      ;No short or long description,
          clear st_desc           ; clear it
      sinfo(sx) = st_info         ;Load array for the selection wnd
    end
  sx = 1                          ;Start with the first one
  if (st_id)
    xcall u_window (D_DELETE, st_id)    ;Delete any previous
                                  ; version and build the window
  xcall s_selbld(st_id, "STRUCTS", ret, ret, sinfo)
                                  ;Put the title on it
  xcall w_brdr(st_id, WB_TITLE, st_title, WB_TPOS, WBT_TOP,
  &            WBT_CENTER)
  xcall u_logwnd(st_id)           ;Log it with UI Toolkit
  xcall u_window(D_PLACE, st_id, 3, 10) ;Place it at 3,10
  return
```

```
;
; Description: Process the structure selection window
;
process_structs,
  xcall s_select(st_id, sx, struct,, sx);Let user select one
  if (g_select) then                    ;If user chose a menu entry
    begin
      case g_entnam of
       begincase
        'O_EXIT ':
          return                        ;Exit
        'O_NXTPG ':
          call next_struct_page         ;Load next page
        'O_PRVPG ':
         call prev_struct_page          ;Load previous page
       endcase
    end                                 ;Note that any other menu entry
                                        ; returns as well
  else
    begin                               ;Select the structure
      xcall dd_struct(dcs, DDS_INFO, struct, s_info)
      if (error)
        call error
      nmflds = si_nmflds                ;Load number of fields
      fld_page = 1                      ;Start at the first page
      clear fld_base(1)
      fld_last = nmflds/SELWND_SIZE     ;Compute last page
      if (fld_last*SELWND_SIZE .lt. nmflds)
        incr fld_last
      fld_sname = struct                ;Load structure name in title
      call load_fields                  ;Load a selection window page
      do
        call process_fields             ;Process selection window
      until (g_select)                  ;Until unsatisfied menu entry
      if (g_entnam .eq. "O_EXIT ")      ;Exit only one level
        clear g_select
      xcall u_window(D_DELETE, fld_id)  ;Delete fields window
    end
  return

;
; Description: Go to the next page of structures
;
next_struct_page,
  if (st_page .ge. st_last) then              ;Check for overflow
    call ding
```

```
  else
    begin
      incr st_page
      st_base(st_page) = structs(SELWND_SIZE   ;Start w/ last
                                               ; structure on prev page
      call load_structs                        ;Load the window
    end
  clear g_select                               ;Menu entry satisfied
  return

;
; Description: Go to the previous page of structures
;
prev_struct_page,
  if (st_page .le. 1) then                     ;Avoid underflow
    call ding
  else
    begin
      decr st_page
      call load_structs                        ;Load the window
    end
  clear g_select                               ;Menu entry satisfied
  return

;
; Description: Load a page of fields into a selection window
;
load_fields,
                                               ;Load a page of field
names
  xcall dd_field(dcs, DDF_LIST, SELWND_SIZE, fields,
  &              fld_base(fld_page), ret)
  if (error)
    call error
  for fx from 1 thru ret                   ;For each field loaded
    begin
      fld_name = fields(fx)                ;Get field information
      xcall dd_field(dcs, DDF_INFO, fld_name, f_info)
      if (error)
        call error
      fld_type = fi_type
      fld_size = fi_size [left]
      if (fi_desc) then                    ;Is there a short description?
        begin
          xcall dd_field(dcs, DDF_TEXT, fi_desc, fld_desc)
          if (error)
            call error
        end
```

```
      else if (fi_ldesc) then           ;No, is there a long description?
        begin
          xcall dd_field(dcs, DDF_TEXT, fi_ldesc, fld_desc)
          if (error)
            call error
        end
      else
        clear fld_desc
    finfo(fx) = fld_info                ;Load selection window array
    end
  fx = 1                                ;So we start with the first one
  if (fld_id)
    xcall u_window (D_DELETE, fld_id)   ;Delete any prev version
                                        ;and build the window
  xcall s_selbld(fld_id, "FIELDS", ret, ret, finfo)
                                        ;Put the title on it
  xcall w_brdr(fld_id, WB_TITLE, fld_title, WB_TPOS, WBT_TOP,
  &            WBT_CENTER)
  xcall u_logwnd(fld_id)                ;Log it with UI Toolkit
  xcall u_window(D_PLACE, fld_id, 5, 20);Place it at 5,20
  return

;
; Description: Process the field selection window
;
process_fields,
  xcall s_select(fld_id, fx, field,, fx);Let user select one
  if (g_select) then                    ;If user chose a menu entry
    begin
      case g_entnam of
       begincase
        'O_EXIT ':
          return                        ;Exit
        'O_NXTPG ':
          call next_field_page          ;Load next page
        'O_PRVPG ':
          call prev_field_page          ;Load previous page
       endcase
    end                                 ;Note that any other menu entry
                                        ; returns also
  else
    begin                               ;Select the structure
      xcall dd_field(dcs, DDF_INFO, field, f_info)
      if (error)
        call error
    end
  return
```

```
;
; Description: Go to the next page of fields
;
next_field_page,
  if (fld_page .ge. fld_last) then      ;Check for overflow
    call ding
  else
    begin
      incr fld_page
      fld_base(fld_page) = fields(SELWND_SIZE)
      call load_fields                  ;Load the window
    end
  clear g_select                        ;Menu entry satisfied
  return

;
; Description: Go to the previous page of fields
;
prev_field_page,
  if (fld_page .le. 1) then             ;Avoid underflow
    call ding
  else
    begin
      decr fld_page
      call load_fields                  ;Load the window
    end
  clear g_select                        ;Menu entry satisfied
  return

;
; Description: Abort on a repository access error. This
; routine is called to provide a full traceback of where
; the error occurred.
;
error,
  xcall u_abort("Error in Repository info routines", %a(error))

;
; Description: Ring the terminal bell
;
ding,
  display (g_terminal, 7)
  return
.end
```

Program 2

```
; This program traverses all fields and groups in a structure. It assumes
; all groups are explicit groups.
;
.include "RPSLIB:ddinfo.def"

.define MAX_FLDS      ,99
.define MAX_LVLS      ,10
.define PUTOUT(msg)   writes(chan, (msg))
.define ERROUT(msg)   PUTOUT("Error # " + %string(error) + " " + (msg))

common
  chan    ,i4

proc
  xcall u_start
  xcall u_open(chan, "o:s", "TST:output.txt")

  xcall dd_init(dcs, "TST:testmain.ism", "TST:testtext.ism")
  if (error)
    begin
      xcall u_message("Cannot open repository file due to error " +
  &                    %string(error))
      xcall u_finish
      stop
    end

  xcall dd_struct(dcs, DDS_INFO, "COMPANY", s_info)
  if (error) then
    ERROUT("Cannot load COMPANY")
  else
    PUTOUT("Structure COMPANY")

  ; Traverse the structure
  xcall check_level(dcs)
  xcall u_close(chan)
  xcall u_finish
  stop
.end

subroutine check_level,  reentrant, stack
  a_dcs        ,a

.include "RPSLIB:ddinfo.def"

common
  chan         ,i4
```

```
record clear_a
  ix            ,d4               ;Loop index
  num_fields    ,d3               ;Number of fields returned
  field_names   ,MAX_FLDS a30     ;Array of field names
  name          ,a30              ;Current name for optimization

static record
  level         ,i4               ;Group level (1 = main structure)

proc
  clear clear_a
  dcs = a_dcs
  incr level

  xcall dd_field(dcs, DDF_SLIST, MAX_FLDS, field_names(1),, num_fields)
  if (error)
    ERROUT("Cannot load level " + %string(level))

  for ix from 1 thru num_fields
    begin
      name = field_names(ix)
      xcall dd_field(dcs, DDF_INFO, name, f_info)
      if (fi_group) then
        begin
          PUTOUT("Group " + (name))
          xcall dd_field(dcs, DDF_GROUP, name)
          xcall check_level(dcs)               ;Recurse to its members
          xcall dd_field(dcs, DDF_ENDGROUP)
          PUTOUT("Endgroup")
        end
      else
          PUTOUT("Field " + (name))
    end
  decr level
  a_dcs = dcs
  xreturn
endsubroutine
```

Definition file

The information below is included in the definition file **RPSLIB:ddinfo.def**, which you must .INCLUDE in your Synergy DBL source code in order to use the Repository information subroutines. This file contains .DEFINEs for the various function codes and the data structures returned from the routines.

See "The ddinfo.def file" on page 7-2 for details on using this file.

```
.ifndef DDF_LIST        ; Prevent duplicate definition in RW and Composer

.ifndef DBLV5
  .include "DBLDIR:dbl.def"
.endc

; Defines for XCALL DD_ALIAS

  .define DDA_INFO        ,1              ; General alias information
  .define DDA_SLIST       ,2              ; List of alias/aliased fields

; Defines for XCALL DD_CONTROL

  .define DDC_INFO        ,1              ; Control record information

; Defines for XCALL DD_ENUM

  .define DDE_INFO        ,1              ; General enumration information
  .define DDE_MBRS        ,2              ; List of member names/values
  .define DDE_TEXT        ,3              ; Textual information

; Defines for XCALL DD_FIELD

  .define DDF_LIST        ,1              ; List of field names
  .define DDF_INFO        ,2              ; General field information
  .define DDF_TEXT        ,3              ; Textual information
  .define DDF_SLIST       ,4              ; List of field names
                                          ;  (in sequence order)
  .define DDF_FORMAT      ,5              ; Field format
  .define DDF_GROUP       ,6              ; Set current group context
  .define DDF_ENDGROUP    ,7              ; End current group context

  .define DDF_NSGET       ,8              ; Get namespace(s) for fields in
                                          ;  structure
  .define DDF_NSINFO      ,9              ; Get field info using namespace
  .define DDF_NSREL       ,10             ; Release namespace(s) and
                                          ;  associated data

; Defines for XCALL DD_FILE

  .define DDL_INFO        ,1              ; General file information
  .define DDL_STRS        ,2              ; List of assigned structures
  .define DDL_LIST        ,2              ; (For compatibility with v2)
  .define DDL_TEXT        ,3              ; Textual information
```

```
; Defines for XCALL DD_FORMAT

  .define DDM_INFO        ,1              ; General format information
  .define DDM_LIST        ,2              ; List of (local) format names
  .define DDM_SINFO       ,3              ; General (local) format info

; Defines for XCALL DD_KEY

  .define DDK_LIST        ,1              ; List of key names
  .define DDK_INFO        ,2              ; General key information
  .define DDK_TEXT        ,3              ; Textual information
  .define DDK_SLIST       ,4              ; List of key names
                                          ;  (in sequence order)
; Defines for XCALL DD_NAME

  .define DDN_COUNT       ,1              ; Definition count
  .define DDN_LIST        ,2              ; Definition list

  .define DDN_STRUCT      ,1              ; Structure
  .define DDN_FILE        ,2              ; File
  .define DDN_TEMPLATE    ,3              ; Template
  .define DDN_FMT         ,4              ; Format
  .define DDN_DFMT        ,5              ; Pre-defined Date Format
  .define DDN_TFMT        ,6              ; Pre-defined Time Format
  .define DDN_ENUM        ,7              ; Enumeration

; Defines for XCALL DD_RELATION

  .define DDR_LIST        ,1              ; List of relation names
  .define DDR_INFO        ,2              ; General relation information

; Defines for XCALL DD_STRUCT

  .define DDS_INFO        ,1              ; General structure info
  .define DDS_FILS        ,2              ; List of files assigned to
  .define DDS_LIST        ,2              ; (For compatibility with v2)
  .define DDS_TEXT        ,3              ; Textual information

; Defines for XCALL DD_TAG

  .define DDTG_LIST       ,1              ; List of tag names
  .define DDTG_INFO       ,2              ; General tag information

; Defines for XCALL DD_TEMPLATE

  .define DDT_INFO        ,1              ; General template information
  .define DDT_TEXT        ,2              ; Textual information

; Possible error codes returned in the control structure

  .define E_OK      ,0    ; No error
  .define E_NOFIND  ,1    ; Record not found
```

```
  .define E_OPNERR  ,2    ; Cannot open file
  .define E_INVFNC  ,3    ; Invalid function
  .define E_OPNERRM ,4    ; Cannot open main repository file
  .define E_OPNERRT ,5    ; Cannot open repository text file
  .define E_BADVERS ,6    ; Incompatible repository version
  .define E_OLDDCS  ,7    ; Old version of dcs structure

; Defines for Data type fields (fi_type or ti_type)

  .define T_ALP   ,'A'    ; fi_type or ti_type = ALPHA
  .define T_DEC   ,'D'    ; fi_type or ti_type = DECIMAL
  .define T_INT   ,'I'    ; fi_type or ti_type = INTEGER
  .define T_USR   ,'U'    ; fi_type or ti_type = USER DEFINED

; Defines for Data type subclass (fi_class or ti_class)

; If fi_type = T_DEC, and fi_class equals one of the following,
; OR, ti_type = T_DEC, and ti_class equals one of the following,
; then it's a DATE or TIME type.

  .define C_YYMMDD        ,1
  .define C_YYYYMMDD      ,2
  .define C_YYJJJ         ,3
  .define C_YYYYJJJ       ,4
  .define C_YYPP          ,5
  .define C_YYYYPP        ,6

  .define C_HHMMSS        ,8
  .define C_HHMM          ,9

; If fi_type = T_ALP, then fi_class defines the subtype, OR,
; if ti_type = T_ALP, then ti_class defines the subtype.

  .define C_BINARY        ,1
  .define C_STRFLD        ,2  ; For Fields only, not Templates

; If fi_type = T_INT, then fi_class defines the subtype, OR,
; if ti_type = T_INT, then ti_class defines the subtype.

  .define C_BOOLEAN       ,1
  .define C_ENUM          ,2
  .define C_AUTOSEQ       ,3
  .define C_AUTOTIME      ,4

; If fi_type = T_USR, then fi_class defines the subtype, OR,
; if ti_type = T_USR, then ti_class defines the subtype.

  .define C_ALPHA         ,0
  .define C_NUMERIC       ,1
  .define C_DATE          ,2
  .define C_UBINARY       ,3
```

```
; If fi_type = T_DEC, T_INT, T_USR, fi_coertype defines the
; coerced type, OR, if ti_type = T_DEC, T_INT, or T_USR, then
; ti_coertype defines the coerced type.

  .define CT_DEFAULT      ,0
  .define CT_BYTE         ,1
  .define CT_SHORT        ,2
  .define CT_INT          ,3
  .define CT_LONG         ,4
  .define CT_SBYTE        ,5
  .define CT_USHORT       ,6
  .define CT_UINT         ,7
  .define CT_ULONG        ,8
  .define CT_BOOLEAN      ,9
  .define CT_DEC          ,10
  .define CT_NULLDEC      ,11
  .define CT_AUTOTIME     ,12     ; only used by genxml

;        .define CT_DEFAULT      ,0     ; Already defined above
  .define CT_DOUBLE       ,1
  .define CT_FLOAT        ,2
  .define CT_NULLDECIMAL  ,3
  .define CT_DECIMAL      ,4

  .define CT_DATETIME     ,0
  .define CT_NULLDATETIME ,1

; Defines for view flags.  NOTE THAT 0 = YES, 1 = NO.

  .define FI_VW           ,0      ; fi_dblvw, fi_rptvw, fi_scrptvw, fi_webvw = YES
  .define FI_NVW          ,1      ; fi_dblvw, fi_rptvw, fi_scrptvw, fi_webvw = NO

; Defines for selection window types

  .define FI_SELWND       ,1      ; fi_seltyp = Name of window supplied
  .define FI_SELLST       ,2      ; fi_seltyp = List of entries supplied

; Defines for Negative value allowed?

  .define FI_NONEG        ,0      ; fi_negalw = NO
  .define FI_NEG          ,1      ; fi_negalw = YES
  .define FI_NEGONLY      ,2      ; fi_negalw = ONLY
  .define FI_NEGORZERO    ,3      ; fi_negalw = ORZERO

; Defines for Break value

  .define FI_NOBRK        ,0      ; fi_break = NO
  .define FI_BRK          ,1      ; fi_break = YES
  .define FI_BRKALL       ,2      ; fi_break = ALWAYS
  .define FI_BRKRET       ,3      ; fi_break = RETURN
```

```
; Defines for Null allowed

  .define FI_NULLDFLT     ,0      ; fi_null = DEFAULT
  .define FI_NULLALW      ,1      ; fi_null = YES
  .define FI_NONULL       ,2      ; fi_null = NO

; Defines for Default action

  .define FI_NONE         ,0      ; fi_defact = NONE
  .define FI_DEFAULT      ,1      ; fi_defact = DEFAULT
  .define FI_COPY         ,2      ; fi_defact = COPY
  .define FI_INCR         ,3      ; fi_defact = INCREMENT
  .define FI_DECR         ,4      ; fi_defact = DECREMENT

; Defines for justification

  .define FI_LEFT         ,0      ; fi_inpjust = LEFT
  .define FI_RIGHT        ,1      ; fi_inpjust = RIGHT
  .define FI_CENTER       ,2      ; fi_inpjust = CENTER

; Defines for input field position type

  .define FI_ABS          ,1      ; fi_postyp or fi_fpostyp = ABSOLUTE
  .define FI_REL          ,2      ; fi_postyp or fi_fpostyp = RELATIVE

; Defines for input field wait

  .define FI_WAITNON      ,0      ; fi_wait or ti_wait = NO
  .define FI_WAIT_TIME    ,1      ; fi_wait or ti_wait = WAIT TIME
  .define FI_WAITIMMD     ,2      ; fi_wait or ti_wait = IMMEDIATE
  .define FI_WAITGLBL     ,3      ; fi_wait or ti_wait = GLOBAL
  .define FI_WAITFRVR     ,4      ; fi_wait or ti_wait = FOREVER

; Defines for View as

  .define FI_FIELDVW      ,0      ; fi_view or ti_view = FIELD
                                  ; <reserved>
  .define FI_RADIOVW      ,2      ; fi_view or ti_view = RADIO BUTTONS
  .define FI_CHECKVW      ,3      ; fi_view or ti_view = CHECKBOX

; Defines for Group flag

  .define F_NOGROUP       ,0      ; fi_group = NO
  .define F_GROUPFLD      ,1      ; fi_group = YES
  .define F_GROUPOVRFLD   ,2      ; fi_group = OVERLAY

; Defines for testing bits in fi_flags (15i1) and ti_flags (15i1)

  ; fi_flags(1) and ti_flags(1)
  .define B_DESC          ,0      ; fi_desc, ti_desc
  .define B_LDESC         ,1      ; fi_ldesc, ti_ldesc
  .define B_DBLTYPE       ,2      ; fi_dbltype, ti_dbltype
  .define B_USRTYP        ,3      ; fi_usrtyp, ti_usrtyp
  .define B_DBLSIZE       ,4      ; fi_dblsize, ti_dblsize
```

```
.define B_PREC          ,6      ; fi_prec, ti_prec
.define B_DIM           ,7      ; fi_dim, ti_dim

; fi_flags(2) and ti_flags(2)
.define B_DBLVW         ,1      ; fi_dblvw, ti_dblvw
.define B_RPTVW         ,2      ; fi_rptvw, ti_rptvw
.define B_SCRPTVW       ,3      ; fi_scrptvw, ti_scrptvw
.define B_WEBVW         ,4      ; fi_webvw, ti_webvw

; fi_flags(4) and ti_flags(4)
.define B_HDG           ,0      ; fi_hdg, ti_hdg
.define B_FMT           ,1      ; fi_fmt, ti_fmt
.define B_RPTJUST       ,2      ; fi_rptjust, ti_rptjust
.define B_INPJUST       ,3      ; fi_inpjust, ti_inpjust
.define B_FPOSTYP       ,4      ; fi_fpostyp, ti_fpostyp
.define B_POSTYP        ,6      ; fi_postyp, ti_postyp
.define B_BZRO          ,7      ; fi_bzro, ti_bzro

; fi_flags(5) and ti_flags(5)
.define B_PAINT         ,0      ; fi_paint, ti_paint
.define B_VIEW          ,1      ; fi_view, ti_view
.define B_PROMPT        ,2      ; fi_prompt, ti_prompt
.define B_HELP          ,3      ; fi_help, ti_help
.define B_INFOLN        ,4      ; fi_infoln, ti_infoln
.define B_UTEXT         ,6      ; fi_utext, ti_utext
.define B_ALTNM         ,7      ; fi_altnm, ti_altnm
.define B_ODBCNM        ,7      ; (For compatibility with pre-v7.5)

; fi_flags(6) and ti_flags(6)
.define B_FONT          ,0      ; fi_font, ti_font
.define B_PRMPTFONT     ,1      ; fi_prmptfont, ti_prmptfont
.define B_RENDINFO      ,2      ; fi_color, ti_color, fi_attrib,
                                ; ti_attrib, fi_highlight, etc
.define B_READONLY      ,3      ; fi_readonly, ti_readonly
.define B_DISABLED      ,4      ; fi_disabled, ti_disabled
.define B_DISPLEN       ,6      ; fi_displen, ti_displen
.define B_VIEWLEN       ,7      ; fi_viewlen, ti_viewlen

; fi_flags(7) and ti_flags(7)
.define B_NOECHO        ,0      ; fi_noecho, ti_noecho
.define B_ECHOCHR       ,1      ; fi_echochr, ti_echochr
.define B_DEFACT        ,2      ; fi_defact, ti_defact
.define B_AUTO          ,3      ; fi_auto, ti_auto
.define B_TODAY         ,4      ; fi_today, ti_today
.define B_SHORT         ,6      ; fi_short, ti_short
.define B_NOW           ,7      ; fi_now, ti_now

; fi_flags(8) and ti_flags(8)
.define B_AMPM          ,0      ; fi_ampm, ti_ampm
.define B_WAIT          ,1      ; fi_wait, ti_wait
.define B_UC            ,2      ; fi_uc, ti_uc
.define B_NODEC         ,3      ; fi_nodec, ti_nodec
.define B_NOTERM        ,4      ; fi_noterm, ti_noterm
```

```
  .define B_RETPOS        ,6      ; fi_retpos, ti_retpos
  .define B_INPLEN        ,7      ; fi_inplen, ti_inplen

  ; fi_flags(10) and ti_flags(10)
  .define B_REQ           ,0      ; fi_req, ti_req
  .define B_BREAK         ,1      ; fi_break, ti_break
  .define B_NEGALW        ,2      ; fi_negalw, ti_negalw
  .define B_ALWLST        ,3      ; fi_alwlst, ti_alwlst
  .define B_MATCHCS       ,4      ; fi_matchcs, ti_matchcs
  .define B_MATCHEX       ,6      ; fi_matchex, ti_matchex
  .define B_RANGE         ,7      ; fi_range, ti_range

  ; fi_flags(11) and ti_flags(11)
  .define B_ENUM          ,0      ; fi_enum, ti_enum
  .define B_SELLST        ,1      ; fi_sellist, ti_sellist
  .define B_NULL          ,2      ; fi_null, ti_null

  ; fi_flags(13) and ti_flags(13)
  .define B_ARVMETH       ,0      ; fi_arrivemeth, ti_arrivemeth
  .define B_LVMETH        ,1      ; fi_leavemeth, ti_leavemeth
  .define B_DRLMETH       ,2      ; fi_drillmeth, ti_drillmeth
  .define B_HYPMETH       ,3      ; fi_hypermeth, ti_hypermeth
  .define B_CHGMETH       ,4      ; fi_changemeth, ti_changemeth
  .define B_DSPMETH       ,6      ; fi_dispmeth, ti_dispmeth
  .define B_EDTFMTMETH    ,7      ; fi_editfmtmeth, ti_editfmtmeth

; Defines for file record type

  .define FLI_FIXED       ,0      ; fli_rectyp = FIXED
  .define FLI_VAR         ,1      ; fli_rectyp = VARIABLE
  .define FLI_MULT        ,2      ; fli_rectyp = MULTIPLE

; Defines for file page size

  .define FLI_PS1024      ,0      ; fli_pagesize = 1024
  .define FLI_PS512       ,1      ; fli_pagesize =  512
  .define FLI_PS2048      ,2      ; fli_pagesize = 2048
  .define FLI_PS4096      ,3      ; fli_pagesize = 4096
  .define FLI_PS8192      ,4      ; fli_pagesize = 8192
  .define FLI_PS16384     ,5      ; fli_pagesize = 16384
  .define FLI_PS32768     ,6      ; fli_pagesize = 32768

; Defines for file addressing

  .define FLI_32BIT       ,0      ; fli_addressing = 32BIT
  .define FLI_40BIT       ,1      ; fli_addressing = 40BIT

; Defines for key record length

  .define K_INFO_LEN      ,888    ; Size of k_info record
```

```
; Defines for key type

  .define KI_FOR          ,0      ; ki_ktype = FOREIGN key
  .define KI_ACC          ,1      ; ki_ktype = ACCESS key

; Defines for key order

  .define KI_ASC          ,0      ; ki_order = ASCENDING
  .define KI_DES          ,1      ; ki_order = DESCENDING

; Defines for key duplicates value

  .define KI_NDPS         ,0      ; ki_dups = NO DUPS allowed
  .define KI_DPS          ,1      ; ki_dups = DUPS allowed

; Defines for duplicate key insert value

  .define KI_FRT          ,0      ; ki_insert = INSERT AT FRONT
  .define KI_END          ,1      ; ki_insert = INSERT AT END

; Defines for modifiable key value

  .define KI_NMDF         ,0      ; ki_mod = NOT MODIFIABLE
  .define KI_MDF          ,1      ; ki_mod = MODIFIABLE

; Defines for null key types

  .define KI_NONULL       ,0      ; ki_null = Not a NULL key
  .define KI_REP          ,1      ; ki_null = REPLICATING
  .define KI_NONREP       ,2      ; ki_null = NON_REPLICATING
  .define KI_SHORT        ,3      ; ki_null = SHORT

; Defines for key segment types

  .define KI_SG_FLD       ,'F'    ; ki_segtyp = FIELD
  .define KI_SG_LIT       ,'L'    ; ki_segtyp = LITERAL
  .define KI_SG_EXT       ,'E'    ; ki_segtyp = EXTERNAL
  .define KI_SG_REC       ,'R'    ; ki_segtyp = RECORD NUMBER

; Defines for key segment order override
; If ki_segord = 0, use the key order, ki_order

  .define KI_SG_ASC       ,1      ; ki_segord = ASCENDING
  .define KI_SG_DES       ,2      ; ki_segord = DESCENDING

; Defines for optional key segment data type
; If ki_segdtyp = 0, look up the field type, fi_type

  .define KI_SG_ALP       ,1      ; Alpha key segment (Used for k_segdtyp)
  .define KI_SG_NOC       ,2      ; NoCase Alpha key segment "     "
  .define KI_SG_DEC       ,3      ; Decimal key segment "    "     "
  .define KI_SG_INT       ,4      ; Integer key segment "    "     "
  .define KI_SG_UNS       ,5      ; Unsigned Integer key segment "    "
```

```
  .define KI_SG_SEQ       ,6      ; Sequence (integer) key segment "    "
  .define KI_SG_TIM       ,7      ; Timestamp (integer) key segment "    "
  .define KI_SG_CRT       ,8      ; Create timestamp (integer) key segment "    "

; Defines for structure tag types

  .define TAGNON          ,0      ; si_tagtyp = No tag
  .define TAGFLD          ,1      ; si_tagtyp = Field & value
  .define TAGSIZ          ,2      ; si_tagtyp = Record size

; Defines for tag field comparison operators

  .define TGI_EQ          ,1      ; tgi_tagcmp = EQ
  .define TGI_NE          ,2      ; tgi_tagcmp = NE
  .define TGI_LE          ,3      ; tgi_tagcmp = LE
  .define TGI_LT          ,4      ; tgi_tagcmp = LT
  .define TGI_GE          ,5      ; tgi_tagcmp = GE
  .define TGI_GT          ,6      ; tgi_tagcmp = GT

; Defines for tag field comparison connectors

  .define TGI_AND         ,1      ; tgi_tagcon = AND
  .define TGI_OR          ,2      ; tgi_tagcon = OR

; Defines for namespace inclusion

  .define NSI_DBL         ,^x(0001) ; Include if available to Language
  .define NSI_UI          ,^x(0002) ; Include if available to UI Toolkit
  .define NSI_RW          ,^x(0004) ; Include if available to Report Writer
  .define NSI_WEB         ,^x(0008) ; Include if available to Web

  .define NSI_NAMES       ,^x(0100) ; Index by name
  .define NSI_SEQ         ,^x(0200) ; Index by sequence number

  .define NSI_ALL         ,^x(FFFF) ; Index all fields both ways

; Macro for creating a key in the sequence namespace

  .define M_SEQKEY(sequence) %string(sequence,"SXXXXXXXX")

  .define DCS_SIZE    ,262        ; dcs size in version 9.3 (was 198)
.endc

; Repository Control Structure

.ifndef DDINFO_DEFINES_ONLY  ; Prevent getting data when only defines needed
.ifdef DBL8CMP
.define endrecord
.define endstructure
.endc
.ifndef DDINFO_INGLOBAL      ; Exclude STACK records when used in global section
.ifdef DDINFO_STRUCTURE
.define record structure
```

```
.define endrecord endstructure
.else
  .define record stack record
.endc
.endc
  record dcs
    error          ,d2     ; Non-zero = error (See .defines above)
    mchn_r         ,i4     ; Main repository channel
    tchn_r         ,i4     ; Repository text channel
    sname          ,a30    ; Current structure name
    sid            ,a4     ; Current structure ID
    flname         ,a30    ; Current file name
    tname          ,a30    ; Current template name
    fname          ,a30    ; Current field name
    aid            ,a4     ; Current alias' aliased structure ID
    kname          ,a30    ; Current key name
    grpid          ,a4     ; Current group ID (0 if group = AC_STR)
    ename          ,a30    ; Current enumeration name
    dcs_filler     ,a60    ; Room to grow
  endrecord

; Alias information structure

  record a_info
    ai_name        ,a30    ; Aliased structure name
  endrecord

; Control record information structure

  record c_info
    ci_tstamp      ,a14    ; Timestamp of last repository modification
    ci_ver         ,a8     ; Repository version
    ci_str_tstamp  ,a14    ; Timestamp of last structure addition/deletion
  endrecord

; Field information structure

  record f_info
    fi_seqnm       ,d3     ; Sequence number
    fi_mbrct       ,d3     ; Number of members if field is a group
    fi_nosize      ,d1     ; TRUE if group size = size of all members
    fi_pos         ,d5     ; Starting position within the record or group
    fi_ovrfld      ,a30    ; Name of the field being overlaid
    fi_ovroff      ,d5     ; Overlay offset within the above field
    fi_group       ,d1     ; Group flag (see .defines above)
    fi_struct      ,d2     ; Non-zero = struct name for implicit group
    fi_prefix      ,d2     ; Non-zero = group member prefix exists
    fi_template    ,a30    ; Template referenced by this field
    fi_desc        ,d2     ; Non-zero = short description exists
    fi_ldesc       ,d2     ; Non-zero = long description exists
    fi_type        ,a1     ; Data type (see .defines above)
    fi_class       ,d2     ; DBL type subclass (see .defines above)
    fi_usrtyp      ,d2     ; Non-zero = user data type string exists
```

```
     fi_enmfld     ,d2@fi_usrtyp ; Or...enumeration name for Enum fields
     fi_strfld     ,d2@fi_usrtyp ; Or...structure name for Struct fields
    fi_ndtype     ,a1     ; Native data type
    fi_size       ,d5     ; Data size
    fi_prec       ,d2     ; # digits to the right of the decimal pt.
.ifdef DDINFO_STRUCTURE
    fi_dim        ,[4]d3  ; Dimension (used for arrays)
.else
    fi_dim        ,4d3    ; Dimension (used for arrays)
.endc
    fi_ndsize     ,d5     ; Native data size
    fi_dblvw      ,d1     ; Is field NOT available to DBL? (see .defines)
    fi_rptvw      ,d1     ; Is field NOT available to RW? (see .defines)
    fi_scrptvw    ,d1     ; Is field NOT available to UI? (see .defines)
    fi_nolnk      ,d1     ; Field name is the name link (boolean)
    fi_cmppref    ,d1     ; TRUE if member prefix will be used by Compiler
    fi_webvw      ,d1     ; Is field NOT available to WEB? (see .defines)
    fi_coertype   ,d2     ; Coerced type (see .defines)
    fi_filler1    ,a18    ; (Reserved for future use)
    fi_heading    ,d2     ; Non-zero = column heading exists
    fi_fmt        ,a30    ; Defines a global or local format name
    fi_rptjust    ,d1     ; RW field justification (see .defines above)
    fi_inpjust    ,d1     ; Input field justification (see .defines above)
    fi_fpostyp    ,d1     ; Input field position type (see .defines above)
    fi_finprow    ,d3     ; Input field row position
    fi_finpcol    ,d3     ; Input field column position
    fi_postyp     ,d1     ; Input prompt position type (see .defines above)
    fi_inprow     ,d3     ; Input prompt row position
    fi_inpcol     ,d3     ; Input prompt column position
    fi_bzro       ,d1     ; Blank if zero? (boolean)
    fi_paint      ,d1     ; Is a paint character specified? (boolean)
    fi_pntchr     ,a1     ; Paint character for empty fields
    fi_view       ,d1     ; View as Field, Radio buttons, or Checkbox
    fi_prompt     ,d2     ; Non-zero = prompt text exists
    fi_help       ,d2     ; Non-zero = help id exists
    fi_infoln     ,d2     ; Non-zero = info line text exists
    fi_utext      ,d2     ; Non-zero = user text string exists
    fi_altnm      ,d2     ; Non-zero = alternate field name exists
     fi_odbcnm     ,d2 @fi_altnm  ; (For compatibility with pre-v7.5)
    fi_font       ,d2     ; Non-zero = field font name exists
    fi_prmptfont  ,d2     ; Non-zero = prompt font name exists
    fi_color      ,d2     ; Color palette (1-16)
    fi_attrib     ,d1     ; If non-zero, then the following four fields
                          ; override the window's attributes
    fi_highlight  ,d1     ; Highlight attribute
    fi_reverse    ,d1     ; Reverse attribute
    fi_blink      ,d1     ; Blink attribute
    fi_underline  ,d1     ; Underline attribute
    fi_readonly   ,d1     ; Is the field read-only? (boolean)
    fi_disabled   ,d1     ; Is the field disabled? (boolean)
    fi_displen    ,d5     ; Display length
    fi_viewlen    ,d5     ; View length
    fi_filler2    ,a10    ; (Reserved for future use)
```

```
    fi_noecho     ,d1     ; Do not display text input in field? (boolean)
    fi_echochr    ,a1     ; No echo character. Replaces text input.
    fi_defact     ,d1     ; Default action (see .defines above)
    fi_def        ,d2     ; Non-zero = default value exists
    fi_auto       ,d1     ; Automatic default action? (boolean)
    fi_today      ,d1     ; Default date to TODAY? (boolean)
    fi_short      ,d1     ; Allow short date? (boolean)
    fi_now        ,d1     ; Default time to NOW? (boolean)
    fi_ampm       ,d1     ; Display meridian indicator? (boolean)
    fi_wait       ,d1     ; Input timeout value (see .defines above)
    fi_waittime   ,d5     ; Input timeout time if fi_wait = WAIT TIME
    fi_uc         ,d1     ; Convert all input to uppercase? (boolean)
    fi_nodec      ,d1     ; No decimal needs to be input? (boolean)
    fi_noterm     ,d1     ; Field terminates auto. when filled? (boolean)
    fi_retpos     ,d1     ; Retain cursor position in text field (boolean)
    fi_inplen     ,d5     ; Input length
    fi_filler3    ,a5     ; (Reserved for future use)
    fi_req        ,d1     ; Is field input required? (boolean)
    fi_break      ,d1     ; Break field? (see .defines above)
    fi_negalw     ,d1     ; Negative allowed? (see .defines above)
    fi_alwlst     ,d2     ; Non-zero = allow list exists
    fi_alwct      ,d2     ; Number of allow list entries
    fi_alwlen     ,d3     ; Length of longest allow list entry
    fi_matchcs    ,d1     ; Match case? (boolean)
    fi_matchex    ,d1     ; Match exact? (boolean)
    fi_range      ,d2     ; Non-zero = min/max values exist
    fi_enum       ,d2     ; Non-zero = enumerated field info exists
    fi_sellist    ,d2     ; Non-zero = selection list exists
    fi_selct      ,d2     ; Number of selection list entries
    fi_sellen     ,d3     ; Length of longest selection list entry
    fi_seltyp     ,d1     ; Current selection type (see .defines above)
    fi_selrow     ,d2     ; Current selection window row
    fi_selcol     ,d2     ; Current selection window column
    fi_selwin     ,a15    ; Current selection window name...
    fi_selht      ,d2     ; ...OR selection window height
    fi_null       ,d1     ; Null allowed? (see .defines above)
    fi_filler4    ,a9     ; (Reserved for future use)
    fi_arrivemeth ,d2     ; Non-zero = arrive method exists
    fi_leavemeth  ,d2     ; Non-zero = leave method exists
    fi_drillmeth  ,d2     ; Non-zero = drill method exists
    fi_hypermeth  ,d2     ; Non-zero = hyperlink prompt method exists
    fi_changemeth ,d2     ; Non-zero = change method exists
    fi_dispmeth   ,d2     ; Non-zero = display method exists
    fi_editfmtmeth,d2     ; Non-zero = edit format method exists
    fi_filler5    ,a4     ; (Reserved for future use)
.ifdef DDINFO_STRUCTURE
    fi_flags      ,[15]i1 ; Template override flags (see .defines above)
.else
    fi_flags      ,15i1   ; Template override flags (see .defines above)
.endc
  endrecord
```

```
; Field range information structure

  record fti_range
    fti_rgmin     ,d28.10 ; Current field range minimum
    fti_rgmax     ,d28.10 ; Current field range maximum
  endrecord

; Enumerated field information structure

  record fti_enum
    fti_enmlen    ,d2     ; Current enumerated field display length
    fti_enmbase   ,d2     ; Current enumerated field base
    fti_enmstep   ,d2     ; Current enumerated field step
  endrecord

; Field selection information structure

  record  fti_entlst
.ifdef DDINFO_STRUCTURE
    fti_entlstary ,[100]a80 ; Current selection or allow list entries
.else
    fti_entlstary ,100a80   ; Current selection or allow list entries
.endc
  endrecord

; File information structure

  record fl_info
    fli_tstamp      ,a14    ; Timestamp of last modification
    fli_filtyp      ,a15    ; File type (e.g., "DBL ISAM", "ASCII")
    fli_fname       ,a255   ; Actual filename
    fli_temp        ,d1     ; Is file definition "temporary"? (boolean)
    fli_nmstructs   ,d3     ; Number of structures assigned to the file
    fli_struct      ,a30    ; First (or only) assigned structure
    fli_desc        ,d2     ; Non-zero = short description exists
    fli_ldesc       ,d2     ; Non-zero = long description exists
    fli_utext       ,d2     ; Non-zero = user text string exists
    fli_rectyp      ,d1     ; Fixed, Variable, Multiple records
    fli_pagesize    ,d1     ; Page (index block) size
    fli_density     ,d3     ; File density
    fli_addressing  ,d1     ; File addressing
    fli_compress    ,d1     ; Compress record data? (boolean)
    fli_staticrfa   ,d1     ; Static RFA's? (boolean)
    fli_portable    ,d2     ; Non-zero = portable int specs exist
    fli_track       ,d1     ; Track changes? (boolean)
    fli_tbyte       ,d1     ; Terabyte? (boolean)
    fli_sgrfa       ,d1     ; Stored GRFA? (boolean)
    fli_netencrypt  ,d1     ; Network encrypt? (boolean)
    fli_norollback  ,d1     ; No change tracking rollback (boolean)
    fli_sizelimit   ,d5     ; File size limit in MB
    fli_reclimit    ,d8     ; Number of records limit
    fli_filetext    ,d2     ; Non-zero = file text exists
    fli_filler      ,a30    ; Room to grow
  endrecord
```

```
; File specification structure

  record fls_info
    flsi_name       ,a255   ; Actual filename
    flsi_filtyp     ,a15    ; File type (e.g., "DBL ISAM", "ASCII")
    flsi_sname      ,a30    ; Structure name
    flsi_recsz      ,d5     ; Record size
    flsi_nmkeys     ,d2     ; Number of ACCESS keys
    flsi_rectyp     ,d1     ; Fixed, Variable, Multiple records
    flsi_pagesize   ,d1     ; Page (index block) size
    flsi_density    ,d3     ; File density
    flsi_addressing ,d1     ; File addressing
    flsi_compress   ,d1     ; Compress record data? (boolean)
    flsi_staticrfa  ,d1     ; Static RFA's? (boolean)
    flsi_portable   ,a120   ; Portable integer specs
    flsi_track      ,d1     ; Track changes? (boolean)
    flsi_tbyte      ,d1     ; Terabyte? (boolean)
    flsi_sgrfa      ,d1     ; Stored GRFA? (boolean)
    flsi_netencrypt ,d1     ; Network encrypt? (boolean)
    flsi_norollback ,d1     ; No change tracking rollback (boolean)
    flsi_sizelimit  ,d5     ; File size limit in MB
    flsi_reclimit   ,d8     ; Number of records limit
    flsi_filler     ,a2     ; Room to grow
    endrecord

; Enumeration information structure

  record e_info
    ei_tstamp       ,a14    ; Timestamp of last modification
    ei_nmmbrs       ,d3     ; Number of member definitions
    ei_desc         ,d2     ; Non-zero = short description exists
    ei_ldesc        ,d2     ; Non-zero = long description exists
    ei_filler       ,a50    ; Room to grow
  endrecord

; Key information structure

  record k_info
    ki_seqnm      ,d3     ; Sequence number
    ki_name       ,a30    ; Key name (for use with dd_filespec)
    ki_ktype      ,d1     ; Key type (see .defines above)
    ki_size       ,d3     ; Total size of all key segments
    ki_desc       ,d2     ; Non-zero = short description exists
    ki_filler1    ,d4     ; (Reserved for future use)
    ki_order      ,d1     ; Sort order (see .defines above)
    ki_dups       ,d1     ; Are dups allowed? (see .defines above)
    ki_insert     ,d1     ; If dups, insert at front or end (see .defines)
    ki_mod        ,d1     ; Modifiable? (boolean)
    ki_null       ,d1     ; Null key? (see .defines above)
    ki_nullval    ,d2     ; Non-zero = null value string exists
    ki_krf        ,d3     ; Optional explicit key of reference
```

```
    ki_density    ,d3     ; Key density
    ki_cmpidx     ,d1     ; Compress index? (boolean)
    ki_cmprec     ,d1     ; Compress record? (boolean)
    ki_cmpkey     ,d1     ; Compress key? (boolean)
    ki_nmseg      ,d1     ; Number of segments in this key
.ifdef DDINFO_STRUCTURE
    ki_segtyp     ,[8]a1    ; Segment types (see .defines above)
    ki_segpos     ,[8]d5    ; Segment field positions
    ki_seglen     ,[8]d3    ; Segment field/literal lengths
    ki_fldnam     ,[8]a30   ; Segment field names
    ki_strnam     ,[8]a30   ; Segment structure names
    ki_litval     ,[8]a30   ; Segment literal values (lengths in ki_seglen)
    ki_segdtyp    ,[8]d1    ; Optional, segment data type
    ki_segord     ,[8]d1    ; Optional, segment order
.else
    ki_segtyp     ,8a1    ; Segment types (see .defines above)
    ki_segpos     ,8d5    ; Segment field positions
    ki_seglen     ,8d3    ; Segment field/literal lengths
    ki_fldnam     ,8a30   ; Segment field names
    ki_strnam     ,8a30   ; Segment structure names
    ki_litval     ,8a30   ; Segment literal values (lengths in ki_seglen)
    ki_segdtyp    ,8d1    ; Optional, segment data type
    ki_segord     ,8d1    ; Optional, segment order
.endc
    ki_odbcvw     ,d1     ; Is key accessible to ODBC? (boolean)
    ki_filler2    ,a19    ; Room to grow
  endrecord

; Structure information

  record s_info
    si_tstamp       ,a14    ; Timestamp of last modification
    si_filtyp       ,a15    ; File type (ie "DBL ISAM", "ASCII")
    si_desc         ,d2     ; Non-zero = short description exists
    si_ldesc        ,d2     ; Non-zero = long description exists
    si_utext        ,d2     ; Non-zero = user text string exists
    si_recsz        ,d5     ; Record size
    si_nmflds       ,d3     ; Number of associated fields
    si_nmkeys       ,d3     ; Number of associated keys
    si_nmrels       ,d2     ; Number of associated relations
    si_nmfils       ,d3     ; Number of files assigned to
    si_nmfmts       ,d3     ; Number of associated local formats
    si_nmtags       ,d2     ; Number of associated tags
    si_tagtyp       ,d1     ; Structure tag type (see .defines above)
    si_childct      ,d3     ; Number of fields/groups in first level
    si_file         ,a30    ; First (or only) file assigned to
    si_filler       ,a50    ; Room to grow
  endrecord

; Tag information structure

  record tg_info
    tgi_seqnm     ,d3     ; Sequence number
    tgi_tagcon    ,d1     ; Tag field comparison connector (see .defines)
```

```
    tgi_tagfld    ,a30    ; Field name if si_tagtyp = TAGFLD
    tgi_tagcmp    ,d1     ; Tag field comparison operator (see .defines)
    tgi_tagval    ,a15    ; Tag field comparison value
    tgi_filler    ,a20    ; Room to grow
  endrecord

; Template information structure

  record t_info
    ti_tstamp     ,a14    ; Timestamp of last modification
    ti_template   ,a30    ; Template referenced by this template
    ti_desc       ,d2     ; Non-zero = short description exists
    ti_ldesc      ,d2     ; Non-zero = long description exists
    ti_type       ,a1     ; Data type (see .defines above)
    ti_class      ,d2     ; DBL type subclass (see .defines above)
    ti_usrtyp     ,d2     ; Non-zero = user data type string exists
     ti_enmfld     ,d2@ti_usrtyp ; Or...enumeration name for Enum templates
    ti_ndtype     ,a1     ; Native data type
    ti_size       ,d5     ; Data size
    ti_prec       ,d2     ; # digits to the right of the decimal pt.
.ifdef DDINFO_STRUCTURE
    ti_dim        ,[4]d3  ; Dimension (used for arrays)
.else
    ti_dim        ,4d3    ; Dimension (used for arrays)
.endc
    ti_ndsize     ,d5     ; Native data size
    ti_dblvw      ,d1     ; Is field NOT available to DBL? (see .defines)
    ti_rptvw      ,d1     ; Is field NOT available to RW? (see .defines)
    ti_scrptvw    ,d1     ; Is field NOT available to UI? (see .defines)
    ti_nolnk      ,d1     ; Field name is the name link (boolean)
    ti_webvw      ,d1     ; Is field NOT available to WEB? (see .defines)
    ti_coertype   ,d2     ; Coerced type (see .defines)
    ti_filler1    ,a19    ; (Reserved for future use)
    ti_heading    ,d2     ; Non-zero = column heading exists
    ti_fmt        ,a30    ; Defines a global or local format name
    ti_rptjust    ,d1     ; RW field justification (see .defines above)
    ti_inpjust    ,d1     ; Input field justification (see .defines above)
    ti_fpostyp    ,d1     ; Input field position type (see .defines above)
    ti_finprow    ,d3     ; Input field row position
    ti_finpcol    ,d3     ; Input field column position
    ti_postyp     ,d1     ; Input prompt position type (see .defines above)
    ti_inprow     ,d3     ; Input prompt row position
    ti_inpcol     ,d3     ; Input prompt column position
    ti_bzro       ,d1     ; Blank if zero? (boolean)
    ti_paint      ,d1     ; Is a paint character specified? (boolean)
    ti_pntchr     ,a1     ; Paint character for empty fields
    ti_view       ,d1     ; Is field viewed as radio buttons or checkbox
    ti_prompt     ,d2     ; Non-zero = prompt text exists
    ti_help       ,d2     ; Non-zero = help id exists
    ti_infoln     ,d2     ; Non-zero = info line text exists
    ti_utext      ,d2     ; Non-zero = user text string exists
    ti_altnm      ,d2     ; Non-zero = alternate field name exists
     ti_odbcnm     ,d2 @ti_altnm  ; (For compatibility with pre-v7.5)
```

```
ti_font       ,d2     ; Non-zero = field font name exists
ti_prmptfont  ,d2     ; Non-zero = prompt font name exists
ti_color      ,d2     ; Color palette (1-16)
ti_attrib     ,d1     ; If non-zero, then the following four fields
                      ; override the window's attributes
ti_highlight  ,d1     ; Highlight attribute
ti_reverse    ,d1     ; Reverse attribute
ti_blink      ,d1     ; Blink attribute
ti_underline  ,d1     ; Underline attribute
ti_readonly   ,d1     ; Is the field read-only? (boolean)
ti_disabled   ,d1     ; Is the field disabled? (boolean)
ti_displen    ,d5     ; Display length
ti_viewlen    ,d5     ; View length
ti_filler2    ,a10    ; (Reserved for future use)
ti_noecho     ,d1     ; Do not display text input in field? (boolean)
ti_echochr    ,a1     ; No echo character. Replaces text input.
ti_defact     ,d1     ; Default action (see .defines above)
ti_def        ,d2     ; Non-zero = default value exists
ti_auto       ,d1     ; Automatic default action? (boolean)
ti_today      ,d1     ; Default date to TODAY? (boolean)
ti_short      ,d1     ; Allow short date? (boolean)
ti_now        ,d1     ; Default time to NOW? (boolean)
ti_ampm       ,d1     ; Display meridian indicator? (boolean)
ti_wait       ,d1     ; Input timeout value (see .defines above)
ti_waittime   ,d5     ; Input timeout time if fi_wait = WAIT TIME
ti_uc         ,d1     ; Convert all input to uppercase? (boolean)
ti_nodec      ,d1     ; No decimal needs to be input? (boolean)
ti_noterm     ,d1     ; Field terminates auto. when filled? (boolean)
ti_retpos     ,d1     ; Retain cursor position in text field (boolean)
ti_inplen     ,d5     ; Input length
ti_filler3    ,a5     ; (Reserved for future use)
ti_req        ,d1     ; Is field input required? (boolean)
ti_break      ,d1     ; Break field? (see .defines above)
ti_negalw     ,d1     ; Negative allowed? (see .defines above)
ti_alwlst     ,d2     ; Non-zero = allow list exists
ti_alwct      ,d2     ; Number of allow list entries
ti_alwlen     ,d3     ; Length of longest allow list entry
ti_matchcs    ,d1     ; Match case? (boolean)
ti_matchex    ,d1     ; Match exact? (boolean)
ti_range      ,d2     ; Non-zero = min/max values exist
ti_enum       ,d2     ; Non-zero = enumerated field info exists
ti_sellist    ,d2     ; Non-zero = selection window info exists
ti_selct      ,d2     ; Number of selection list entries
ti_sellen     ,d3     ; Length of longest selection list entry
ti_seltyp     ,d1     ; Current selection type (see .defines above)
ti_selrow     ,d2     ; Current selection window row
ti_selcol     ,d2     ; Current selection window column
ti_selwin     ,a15    ; Current selection window name
ti_selht      ,d2     ; ...OR selection window height
ti_null       ,d1     ; Null allowed? (see .defines above)
ti_filler4    ,a9     ; (Reserved for future use)
ti_arrivemeth ,d2     ; Non-zero = arrive method exists
ti_leavemeth  ,d2     ; Non-zero = leave method exists
```

```
    ti_drillmeth  ,d2     ; Non-zero = drill method exists
    ti_hypermeth  ,d2     ; Non-zero = hyperlink prompt method exists
    ti_changemeth ,d2     ; Non-zero = change method exists
    ti_dispmeth   ,d2     ; Non-zero = display method exists
    ti_editfmtmeth,d2     ; Non-zero = edit format method exists
    ti_filler5    ,a4     ; (Reserved for future use)
.ifdef DDINFO_STRUCTURE
    ti_flags      ,[15]i1 ; Template override flags (see .defines above)
.else
    ti_flags      ,15i1   ; Template override flags (see .defines above)
.endc
  endrecord

    ; Namespace control argument

  record ns_ctl
    ns_name       ,i4     ; Handle to namespace ordered by name (or 0)
    ns_seq        ,i4     ; Handle to namespace ordered by sequence (or 0)
  endrecord

.ifndef DDINFO_INGLOBAL   ; Exclude STRUCTUREs when used in global section
  structure nsf_info      ; Namespace information structure for fields
    nsf_pos       ,i4     ; Absolute structure position
    nsf_dup       ,i4     ; Is the name non-unique?
    nsf_invisible ,i4     ; Is the field invisible? (not in name namespace)
    nsf_implicit  ,i4     ; Is the field a member of an implicit group?
    nsf_seqnm     ,i4     ; Absolute sequence number
    nsf_parent    ,i4     ; Sequence number of parent group, or 0 if struct
    nsf_parentid  ,a4     ; ID of parent group or structure
    nsf_name      ,a30    ; Name of the field (including any prefix)
    nsf_nsp       ,a30    ; Name sans prefix
endstructure

.undefine record
.ifdef DDINFO_STRUCTURE
.undefine endrecord
.endc
.ifdef DBL8CMP
.undefine endrecord
.undefine endstructure
.endc

.endc           ; DDINFO_INGLOBAL

.endc           ; DDINFO_DEFINES_ONLY
```

Appendices

Appendix A: Maximums

Lists all size and number maximums permitted by Repository.

Appendix B: Date and Time Formats

Lists the date and time storage and display formats that Repository supports.

Appendix C: Error Messages

Lists error messages that may appear in Repository, along with an explanation of the problem that caused the error to occur.

Appendix D: Data Formats

Explains Synergy DBL data formatting.

Appendix E: Distributed Shortcuts

Lists the Repository shortcuts as they are originally distributed.

A

Maximums

Maximum Values Permitted by Repository	
Item	**Maximum**
Aliases	999
Alias fields per alias structure	650
Array dimensions	4
Array elements per dimension	999
Digits to right of decimal point	28
Enumerations	9,999
Enumeration members	999
Enumeration definition name (size in characters)	30
Enumeration description (short) (size in characters)	40
Enumeration description (long) (size in characters)	1,800
Enumeration member name (size in characters)	30
Enumeration value (size in characters))	11
Field (size in characters)	99,999
Field allow list entries	99
Field allow list entry (size in characters)	80
Field default input value (size in characters)	80
Field definition name (size in characters)	30
Field description (long) (size in characters)	1,800
Field description (short) (size in characters)	40

Maximum Values Permitted by Repository (Continued)	
Item	**Maximum**
Field display length	65,535
Field font and prompt font names (size in characters)	80
Field help string (size in characters)	80
Field information line (size in characters)	80
Field input length	65,535
Field input prompt (size in characters)	80
Field minimum/maximum range value (size in characters)	18
Field report heading (size in characters)	40
Field selection window entries	99
Field selection window entry (size in characters)	80
Field selection window name (size in characters)	15
Field user data type string (size in characters)	30
Field user-defined text string (size in characters)	80
Field view length	9,999
Fields per structure or group	999
Fields to which a template is assigned	6,000
Fields which may reference a structure as an implicit group	200
File definition name (size in characters)	30
File definition's ODBC table name (size in characters)	30
File definition's open filename (size in characters)	64
File definition's portable integer specification (size in characters)	120
File definitions	9,999
File description (long) (size in characters)	1,800
File description (short) (size in characters)	40

Maximum Values Permitted by Repository (Continued)	
Item	**Maximum**
File user-defined text string (size in characters)	60
Files which may be assigned to a structure	200
Format definition name (size in characters)	30
Format string (size in characters)	30
Global data section name (size in characters)	30
Global formats	9,999
Group member prefix (size in characters)	30
Implied-decimal field (size in bytes)	28
Key (size in bytes)	255
Key definition name (size in characters)	30
Key description (short) (size in characters)	40
Keys per structure	99
Record	99,999
Relations per structure	99
Segment (field or external type) (size in bytes)	255
Segment (literal type) (size in characters)	30
Segments per key	8
Structures	9,999
Structure definition name (size in characters)	30
Structure description (long) (size in characters)	1,800
Structure description (short) (size in characters)	40
Structure user-defined text string (size in characters)	60
Structures assigned to a file	200
Structure-specific formats per structure	250

Maximum Values Permitted by Repository (Continued)	
Item	**Maximum**
Structure use in an external relation or external key segment definition (number of times)	200
Tags per structure	10
Templates	9,999
Template definition name (size in characters)	30
Template description (long) (size in characters)	1,800
Template description (short) (size in characters)	40
Templates to which a template is assigned	3,000

B

Date and Time Formats

This appendix lists the date and time storage formats supported by Repository and xfODBC, as well as the date and time display formats built into every repository.

Date and time formats selected in Repository also how data is converted when a structure is included in a Synergy component (JAR file or assembly). For information on how the various date and time formats are handled by xfNetLink clients, see "Appendix B: Data Type Mapping" in the *xfNetLink & xfServerPlus User's Guide*.

Date Formats

Date storage formats supported by Repository

YYMMDD	two-digit year, month, day
YYYYMMDD	four-digit year, month, day
YYJJJ	two-digit year, Julian day
YYYYJJJ	four-digit year, Julian day
YYPP	two-digit year, period
YYYYPP	four-digit year, period

Date storage formats supported by xfODBC

To specify these formats in your repository, define the field as a user type and include the following within the field's 30-character user data string:

^CLASS^ = *format*

where *format* is one of the following:

DDMMYY
DDMMYYYY
MMDDYY
MMDDYYYY
YYYYMMDDHHMISS
YYYYMMDDHHMISSUUUUUU
DDMonYY
DDMonYYYY
MonDDYY
MonDDYYYY
YYMonDD
YYYYMonDD
JJJYY
JJJYYYY
JJJJJJ
PPYY
PPYYYY

Date display formats that are built into every repository

#01	MM/DD/YYYY
#02	MM/DD/YY
#03	MM-DD-YYYY
#04	MM-DD-YY
#05	Mon/DD/YYYY
#06	Mon/DD/YY
#07	Mon-DD-YYYY
#08	Mon-DD-YY
#09	YYYY/MM/DD
#10	YY/MM/DD
#11	YYYY-MM-DD
#12	YY-MM-DD
#13	YYYY/Mon/DD
#14	YY/Mon/DD
#15	YYYY-Mon-DD
#16	YY-Mon-DD
#17	DD/MM/YYYY
#18	DD/MM/YY
#19	DD-MM-YYYY
#20	DD-MM-YY
#21	DD/Mon/YYYY
#22	DD/Mon/YY
#23	DD-Mon-YYYY
#24	DD-Mon-YY
#25	PP/YYYY
#26	PP/YY
#27	PP-YYYY
#28	PP-YY

where

MM	is the one- or two-digit month.
Mon	is the three-letter abbreviation for the month.
DD	is the one- or two-digit day.
YYYY	is the year, including the century.
YY	is the last two digits of the year.
PP	is the period.

The format size and the display size don't have to match. For example, you can pick a two-digit year format for a field that's stored as a four-digit year, and ReportWriter will automatically omit the century from the display format.

Time Formats

Time storage formats supported by Repository

HHMM	(hour and minute)
HHMMSS	(hour, minute, and second)

Time display formats that are built into every repository

#01	12:MM
#02	24:MM
#03	12:MMm
#04	12:MM:SS
#05	24:MM:SS
#06	12:MM:SSm

where

12	is the hour in 12-hour notation.
24	is the hour in military (24-hour) notation.
MM	is the minute.
SS	is the second.
m	is the meridian a or p (ante or post).

C

Error Messages

This appendix lists messages that may appear in Repository, along with explanations of the problems that may have caused the error. If you receive messages that are not listed in this appendix, contact your Synergy/DE Developer Support engineer at 800.366.3472 (in the U.S. and Canada) or 916.635.7300.

Filename is not a version 7 or version 10 repository.

Repository version 10 can only open repository files that are in version 7 format or version 10 format. (Version 7 format was used by Repository versions 7, 8, and 9; it can be converted to version 10 format by the version 10 Repository.) It is likely that the specified repository predates version 7 and therefore can be neither opened nor converted by version 10. If this is the case, you must use the repository conversion program to convert it. Refer to the Repository release notes (REL_RPS.TXT) for instructions.

Filename is not a version 10 repository.

The specified repository is not in version 10 format. To convert it to version 10 format, open it for modification in Repository and select Yes when prompted to convert. Once a repository has been converted, it cannot be accessed by pre-version 10 Repository programs and tools.

Filename is not a version 10 repository. Do you want to convert it?

The repository being opened for modification is not in version 10 format. Select Yes to convert it. Once a repository has been converted, it cannot be accessed by pre-version 10 Repository programs and tools. Select No to leave the repository in its current format and return to the menu or input window.

A field or group with the same name already exists.

While adding or copying a field, you've specified the name of an existing field or group. Field names must be unique within the current structure or group.

A file with the same name already exists.

While adding or copying a file, you've specified the name of an existing file. Filenames must be unique within the repository.

A format with the same name already exists.

While adding or copying a format, you've specified the name of an existing format. Structure-specific format names must be unique within the current structure, and global format names must be unique within the repository.

A key with the same name already exists.

While adding or copying a key, you've specified the name of an existing key. Key names must be unique within the current structure.

A member with the same name already exists.

While adding an enumeration member, you've specified the name of an existing member. Member names must be unique within the current enumeration.

A relation with the same name already exists.

While adding or copying a relation, you've specified the name of an existing relation. Relation names must be unique within the current structure.

A structure with the same name already exists.

While adding or copying a structure, you've specified the name of an existing structure. Structure names must be unique within the repository.

A tag with the same name already exists.

While adding or copying a tag, you've specified the name of an existing tag. Tag names must be unique within the current structure.

A template with the same name already exists.

While adding or copying a template, you've specified the name of an existing template. Template names must be unique within the repository.

Access keys are defined for this structure. File type cannot be modified.

You cannot change a file type to **Relative** if access keys already exist for the structure, because relative structures can only have one access key (the record number). If the original file type is relative and the record number key has not been deleted, the only file type to which you can change is **User defined**.

Access keys cannot contain external segments.

You've specified the **External** segment type for an access key. Access keys define true keys in the data file and therefore cannot contain external segments.

Access keys cannot contain literal segments.

You've specified the **Literal** segment type for an access key. Access keys define true keys in the data file and therefore cannot contain literal segments.

Alias cannot be deleted. Field reference exists.

You cannot delete the current alias because it is referenced within one or more struct type field definitions. Use the Print Repository Definitions utility to see where it is referenced. See "Printing Repository Definitions" on page 5-5.

An access key already exists for this relative file type.

Only one access key, the record number, is allowed for a relative file. If you have modified the record number segment, reselect **Access** as the key type.

An alias with the same name already exists.

While adding an alias structure, you've specified an existing alias or structure name. Alias names must be unique within the repository not only among alias names, but among structure names as well.

An enumeration with the same name already exists.

While adding or copying an enumeration, you've specified the name of an existing enumeration. Enumeration names must be unique within the repository.

At least one field must be defined for the structure you are trying to assign.

The structure you're attempting to assign to a file doesn't contain any field definitions.

At least one field must be defined for the structure you are trying to reference.

The structure you're attempting to reference as an implicit group or Struct type field doesn't contain any field definitions.

Cannot open main repository file *filename*.

Repository can't find the repository main file specified either by the RPSMFIL logical or the RPSDAT logical or in the current directory. Another possible explanation is that you don't have the necessary system privileges to access this file.

Cannot open Repository cross-reference file *filename*.

Repository can't find the repository cross-reference file specified by either the RPSXFIL logical, the RPSDAT logical, or the current directory. Another possible explanation is that you don't have the necessary system privileges to access this file.

Cannot open Repository message file *filename*.

Repository can't find the message file in either the RPSDAT logical or the current directory. Another possible explanation is that you don't have the necessary system privileges to access this file.

Cannot open repository text file *filename*.

Repository can't find the repository text file specified either by the RPSTFIL logical or the RPSDAT logical or in the current directory. Another possible explanation is that you don't have the necessary system privileges to access this file.

Cannot open Repository window library *filename*.

Repository can't find its window library file in either the directory specified by the RPS logical or the current directory. Another possible explanation is that you don't have the necessary system privileges to access this file.

Creation specifications are incomplete.

You have not entered all the required information. To create a new repository, you must enter both repository main and text filenames.

Cross-reference specifications are incomplete.

You have not entered all the required information. To generate a cross-reference file, you must specify the repository filenames and the name of the cross-reference file into which to generate the information.

Definition specifications are incomplete.

You have not entered all the required information. To generate a definition file, you must specify both the structure name and the name of the .INCLUDE file into which your definition is to be generated. If you've designated that you want your definition to include a global data section, you must specify the global data section name.

Deleting the current field will invalidate overlay field *field name*.

You cannot delete the current field, because to do so would invalidate one or more overlay fields that are defined after it. For example, if your structure contains the following two fields:

```
FIELD_ONE        ,A20
FIELD_TWO        ,A15   @FIELD_ONE
```

and you try to delete FIELD_ONE, Repository won't allow the deletion because it would cause FIELD_TWO to be an invalid overlay definition.

Enumeration cannot be deleted. Field reference exists.

You cannot delete the current enumeration because it is referenced within one or more field definitions. Use the Print Repository Definitions utility to see where it is referenced.

Enumeration cannot be deleted. Template reference exists.

You cannot delete the current enumeration because it is referenced within one or more template definitions. Use the Print Repository Definitions utility to see where it is referenced.

Enumeration no longer exists.

The enumeration you tried to select was deleted by another user between the time it was displayed in a list and the time you selected it.

External segment structure cannot be the current structure.

The structure name you specified in an external segment definition is the current structure. Because external segments define fields in other structures, you are not allowed to specify the current structure.

Field *field name* is a group. Deleting it will delete all group members as well. Do you want to continue?

The field you are deleting is defined as a group. To delete it will delete all its members, and all their members, and so forth. If you don't want to delete the group, select No.

Field *field name* is a group. Clearing the "Group" field will delete all group members. Do you want to continue?

Saving the current group field definition will delete all group members, and all their members, and so forth. If you don't want to delete the group, select No.

Field cannot be deleted. It is defined as a key.

The field you're trying to delete is used as a key segment in the current structure. Before you can delete the field, you must delete all keys that reference it.

Field cannot be deleted. It is defined as a tag field.

The field you're trying to delete is a tag field for the current structure. Before you can delete the field, you must remove it as a structure tag.

Field cannot be deleted. It is defined as an external key segment.

The field you're trying to delete is used in an external key segment that is defined by another structure. Before you can delete the field, you must delete all references to it in any external structures.

Field data type modification will invalidate key *key name*.

You have modified the data type of a field that is referenced within a key definition, and whose data type is overridden within that key. The modified data type is invalid for the key's overridden data type.

Field definition is incomplete.

You have not entered all the required information. When you define a field, you must specify the field name, data type, and size. Size is optional if you are defining a group field.

Field modifications will invalidate overlay field *field name*.

The field modifications you're trying to make would cause one or more overlay field definitions to become invalid. For example, if your structure contains the following two fields:

```
FIELD_ONE         ,A20
FIELD_TWO         ,A15   @FIELD_ONE
```

and you're trying to modify FIELD_ONE to be an **A10** field, the modification would make FIELD_TWO's overlay information invalid because it would attempt to overlay itself. You cannot change the size of FIELD_ONE unless you first modify FIELD_TWO appropriately.

File definition is incomplete.

You haven't entered all the required information. When you define a file, you must specify the name, file type, description, and open filename.

File no longer exists.

The file you attempted to select was deleted by another user between the time it was displayed in a list and the time you selected it.

File type of specified structure does not match that of the file definition.

The file type of the structure you're attempting to assign to a file doesn't match the file type of the file definition.

Foreign keys cannot contain record number segments.

You've modified the key type of a record number key from access to foreign. You must now change the record number segment to be a field, literal, or external segment.

Format *format name* referenced by field *field name* no longer exists.

The global format that was assigned to this field has since been deleted. You can select another format, or you can press the "Abandon" shortcut to abandon your changes and then create a new format with the same name.

Format cannot be deleted. It is assigned to a field.

One or more fields use the structure-specific format you're trying to delete.

Format definition is incomplete.

You have not entered all the required information. When you define a format, you must specify the format name, format type, and format string.

Format no longer exists.

The format you attempted to select was deleted by another user between the time it was displayed in a list and the time you selected it.

Internal reference buffer is full. Please exit to save changes.

Repository maintains many types of references so that it can preserve the integrity of your data base. It maintains a temporary reference buffer when you're working with a structure, and then the temporary buffer becomes permanent when you save your changes. If this temporary buffer becomes full while you're working with a structure, you will get this message. You must save your changes at this point by pressing the "Exit" shortcut. Once you've saved your changes, you can further modify the structure.

Invalid autokey specification.

Autokeys (sequence, timestamp, and create timestamp) must be a single segment consisting of an 8-byte field. They cannot be null, modifiable, or allow duplicates.

Invalid display format selected for given Date storage type.

Either the storage format for this date field is a period format (for example, YYYYPP) and you're trying to assign a non-period display format to it, or the storage format is non-period (for example, YYMMDD) and you're trying to assign a period display format to it.

Invalid group size specified. Group size must be equal to or larger than the size of its members.

In your group field definition, you've specified an explicit size, but the total size of the group members exceeds it. Edit the group members and adjust their total size, or increase the size of the group field.

Invalid key segment type.

In your key definition, you've specified a segment data type that is invalid for the data type of the field defined in that segment. When the field type is alpha, valid segment types are **A** (alpha) and **N** (nocase alpha). When the field type is integer, valid segment types are **I** (integer), **S** (sequence), **T** (timestamp), **C** (create timestamp), and **U** (unsigned integer). **D** (decimal) is the only valid segment type for decimal fields.

Invalid overlay field specified.

In your field definition, you've specified an invalid field name as the optional field to overlay. Press the "List Selections" shortcut to display a list of valid overlay fields.

Invalid parent template specified. Recursive references not allowed.

In your template definition, you've specified a parent template that is the descendant of the template you are currently editing. Attempting to reference this template creates a recursive reference.

Invalid precision specified.

For implied-decimal fields, the precision value must be less than or equal to the field size.

Invalid structure name specified. Recursive reference not allowed.

In your implicit group or Struct type field definition, you've specified a structure that contains a reference to the structure you are currently editing. Attempting to reference this structure creates a recursive reference.

Invalid tag comparison connector.

In your tag criteria, you've specified an invalid comparison connector.

Invalid tag comparison operator.

In your tag criteria, you've specified an invalid comparison operator.

Invalid value.

In your tag criteria, you've attempted to compare a numeric field with an alphanumeric literal.

Key cannot be deleted. It is defined as a "FROM" KEY in a relation.

You cannot delete the current key because it is used in a relation defined by the current structure.

Key cannot be deleted. Key reference exists.

You cannot delete the current key because it is used as a "to" key in a relation defined by another structure.

Key definition is incomplete.

You have not entered all the required information. When you define a key, you must specify the key name, the key type, and at least one segment definition.

Key type cannot be foreign. It is defined as a "TO" KEY in a relation.

You've attempted to change the type of the current key to **Foreign**. Repository won't allow this, because the key is defined as a "to" key in a relation, and only access keys can be "to" keys. Access keys are used to specify relationships between files and are true keys in the data file.

Line *line number* too long (len=*line length*, max=150)... Truncated.

A line in the definition record that is being loaded from your .INCLUDE file exceeds the maximum allowed. It was truncated.

Load fields information is incomplete.

You haven't entered all the required information. When you load fields, you must specify the .INCLUDE file and the record number.

Maximum field size (99999) has been exceeded.

The current field specification is larger than the maximum field size of 99,999. (If the field is arrayed, this is the maximum for each array element.)

Maximum implied-decimal field size (28) has been exceeded.

The current implied-decimal field specification is larger than the maximum implied-decimal field size of 28.

Maximum key size (255) has been exceeded. Data beyond 255 will be ignored.

The current key specification is larger than the maximum key size of 255. ReportWriter will use only the first 99 bytes of the key.

Maximum number of alias structures (999) has been defined.

You attempted to define more than 999 aliases. You cannot add any more alias definitions to the repository until you delete one or more.

Maximum number of enumerations (9999) has been defined.

You attempted to define more than 9,999 enumerations. You cannot add any more enumeration definitions to the repository until you delete one or more.

Maximum number of field references (6000) has been defined for template *template name*.

A field reference documents each time a template is used by a field. The maximum number of field references per template is 6,000. You cannot assign this template to any more fields until you remove it from one or more other fields.

Maximum number of fields (999) has been defined.

You attempted to define more than 999 fields. You cannot add any more field definitions to the current structure until you delete one or more.

Maximum number of fields (999) has been loaded.

Your .INCLUDE file contains more than 999 fields to load. The remaining fields have been ignored.

Maximum number of file references (200) has been defined for structure *structure name*.

You attempted to assign the specified structure to more than 200 files. You cannot assign the current structure to any more files until you disassociate it from one or more files.

Maximum number of files (9999) has been defined.

You attempted to define more than 9,999 files. You cannot add any more file definitions to the repository until you delete one or more.

Maximum number of formats (9999) has been defined.

You attempted to define more than 9,999 global formats. You cannot add any more global format definitions to the repository until you delete one or more.

Maximum number of key references (200) has been defined for structure *structure name*.

A key reference documents each time a structure is used as a "to" structure in a relation and each time a structure is used in an external key segment definition. The maximum total number of these two types of references is 200 per structure. You cannot define any more external key segments or relations for the specified "to" structure until you reduce the number of key references.

Maximum number of keys (99) has been defined.

You've attempted to define more than 99 keys for the current structure. You cannot add any more key definitions to the structure until you delete one or more.

Maximum number of members (100) has been defined.

You've attempted to define more than 100 members for an enumeration. You cannot add any more member definitions to the enumeration until you delete one or more.

Maximum number of relations (99) has been defined.

You've attempted to define more than 99 relations for the current structure. You cannot add any more relation definitions to the current structure until you delete one or more.

Maximum number of structure-specific formats (250) has been defined.

You've attempted to define more than 250 structure-specific formats for the current structure. You cannot add any more format definitions to the structure until you delete one or more.

Maximum number of structures (200) has been assigned to the file.

You've attempted to assign more than 200 structures to the current file. You cannot assign any more structures to the file until you disassociate one or more structures.

Maximum number of structures (9999) has been defined.

You've attempted to define more than 9999 structures. You cannot add any more structure definitions to the repository until you delete one or more.

Maximum number of tags (10) has been defined.

You've attempted to define more than 10 tags for the current structure. You cannot add any more tag definitions to the structure until you delete one or more.

Maximum number of template references (3000) has been defined for parent template *template name*.

A template reference is created each time a parent template is used by another template. The maximum number of parent references per template is 3000. You cannot assign this parent template to any more templates until you remove it from one or more other templates.

Maximum number of templates (9999) has been defined.

You've attempted to define more than 9999 templates. You cannot add any more template definitions to the repository until you delete one or more.

Maximum precision (28) has been exceeded.

The current precision specification is larger than the maximum precision size of 10.

Missing array dimension.

An array dimension is missing. For example, you may have specified **Dim1** and **3**, but not **2**. Array dimension specifications must be contiguous.

Missing field name.

You've defined a key segment of type **Field**, but you haven't entered a field name in the **Field name or Literal** column.

Missing group member definition. At least one member must be defined.

You have not defined any members for the current group field. Define one or more group members by selecting "Edit Group Members" from the Edit Field Functions menu, or clear the **Group** field.

Missing member definition. At least one member must be defined.

You attempted to save an enumeration definition containing no members. Enumerations must contain at least one member.

Missing segment definition.

You skipped a key segment definition. Segment definitions must be contiguous.

Missing segment definition. At least one segment must be defined.

You have not defined any segments for the current key. Segment number 1 must be defined.

Missing segment type.

You entered a field name, a structure, and/or a literal in the appropriate column(s), but you haven't selected a segment type in the **Seg type** column.

Missing structure name.

You've defined a key segment of type **External**, but you haven't entered an external structure name in the **Structure name** column.

Modifying enumeration *enumeration name* will affect one or more template and field definitions. Are you sure you want to save your modifications?

A change to the current enumeration data could affect field definitions that are .INCLUDEd into Synergy DBL source files or referenced in xfServerPlus method catalog definitions. If you don't want to save the modifications, select No.

Modifying template *template name* will affect one or more template, field, and key definitions. Are you sure you want to save your modifications?

A change to the current template data will cause all affected field and template definitions to be modified. In addition, all keys (the size and type of keys in the current structure and external segments in other structures) that use those fields will be updated. The record sizes of all affected structures will be updated as well. If you don't want to make such modifications, select No.

Moving the current field will invalidate its (an) overlay specification.

Either the field you're attempting to move is an overlay field whose specification would be invalid at the new location, or moving the current field would invalidate one or more other overlay fields. If your goal is to redefine overlay attributes, remove all overlay attributes before reordering the fields. When the fields are in the desired order, add the overlay information.

New repository is invalid. Correct errors in schema file and reload.

Errors have been found in the schema file being loaded by the Load Repository Schema utility. Do not use the new or copied repository. Look at the log file that was created to determine which definitions contain errors. Correct the schema file and reload it.

Original repository cannot be copied. Files are in use.

You are merging a schema into an existing repository. The merge takes place using a copy of the original repository, but the copy cannot be created because the repository is in use by another user. Try again later when the repository is not in use.

Original repository cannot be replaced. Files are in use.

You are merging a schema into an existing repository. The merge takes place using a copy of the original repository, and you have chosen to replace the original when the merge is complete. But the original cannot be replaced because it is use by another user. You can either keep trying or clear the Replace original repository check box. If you are running **rpsutl** from the command line, repeat the command and suppress (**-s**) the replacement of the original repository.

Overlaid field does not exist.

An overlay specification in your .INCLUDE file refers to a field that doesn't exist.

Print specifications are incomplete.

You have not entered all the required information. To print repository definitions, you must specify the repository filenames, the name of the file into which to print, and the print option. Additionally, if you've selected the **Single** option, you must specify the structure, file, template, and/or format name.

For detailed information on record locking in Repository, see "Record locking" on page 1-19.

Record locked. Do you want to view the definition?

You have selected to modify a structure, file, template, format, or enumeration that another user is already modifying. Only one user at a time can modify these definitions. To view the definition without being able to modify it, select Yes.

Record locked. Please try again later.

Another user is currently modifying a record that you need to modify. This message can occur in any of the following circumstances:

- Deleting a structure, file, template, format, or enumeration
- Saving changes to a structure, file, template, format, or enumeration
- Adding or copying a structure
- Adding a group
- Saving a field that is a group
- Selecting the Attributes menu for the first time after having copied a structure

Record locked. Structure references cannot be saved.

Another user is currently modifying a structure that either references or is referenced by the definition that you are attempting to modify. Try again later.

Record locked. Template references cannot be saved.

Another user is currently modifying a template or enumeration that is referenced by the definition that you are attempting to modify. Try again later.

Referenced structure cannot be the current structure.

The referenced structure name you've specified within an implicit group or Struct type field definition is the same as the current structure name.

Relation cannot be examined. Specifications are invalid.

You've tried to examine a relation without entering all of the required information. You must specify the "from" and "to" structure and the "from" and "to" key before you can examine the relation.

Relation definition is incomplete.

You have not entered all the required information. When you define a relation, you must specify the "from" key, the "to" structure, and the "to" key.

Repository filenames cannot be the same.

When specifying the two repository files to verify and the two repository files to create, you did not enter unique filenames.

Repository files already exist. Do you want to merge the schema?

You have specified the name of an existing repository for the Load Repository Schema utility. If you want the utility to create a new repository into which to load the schema, select No and specify another repository. If you want to merge the schema definitions into the existing repository, select Yes. You must then specify how you want both new and existing definitions to be handled.

Schema specifications are incomplete.

You have not entered all the required information. To generate the schema (Synergy Data Language), you must specify the repository filenames and the name of the file into which the schema is to be generated. To load the schema, you must specify the name of the file that contains the schema to be loaded, the repository filenames, and the log filename.

Set specifications are incomplete.

You have not entered all the required information to set the current repository. You must enter both repository main and text filenames.

Size of selected Date type field cannot be modified.

Repository determines the field size based on the storage format selected for a Date type field. You cannot alter this size.

Size of selected Struct type field cannot be modified.

Repository determines the field size based on the structure selected for a Struct type field. You cannot alter this size.

Size of selected Time type field cannot be modified.

Repository determines the field size based on the storage format selected for a Time type field. You cannot alter this size.

"Structure" can only be specified when no explicit group members are defined.

You've specified a structure name (which designates an implicit group) when explicit group members are already defined. Either remove the structure name or delete the explicit group members.

Structure cannot be deleted. Field reference exits.

You cannot delete the current structure because it is referenced within one or more implicit group or Struct type field definitions. Use the Print Repository Definitions utility to see where it is referenced.

Structure cannot be deleted. File reference exists.

The current structure is assigned to one or more file definitions. You cannot delete it until you disassociate it from all files. Use the Print Repository Definitions utility to see where it is referenced.

Structure cannot be deleted. Key reference exists.

Either this structure is used as a "to" structure in a relation defined by another structure, or a field in this structure is defined as an external key segment by another structure. You cannot delete the structure.

Structure definition is incomplete.

You haven't entered all the required information. When you define a structure, you must specify the name, file type, and description.

Structure has been modified. Do you want to save changes?

Whenever you modify a structure's attributes, a temporary copy of that structure is created. As a result, you have the opportunity to abandon any changes you've made since entering the "Edit Attributes" function.

Structure is assigned to a file. File type cannot be modified.

You cannot modify the file type of this structure, because it must match the file type of the file it's assigned to. Before you can change a structure's file type, you must disassociate it from all files.

Structure is referenced as a group, and therefore at least one field must be defined.

The current structure is referenced within an implicit group specification. Groups must contain at least one member. Use the Print Repository Definitions utility to see where the structure is referenced.

Structure no longer exists.

The structure you tried to select was deleted by another user between the time it was displayed in a list and the time you selected it.

Structure *structure name* is referenced as an implicit group or structfield. Modifying it will affect one or more field and key definitions. Are you sure you want to save your modifications?

A change to the current structure data will cause all affected field definitions to be modified. In addition, all keys (the size and type of keys in the current structure and external segments in other structures) that use those fields will be updated. The record sizes of all affected structures will be updated as well. If you don't want to make such modifications, select No.

Structure's primary key does not match those of other assigned structures.

If more than one structure is assigned to a file, the primary key definition must be the same for all assigned structures. (The primary key is assumed to be the first key defined and must be an access key.) The following primary key information will be compared: key size; sort order; dups allowed flag; key data type; number of segments; and type, position, length and order of each segment. You will get this message if the key information is not the same.

Structures are assigned to this file. File type cannot be modified.

You cannot modify the file type of this file, because it must match the file type of the structures assigned to it.

Tag definition is incomplete.

You haven't entered all the required information. When you define a field type tag, you must specify the field name and the comparison operator.

Tag fields are defined for this structure. Tag type cannot be modified.

You cannot modify the tag type of this structure because there are tag fields defined. Before you can change the structure's tag type, you must delete all tag definitions.

Template cannot be deleted. Field reference exists.

You cannot delete the current template because one or more field definitions use it. Use the Print Repository Definitions utility to see where it is referenced.

Template cannot be deleted. Template reference exists.

You cannot delete the current template because one or more template definitions use it as a parent, which means you cannot delete it. Use the Print Repository Definitions utility to see where it is referenced.

Template definition is incomplete.

You haven't entered all the required information. When you define a template, you must specify the name, data type, and size.

Template name and parent template cannot be the same.

The parent template name you've specified is the same as the current template name. The parent must be a template other than the current one.

Template no longer exists.

The template you attempted to select was deleted by another user between the time it was displayed in a list and the time you selected it.

This field is defined as a key. All affected keys in the current structure will be updated when you save changes. Are you sure you want to make modifications?

A change to the current field definition will cause the key size and type of all affected key definitions in the current structure to be modified. In addition, the current structure's record size will be updated. If you don't want to make modifications, select No.

This field is defined as an external segment by another structure. All affected keys in all external structures will be updated when you save changes. Are you sure you want to make modifications?

A change to the current field definition will cause the key size and type of all affected key definitions in external structures to be updated. The record sizes of all affected structures will be updated as well. If you don't want to make modifications, select No.

"TO" KEY must be an ACCESS key.

You've tried to specify a foreign key as a "to" key in a relation. Only true keys in the data file (that is, access keys) can be used as "to" keys.

Validation specifications are incomplete.

You have not entered all the required information to validate the repository. You must specify the repository filenames to validate, as well as the log filename.

Verification specifications are incomplete.

You have not entered all the required information. To verify the repository, you must specify the repository filenames to verify, the repository filenames to create (if inconsistencies are found), and the log filename.

D

Data Formats

The format specification is a sequence of characters that depicts how the contents of the field will look. The characters listed in the table below have special meanings. Any other character that appears in a format string is inserted directly into the field.

If the format size (number of data characters) is larger than the field size, an alpha field will be left-justified and the format string truncated, while a decimal field will be right-justified and the format string truncated. If the format size is smaller than the field size, the data will be truncated.

Character	Meaning
@	Represents a single alpha character.
X	Represents a single digit. It causes a digit from the source data to be inserted into the specified position in the field. Digits are extracted from the source data beginning at the right and continuing to the left; the rightmost X in the format is loaded with the rightmost digit in the source data. If there are more X characters than there are significant digits in the source data, the leftmost X positions are loaded with zeros.
Z	Represents a single digit. Z differs from X if there are more Z characters than significant digits to be transferred from the source data. The Z position is loaded with a blank if no X character or period appears to the left of it in the format. If a period appears to the left but not to the right, the Z position is loaded with a zero. A zero is also loaded if an X appears to the left of the Z.
*	Represents a digit position. The * is similar to an X, except that when there are no more significant digits, the position is loaded with an asterisk rather than a zero.
money sign	Represents a digit position. If there are more significant digits to be transferred, the position is loaded with the next digit. If no more significant digits remain to be transferred, the position is loaded with a blank. The rightmost money character is changed to a dollar sign when field loading is concluded. The default money sign is “$”, but you can use the MONEY subroutine to change it to any character. See MONEY in the “System-Supplied Subroutines and Functions” chapter of the *Synergy DBL Language Reference Manual*.

Character	Meaning
–	Causes a minus sign to be inserted at the corresponding position in the field if the – is the first or last character of a format. If the source data is positive, a blank will be inserted. If the – is used within the format, a hyphen is inserted at the corresponding position in the field.
.	Causes a decimal point to be inserted at the corresponding position in the field. In addition, any Z that appears to the right of the decimal point is treated as an X.
,	Causes a comma to be inserted at the corresponding destination position, but only if one of the following conditions is true: ▸ More significant source digits remain to be transferred. ▸ There is an X character to the left of the comma.

E

Distributed Shortcuts

The table below lists some of the Repository shortcuts as they appear in your original distribution.

Menu entry	VT-style shortcut	PC-style shortcut
Abandon	CTRL+A	CTRL+A
Access Template Overrides	F12	SHIFT+F2
Add Field (File, Format, Group, Key, Relation, Structure, Tag, Template)	INS	INS
Assign Structures	CTRL+G	CTRL+G
Copy Field (File, Format, Key, Relation, Structure, Tag, Template)	F18	SHIFT+F8
Delete Field (File, Format, Key, Relation, Structure, Tag, Template)	REM	DEL
Edit	ENTER	ENTER
Edit Attributes	PF3	F3
Edit Display Information	CTRL+H	CTRL+H
Edit Group Members	CTRL+G	CTRL+G
Edit Input Information	CTRL+W	CTRL+W
Edit Validation Information	CTRL+V	CTRL+V
Examine Key (Relation)	CTRL+E	CTRL+E
Exit	PF4	F4
First Entry	CTRL+F	CTRL+F
Help	PF1	F1
Last Entry	CTRL+L	CTRL+L

Menu entry	VT-style shortcut	PC-style shortcut
List Selections	F7	F7
Load Fields	F9	F9
Next Tab	TAB	CTRL + TAB
Previous Tab	F8	CTRL + SHIFT + TAB
Reorder Fields (Formats, Keys, Relations, Tags)	F17	SHIFT + F7
Toggle View	CTRL + V	CTRL + V

Glossary

access key — Represents a true key in a data file and is used to specify relationships between files.

alias — An alternate name for a structure or field. Your repository can contain aliases for your Synergy DBL identifier names.

arrayed field — A field definition from the repository that represents a group of fields, each of the same size and data type. Arrayed fields can have between one and four dimensions.

attributes — The characteristics of a structure that describe fields, keys, relations, tags, and redisplay formats.

autokey — A DBL ISAM key that is automatically filled in by Synergy DBMS with the appropriate value. Supported autokey types are SEQUENCE, TIMESTAMP, and CTIMESTAMP (creation timestamp). They must be 8 bytes and defined as read-only. The corresponding data types are AutoSeq and AutoTime (used by both timestamps).

definition file — Contains standard Synergy DBL local, common, or global record layouts.

enumeration — A set of related values. An enumeration has a name and one or more members, which may have values assigned to them.

explicit group — A group in which the members are defined within the Field Definitions list. See also implicit group.

external key — An external key allows for a key to be composed of a segment from the defining structure and one or more fields from another structure or structures.

field — A named area of computer memory used to store a specific type of data.

field header — The text that appears at the top of the column for a field (if you choose to print page headers) on each page of a report in ReportWriter. The default field header is the header that you specify during the “Define Fields” phase of repository maintenance.

file A physical data file.

file definition Describes a particular file and defines which structures can be used to access it.

file type Defines a specific class of data file (for example, ASCII or DBL ISAM). File types are assigned to structures and file definitions. User-defined file types enable developers to supply subroutines to handle I/O for file types that are not supported by ReportWriter.

foreign key A key used to specify relationships between files but which does not have to be a true key in the data file. Foreign keys may consist of one or more literal segments. Foreign keys may also consist of one or more external segments (fields from another structure).

format Designates the way a field will be displayed in your report. Global formats are available for use by any field definition in Repository and also in ReportWriter. Structure-specific formats are defined and only valid for a particular structure. Predefined formats for date and time fields are built into every repository.

“from” key Refers to the key in the structure that is defining a relation. The “from” key may be an access key or a foreign key.

“from” structure Refers to the structure that is defining a relation.

group A structure within a structure, as defined by the GROUP statement. Fields or other group definitions can be members of a group.

implicit group A group in which the members are defined by referencing another structure. See also explicit group.

information line A single line at the bottom of the screen body that Repository uses to display messages and general information.

input window A window that can contain text, input fields, and buttons, in which the user enters information.

key The portion of a data record that identifies and is used to access the record. A key can be composed of discontiguous data segments from within the record.

literal A specific value that represents itself (as opposed to a variable). Both numbers and text can be literal values.

modifiable list A list of entries that you can edit. You can also add, delete, and sometimes move entries in the list.

name link — Used to associate fields with access keys in other structures. These associations can then be used by ReportWriter to access related files.

non-modifiable list — A list of entries that you can only select from.

numeric type — Refers to decimal, implied-decimal, and integer types.

overlaid field — A field that is "covered" by one or more overlay fields. It must precede the overlay field(s) in your field definition list.

overlay field — A field that lays on top of another field (the overlaid field) so that the two fields share all or part of the same data space. For example, if your overlaid field is a date, your overlay fields might be the month, the day, and the year.

overlay offset — The overlay offset is the amount to add to the first character in the overlaid field to determine where the overlay begins. (For example, if the overlay begins at the first character of the overlaid field, the offset is 0; if the overlay begins at the second character, the offset is 1; and so on.)

parent — A template referenced by another template. Once a template references another template, the attributes of the parent template are inherited.

precision — The number of places after the decimal point in an implied-decimal field.

pseudo array — A single-dimension array of type **a**, **d**, **d.**, **i1**, **i2**, **i4**, **i8**, **p**, or **p.** (for example, **10a5**).

real array — A single or multi-dimensional array of type **a**, **d**, **d.**, **i1**, **i2**, **i4**, **i8**, **p**, or **p.** that is specified in square brackets immediately preceding the data type (for example, **[10]a5**).

record — A unit of data. Each record is divided into one or more fields.

relation — Enables you to link the keys of one structure with the keys of other structures.

rendition — A combination of the display attributes and color of portions of the screen.

renditions file — Contains predefined and default renditions. Renditions files can be defined and modified with UI Toolkit.

repository — Where your data definitions are stored.

Repository The Synergy/DE application that orders and defines your data structures, files, and attributes.

schema A description of a repository written in Synergy Data Language.

script information The data associated with a field that affects how the field is used in UI Toolkit input windows. When a field is defined, Repository creates default script information for that field.

selection window A window that contains a choice of one or more entries that can be selected (usually with arrow keys).

struct A data type, which is represented as a structfield in code and definition files.

structure A record definition or compilation of field and key characteristics for a particular file or files.

tag The information that uniquely identifies a particular record structure (or record type) in a multi-record structure file. A structure tag can be either the record size or one or more particular fields in the structure and their associated comparison values.

template A set of field characteristics that can be assigned to one or more field definitions or templates.

“to” key The key in the structure which is being related to in a relation. The “to” key must be an access key.

“to” structure The structure being related to in a relation.

Index

D

N

S

T

X

Z