Thoroughbred® Script-IV™
Developer Guide

Version 8.8.0

46 Vreeland Drive, Suite 1 • Skillman, NJ 08558-2638
Telephone: 732-560-1377 • Outside NJ 800-524-0430
Fax: 732-560-1594

Internet address: **http://www.tbred.com**

Published by:
Thoroughbred Software International, Inc.
46 Vreeland Drive, Suite 1
Skillman, New Jersey 08558-2638

Document Number: SD8.8.0M101

INTRODUCTION

Thoroughbred Dictionary-IV is an application development environment that contains a set of integrated tools designed to remove much of the burden from programming. Thoroughbred Script-IV is a high-level programming language included in the Dictionary-IV product set.

Operating System Support: UNIX, Linux, OpenVMS, and Windows
For specific information, please contact your Thoroughbred Sales Representative.

Overview of Thoroughbred Script-IV

Script-IV is a fourth-generation programming language that provides a set of structured commands. The language was designed to be concise but comprehensive. Because the commands are based on English language constructs, most software developers find them easy to learn and easy to remember.

The language enables developers to write task-oriented code. A script tells the system what to do but it does not necessarily specify how to do it. Script-IV determines how to manage most of the details.

For example, suppose that you want to write a script that will close all open purchase orders with a balance of less than 500 dollars. The following Script-IV code fragment accomplishes that task:

```
PRINT SCREEN POCHECKS
CHANGE PODATA USING
   KEY RANGE FROM FIRST TO LAST
   SELECT WHEN STATUS = "OPEN" AND
      PURCHASE-AMOUNT <500
   PROCESSING IS CLOSE-PORDERS
```

The sample code above uses system resources defined in Dictionary-IV. Script-IV can access the Dictionary-IV database and system dictionary, which contain the following system resources:

Format defines the format of a logical group of data or the physical record layout of a data file.

View defines a way of specifying and displaying multiple data records.

Help defines context-sensitive, on-line help for menus, data entry, and messages.

Screen defines a screen that is displayed on the terminal. The sample code fragment above uses the **POCHECKS** screen.

Link specifies a sort, text, or data file and links it to a format, screen, or view. The sample code fragment above uses the **PODATA** link.

Menu provides a list of items to select.

Messages contain common messages that can be used by scripts.

These resources enable developers to write terse code and enable Script-IV to manage the implicit details, such as how information will be displayed. Because Script-IV was designed as a flexible language, developers can override Dictionary-IV defaults and specify explicit sets of instructions. For more information on the resources listed above see to the Dictionary-IV Developer Guide.

Script-IV works in concert with other Dictionary-IV components. Developers can use Source-IV to write, compile, and edit scripts. A script can execute reports created under Report-IV and execute queries created under Query-IV.

Script-IV was designed to provide alternatives to third-generation programming techniques, but it also provides an interface to the Thoroughbred Basic third generation language. Developers can migrate existing Thoroughbred Basic applications to Script-IV or use scripts to integrate existing Thoroughbred Basic programs with Script-IV.

Script-IV was designed to help developers create, maintain, and enhance applications:

- Script-IV enables developers to produce readable, self-documenting code.
- Script-IV is dictionary-driven. Resources can be defined once in a common dictionary and shared among applications. If a developer needs to change a resource definition, the change applies to every script that uses the definition.
- Script-IV supports a multiple spoken-language interface.
- Script-IV can be run with Thoroughbred VIP and Gateway for Windows for display and full use under the Microsoft Windows graphical user interface.

For More Information

- about creating applications please refer to the Thoroughbred Dictionary-IV Developer Guide.
- about software conventions and databases, refer to the Dictionary-IV User Guide.
- about using an Dictionary-IV product please refer to the individual product reference manual. The Query-IV, Report-IV, Script-IV, and Source-IV Reference Manuals are available from Thoroughbred Software.
- about Thoroughbred Basic please refer to the Thoroughbred Basic Reference Manual.
- about Thoroughbred Basic Utilities please refer to the Thoroughbred Basic Utilities Manual.

CREATING SCRIPTS

The Script-IV Editor and Compiler

Scripts are created, maintained and compiled using Source-IV. Source-IV is a source code management system distributed with Dictionary-IV.

For details on how to maintain and compile Scripts in Source-IV, please refer to the Source-IV manual.

It is **strongly recommended** that all scripts be imported to Source-IV. For details on how to import script into Source-IV, see the Source-IV manual.

A few advantages to Source-IV include:

- Compiler includes more robust run time error processing and debugging
- Compiler supports compiling by range
- Editor supports source locking and edit history including edit history rollback
- Editor supports copy and paste
- Editor supports search and replace by module or library
- Editor supports F3 on a #format_name to display the format definition
- Supports “S” type Script Methods for OPENworkshop

For details on how to maintain and compile Scripts using the older IDOL Script editor, please refer to the appendix later in this manual.

How to Structure a Script

Before you enter a script into Source-IV or the script editor you must consider the enforced structure and the optional structure of the script. Enforced structure is the set of requirements a script must meet before it can be compiled and executed. Optional structure is the set of techniques you can use to make scripts more readable and easier to maintain.

Enforced Structure

Enforced structure includes the following requirements:

- identifying the beginning of a script.

- identifying the procedures and commands in a script. Data declarations, procedure names, and the **.LONGVAR**, **.SHORTVAR**, and **.PREC** commands must begin in the leftmost column. All other Script-IV commands must begin at least one tab stop from the left margin. Command elements must be separated by at least one space.

Scripts are divided into the Data Environment Section, which is optional, and the Procedures Section, which is required for most scripts. Only the Type 5 (copy) script, does not require a Procedures Section. The Data Environment Section and the Procedures Section are described below.

DATA ENVIRONMENT SECTION

This section is located at the beginning of the script. If you plan to use dictionary definitions, such as formats, screens, views, or links, you must declare them in this section. The type of script you plan to write and the order of the declarations of dictionary definitions help to create the data environment for a script. For more information on script types, see the section on **Different Types of Scripts** in this manual.

Data Declarations

Each data declaration, except for **DN**, consists of a data declaration command followed by the name of a dictionary definition or 4GL dataname. The command must begin in the leftmost column of the screen and the definition name must be indented at least one tab stop on the same line. For example:

```
VN 4SCUST, 4SSLSRP, 4SINVEN
LN 4SSALDT
SN 4STOPSC1, 4STOPSC2, 4SBOTSCR
DN INPUT-FLAG (1), VIEW -FLAG (1),
DN TEXT-FLAG (1)
DN TAX-RATE (2.0)
```

VN, **LN**, **SN**, and **DN** are data declaration commands. Each data declaration, except for the **DN** declarations, specifies at least one dictionary definition or 4GL dataname. More than one declaration is allowed on a line if the definition names are separated by commas or spaces.

The order of data declarations is important. When a script is compiled, the Script-IV compiler builds a data name table. Data names are placed in the table as they are encountered. In the example above, the **4SCUST** view defined in the **VN** statement is associated with the **4SCUST** format and the **4SSLSRP** view is associated with the **4SSLSRP** format. When a script that uses these data declarations is compiled, the data names defined in the **4SCUST** format precede the data names defined in the **4SSLSRP** format.

There is a possibility that duplicate data names will be placed in the table. When a script specifies a data name, the compiler searches for the first occurrence of the data name in the table. Using the example above, the **4SCUST** format and the **4SSLSRP** format both contain the **SALES-REP-CODE** data element. A script that specifies **SALES-REP-CODE** accesses the value defined for **SALES-REP-CODE** in the **4SCUST** format because **4SCUST** is declared before **4SSLSRP**.

To access a different occurrence of the data name, the data name must be qualified by preceding it with a format name and a period. Using the example above, to access the value of the **SALES-REP-CODE** data element contained in the **4SSLSRP** format, the script must specify the data name as **4SSLSRP.SALES-REP-CODE**. However, to access the value of the **SALES-REP-CODE** data element contained in the **4SCUST** format, the script can specify **SALES-REP-CODE** or **4SCUST.SALES-REP-CODE**.

For more information on data declaration commands see the Script-IV Language Reference. For more information on dictionary definitions see the Dictionary-IV Reference Manual.

Compile-Time Definitions

Compile-time definitions contain data that is resolved when the script is compiled. They include the definition names specified in data declarations and script names used in **INCLUDE** commands:

Definition	**Data Declaration Command**
Data Name	**DN**
Format Name	**FN**
Link Alias	**LA**
Link Name	**LN**
Screen Name	**SN**
View Name	**VN**

Definition	**Command**
Script Name	**INCLUDE**

When specified in Script-IV commands, compile-time definitions are not enclosed by quotation marks. They cannot be used as parameters or passed to a command in a data name or variable. They must be fully specified, for example, **OPEN SCREEN CUSSCRN1**.

Run-Time Definitions

Run-time definitions contain data that is resolved when the script is executed. They include all program names, on-line help, message dictionary definitions, and script names except when used in the **INCLUDE** command:

Definition	Command
Message Dictionary	All Applicable
On-Line Help	All Applicable
Program Name	All Applicable
Script Name	All except **INCLUDE**

You do not have to declare these definitions before they are specified in Script-IV commands. They can be used in string constants, variables, data names, or expressions. These definitions can be soft-coded in your script, for example, OPEN MESSAGES "ARMSGS" or OPEN MESSAGES MESSAGE-LIST.

Data Names

Data names can be defined to hold string, integer, or decimal data. Additional attributes can be specified. Data names only hold a single type of data. For example, if **CUS-NAME** is defined, it is handled as a single element. If it needs to be handled as a first and last name, you must define it as two parts, for example, **CUS-NAME-FIRST** and **CUS-NAME-LAST**.

Data names can be used in expressions, functions, assignments, and calculations. A data name is limited to a length of 20 characters. Valid characters are uppercase and lowercase alphabetic characters, numerals, the hyphen character, and the underscore character. Data names must not contain a period or any special characters that may cause a conflict in Script-IV syntax. Data names must not conflict with any procedure names, declared format, link, screen, or view names, or keywords. If a data name has the same name as a system variable, the data name takes precedence.

Because data names are treated as single elements in scripts, substring operations cannot be performed on data names. However, the value in a data name can be moved to a variable on which string operations can be performed. Data names and variables can be used interchangeably in Script-IV syntax, except when specifically stated otherwise.

The four types of data names that can be used in Script-IV are local format data names, global format data names, link alias data names, and local variables.

Local Format Data Names

Local format data names defined in a format can be used in a script if the format has been declared by an FN command. These local format data names are dictionary-based because the format definition resides in Dictionary-IV.

When used in a script, these data names can, and in some cases must, be qualified by the format name. The name of the format must precede the data name delimited by a period, for example: **ARFORMAT.CUS-NUMBER**. This is necessary if your script uses data names that match in multiple formats.

For more information on local format data names see the description of the **FN** command in the Script-IV Language Reference.

Global Format Data Names

Global format data names are also defined in a format that resides in Dictionary-IV. The global format name always consists of a **#** (pound sign) followed by three to eight characters.

The global format is not declared in the Data Environment section, but there are a number of ways to include global format data names in a script. For more information on global format data names see the section on **Formats and data names** in the Thoroughbred Basic Reference Manual.

The data in a global format is separate and independent from the data in a local format.

Link Alias Data Names

For each link alias declared in the script, a duplicate format with an additional set of matching data names is available to the script. To be accessed, these data names must be qualified by the link-name-alias rather than the format name. For more information on link alias data names see the description of the **LA** command in the Script-IV Language Reference.

Local Variables

Local variables can be defined in a script using the **DN** data declaration command. These local variables do not reside in the dictionary but are handled much like a local format data name. They are not qualified by a format name or other name, and therefore must be unique among any other local variables or data names used in a format. For more information on the **DN** data declaration command see the description of the **DN** command in the Script-IV Language Reference.

Physical and Logical Formats

During file maintenance, formats are used to access data in files. The format is linked to a physical data file through the link definition. This is a physical format, which describes a record and the data elements in the record along with element characteristics, defaults, valid values, and related data entry restrictions.

In scripts, a format can be used independently of a data file or link. This is a logical format, which can function like a data name or a variable. It can be assigned a value, its value can be printed or passed to another script, and it can generally be manipulated as an item of data in several commands.

The ability to manipulate a logical format provides greater freedom in script design. The following two examples each describe a different way to collect two data records and write them to two files.

Formats for Examples 1 and 2:

```
Format ONE(for screen ONE and LINK-ONE):

CUS-CODE
CUS-NAME
CUS-ADDRESS
SLS-CODE

Format TWO(for screen TWO and LINK-TWO):

SLS-CODE
SLS-NAME

Format ABC (for screen ABC; no link):

CUS-CODE
CUS-NAME
CUS-ADDRESS
SLS-CODE
SLS-NAME
```

The first example uses two independent screens and two physical formats.

```
Example 1:

PRINT SCREEN ONE
INPUT SCREEN ONE
ADD LINK-ONE
PRINT SCREEN TWO
INPUT SCREEN TWO
ADD LINK-TWO
```

Screen ONE is printed, then used to input data into format ONE. The data in format ONE is added to the data file using LINK-ONE. The same procedure is performed for screen TWO, format TWO, and LINK-TWO. In this example, the two screens can be displayed and manipulated independently of each other.

The second example accomplishes the same task as the first example using one screen with one logical format and two physical formats.

```
Example 2:

PRINT SCREEN ABC
INPUT SCREEN ABC
LET ONE = ABC
ADD LINK-ONE
LET TWO = ABC
ADD LINK-TWO
```

Screen ABC is printed then used to input data into format ABC. Format ABC contains data names from two different physical formats: ONE and TWO. Format ONE is loaded with data in a format assignment statement, and the data in format ONE is added to the data file using LINK-ONE. Format TWO is loaded with data in a format assignment statement, and the data in format TWO is added to the data file using LINK-TWO. In this example, a single screen is used to collect data.

Constants

Constants are data elements that do not change value during script execution. Constants are also called literals because values such as **1.25** or **"string"** are literal values. There are two types of constants: .

- Numeric constants can be positive or negative numerals in integer, fixed point, or floating-point format.
- String constants include ASCII characters delimited by quotes, such as **"ABC"**, or hexadecimal values delimited by dollar signs, such as **414243**.

For more information on constant values see the Thoroughbred Basic Reference Manual.

Elastic Variables

Elastic variables contain values that can change during execution. Elastic variables can dynamically change length. They are not declared in the data declaration area. Do not confuse local variables, which are dictionary-based data names, with the elastic variables described below.

Elastic variables are not dictionary-based. They provide another way of holding and manipulating data. Elastic variables do have the requirements of the data name definitions described above, but elastic variables do not promote data independence. They can be used to manipulate strings, to create temporary work areas, and to fulfill various formatting requirements.

Data names and elastic variable names can be used interchangeably in Script-IV syntax, except when specifically stated otherwise.

The types of elastic variables that can be used in Script-IV are elastic numeric variables and elastic string variables.

Elastic Numeric Variables

Elastic numeric variables contain numeric values. These values can be integers, fixed-point numbers, or floating point numbers. You can use the LET command to assign numeric values to the elastic numeric variables you define.

For more information on how to define elastic numeric variables see the Thoroughbred Basic Reference Manual. To establish control of naming conventions for elastic variables please refer to the descriptions of the **.LONGVAR** and **.SHORTVAR** commands in the Script-IV Language Reference. To specify how integer values are rounded please refer to the descriptions of **.PREC** and **PRECISION** in the Script-IV Language Reference.

Elastic String Variables

Elastic string variables contain string values. These values are non-numeric values that can range up to 65000 bytes long. You can use the **LET** command to assign string values to the elastic string variables you define.

For more information on how to define elastic string variables see the Thoroughbred Basic Reference Manual.

Restrictions on Elastic Variables

Under some circumstances you cannot define elastic variables with certain names. This restriction applies to 8INPUT Pre/Post Processing scripts and File Maintenance Pre/Post Processing scripts. For a list of restricted elastic variable names, please refer to on-line documentation.

You cannot define a Script-IV reserved word or a Thoroughbred Basic reserved word as an elastic variable name. For a list of reserved words please refer to the Script-IV Language Reference.

System Variables

System variables are numeric or string variables that Script-IV defines to help you manage certain types of tasks. In most cases, these variables interact with a Script-IV command. .

Examples of Script-IV system variables include:

VARIABLE	Interacts with
COLUMN	**INPUT SCREEN**
ESCAPE	**ESCAPE-KEY**
FIELD	**INPUT SCREEN**
FILE-SUFFIX	**OPEN**
LENGTH	**INPUT SCREEN**
LINE	**INPUT SCREEN**
MENU-PARMS	Not Applicable
SYSTEM-DATE	**SET**
SYSTEM-TIME	**SET**
TERM-KEY	**INPUT MESSAGE**
	INPUT SCREEN
TERMINAL-DATE	**SET**
TEXT-END	**READ**

For more information on these system variables see the descriptions in the Script-IV Language Reference.

PROCEDURES SECTION

The procedures section consists of at least one independent procedure that contains Script-IV commands. Procedures are the main body of the script.

The first procedure in a script is the main procedure and controls all other procedures. When the main procedure is completed, the script automatically terminates. If there are no commands in the main procedure, the script will automatically terminate without processing further procedures.

Give some thought to the design and organization of script procedures. Long-term productivity can be increased by spending time in the analysis and design phases of product development. Prior planning can decrease development time, enhance script readability, and facilitate script maintenance.

A procedure consists of a procedure name followed by one or more script commands.

Procedure Names

The procedure name:

- Identifies the body of commands as a unique procedure within the script.
- Must be different from all other procedure names in the script or in any included script. The procedure name must not conflict with any reserved words.

- Must begin in the leftmost column of the screen and must appear on a line by itself. However, comments preceded by ! (exclamation point) can follow the procedure name.

- Can be from 1 through 64 characters long. It cannot contain space characters. The first 20 characters must be unique.

- Must not be broken or fall onto two lines when referred to by a command.

- Can consist of uppercase or lowercase characters, numerals, and the - (hyphen) character.

Script-IV Commands

Script-IV commands can be grouped into a procedure that performs a task. Since the commands tell the system what to do rather than how to do it, the procedure is self-documenting.

Commands perform operations on data elements such as constants and variables, control input and output, and specify how scripts are executed and processed. For example:

- To assign a value to a variable, you can use the **LET** or **SET** command.

- To specify a branch in execution, you can use the **IF/THEN/ELSE/ENDIF** command.

- To control file access, you can use the **OPEN**, **LOCK**, **UNLOCK**, and **CLOSE** commands.

- To control I/O to disk, you can use the **ADD**, **CHANGE**, **DELETE**, or **READ** command.

- To use Dictionary-IV definitions, you can use the **INPUT SCREEN**, **INPUT MESSAGE**, **PRINT HELP**, **PRINT VIEW**, or **CONNECT** commands.

Commands must be indented at least one tab stop from the first column on the screen. Additional indention and line spacing can be used for readability.

Commands are sensitive to spaces. Use a space to separate all elements within a command such as the command, clauses, options, parameters, data names, and so on. This is required for the compiler to accept the commands and, if not used, may cause a syntax error. The only exceptions to this are the **( )** (parentheses) characters when they are used for grouping. In this case, a space precedes but does not follow a left parenthesis, and a space follows but does not precede a right parenthesis.

Optional Structure

Optional structure, or appearance, includes the following:

- command indention
- line spacing
- punctuation

Although optional structure does not affect script compilation and execution, it can have an impact on readability and maintenance. Because optional structure provides flexibility, it is important to set standards of consistency for your scripts.

The Script-IV language is a self-documenting language designed for readability and easy maintenance. Command syntax is simple and descriptive. Software developers who prefer terse code can build readable procedures. However, script readability can be enhanced by using the following options.

Command Placement

More than one command can be placed on one line of a script, but this coding style can produce procedures that are hard to read. Starting a Script-IV command on a new line helps produce readable code.

Many Script-IV commands contain clauses. Placing each clause on a new line enhances readability.

Comments

Comment or remark lines are not compiled. Any line that contains an ***** (asterisk) in the leftmost column is treated as a comment line. You can use comment lines as dividers to separate procedures or segments within a procedure. For example:

```
*----------------------------------------
* Customer Record Maintenance
*----------------------------------------
```

Optional Syntax Elements

Optional syntax elements do not affect command function; they enhance readability. Three common optional elements are **IS**, **ARE**, and **PROCESS**. For example, the following two clauses perform the same function:

```
MISSING TOTALS-PROCEDURE

MISSING KEY PROCESS IS
   TOTALS-PROCEDURE
```

Optional Punctuation

The **;** (semicolon) and **.** (period) can be used to mark the end of a command. The script compiler ignores these punctuation marks.

Optional Line Spacing

Blank lines can be used to separate logical segments of your script. You can separate procedures or groups of procedures from each other, or separate one command from another. Blank lines are not compiled. Because scripts are compressed before they are stored on disk, blank lines do not require storage space.

Optional Indention

You can indent commands and their subordinate clauses to highlight the clauses and display processing hierarchy. If you develop and follow your own standards for indenting, it can strengthen the structure of your scripts.

Example 1

You can use indention for a command that requires many clauses, and use a blank line to separate the command from the next command:

```
CHANGE CUSFIL USING KEY CUS-NUMBER
   BUSY PROCESS IS   BUSY-MESSAGE
   END PROCESS IS    END-OF-FILE
   PROCESSING IS  UPDATE-CUS
   TEXT "A"
      WINDOW LINE IS 15
             COLUMN IS 0
             CHARACTERS PER-LINE ARE 60
             NUMBER LINES ARE 6

IF CUS-NUMBER > "T0000" THEN
   PRINT SCREEN CUSSCRN1 CLEAR
   DO LOCAL-CUSTOMERS
ELSE
   DO COUNT-MAIL-ORDER-CUSTOMERS
   IF COUNT1 > 1000 THEN
       PRINT MESSAGE "N,150"
       IF MAIL-ORDER-FLAG = "y" THEN
           DO BULK-MAIL
       ENDIF
   ENDIF
   DO CLOSE-MAIL-ORDERS
ENDIF
```

Example 2

The first part of this example demonstrates lack of structure and poor readability:

```
IF SORT-NO = 0 THEN
   CHANGE ATAPMSTR USING KEY NEXT
   PROCESSING IS EDIT-RECORD;BUSY IS
   BUSY-RECORD;END IS END-OF-MAIN-FILE
   ELSE CHANGE ATAPMSTR USING KEY SORT
   SORT-NO NEXT
   PROCESSING IS EDIT-RECORD;BUSY IS
   BUSY-RECORD;
   END IS END-OF-MAIN-FILE
ENDIF
```

The second part of this example uses the same command and demonstrates how optional structuring can increase readability:

```
IF SORT-NO = 0 THEN
   CHANGE ATAPMSTR USING KEY NEXT
      PROCESSING IS  EDIT-RECORD
      BUSY IS        BUSY-RECORD
      END IS         END-OF-MAIN-FILE

ELSE
   CHANGE ATAPMSTR USING KEY SORT SORT-NO
   NEXT
      PROCESSING IS  EDIT-RECORD
      BUSY IS        BUSY-RECORD
      END IS         END-OF-MAIN-FILE
ENDIF
```

Different Types of Scripts

You must specify the script type when you define the script. You can select one of several different types: primary, continuation, overlay, API pre/post processing, file maintenance pre/post processing, copy, public, and utility.

The type of script determines how the script is compiled. This affects the command used to start the script, the script data environment, what will happen when the script terminates, and other execution characteristics.

In this section's description of script types, the following terms are used:

Script-IV Data Environment — The sequence and type of declared data and open message dictionaries that are shared by a script set. Declared data can include links, screens, views, formats, and local data names but not elastic variables or arrays.

3GL Data Environment — Elastic variables and arrays.

Parent Script	A primary script that is a starting point for execution of a script set. The parent script contains the initial data declarations that are common to the script set.
Script Set	A script or group of related scripts that share a common parent and Script-IV data environment.
Executing Script	A script that uses the **RUN** command to execute another script.

Type S – Script Method

A Script-IV Method written specifically for OPENworkshop. This type is preferred over Type 1. For more information please refer to the OPENworkshop manual.

Type 1 - Primary Script

The primary script is used as a starting point for processing. You can execute this script from:

- Any Dictionary-IV menu definition using the type "P".
- An Dictionary-IV menu, using the /script-name command.
- Another script, using the RUN script-name command.
- A 3GL program or Thoroughbred Basic Console Mode, using the RUN program-name directive.

This script automatically clears the screen at the beginning of execution and initializes the Script-IV and 3GL data environments. It closes all files and clears all variables, replaces any other program in memory, and automatically passes data to certain other script types. When it terminates, execution returns to the last selected Dictionary-IV menu.

Type 2 - Continuation Script

This script serves as the continuation of a primary script or another continuation script.

You can execute this script from a primary or continuation script using the **RUN** *script-name* command. This script keeps all files open, retains the values contained in variables, replaces the current primary or continuation script in memory, accepts data from a primary or continuation script, and passes data to certain other script types. When it terminates, execution returns to the last selected Dictionary-IV menu.

The Script-IV data environment is shared with the parent script and script set. It must be declared in the continuation script using the same sequence and type of data as the parent script. You can create a copy module containing the data declarations and use the **INCLUDE** command to incorporate them into any script.

The 3GL data environment is shared with the parent script.

Type 3 - Overlay Script

This script serves as an overlay to a primary, continuation, or another overlay script. It is a specialized type of continuation script that conserves memory and functions somewhat differently from a continuation script.

You can execute an overlay script from a primary, continuation, or overlay script using the **RUN OVERLAY** *script-name* command. The overlay script accepts the entire 4GL environment from the parent script and returns the environment to the parent script. This script operates in its own memory segment. When it terminates, execution returns to the executing script at the command following the **RUN OVERLAY** command.

The Script-IV data environment is shared with the parent script and script set. It must be declared in the overlay script using the same sequence and type of data as the executing script. You can create a copy module containing the data declarations and use the **INCLUDE** command to incorporate them into any script.

This script provides an independent 3GL data environment, which can include variables and numeric arrays, which is not affected by and does not affect the 3GL data in the executing script.

The **RUN** *script-name* command is not allowed in an overlay script. However, you can use the **RUN PUBLIC** *script-name* and **RUN OVERLAY** *script-name* commands.

Only one **ESCAPE-KEY** *procedure* command can be specified in an overlay script.

Type P - API Pre/Post Processing Script

This script type can be executed before or after a field is entered in file maintenance. It is designed to be executed from the **8INPUT** API, the **CONNECT SCREEN** command, or the **CONNECT VIEW** command. The script name is specified in the pre-process or post-processing attribute of a data element in a format definition.

This script is invoked from file maintenance. It cannot be invoked from another script or by the **INPUT SCREEN** command.

As an example, you can use this script type to calculate sets of numbers before a user performs data entry or use this script type to calculate sets of numbers after a user performs data entry.

When you specify the script name in the pre/post-processing attribute in the format definition, the program execution indicator (exec-indicator) must be **0** (**CALL**). If you specify a value for string-value, it will be contained in the **VAV$** variable, which is described below.

The script must contain the data declaration for the screen definition as the first line in the script. For Dictionary-IV database maintenance, you must use a custom screen definition rather than the default screen.

Following is a list of 3GL variables used by an API Pre/Post Processing Script:

VAV$ contains the string value defined in the Pre/Post processing definition.

SPARM$[ALL] is the screen array.

FPARM$[ALL] is the format array.

FD$ is the entire data record (before field edit).

]7$ is reserved for system use.

W1$ contains the field entry, which contains the entered data.

SE is the field control value.

W$ is reserved for system use.

For more information on these variables see the on-line documentation under **API Services**, **8INPUT**, **Pre/Post Proc**. For more information on the pre/post processing section of a format see the Formats section of the Dictionary-IV Developer Guide

Although this script type was designed to be executed from the **8INPUT** API, the **CONNECT SCREEN** command, or the **CONNECT VIEW** command, it can also be executed from single-record maintenance (SRM) or multi-record maintenance (MRM). For information on a script type designed to be executed from SRM or MRM, please refer to the following section.

Type 4 - File Maintenance Pre/Post Processing Script

This script type can be executed before or after a field is entered in file maintenance. It is designed to be executed from single record maintenance (SRM) or multi-record maintenance (MRM). The script name is specified in the pre-process or post-processing attribute of a data element in a format definition.

This script is invoked from file maintenance. It cannot be invoked from another script or by the **INPUT SCREEN** command.

As an example, you can use this script type to calculate sets of numbers before a user performs data entry or use this script type to calculate sets of numbers after a user performs data entry.

When you specify the script name in the pre/post-processing attribute in the format definition, the program execution indicator (exec-indicator) must be **0** (**CALL**). If you specify a value for *string-value*, it will be contained in the **S8$** variable, which is described below.

The script must contain the data declaration for the screen definition as the first line in the script. For Dictionary-IV database maintenance, you must use a custom screen definition rather than the default screen.

Following is a list of 3GL variables used by a File Maintenance Script:

S8$ contains the string value defined in the Pre/Post processing definition.

E1$ is reserved for system use.

A8$ is the screen attribute entry for the current data element:

1,1 Screen column (binary)
2,1 Screen line (binary)
3,1 Screen entry length (binary)
4,1 Fixed attribute entry number (binary)

V9 is the current screen attribute table entry number.

F3$ contains data name contents before input.

F4$ contains data name contents after input. If you want to automatically generate the data field contents before input, using preprocessing procedures sets the contents of this variable, and, when returned to maintenance, the contents will automatically be displayed.

C is the Terminal Control Value (**CTL**).

S$ contains the data record read for File Lookup.

For more information on these variables see the on-line documentation from the pre-process or post-processing attribute field of a data element in a format definition. For more information on the pre/post processing section of a format see the **Formats** section of the Dictionary-IV Developer Guide.

Although this script type was designed to be executed from single-record maintenance or multi-record maintenance, it can also be executed from the **8INPUT** API, the **CONNECT SCREEN** command, or the **CONNECT VIEW** command. For information on a script type designed to be executed from the **8INPUT** API or the **CONNECT** commands, please refer to the preceding section on the **API Pre/Post Processing Script.**

Type 5 - Copy Script

This script is not executed or compiled by itself. A copy script is a script fragment that can contain common groups of directives, functions, or a data environment. Other scripts can use the **INCLUDE** command to copy this information, which becomes part of that script at compile time. By placing common code in one location, copy scripts help you avoid duplication and make application maintenance easier.

Type 6 - Public Script

This script serves as an independent subroutine to a primary, continuation, overlay, or another public script. Having an independent data environment, the public script does not belong to a script set or have a parent script.

You can execute a public script from a primary, continuation, overlay, or public script using the **RUN PUBLIC** *script-name* command. Public scripts do not automatically pass any data and operate with an entirely independent data environment. A public script only knows what is explicitly passed to it and what it declared within it. When it terminates, execution returns to the executing script at the command following the **RUN PUBLIC** command.

This script provides an independent Script-IV data environment that is not affected by and does not affect the data environment in the executing script, but values can be passed to and returned from a public script.

A public script must contain the **ENTER PUBLIC** command as the first command line in the script after the data declaration. The 3GL data environment is initialized for a public script, except for the variables received through the **ENTER PUBLIC** command.

The **RUN** *script-name* command is not allowed in a public script. However, you can use the **RUN PUBLIC** *script-name* command.

Only one **ESCAPE-KEY** *procedure* command can be specified in a public script.

Type U - Utility Script

This script is like a primary script except that it does not automatically clear the screen at the beginning of execution and it does not initialize the Script-IV or 3GL data environments. When it terminates, execution returns to the last selected Dictionary-IV menu.

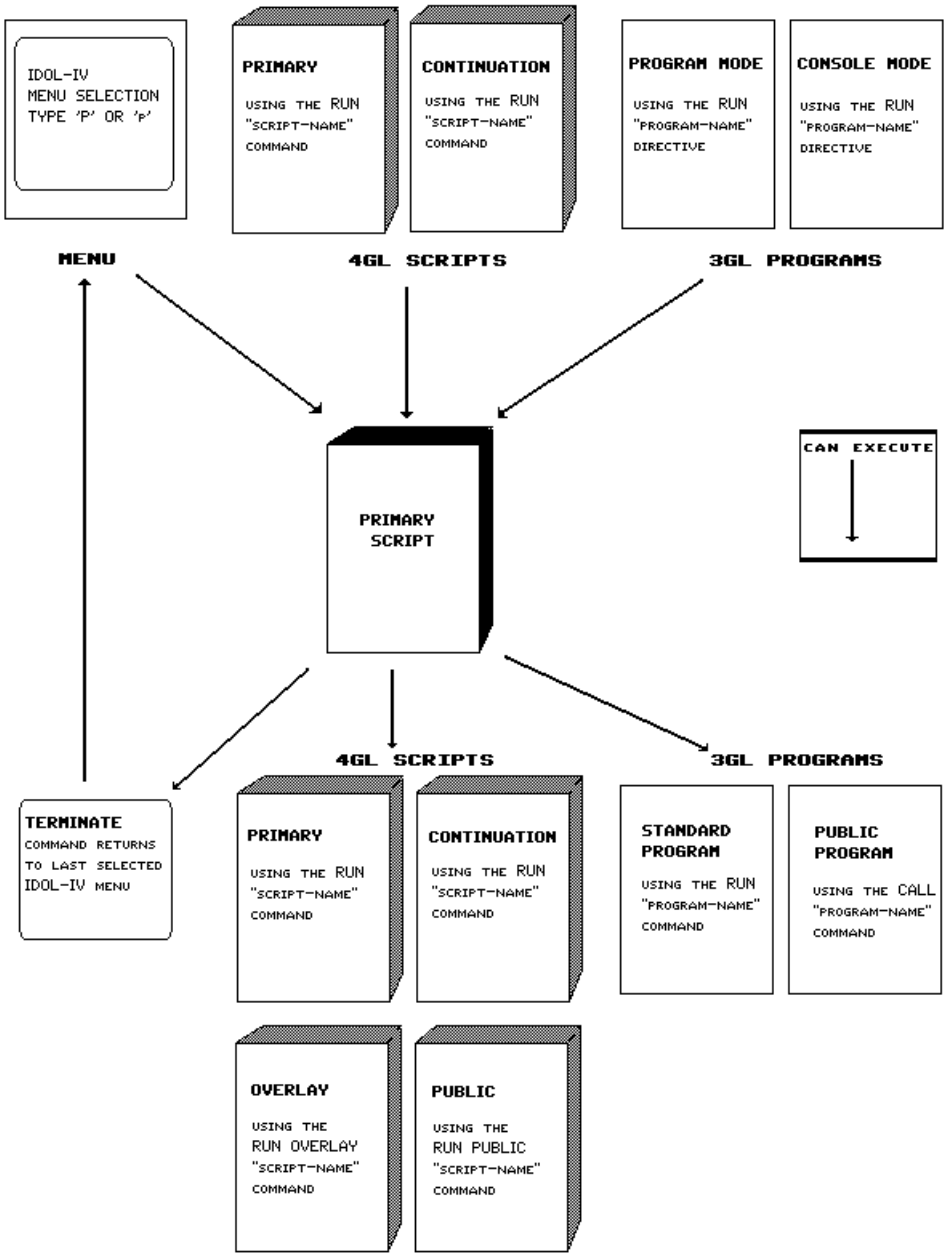
Script Execution Diagrams

The following pages contain diagrams and charts that illustrate how different types of scripts execute and interact with one another, the data environment for some types of scripts, and how memory is used as scripts execute.

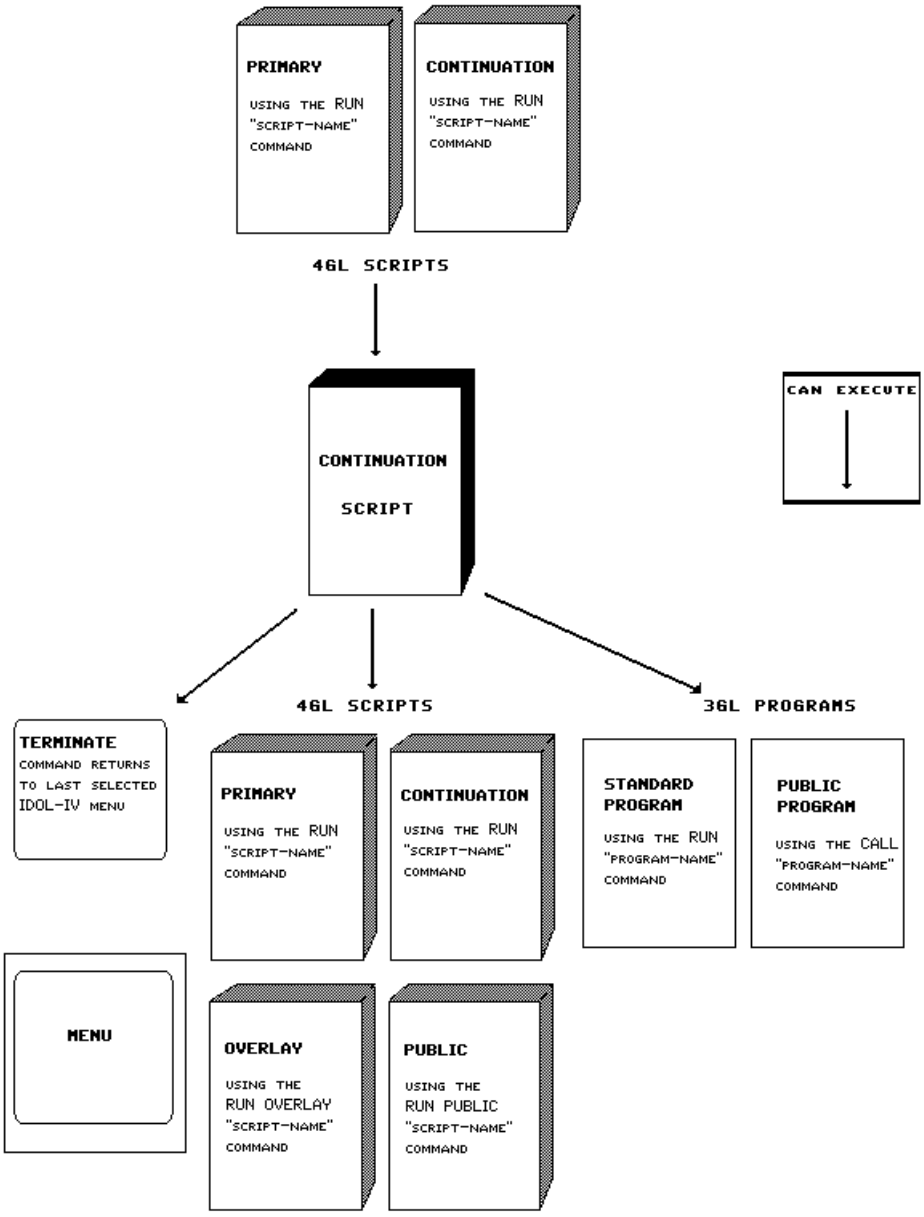
The diagrams and charts are:

- Primary Script Execution
- Continuation Script Execution
- Overlay Script Execution
- Public Script Execution
- Script Type Execution and Memory Usage

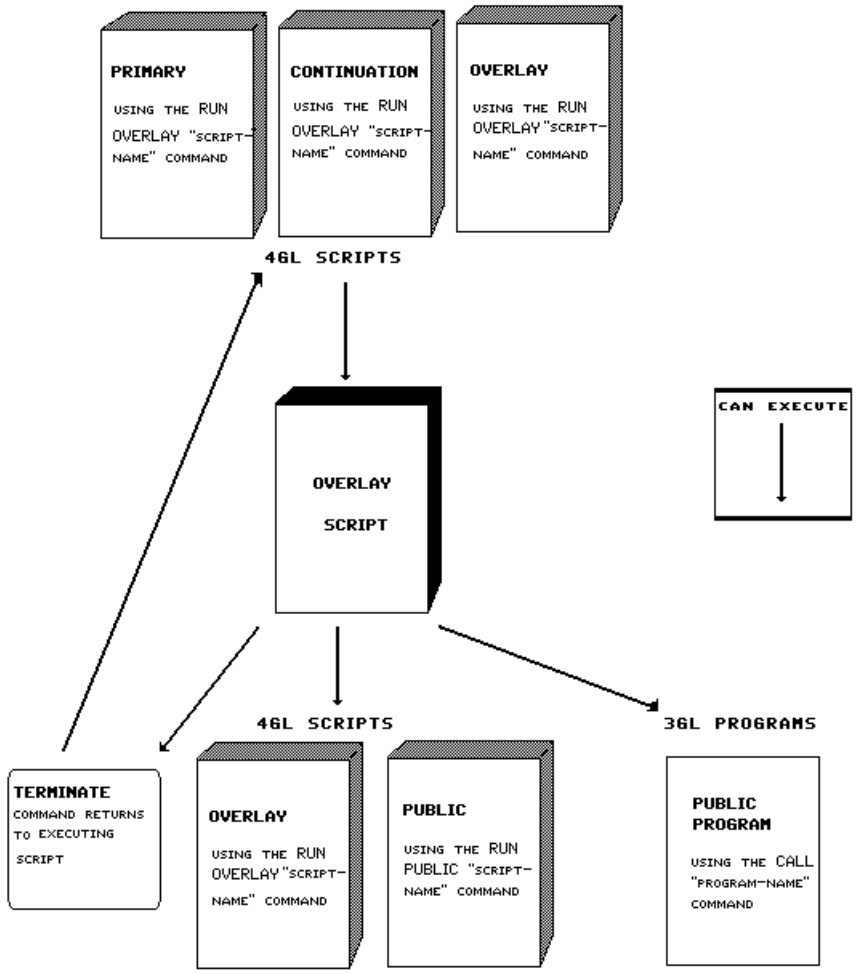
Primary Script Execution



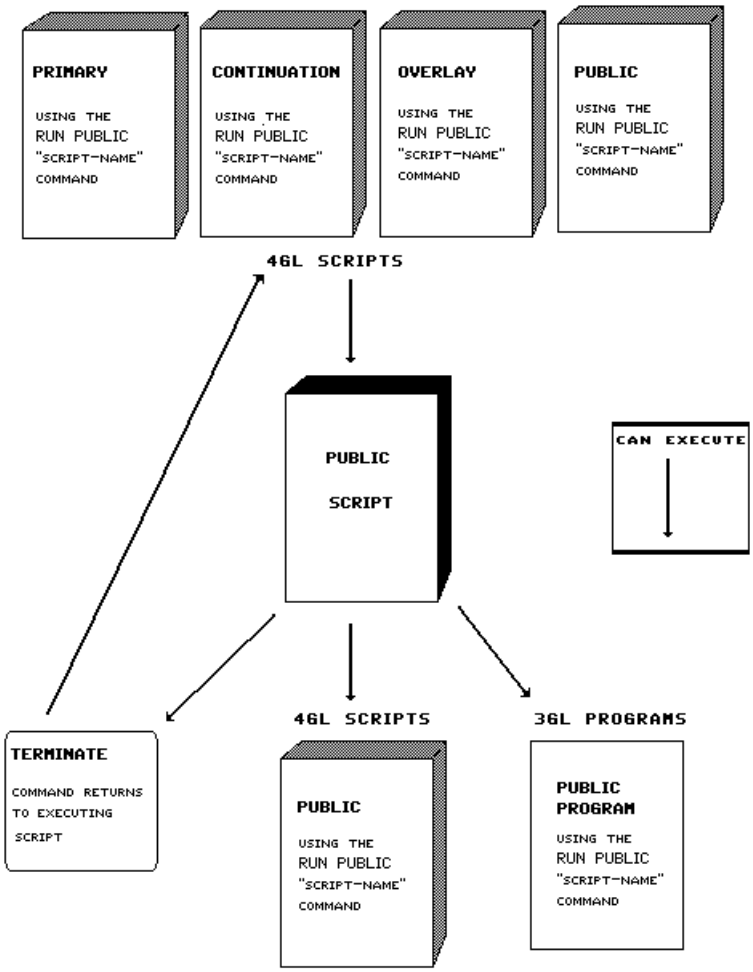
Continuation Script Execution



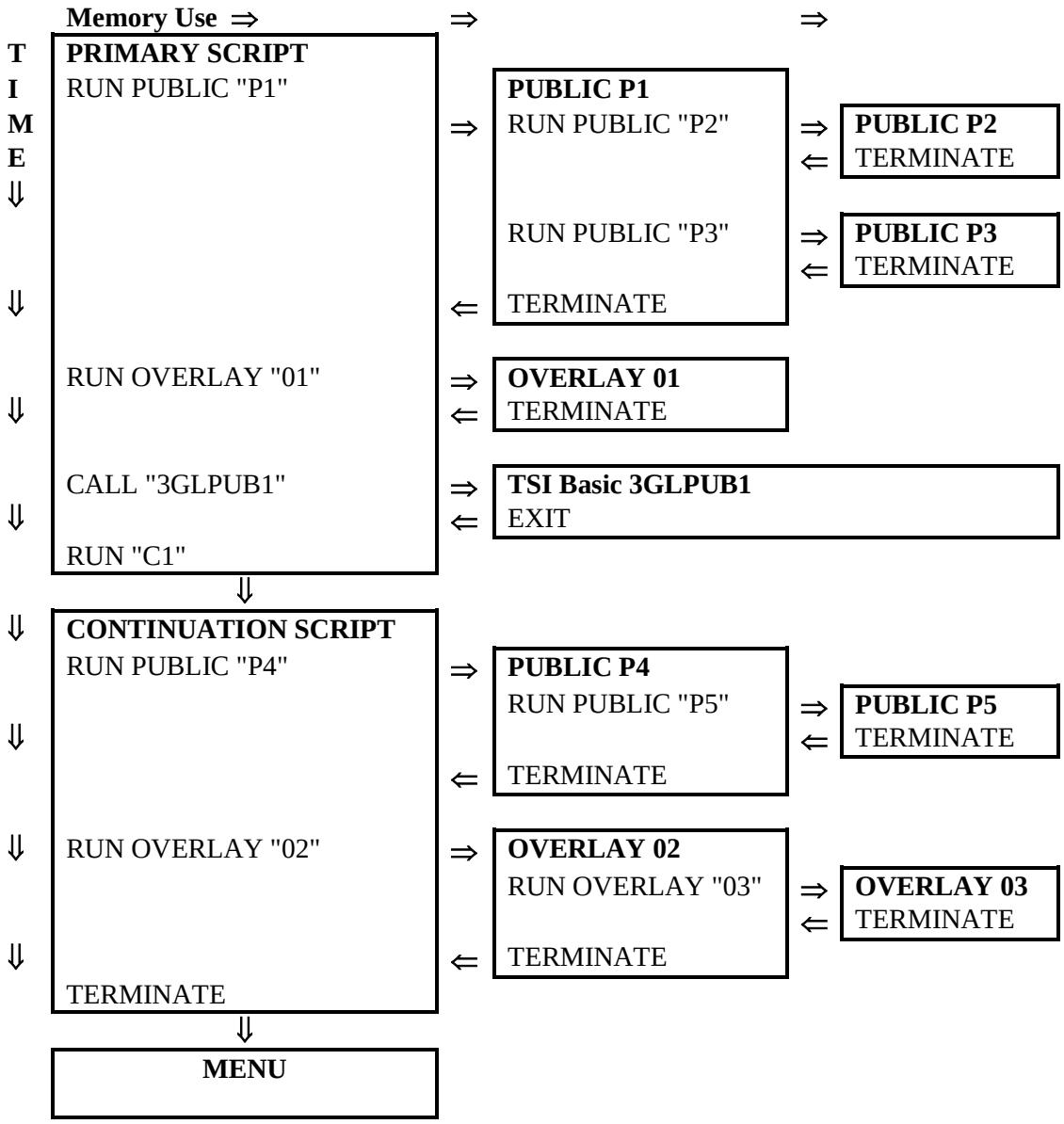
Overlay Script Execution



Public Script Execution



Script Type Execution and Memory Usage



Note: Primary and continuation scripts displace each other in memory. Overlay, public, and 3GL public scripts accumulate in memory until terminated. In all cases, an EXIT or TERMINATE returns control to the command that follows the RUN or CALL.

SCRIPT-IV TIPS AND TECHNIQUES

Script-IV provides advanced data handling features. This section contains the following:

- **Terminal Keyboard Values** describes how to manage certain types of data generated by Script-IV.
- **Escape Processing** describes how to use the **Escape** key to interrupt script execution.
- **Using Keys** describes how to use KEY references with secondary keys and with numeric and date keys.
- **CONNECT Commands** describes how to use CONNECT commands.
- **Database Maintenance and Script-IV** provides a list of features exclusive to database maintenance.
- **Interface to Thoroughbred Basic** describes the interface between Script-IV and the Thoroughbred Basic third generation language.

Terminal Keyboard Values

The Script-IV **TERM-KEY** variable enables you to:

- specify actions that will be taken when a function key is pressed.
- keep track of which key was last pressed during script execution.
- set a time-out value for data entry.

For more information on the **TERM-KEY** variable see the description in the Script-IV Language Reference.

Escape Processing

Normal script execution can be interrupted by pressing the **Escape** key. Most terminals provide an **Escape** key or a keystroke sequence that performs the escape function. The script specifies how the escape will be processed. There are two ways of handling escape processing in Thoroughbred Script-IV:

- Standard escape processing, which is controlled by the **ESCAPE** variable
- Custom escape processing, which is controlled by the **ESCAPE-KEY** command

Standard Escape Processing

The operator presses the **Escape** key during script execution. The following message is displayed:

```
Terminate (Y/N)?
```

Standard escape processing means that the operator can choose to halt the script or continue script execution. If standard escape processing is disabled, pressing the **Escape** key will have no effect and the script will continue to execute. You can enable or disable standard escape processing by setting the value of the **ESCAPE** variable in the script.

Setting the **ESCAPE** variable to "**Y**" enables the standard escape procedure:

```
LET  ESCAPE = "Y"
```

Setting the **ESCAPE** variable to "**N**" disables the standard escape procedure:

```
LET  ESCAPE = "N"
```

The standard processing procedure checks the value of this variable to see if it is enabled. If the ESCAPE variable is not set in a script, the default is **Y**, and standard escape processing is enabled. For more information on the **ESCAPE** variable see the Script-IV Language Reference.

Custom Escape Processing

The **ESCAPE-KEY** command provides the ability to design custom escape processing. The command syntax is **ESCAPE-KEY** *procedure*|**OFF**|**ON**.

ESCAPE-KEY *procedure* — This **ESCAPE-KEY** command specifies a procedure to execute when the **Escape** key is pressed. This command overrides the value of the **ESCAPE** variable and disables standard escape processing. The **ESCAPE-KEY** *procedure* command also replaces any previous custom escape processing set in the script.

ESCAPE KEY OFF — This command turns off custom escape processing. The standard escape processing procedure is enabled.

ESCAPE-KEY ON — This command reactivates the last specified custom escape processing procedure. If the script did not previously specify a custom escape processing procedure, this command is ignored.

In primary and continuation scripts, the **ESCAPE-KEY** command can be used multiple times. The custom escape processing set with the **ESCAPE-KEY** command overrides any previously defined custom processing.

In public and overlay scripts, only one custom escape procedure should be specified. If multiple **ESCAPE-KEY** commands are used, the last one encountered by the compiler is the only one executed.

For more information on the **ESCAPE-KEY** command see the Script-IV Language Reference.

Using Keys

This section describes how to use KEY references with the following types of keys:

- Secondary Keys
- Numeric and Date Keys

Secondary Keys

When you use a KEY reference with secondary keys more than one match is possible. Because of this, when a processing procedure is specified all records that match the specified key will be processed. If no processing procedure is specified only the first match will be read.

Please consider the following examples:

```
READ link-name USING SORT n IS key-value
READ link-name USING SORT n IS NEXT
```

This example returns one record beyond the record where the closest match with **key-value** occurred.

```
READ link-name USING KEY SORT n IS key-value
      PROCESSING IS your-process
```

This example processes all records that match the **key-value**.

```
READ link-name USING KEY SORT n IS key-value
```

This example only processes the first record that matches the **key-value**.

Note: The **SORT** option specifies a sort definition that can be used to access the records in the file by a secondary key. When a **SORT** option is used, a string value must be specified and the **MISSING KEY** procedure is not valid. File access using a secondary key will access the record closest to the specified key. A record will always be accessed unless an end of file is reached. For more information see the description of the **READ** command in the Script-IV Language Reference.

Numeric and Date Keys

Note: The following information applies to date fields because they are numeric fields.

To use Script-IV to perform file input or output using a numeric key, you must manually pad the data in Script-IV so that it matches the specifications in the format.

For example, if the **OPLINES** format uses a numeric key called **SEQ-NUM** defined with a length of **4.0** and a padding type of **3** (right justify and zero fill), the following command will pad the data when adding a new record:

```
ADD OPLINES USING KEY STR( SEQ-NUM :"0000")
```

The spacing in the syntax of this command is significant.

For multi-part keys, you must pad any part of the key that is a numeric field. For example:

```
ADD OPLINES USING KEY CUS-CODE + STR( SEQ-NUM :"0000" )
```

There is an alternative to padding the numeric data every time you use it in a file input or output command. To use a numeric key to maintain a sequence number, such as an invoice number, it may be easier to define an alphanumeric key and manually convert the data to numeric form to increment it for the next sequence number. For example:

```
LET ALPHA-SEQ-NUM = STR( NUM( ALPHA-SEQ-NUM ) + 1:"0000" )
```

A valid entry range specified in the format will not validate that only numbers are entered into the alphanumeric sequence number field. For example, if **ALPHA-SEQ-NUM** has a length of **4** and a valid entries range of **1,0000,9999**, an entry of **000Z** is accepted as valid (the valid entries range check performs an entire string comparison).

CONNECT Commands

The CONNECT commands enable a script to connect to various object classes defined in Dictionary-IV, such as help, menus, screens, and views. Additionally, CONNECT commands enable scripts to connect to Query-IV queries or Report-IV reports.

The data in the objects and the functional capabilities of each class are available during script execution. Use of CONNECT commands enables developers to focus on data objects and the classes used to define them, which assures data independence and reduces the amount of procedural Script-IV code required to make use of a data object.

The CONNECT commands enable concurrent development of data objects and scripts. They are designed to promote fast, flexible prototyping and development.

CONNECT HELP

This command enables you to connect to a help definition specified in Dictionary-IV and display the help text during script execution.

For more information on the **CONNECT HELP** command see the Script-IV Language Reference. For more information on how to create a help definition see the Dictionary-IV Developer Guide.

CONNECT MENU

This command enables you to connect to a pop-up menu definition specified in Dictionary-IV and display the menu during script execution.

For more information on the **CONNECT MENU** command see the Script-IV Language Reference. For more information on how to create a menu definition see the Dictionary-IV Developer Guide.

CONNECT QUERY

This command enables you to connect to a Query-IV query, and display or print the query during script execution.

For more information on the **CONNECT QUERY** command see the Script-IV Language Reference. For more information on how to create a query see the Query-IV Reference Manual.

CONNECT REPORT

This command enables you to connect to a Report-IV report, and display or print the report during script execution.

For more information on the **CONNECT REPORT** command see the Script-IV Language Reference. For more information on how to create a report see the Report-IV Reference Manual.

CONNECT SCREEN

This command enables you to connect to a screen definition specified in Dictionary-IV, display the screen, and initiate single-record maintenance during script execution.

For more information on the **CONNECT SCREEN** command see the Script-IV Language Reference. For more information on how to create a screen definition see the Dictionary-IV Developer Guide.

CONNECT VIEW

This command enables you to connect to a view definition specified in Dictionary-IV, display the view, and initiate multi-record maintenance during script execution.

For more information on the **CONNECT VIEW** command see the Script-IV Language Reference. For more information on how to create a view definition see the Dictionary-IV Developer Guide.

Database Maintenance and Script-IV

The implementation of a feature sometimes depends upon its context, in this case features in Script-IV and Dictionary-IV Database Maintenance. Whether a feature is implemented depends on how useful the feature is and whether the feature has meaning in a given context.

For example, the format security settings for "Add Only" and "Change Only" have meaning in Dictionary-IV Database Maintenance where "add mode" and "change mode" exist. However, these format security settings have no meaning in Script-IV because these features were not implemented in Script-IV.

The following features are implemented for formats in Dictionary-IV Database Maintenance but not in Script-IV:

- Security: Add Only
- Security: Change Only
- Valid Entries: File Lookup Read Option 1
- Delete Record Value
- Audit

The following features are implemented for links in Dictionary-IV Database Maintenance but not in Script-IV:

- Terminal Access
- Operator Access
- Password
- Audit

For more information on these features see the Dictionary-IV User Guide.

Interface to Thoroughbred Basic

Script-IV provides an interface to Thoroughbred Basic language elements and data constructs. This interface adds functionality and flexibility to the Script-IV language by increasing the number of available language elements and by providing more control over certain types of programming details.

This section contains the following:

- **Thoroughbred Basic Files** describes how Script-IV can use data files created by Thoroughbred Basic applications.
- **Thoroughbred Basic Programs** describes how to execute Thoroughbred Basic programs from scripts.

For more information, please see the Thoroughbred Basic Reference Manual.

Thoroughbred Basic Files

Script-IV can access data files created by a Thoroughbred Basic application. To use an existing data file in a script, follow the procedure below:

1. Define the format definition to be used to access the data record.

2. Create a link, which references the format and the data file.

3. Access the file in Dictionary-IV file maintenance to verify that your format and link definitions are correct.

4. Declare the link in your script.

5. Open the link in your script.

After you complete this procedure, you can use Script-IV file I/O commands to access the file through the link name. For more information on Thoroughbred Basic data files see the Thoroughbred Basic Reference Manual.

Thoroughbred Basic Programs

Script-IV can execute Thoroughbred Basic programs. This feature enables you to access existing Thoroughbred Basic applications in Script-IV. For more information on Thoroughbred Basic programs see the Thoroughbred Basic Reference Manual.

Standard Thoroughbred Basic Program

To execute a standard Thoroughbred Basic program from a script you can use the **RUN** *program-name* command. From this program, you can use the **RUN** *program-name* directive to execute another Thoroughbred Basic program or a primary script or use the **CALL** *program-name* directive to execute a Thoroughbred Basic public program.

For more information on the Script-IV **RUN** command see the Script-IV Language Reference. For more information on the Thoroughbred Basic **CALL** and **RUN** directives see the Thoroughbred Basic Reference Manual.

Thoroughbred Basic Public Program

To execute a Thoroughbred Basic public program from a script you can use the **CALL** *program-name* command. From this program, you can use the **CALL** *program-name* directive to execute another Thoroughbred Basic public program.

For more information on the Script-IV **CALL** command, see the Script-IV Language Reference. For more information on the Thoroughbred Basic **CALL** directive, see the Thoroughbred Basic Reference Manual.

Migrating Programs to Scripts

You can migrate a Thoroughbred Basic application to Script-IV in stages by replacing software programs with scripts. Scripts can emulate the appearance of existing software so that the look and feel remains constant during the conversion. After the Thoroughbred Basic application is fully migrated, you can enhance the applications with features inherent in the Dictionary-IV application development system.

COMPILING SCRIPTS

NOTE: If you use Source-IV to edit scripts, you must compile the scripts from Source-IV. If you use the Script-IV script editor to edit a script, you must compile the script from the script editor.

You cannot execute a script until it has been compiled. The compiler produces a 3GL Thoroughbred Basic program from the source script, a program listing, error messages, and error diagnostics. If no errors occur, the resultant program can be executed.

Please refer to the Source-IV manual for details on how to compile scripts.

If you are using the older IDOL script editor, please refer to the Appendix – IDOL Script Editor and Compiler section of this manual.

SAMPLE SCRIPTS

The scripts illustrated in this section are from the **4S** library. You can edit, view, or print any of these scripts. These scripts demonstrate various features and capabilities of the Script-IV language. If you plan to use any of these scripts in an application, you may have to modify it to meet site requirements.

The following sample scripts are illustrated:

4SEX001 is a primary script that provides an example of line offset with scrolling.

4SSAMPL1 is a copy script used for data declarations.

4SSAMPL2 is an overlay script that initializes demonstration data.

4SSAMPL3 is an overlay script that provides examples of the PRINT VIEW command.

4SSAMPL4 is an overlay script that provides an example of the CHANGE command.

4SSAMPLE is a primary script that provides an example of how scripts can be used in sales. It is the parent script of all the 4SAMPLx scripts listed above.

```
Script: 4S EX001   Type: 1 Primary
Desc: SCRIPT-IV Sample: Line Offset with Scrol                    Page: 1
Last Change Date: 11/26/88     Last Compile Date: 05/09/95
            Time: 12:45:17                   Time: 17:39:07      Date: 05/12/95
================================================================================
    * 4SEX001 - SCRIPT-IV Sample - Shows Line Offset with Scrolling
    SN   4SEX01
    DN   START-LINE (2.0), SCROLL-FLAG (1)
    MAIN-LINE
         OPEN SCREEN 4SEX01;  OPEN 4SEX01 CREATE
         READ 4SEX01 USING KEY RANGE FROM FIRST TO LAST
              PROCESSING IS DELETE-RECS
         INPUT SCREEN 4SEX01 PRE PROCESS  DNA, GET-START-LINE
         LET L = 1, L1 = 0
         DO LOOP UNTIL TERM-KEY = 4
            LET 4SEX01 = "", DNA = STR(L:"000000")
            READ 4SEX01 USING KEY DNA
                 MISSING KEY PROCESS IS NEW-REC
            PRINT SCREEN 4SEX01 DATA LINE OFFSET IS L1
            INPUT SCREEN 4SEX01 DATA-NAME DNB  LINE OFFSET IS L1
                                POST PROCESS  DNB, SCROLL
            IF TERM-KEY <> 4 AND SCROLL-FLAG <> "X" THEN
               ADD 4SEX01 USING KEY STR(L:"000000");  LET L = L + 1
               IF L1 < 4 THEN
                  LET L1 = L1 + 1
               ELSE
                  PRINT @(0, START-LINE), 'LD',
               ENDIF
            ENDIF
            LET SCROLL-FLAG = ""
         ENDLOOP
    NEW-REC
         PRINT @(0,4), "CAN'T FIND REC: ", DNA,
    SCROLL
         IF TERM-KEY <> 0 THEN
            LET SCROLL-FLAG = "X", 4SEX01.FIELD = 99
         ENDIF
         IF TERM-KEY = -4 THEN
            IF L1 > 0 THEN
               LET L1 = L1 - 1
            ELSE
               IF L > 1 THEN
                  PRINT @(0, START-LINE + 4), 'LD', @(0, START-LINE), 'LI',
               ENDIF
            ENDIF
            IF L > 1 THEN LET L = L - 1;  ENDIF
         ENDIF
         IF TERM-KEY = -3 THEN
            IF L1 < 4 THEN
               LET L1 = L1 + 1
            ELSE
               PRINT @(0, START-LINE), 'LD',
            ENDIF
            LET L = L + 1
         ENDIF
    GET-START-LINE
         LET START-LINE = 4SEX01.LINE, 4SEX01.FIELD = 99
    DELETE-RECS
         DELETE 4SEX01 USING KEY DNA



Script: 4S EX001   Type: 1 Primary
Desc: SCRIPT-IV Sample: Line Offset with Scrol                    Page: 2
Last Change Date: 11/26/88     Last Compile Date: 05/09/95
            Time: 12:45:17                   Time: 17:39:07      Date: 05/12/95
================================================================================
    END-SCRIPT
```

```
Script: 4S SAMPL1  Type: 5 Copy
Desc: SCRIPT-IV SAMPLE: Data Declarations                          Page: 1
Last Change Date: 11/26/88     Last Compile Date:
           Time: 11:42:39                 Time:                Date: 05/12/95
================================================================================
    * 4SSAMPL1 - Data Declarations - Copy module for 4SSAMPLE script set

    VN    4SCUST, 4SSLSRP, 4SINVEN
    LN    4SSALDT
    SN    4STOPSC1, 4STOPSC2, 4SBOTSCR
    DN    INPUT-FLAG (1), VIEW-FLAG (1), TEXT-FLAG (1)
    DN    TAX-RATE (2.0)
```

```
Script: 4S SAMPL2  Type: 3 Overlay
Desc: SCRIPT-IV SAMPLE: Initalize Demo Data                    Page: 1
Last Change Date: 11/26/88     Last Compile Date: 07/19/93
           Time: 16:29:10                  Time: 09:16:34      Date: 05/12/95
================================================================================
    * 4SSAMPL2 - Initialize Demo Data - Overlay script to 4SSAMPLE

    * 4SSAMPL1 - Data Declarations - Copy module for 4SSAMPLE script set

    VN   4SCUST, 4SSLSRP, 4SINVEN
    LN   4SSALDT
    SN   4STOPSC1, 4STOPSC2, 4SBOTSCR
    DN   INPUT-FLAG (1), VIEW-FLAG (1), TEXT-FLAG (1)
    DN   TAX-RATE (2.0)

    1-MAIN2
         PRINT MESSAGE "P,8"
         OPEN LINK 4SSLSRP CREATE

         DIM B$(90)
         LET B$(1,30) =  "John Thompson", B$(31,30) = "Helen Addison",
             B$(61,30) = "Steve Watson" , X=0 , A$ = "JTHASW"

         DO LOOP UNTIL X = 3
            LET X = X + 1, 4SSLSRP.SALES-REP-CODE = A$(X*2-1,2),
                SALES-REP-NAME = B$(X*30-29,30)
            ADD 4SSLSRP USING 4SSLSRP.SALES-REP-CODE
         ENDLOOP

         OPEN LINK 4SCUST CLEAR

         DIM A$(60), B$(90), C$(45), D$(45)
         LET A$(1,20)= "Mr. David Kelly", A$(21,20) =  "Ms. Marcia Thomas",
             A$(41,20)= "Robert Marks", B$(1,30) =   "D K & Associates ",
             B$(31,30)= "Computer Inc.",
             B$(61,30)= "Today's Business Company",
             C$(1,15)=  "1010 Main Street", C$(16,15)= "Route 12",
             C$(31,15)= "1650 Kingston Pike", D$(1,15)= "Somerset",
             D$(16,15)= "Madison", D$(31,15) =  "Palisades Park",
             X = 0, CUST-STATE-ZIP = "NJ   07090",
             SALES-TAX-CODE = "NJ"

         DO LOOP UNTIL X = 3
            LET X=X+1, CUST-CODE = STR(X:"000000"),
                CUST-CONTACT = A$(X*20-19,20), CUST-NAME = B$(X*30-29,30),
                CUST-ADDRESS = C$(X*15-14,15), CUST-CITY = D$(X*15-14,15)
            ADD 4SCUST USING KEY CUST-CODE
         ENDLOOP

         OPEN LINK 4SINVEN CLEAR

         DIM A$(120), B(3)
         LET A$(1,40)= "bProduct Description",
             A$(41,40) =  "cProduct Description",
             A$(81,40) =  "aProduct Description",
             B(1) = 12.95, B(2) = 19.95, B(3) = 1.99, X = 0
         DO LOOP UNTIL X = 3
            LET X=X+1, 4SINVEN.ITEM-CODE = STR(X:"00-000-000") ,
                ITEM-DESCRIPTION = A$(X*40-39,40),



Script: 4S SAMPL2  Type: 3 Overlay
Desc: SCRIPT-IV SAMPLE: Initalize Demo Data                    Page: 2
Last Change Date: 11/26/88     Last Compile Date: 07/19/93
           Time: 16:29:10                  Time: 09:16:34      Date: 05/12/95
================================================================================
                4SINVEN.BASE-PRICE = B(X)
            ADD 4SINVEN USING KEY 4SINVEN.ITEM-CODE
         ENDLOOP
         OPEN LINK 4SSALDT CLEAR
    ENDSCRIPT
```

```
Script: 4S SAMPL3  Type: 3 Overlay
Desc: SCRIPT-IV SAMPLE: Print Views                                  Page: 1
Last Change Date: 11/26/88     Last Compile Date: 07/19/93
          Time: 16:06:02                  Time: 09:16:51      Date: 05/12/95
================================================================================
    * 4SSAMPL3 - Print Views - Overlay script to 4SSAMPLE

    * 4SSAMPL1 - Data Declarations - Copy module for 4SSAMPLE script set

    VN   4SCUST, 4SSLSRP, 4SINVEN
    LN   4SSALDT
    SN   4STOPSC1, 4STOPSC2, 4SBOTSCR
    DN   INPUT-FLAG (1), VIEW-FLAG (1), TEXT-FLAG (1)
    DN   TAX-RATE (2.0)

    1-MAIN3
         PRINT MESSAGE "P,5"  USING "4SSAMPLE"
         IF VIEW-FLAG = "C"  THEN
            PRINT VIEW 4SCUST USING KEY SORT 1
               WINDOW LINE 15
                      COLUMN 20
                      NUMBER LINES 3
                      CHARACTERS 40
                      BORDER TYPE "R"
                      FUNCTION "C"
                      HEADING "N"
               KEY INTO 4SCUST.CUST-CODE
            IF TERM-KEY <> 1 THEN
               LET 4SCUST.CUST-CODE = ""
            ENDIF
         ENDIF
         IF VIEW-FLAG = "S" THEN
            PRINT VIEW 4SSLSRP USING KEY SORT 1
               WINDOW LINE 15
                      COLUMN 20
                      NUMBER LINES 3
                      CHARACTERS 40
                      BORDER TYPE "R"
                      FUNCTION "C"
                      HEADING "N"
               KEY INTO 4SSLSRP.SALES-REP-CODE
            IF TERM-KEY <> 1 THEN
               LET 4SSLSRP.SALES-REP-CODE = ""
            ENDIF
         ENDIF
         IF VIEW-FLAG = "I" THEN
            PRINT VIEW 4SINVEN USING KEY SORT 1
               WINDOW LINE 18
                      COLUMN 40
                      NUMBER LINES 3
                      FUNCTION "C"
                      HEADING "N"
                      BORDER TYPE "R"
                      CHARACTERS 30
               KEY INTO 4SSALDT.ITEM-CODE
            IF TERM-KEY <> 1 THEN
               LET 4SSALDT.ITEM-CODE = ""
            ENDIF
         ENDIF


Script: 4S SAMPL3  Type: 3 Overlay
Desc: SCRIPT-IV SAMPLE: Print Views                                  Page: 2
Last Change Date: 11/26/88     Last Compile Date: 07/19/93
          Time: 16:06:02                  Time: 09:16:51      Date: 05/12/95
================================================================================

         OPEN MESSAGES "4SSAMPLE"

    ENDSCRIPT
```

```
Script: 4S SAMPL4  Type: 3 Overlay
Desc: SCRIPT-IV SAMPLE: Change Text                                    Page: 1
Last Change Date: 11/26/88     Last Compile Date: 07/19/93
            Time: 16:59:59                    Time: 09:17:08       Date: 05/12/95
================================================================================
    * 4SSAMPL4 - Change Text - Overlay script to 4SSAMPLE

    * 4SSAMPL1 - Data Declarations - Copy module for 4SSAMPLE script set

    VN   4SCUST, 4SSLSRP, 4SINVEN
    LN   4SSALDT
    SN   4STOPSC1, 4STOPSC2, 4SBOTSCR
    DN   INPUT-FLAG (1), VIEW-FLAG (1), TEXT-FLAG (1)
    DN   TAX-RATE (2.0)

    1-MAIN4
         CHANGE 4SCUST USING 4SCUST.CUST-CODE
            TEXT "A"
            WINDOW FUNCTION TEXT-FLAG
                   HEADING "Additional Customer Information"
                   LINE 15
                   NUMBER LINES 3
                   CHARACTERS 40
                   COLUMN 20
                   BORDER TYPE "R"

    ENDSCRIPT
```

```
Script: 4S SAMPLE  Type: 1 Primary
Desc: SCRIPT-IV SAMPLE: Sales Script                                Page: 1
Last Change Date: 11/26/88     Last Compile Date: 07/19/93
          Time: 13:46:56                Time: 09:18:35      Date: 05/12/95
================================================================================
    * 4SSAMPLE - Sales Script - Parent script of 4SSAMPLE script set

    * 4SSAMPL1 - Data Declarations - Copy module for 4SSAMPLE script set

    VN   4SCUST, 4SSLSRP, 4SINVEN
    LN   4SSALDT
    SN   4STOPSC1, 4STOPSC2, 4SBOTSCR
    DN   INPUT-FLAG (1), VIEW-FLAG (1), TEXT-FLAG (1)
    DN   TAX-RATE (2.0)

    1-MAIN
         OPEN SCREEN 4STOPSC1
         OPEN SCREEN 4STOPSC2
         OPEN SCREEN 4SBOTSCR

         OPEN VIEW 4SCUST
         OPEN VIEW 4SSLSRP
         OPEN VIEW 4SINVEN

         OPEN MESSAGES "4SSAMPLE"

    * -------------------------  BUILD DEMO DATA  ---------------------- *
         RUN OVERLAY "4SSAMPL2"

    * -------------------------  PRINT ALL SCREENS  -------------------- *
         PRINT SCREEN 4STOPSC1
         PRINT SCREEN 4STOPSC2
         PRINT SCREEN 4SBOTSCR

    * -------------------------  SET FIRST INVOICE NUMBER  ------------- *
         LET REFERENCE-NUMBER = "T  100", TAX-RATE = 6
         DO LOOP UNTIL INPUT-FLAG = "D"
            DO 2-GET-REP
         ENDLOOP

    2-GET-REP
         PRINT SCREEN 4STOPSC1 CLEAR DATA
         PRINT SCREEN 4STOPSC2 CLEAR DATA
         LET INPUT-FLAG = "" , 4SSLSRP = ""
         INPUT SCREEN 4STOPSC1
            POST PROCESS SALES-REP-CODE, 3-VERIFY-REP
         IF TERM-KEY = 4 THEN
            LET INPUT-FLAG = "D"
            TERMINATE PROCEDURE
         ELSE
            PRINT SCREEN 4STOPSC1 DATA
            DO LOOP UNTIL INPUT-FLAG = "D"
               DO 5-GET-CUST
            ENDLOOP
            LET INPUT-FLAG = ""
         ENDIF

    3-VERIFY-REP
         IF TERM-KEY = 1 THEN
```

```
            LET VIEW-FLAG = "S"
    * -------------------------  PRINT SALES-REP VIEW  ------------------ *
            RUN OVERLAY "4SSAMPL3"
         ENDIF
         LET 4STOPSC1.FIELD = 99
         READ 4SSLSRP USING KEY 4SSLSRP.SALES-REP-CODE
            MISSING KEY PROCESS IS 4-REDO-INPUT

    4-REDO-INPUT
         LET 4STOPSC1.FIELD = 0

    5-GET-CUST
         PRINT SCREEN 4STOPSC2 CLEAR DATA
         PRINT MESSAGE "P,6"
         LET INPUT-FLAG = "", 4SCUST = ""
         INPUT SCREEN 4STOPSC2 KEY
         DO 6-CUST-STUFF
         IF INPUT-FLAG <> " " THEN
            TERMINATE PROCEDURE
         ENDIF
         PRINT SCREEN 4STOPSC2 DATA
         IF CUST-CODE <> "Cash  " THEN
            DO 7-CUST-READ
            IF TERM-KEY = 4 OR INPUT-FLAG <> " " THEN
               TERMINATE PROCEDURE
            ENDIF
         ENDIF
         DO 14-GET-DETAIL

    6-CUST-STUFF
         IF TERM-KEY = 1 THEN
            LET VIEW-FLAG = "C"

    * -------------------------  PRINT CUSTOMER VIEW  ------------------- *
            RUN OVERLAY "4SSAMPL3"
         ELSE
            IF TERM-KEY = 4 THEN
               LET INPUT-FLAG = "D"
            ELSE
               IF TERM-KEY = 2 THEN
                  LET CUST-CODE = "Cash", CUST-CONTACT = "CASH CUSTOMER"
               ENDIF
            ENDIF
         ENDIF

    7-CUST-READ
         CHANGE 4SCUST USING KEY CUST-CODE
            MISSING KEY PROCESS IS 8-ADD-CUST
            BUSY PROCESS IS 9-LOCKED-CUST
            PROCESSING IS 10-CHG-CUST
         IF INPUT-FLAG = "T"  THEN
            LET TEXT-FLAG = "C"
            DO 13-CHG-COMMENTS
            LET INPUT-FLAG = ""
```

```
        ENDIF

    8-ADD-CUST
        IF 4SCUST.CUST-CODE = "       " THEN
           LET INPUT-FLAG = "N"
           TERMINATE PROCEDURE
        ENDIF

    * -------------------------  CUSTOMER NOT FOUND. ADD (Y/N)?  --------- *
        INPUT MESSAGE "Y,2" INTO INPUT-FLAG
        IF INPUT-FLAG = "Y" THEN
           INPUT SCREEN 4STOPSC2 DATA-NAME CUST-CONTACT
              POST PROCESS CREDIT-LIMIT, 11-ADD-REC
           IF TERM-KEY = 4 THEN
              DELETE 4SCUST USING CUST-CODE
                 MISSING KEY PROCESS IS 21-IGNORE
              LET INPUT-FLAG = "N"
           ENDIF
        ELSE
           LET INPUT-FLAG = "N"
        ENDIF

    9-LOCKED-CUST
        LET INPUT-FLAG = "L"

    10-CHG-CUST
        PRINT SCREEN 4STOPSC2 DATA
        LET TEXT-FLAG = "d"
        DO 13-CHG-COMMENTS

    * -------------------------  MAKE CUSTOMER CHANGES ? ----------------- *
        INPUT MESSAGE "I,2" INTO INPUT-FLAG USING "4SSAMPLE"
        IF INPUT-FLAG <> "C" THEN
           LET INPUT-FLAG = ""
           TERMINATE PROCEDURE
        ENDIF
        INPUT SCREEN 4STOPSC2 DATA-NAME CUST-CONTACT
              POST PROCESS CREDIT-LIMIT, 12-MAKE-CHANGES

    * -------------------------  CHANGE CUSTOMER TEXT FIELD ? ------------ *
        INPUT MESSAGE "Y,5" INTO INPUT-FLAG
        IF INPUT-FLAG = "Y" THEN
           LET INPUT-FLAG = "T"
        ELSE
           IF INPUT-FLAG = "N" THEN
              LET INPUT-FLAG = ""
           ENDIF
        ENDIF

    11-ADD-REC
        ADD 4SCUST USING KEY CUST-CODE
        LET TEXT-FLAG = "C"
        DO 13-CHG-COMMENTS
```

```
    * -------------------------  DATA CORRECT ? ------------------------- *
         INPUT MESSAGE "Y,3" INTO INPUT-FLAG
         IF INPUT-FLAG = "N" THEN
            LET 4STOPSC2.FIELD = 0
         ELSE
            IF TERM-KEY = 4 THEN
               DELETE 4SCUST USING KEY CUST-CODE
            ENDIF
         ENDIF

    12-MAKE-CHANGES

    * -------------------------  CHANGES CORRECT ?  ---------------------- *
         INPUT MESSAGE "Y,4" INTO INPUT-FLAG
         IF INPUT-FLAG <> "Y" THEN
            LET 4STOPSC2.FIELD = 0
         ENDIF

    * -------------------------  DO TEXT FIELD DISPLAY OR  -------------- *
    * -------------------------      CHANGE OR ERASE       -------------- *
    13-CHG-COMMENTS
         RUN OVERLAY "4SSAMPL4"

    14-GET-DETAIL
         LET TEXT-FLAG = "E"
         DO 13-CHG-COMMENTS
         DIM A$(40), Q0(4), P0(4), T0(4), E0(4)
         LET LINE-NUMBER = 1
         DO LOOP UNTIL TERM-KEY = 4
            DO 15-DETAIL-LINE
            LET LINE-NUMBER = LINE-NUMBER + 1
            IF LINE-NUMBER = 5  THEN
               LET TERM-KEY = 4
            ENDIF
         ENDLOOP
         DO 19-DETAIL-CORRECT
         LET A$ = ""

    15-DETAIL-LINE
         LET L0                  = LINE-NUMBER,
             QUANTITY            = Q0(L0),
             4SSALDT.ITEM-CODE   = A$(L0*10-9,10),
             4SSALDT.BASE-PRICE  = P0(L0)

         INPUT SCREEN 4SBOTSCR LINE OFFSET L0-1
            POST PROCESS ITEM-CODE,  16-VERIFY-ITEM
                         BASE-PRICE, 18-TAX-EXTENSION

    16-VERIFY-ITEM
         LET D0$= "", ITEM-DESCRIPTION = ""
         IF TERM-KEY = 1 THEN
            LET VIEW-FLAG = "I"

    * -------------------------  PRINT ITEM CODE VIEW  ------------------- *
```

```
            RUN OVERLAY "4SSAMPL3"
            PRINT SCREEN 4SBOTSCR LINE OFFSET L0-1
                  DATA-NAME LIST ITEM-CODE
         ENDIF
         LET INPUT-FLAG = ""
         READ 4SINVEN USING 4SSALDT.ITEM-CODE
            MISSING KEY PROCESS IS 17-MISSING-ITEM
         IF INPUT-FLAG = "M" THEN
            LET 4SBOTSCR.FIELD = 0
         ELSE
            LET D0$ = ITEM-DESCRIPTION,
                A$(L0*10-9,10) = 4SSALDT.ITEM-CODE,
                4SSALDT.BASE-PRICE = 4SINVEN.BASE-PRICE
            PRINT @(0,22),'CL',
         ENDIF

    17-MISSING-ITEM
         LET INPUT-FLAG = "M"

    18-TAX-EXTENSION
         LET Q0(L0) = QUANTITY , P0(L0) = 4SSALDT.BASE-PRICE ,
             E0 = Q0(L0) * P0(L0) , T0(L0) = E0 * (TAX-RATE / 100),
             E0(L0) = E0 + T0(L0)

    19-DETAIL-CORRECT
         DO LOOP UNTIL INPUT-FLAG = "Y"

    * --------------------------  DETAIL LINES CORRECT ?  ---------------- *
            INPUT MESSAGE "Y,6" INTO INPUT-FLAG
            IF INPUT-FLAG = "N" THEN
               DO LOOP UNTIL TERM-KEY = 4

    * --------------------------  ENTER LINE TO CHANGE OR DELETE  -------- *
               INPUT MESSAGE "I,3" INTO INPUT-FLAG
               IF INPUT-FLAG = "D" THEN
                  DIM A$(LEN(A$))

    * --------------------------  SET VARIABLES TO FORCE ENDLOOPS  ------- *
                  LET TERM-KEY = 4, INPUT-FLAG = "Y"
               ENDIF
               IF INPUT-FLAG => "1" AND INPUT-FLAG <= "4" THEN
                  LET L0 = NUM(INPUT-FLAG)
                  IF A$(L0*10-9,10) <>  "          " THEN
                     LET LINE-NUMBER = L0
                     DO 15-DETAIL-LINE
                  ENDIF
               ENDIF
               ENDLOOP
            ENDIF
         ENDLOOP
         DO LOOP CHANGING L0 FROM 1 TO 4
            IF A$(L0*10-9,10) <>  "          " THEN
               LET SALES-TAX-AMT = T0(L0) ,
                   4SSALDT.CUST-CODE = 4SCUST.CUST-CODE,
```

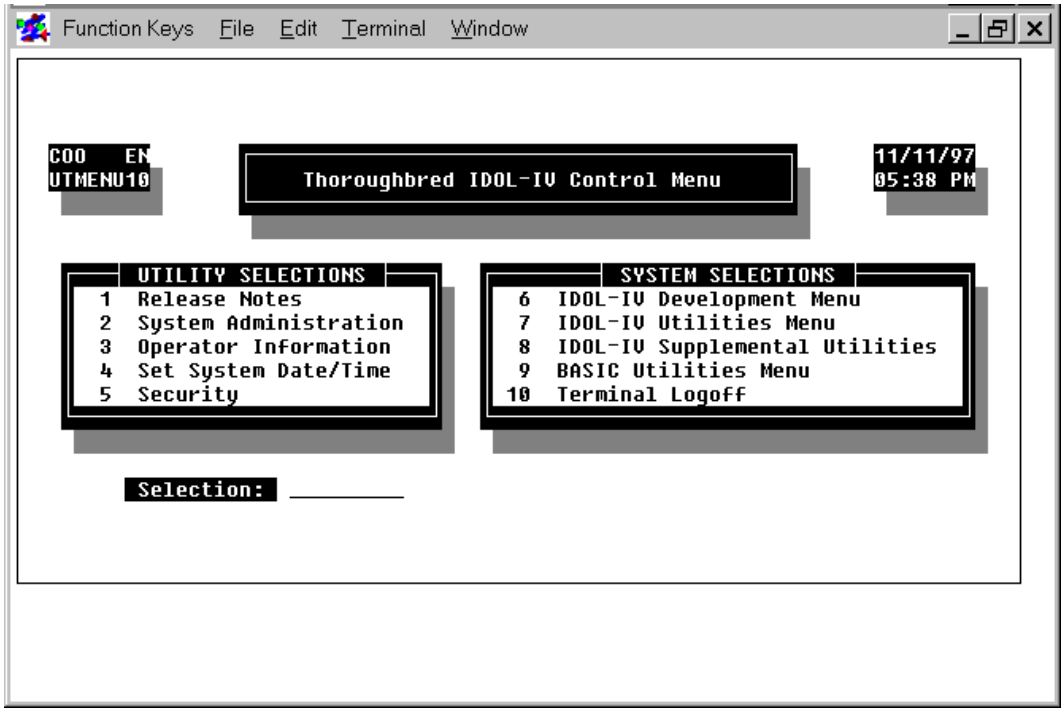
```
                    4SSALDT.DISCOUNT-PERCENT = 4SCUST.DISCOUNT-PERCENT,
                    4SSALDT.SALES-REP-CODE = 4SSLSRP.SALES-REP-CODE,
                    4SSALDT.ITEM-CODE = A$(L0*10-9,10),
                    4SSALDT.BASE-PRICE = P0(L0), QUANTITY = Q0(L0) ,
                    LINE-NUMBER = L0

    * -------------------------  UPDATE SALE DETAIL FILE  --------------- *
              ADD 4SSALDT USING KEY REFERENCE-NUMBER +
                                                  STR(LINE-NUMBER : "00")
                  DUPLICATE KEY PROCESS IS 20-INC-REF-NUM
            ENDIF

    * -------------------------  AFTER UPDATING, CLEAR LINE  ------------ *
            PRINT SCREEN 4SBOTSCR CLEAR DATA
                  LINE OFFSET L0-1
         ENDLOOP

    20-INC-REF-NUM
         LET R$ = REFERENCE-NUMBER, R$(4,3) = STR(NUM(R$(4,3))+1) ,
             REFERENCE-NUMBER = R$
         ADD 4SSALDT USING KEY REFERENCE-NUMBER +
                                        STR(LINE-NUMBER : "00")


    21-IGNORE
```

APPENDIX – IDOL SCRIPT EDITOR AND COMPILER

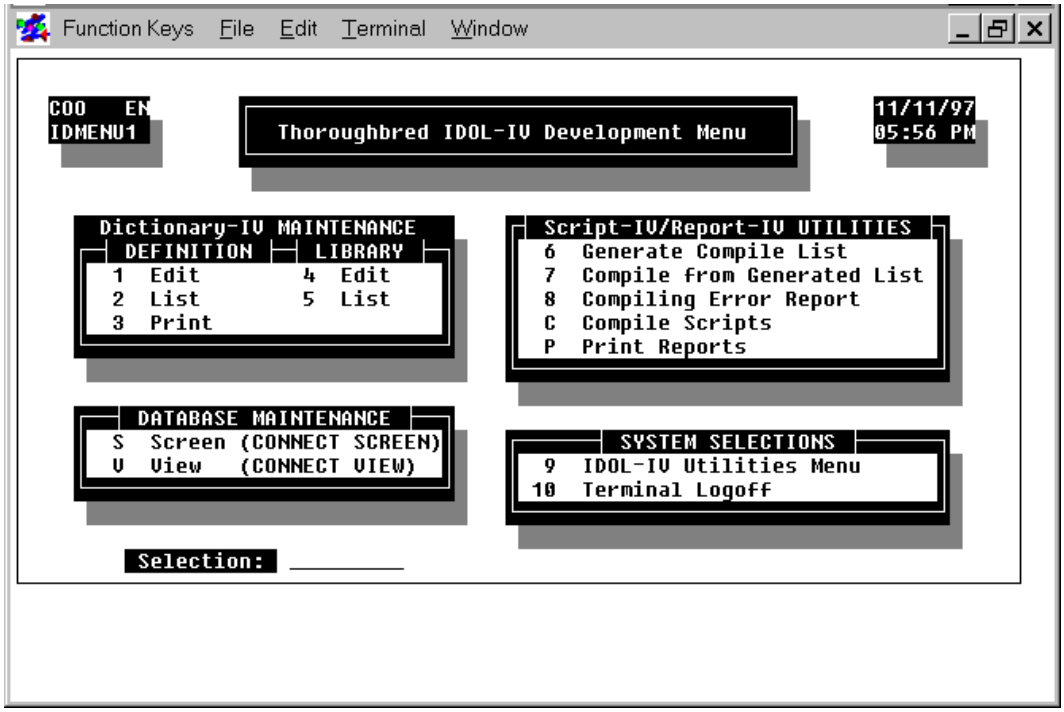
IDOL Script Editor

Thoroughbred provides limited support for the older IDOL Script-IV editor. Scripts maintained by the IDOL script editor cannot be compiled using the Source-IV compiler. It is **strongly recommended** that you import your scripts to Source-IV. Please refer to the Source-IV manual for details.

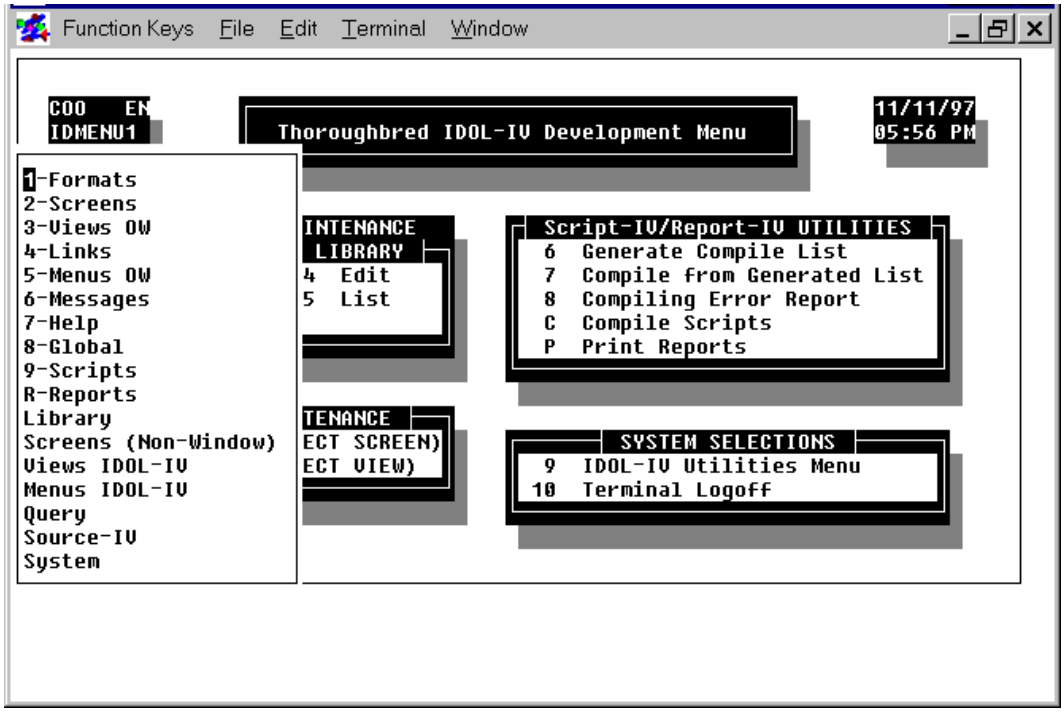
To enter the script editor, log on to Dictionary-IV. The Dictionary-IV Control Menu will be displayed:



Type the number that corresponds to the Dictionary-IV Development Menu and press the **Enter** key. The Dictionary-IV Development Menu will be displayed:

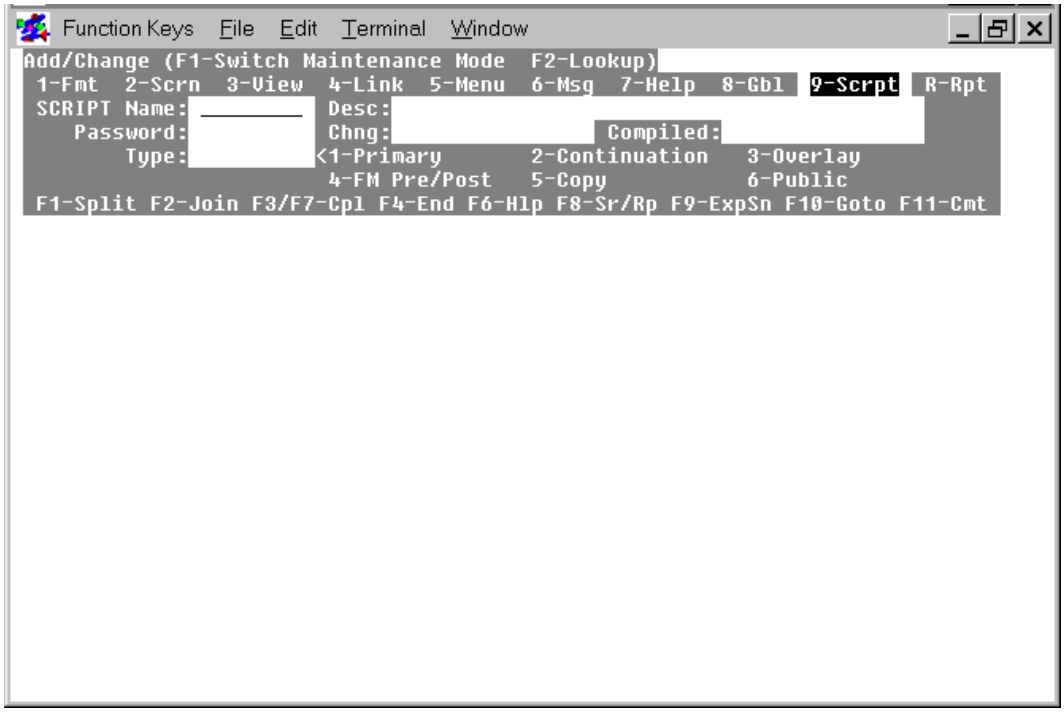


Type the number that corresponds to the Edit Definition function and press the **Enter** key. A pop-up menu will be displayed:



NOTE: The pop-up menu displayed above is available from any Dictionary-IV menu by pressing the **F1** key.

Type the number that corresponds to Scripts and press the **Enter** key. You will be placed in the Script Definition screen:



The Script Definition screen provides the following features:

1. The maintenance mode is displayed on the first line.
2. The definition facilities are displayed on the second line.
3. The components of the script definition are displayed on lines three through six.
4. The function keys active in the script editing area are displayed on the seventh line.
5. The script editing area occupies the remainder of the screen.

These features are described in the following sections.

Maintenance mode

The maintenance mode is displayed on the first line. For more information on maintenance modes see the section on **Components of Script Definition** in this manual.

Definition Facilities

This selector message enables you to switch to other definition facilities. For more information on these definition facilities see the Dictionary-IV
Developer Guide.

Components of Script Definition

SCRIPT Name: Type from 3 to 8 alphanumeric characters for the name of the script and press the **Enter** key. This field is mandatory.

Characters one and two are the library name. Characters three through eight are the script name. If the library does not exist, the system allows you to create it here.

See the Dictionary-IV Reference Manual, which provides a file naming convention you may implement.

The following function keys are available from the **SCRIPT Name:** field:

F1 Switches maintenance mode:

Add/Change	mode enables you to create a new script or edit an existing script.
Delete mode	enables you to delete an existing script.
Rename mode	enables you to rename an existing script.
Copy mode	enables you to copy a script.

F2 Displays a lookup of existing script names. For more information see the section on **Script Definition Lookup**.

F4 Exits. Press once to go to the Definition Facilities, which are discussed below. Press again to return to the Dictionary-IV Development Menu.

F6 Displays on-line help. For more information see the section on **Getting Help**.

Desc: Type up to 40 alphanumeric characters for the description of this script and press the **Enter** key. This field is optional.

Password: Type from 1 to 3 alphanumeric characters and press the **Enter** key. The system will ask you to verify the password. This field is optional.

Chng: Displays the date of the last change to the script definition. The system generates this field.

Compiled: Displays the date when the script was last compiled. The system generates this field.

Type: The following values are available:

1 - Primary Script
2 - Continuation Script
3 - Overlay Script
P - API Pre/Post Processing Script
4 - File Maintenance Pre/Post Processing Script
5 - Copy Script
6 - Public Script
U - Utility Script

Type the number or letter that corresponds to the script type and press the **Enter** key. This field is mandatory.

For more information on script types see the **Different Types of Scripts** section in this manual.

Function Keys Active in the Script Editor

The following function keys are available in the script editor:

F1 Split a line

F2 Join lines

F3 Compile and save a script

F4 Exit the script

F6 Help key, which opens the on-line help system. For more information, see the section on **Getting Help**

F7 Compile and save a script

F8 Search and replace, press twice to reuse a previous search

F9 Expand screen. Press again to reset the screen to its original size.

F10 Goto a procedure

F11 Enter a comment line

Other Editing Keys Active in the Script Editor

The following keys are available in the script editor:

Moving on a Line

Left Arrow	Move left one character
Right Arrow	Move right one character
Back Tab	Move left one tab stop. The Script-IV editor has preset tabs
Tab	Move right one tab stop. The Script-IV editor has preset tabs

Moving Between Lines

Up Arrow	Move up one line. At the bottom of a window, move up half a window
Down Arrow	Move down one line. At the top of a window, move down half a window
Page Up	Move up one window
Page Down	Move down one window
Return/Enter	Move cursor to the beginning of the following line
Home	Moves the cursor according to how many times the key is pressed: 1. Move to the left side of a line 2. Move to the top of the window 3. Move to the beginning of text 4. Move to the end of text

Deleting Characters

Backspace	Delete the character to the left of the cursor
Delete	Delete the current character
Line Clear	Delete characters to the right
Line Delete	Delete the entire line

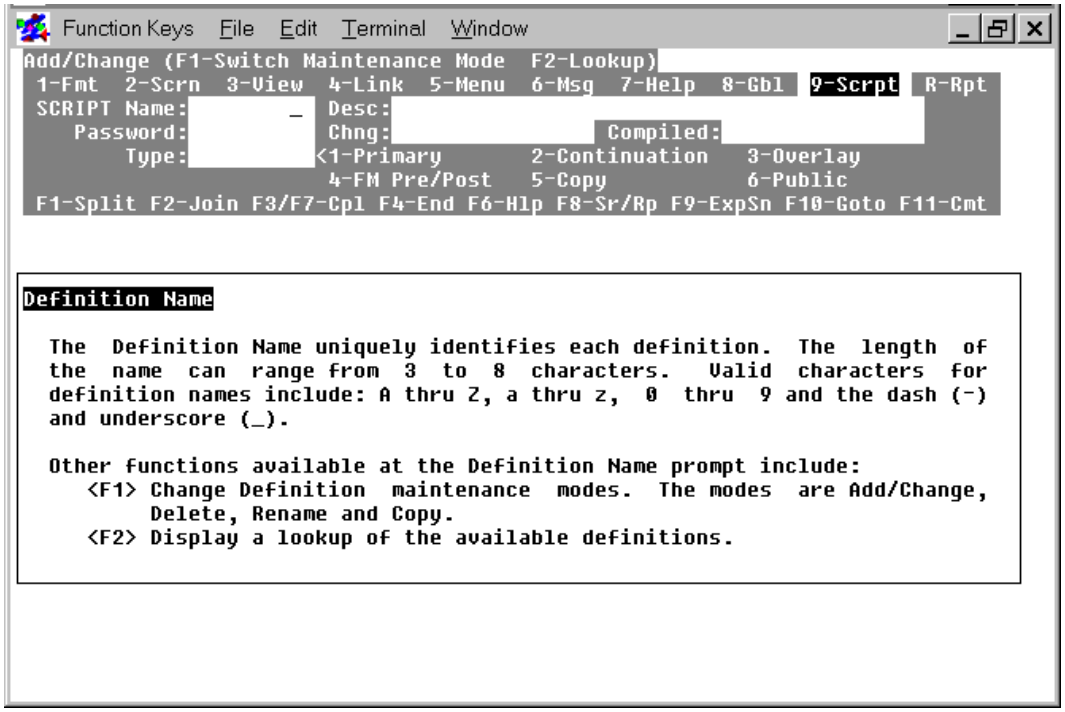
Inserting Characters

Insert	Pushes characters to the right
Line Insert	Inserts a blank line

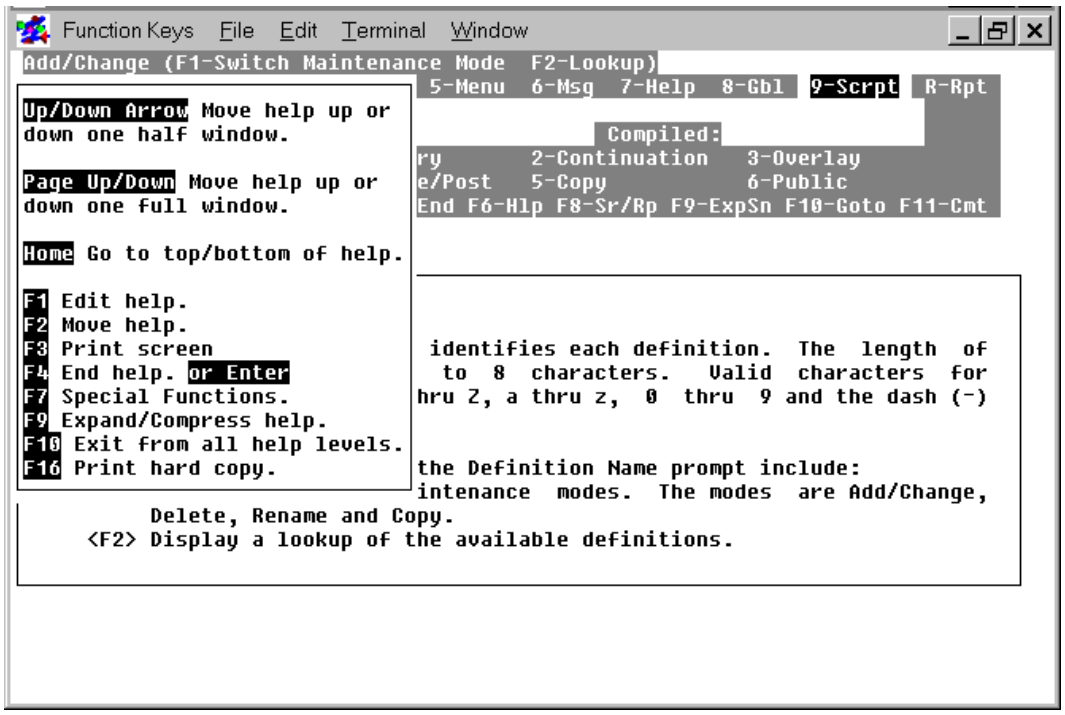
Getting Help

At any point where Script-IV is waiting for you to enter something, you can press the **F6** key and get instructions. In addition to help at menus and prompts, Script-IV provides a help network within the script editor itself. You can use the on-line help in the script editor as a reference when creating scripts. In many cases there are different levels of help you can access by pressing the **F6** key again.

For example, if you are defining a script, you can press the **F6** key at the SCRIPT Name: field to learn more on how to name a script. The following screen will be displayed:



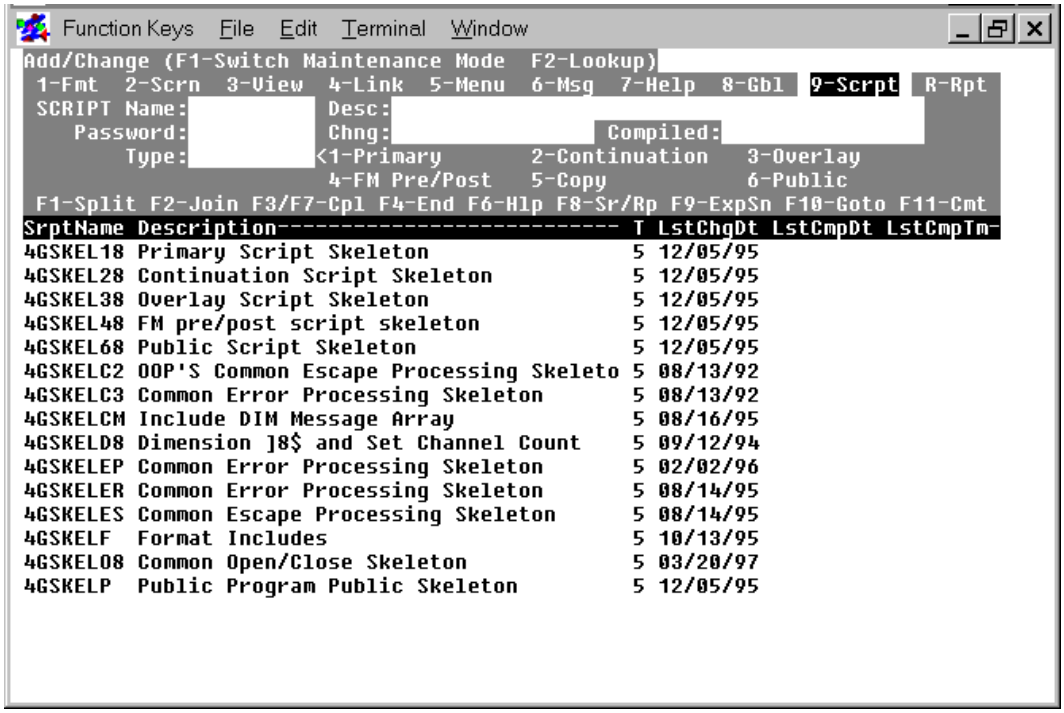
A help window displays information on valid values for script names. Many help windows enable you to retrieve more detailed information on a subject by pressing the **F6** key again. In this case, pressing the **F6** key will display the following screen:



A second help window is displayed. It contains information on the keys you can use in the help system. To return to the previous help window, you can press the **F4** key. To return to the SCRIPT Name: field, press the **F4** key again.

Script Definition Lookup

To lookup existing scripts, you can press the **F2** key on the SCRIPT Name field. A screen similar to the following screen will be displayed:



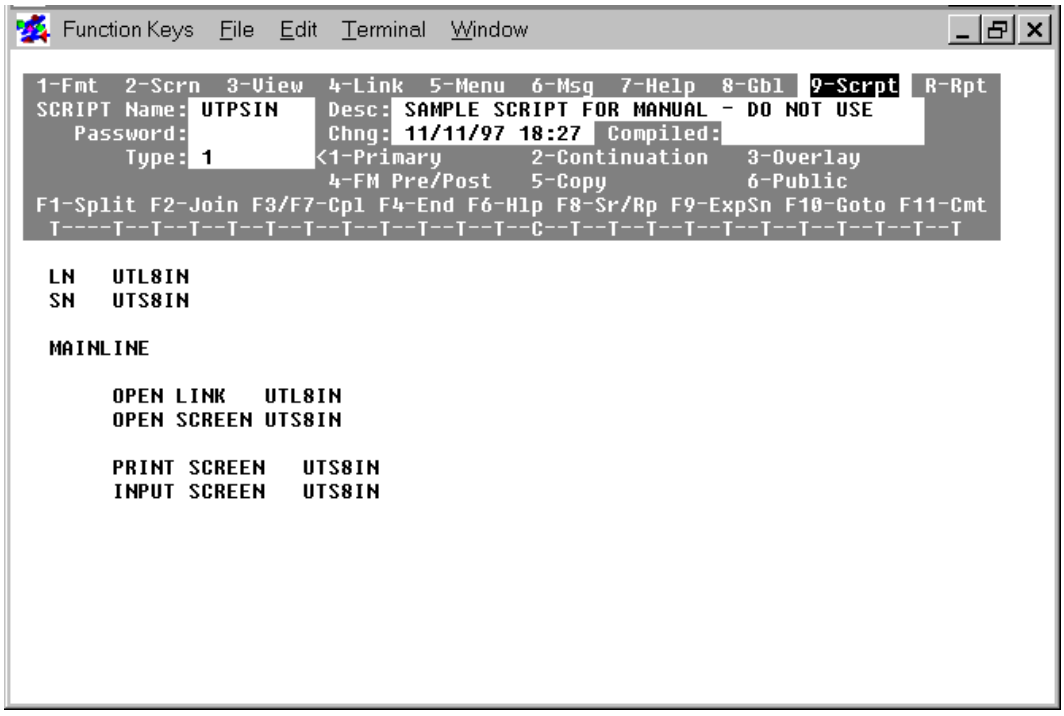
Each entry displays the script name, a description of the script, the number that specifies the type of script, the date the script was last modified, the date the script was last compiled, and the time the script was last compiled. You can use the **Page Down**, **Page Up**, or arrow keys to display information on all the scripts in the list.

To select and edit a script you can use the **Page Down**, **Page Up**, **up arrow**, or **down arrow** keys to move to the name of the script you plan to modify and press the **Enter** key. You will be returned to the Script Definition screen where you can edit the selected script.

If you decide to exit from the script list, press the **F4** key twice.

How to Compile from the Script Editor

Use the script editor to compile a single script. To compile from the script editor, you must access the script you plan to compile, for example:



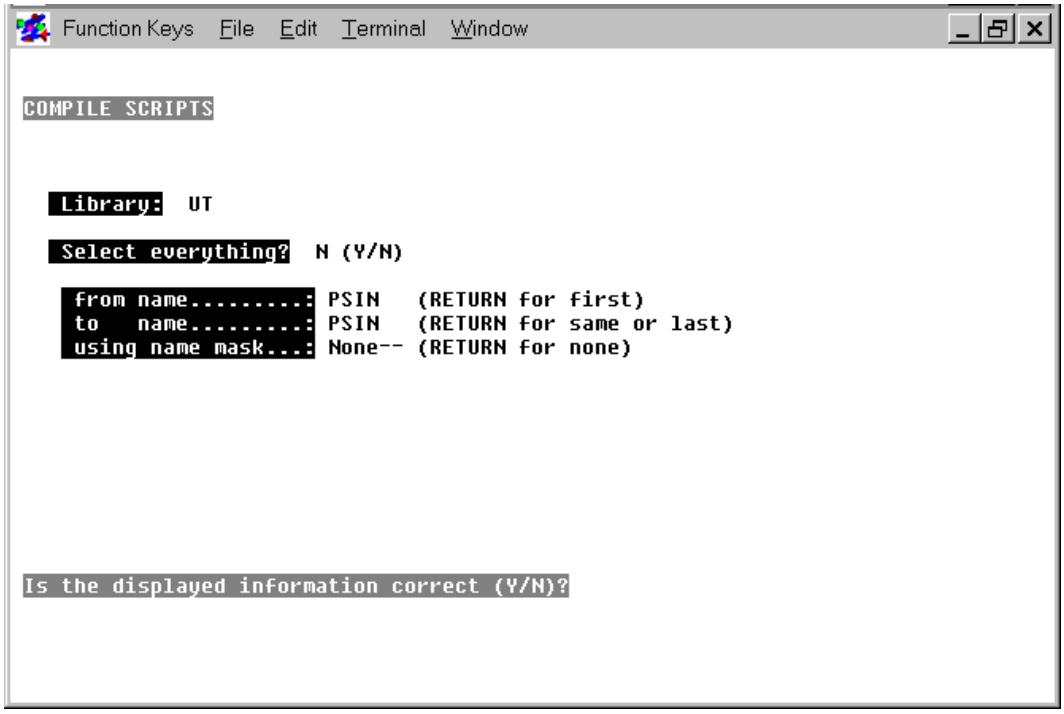
To begin the compilation process, press the **F3** or **F7** key. The following message will be displayed:

```
Do you want to save and compile (Y/N)?
```

Y starts the procedure.

N halts the compilation process. Control returns to the script editor.

Press the **Y** key. The following screen will be displayed:



Script-IV fills in the compilation information. You must answer the question at the bottom of the screen:

```
Is the displayed information correct (Y/N)?
```

Y begins compilation using the displayed information.

N enables you to re-specify the displayed information.

Press the **Y** key to compile the script. The script code will be listed as it is compiled. To stop the compilation process at any time, press the **Escape** key.

Errors pause compilation. Take one of the following actions:

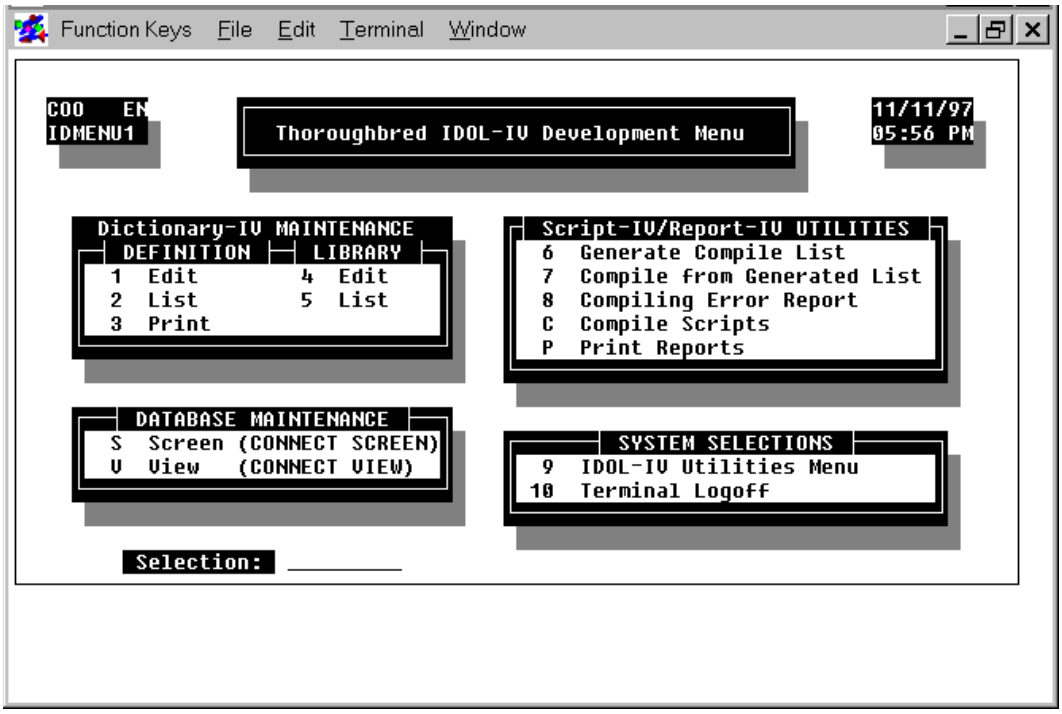
- To halt compilation, press the **F4** key.
- To continue compiling after an error is encountered, press the **Enter** key.
- If there is no response after 15 seconds, compilation continues as if the **Enter** key was pressed.
- If you started the compile from the Script-IV script editor, you can return to the script editor by pressing the **F1** key. You can fix the error, then press the **F3** or **F7** key to compile the script.

For more information on the errors you may encounter see the sections on **How to Manage Compilation Errors** and **Error Messages**.

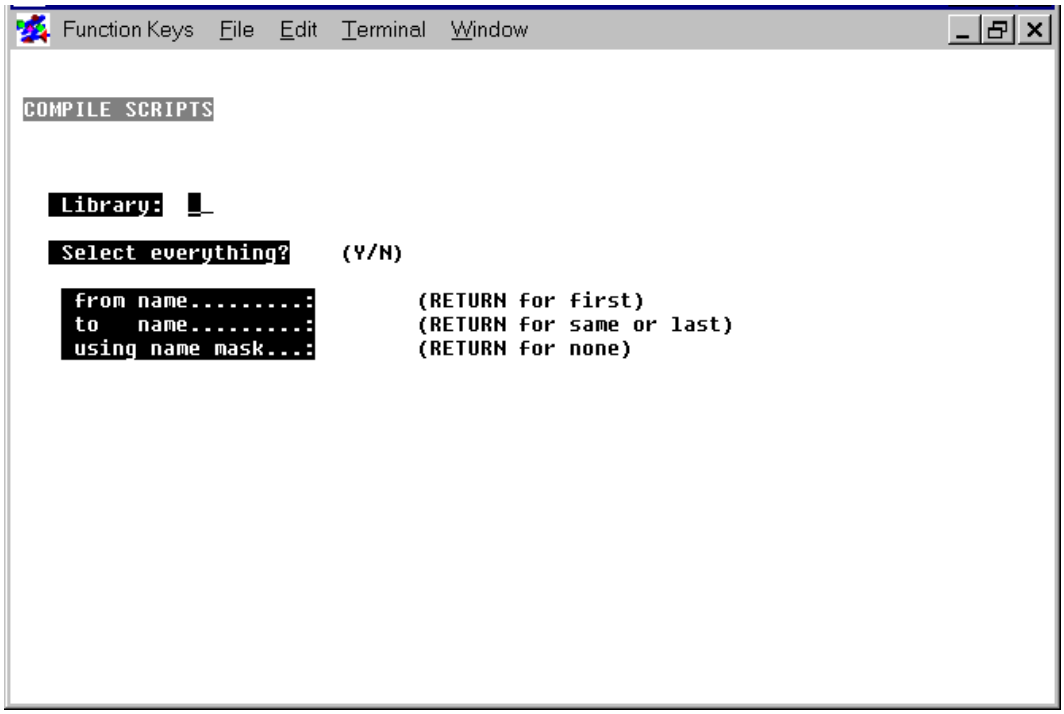
After the compiler finishes processing, you will be returned to the last active menu.

How to Compile from the Dictionary-IV Development Menu

Use the Dictionary-IV Development Menu to compile a single script, a range of scripts, or a library of scripts:



To compile from the Dictionary-IV Development Menu, type **C** and press the **Enter** key. The COMPILE SCRIPTS screen will be displayed:



You must answer the following questions:

Library:

Enter the name of the library that contains the scripts you plan to compile.

Select everything (Y/N)?

Select one of the following:

Y means every script in the library will be compiled.

N means you plan to specify a range of scripts to compile.

from name:
to name:

To define a range of scripts, you can use the following specifications:

- To specify all the scripts in the library, press the **Enter** key in the **from name** field and press the **Enter** key in the **to name** field.
- To specify a range of scripts, type the name of the first script you plan to compile in the **from name** field and the name of the last script you plan to compile in the **to name** field.
- To specify a single script, type the name of the script you plan to compile in the **from name** field and press the **Enter** key in the **to name** field.

In most cases, you only need to type enough of a script name to identify it as a unique name, then press the **Enter** key and let the system fill in the rest of the name.

using name mask:

A name mask enables you to select a set of scripts from a range of scripts. You do not have to compile all the scripts in the range and you do not have to select individual scripts to compile one at a time. If you do not plan to use a mask, press the **Enter** key.

Use the following information to define a mask:

- A mask is a string of characters used to test the string of characters in the script name. A script name that matches the mask is selected for compilation.
- A mask can contain match and passing characters. A character in a script name must be the same character as the match character and occur in the same position. The **?** is the pass character, which means that no match in this position is needed.

Mask	action	definition names		
A?	selects	AB	Ac	A1
	bypasses	aB	cA	11

After you have answered all of these questions, the following message will be displayed:

```
Is the displayed information correct (Y/N)?
```

Enter one of the following:

Y begins compilation using the displayed information.

N enables you to respecify the displayed information.

Press the **Y** key to compile. The name of each script and its description will be displayed as it compiles. The script code will be listed below this information. To stop the compilation process at any time, press the **Escape** key.

Errors pause compilation. To continue compiling after an error is encountered, press the **Enter** key. For more information on the errors you may encounter see the sections on **How to Manage Compilation Errors** and **Error Messages.**

After the compiler finishes processing, you will be returned to the COMPILE SCRIPTS screen.

How to Use a Compile List

A compile list is a list of scripts that need to be compiled before they can be executed. To maintain applications, you can use the Generate Compile List and Compile from Generated List Utilities from the Dictionary-IV Development Menu.

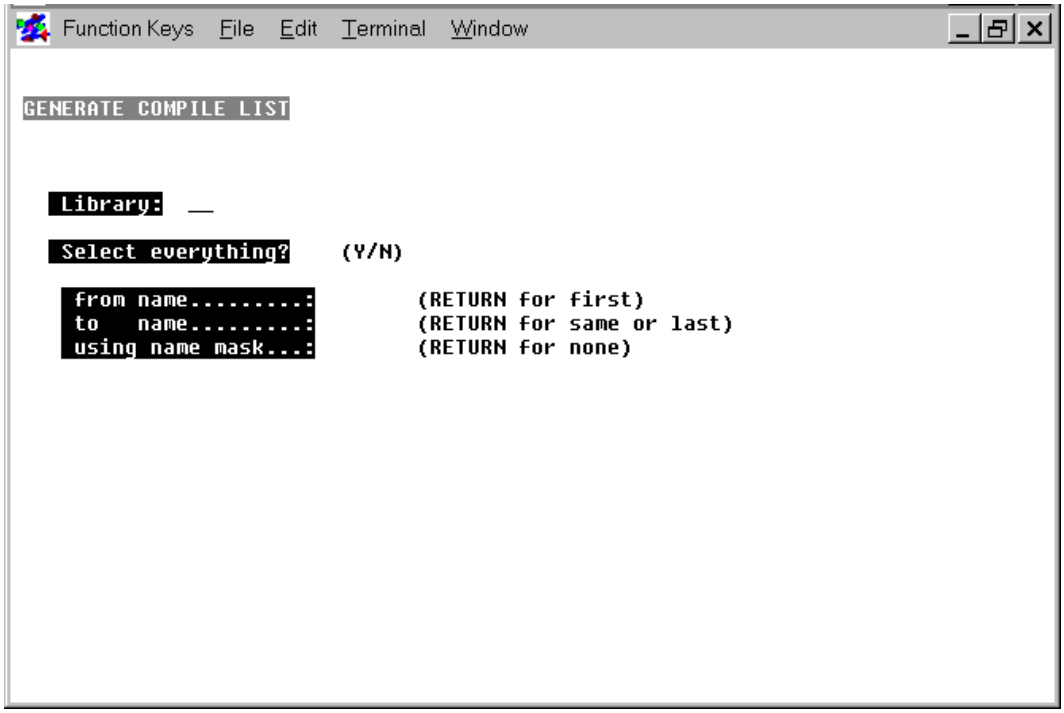
The Generate Compile List Utility searches through a range or library of scripts. It compares the change date of the script, the change date of any copy script used by the script, and the change date of any definition such as a format or screen used by the script, to the compile date. If any of these change dates are more recent than the compile date, the utility puts the script in a compile list. After the library or range is checked, you can specify another range or library to search for script names to add to the list.

The name of the compile list is **4GLCL***tt*, where *tt* is the terminal ID. Any compile list is unique to the terminal where it is created. Each time you select the Generate Compile List Utility the compile list is erased before a new list is created.

The Compile from Generated List Utility compiles the scripts in the compile list.

How to Build a Compile List

To create a compile list, go to the Dictionary-IV Development Menu. Type **6** and press the **Enter** key. The GENERATE COMPILE LIST screen will be displayed:



You must answer the following questions:

`Library:`

Enter the name of the library that contains scripts you plan to include in the compile list.

`Select everything (Y/N)?`

Select one of the following:

Y means every script in the library will be checked.

N means you plan to specify a range of scripts to check.

`from name:`
`to name:`

To define a range of scripts, you can use the following specifications:

- To specify all the scripts in the library, press the **Enter** key in the **from name** field and press the **Enter** key in the **to name** field.
- To specify a range of scripts, type the name of the first script you plan to compile in the **from name** field and the name of the last script you plan to compile in the **to name** field.

- To specify a single script, type the name of the script you plan to compile in the **from name** field and press the **Enter** key in the **to name** field.

In most cases, you only need to type enough of a script name to identify it as a unique name, then press the **Enter** key and let the system fill in the rest of the name.

using name mask:

A name mask enables you to select a set of scripts from a range of scripts. You do not have to compile all the scripts in the range and you do not have to select individual scripts to compile one at a time. If you do not plan to use a mask, press the **Enter** key.

Use the following information to define a mask:

- A mask is a string of characters used to test the string of characters in the script name. A script name that matches the mask is selected for compilation.

- A mask can contain match and passing characters. A character in a script name must be the same character as the match character and occur in the same position. The **?** is the pass character, which means that no match in this position is needed.

Mask	action	definition names		
A?	selects	AB	Ac	A1
	bypasses	aB	cA	11

After you have answered all of these questions, the following message will be displayed:

Is the displayed information correct (Y/N)?

Enter one of the following:

Y begins checking change dates using the displayed information.

N enables you to respecify the displayed information.

Press the **Y** key to check change dates. If no script needs to be compiled, the compile list will be empty. After the utility finishes processing, you will be returned to the GENERATE COMPILE LIST screen.

How to Compile from a Compile List

To compile all of the scripts on the compile list, go to the Dictionary-IV Development Menu. Type **7** and press the **Enter** key. The following message will be displayed:

Do you want to compile from 4GLCL*tt* generated date - time (Y/N)?

Enter one of the following:

Y compiles the scripts on the compile list.

N halts the process and returns you to the Dictionary-IV Development Menu.

Press the **Y** key to compile the scripts on the compile list. If the compile list exists, and if it is not empty, the name of each script and its description will be displayed as it compiles. The script code will be listed below this information. To stop the compilation process at any time, press the **Escape** key.

Errors pause compilation. To continue compiling after an error is encountered, press the **Enter** key. For more information on the errors you may encounter see the sections on **How to Manage Compilation Errors** and **Error Messages**.

After the compiler finishes processing, you will be returned to the Dictionary-IV Development menu.

If the compile list does not exist or is empty, the utility will display a message. Press the **Enter** key to return to the Dictionary-IV Development Menu.

How to Define Your Own Compile List

If the Generate Compile List Utility described in the preceding section does not meet site requirements, you can use alternate methods to build compile lists. The following sample Thoroughbred Basic code is distributed with Dictionary-IV as the IDU006 program. It builds a compile list from a range of scripts. You can modify the code to match your specifications.

```
0010  REM &REM&

0100  SETERR 9000;                  ! Set error trap.
      SETESC 9000;                  ! Set escape trap.
      PRINT 'WC', 'CN',             ! Clear all windows.
        "Build List of Scripts to Compile",;
      C1=UNT;                       ! Get an available channel.
      OPEN (C1) "IDDBD";            ! Open the dictionary.
      FID$=FID(C1), <_>             ! Get the FID of dictionary.
      D=DEC(FID$(20,1))             ! Save disk number.

0200  INPUT @(0,2), 'CL',           ! Get "from script" name.
         "Enter starting script name: ",FROM$;
      IF CTL=4                      ! If F4 key is pressed,
         GOTO 9000                  !  leave the procedure.
      FI;
      EXTRACT (C1,KEY="P"+FROM$,DOM=201)  ! Move key pointer.

0210  K$=KEY(C1,END=200);           ! Get next key.
      READ (C1);                    ! Unextract the key.
      IF K$(1,1) <> "P"             ! If not a script record
         GOTO 200                   !  reenter from script name.
      FI
```

```
0220  INPUT @(0,3),'CL'              ! Get "to script" name.
         "Enter ending script name: ",TO$
      IF CTL=4                       ! If F4 key is pressed
         GOTO 9000                   !  leave the procedure.
      FI;
      IF TO$=FROM$ AND TO$<>""       ! If "to value" = "from value"
         TO$=TO$+"~"                 !  add last value to entry
      FI;
      IF TO$=""                      ! If Enter key pressed
         TO$="~"                     !  set to last key value.
      FI$
      IF TO$<FROM$                   ! If "to value" < "from value"
         GOTO 220                    !  reenter "to value".
      FI;
      FROM$="P"+FROM$,               ! Add "P" key prefix to "from
      TO$="P"+TO$                    ! value" and "to value".

0300  CLF$="4GLCL"+FID(0);           ! Set compile list file name.
      CLEAR ERC;                     ! Clear error control.
      WHILE ERC=0;                   ! Erase compile list file on
         ERASE CLF$, ERC=1;          ! all available disks.
      WEND;
      SORT CLF$,9,2000,D,0;          ! Create compile list file
      C2=UNT;                        ! Get an available channel.
      OPEN (C2) CLF$;                ! Open the compile list file.
      WRITE (C2,KEY=$00$+DAY);       ! Write data header record.
      TO=TIM,T1=FPT(TO)*60,          ! Resolve time.
      TO$=STR(TO:"00")+":"+
         STR(T1:"00")+":"+
         STR(FPT(T1)*60:"00");
      WRITE (C2,KEY=$01$+TO$);       ! Write time header record.
      EXTRACT (C1,KEY=FROM$,DOM=301)! Move key pointer back to
                                     ! from record.

0400  K$=KEY(C1,END=9000);           ! Get next key.
      IF K$>TO$                      ! If next key is out of range
         GOTO 9000                   !  leave the procedure.
      FI;
      IF STL(K$)=9                   ! If key is a header record
         PRINT @(40,0),K$(2),;       !  print current script name
         WRITE (C2,KEY=K$)           !  write it to compile list
      FI;
      READ (C1,KEY=K$+$FF$,DOM=400);! Move pointer to next! header.
      GOTO 400                       ! Go get next key.

9020  RUN "ID"                       ! Return to Dictionary-IV menu.
```

How to Manage Compilation Errors

You must remove errors from scripts to produce working programs. The following sections describe how to manage common compilation problems:

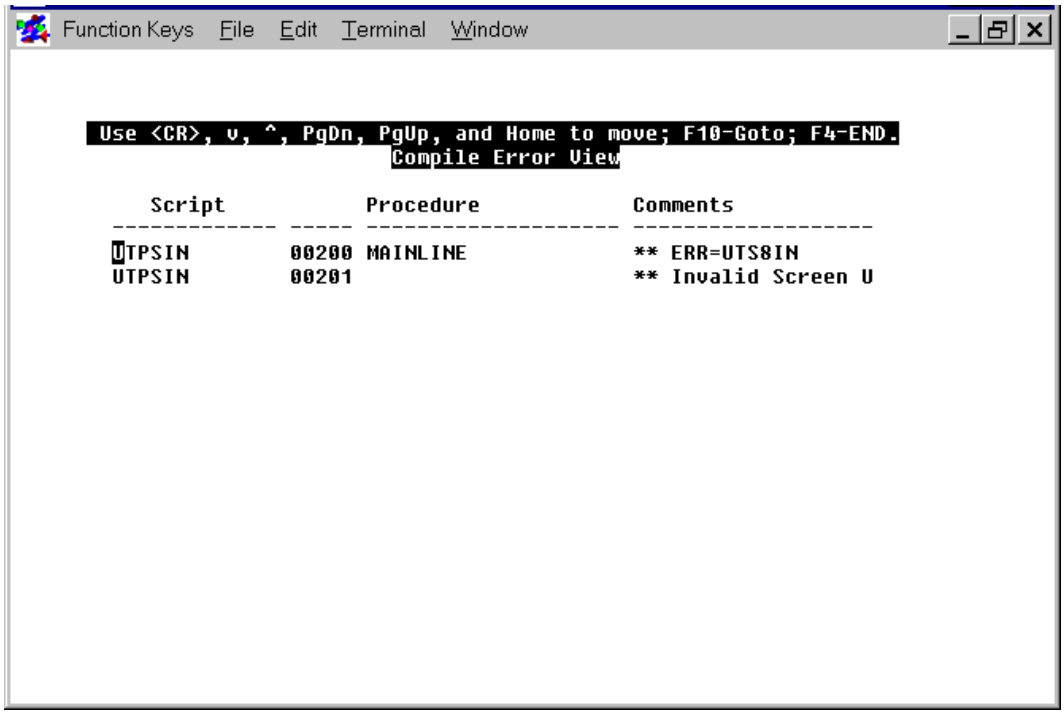
- How to Use the Compile Error Report describes how to get a list of the errors produced during compilation.

- How to Resize Scripts and Programs describes strategies you can use when scripts or programs are too large to be compiled.

How to Use the Compile Error Report

The Compile Error Report makes a list of the errors produced by the last compile. When you compile a script or a group of scripts, all errors are written to the compile error log file. When you perform a new compile, this file is overwritten.

To display the compile error log, go to the Dictionary-IV Development Menu. Type **8** and press the **Enter** key. If the compiler found errors, a screen similar to the following will be displayed:



To move through the screen, press the down arrow, up arrow, **Page Down**, **Page Up**, or **Home** key. To go to a specific line, press the **F10** key.

To exit the utility, press the **F4** key. The following message will be displayed:

```
Do you want a printed copy (Y/N)?
```

Enter one of the following:

Y prompts you to enter a printer code where the hard copy will be produced.

N exits the utility and returns you to the Dictionary-IV Development Menu.

How to Resize Scripts

If a script is too large, the script will not be compiled and the following message will be displayed:

`Generated code is` *`nnnnnn`* **`bytes and cannot be saved`**

A compiled script cannot exceed 64K. If a script is too large, consider turning parts of the script into overlay scripts. For more information on overlay scripts see the section on **Different Types of Scripts** in this manual.

Error Messages

The following sections contain lists of syntax errors and other compilation errors.

Syntax Errors

**** ERR=(script statement where error found)**

The compiler expects a Script-IV key word and one is not found, or a key word other than a data declaration is found in the Data Declaration section of the script.

**** ERR: (statement with basic compile error)**

The script has attempted to open or access undeclared definitions, or the script has attempted to access a definition that has not been opened. Another common cause of this error is making assignments without using the LET keyword.

**** IF/ENDIF mismatch**

The compiler has encountered a new procedure with unmatched IFs and ENDIFs; there must be a corresponding ENDIF for each IF/THEN statement.

**** IF/LOOP nesting error**

The compiler has encountered a procedure with misplaced ENDIFs and ENDLOOPs. Check the order of ENDIFs and ENDLOOPS and make sure they match the nesting order.

**** IF/THEN mismatch**

The compiler has encountered an IF statement with an unmatched THEN; there must be a corresponding THEN for each IF statement.

**** Invalid IF structure**

The compiler has encountered incorrect IF/ENDIF, LOOP/ENDLOOP nesting.

**** Invalid Link Alias**

The link alias is too long. The limit is 8 characters.

**** LOOP/ENDLOOP mismatch**

The compiler has encountered a new procedure with unmatched LOOPs and ENDLOOPs. There must be a corresponding ENDLOOP for each LOOP statement.

**** No statement generated**

The compiler has encountered incomplete syntax. Check for incorrect command structure or key words, which may be used out of context.

**** PRE/POST "ALL" must be first or last**

The compiler has encountered a PRE or POST option in an INPUT SCREEN command that specifies ALL, but ALL is not the first or last specification. Specify ALL in the first or last position for the PRE or POST option.

**** undefined procedure**

The compiler could not locate the referenced procedure.

Other Compile Errors

**** "xxxxxxxx" already exists but is not a program**

The referenced script name is not a program file.

**** Copy source xxxxxx not found**

The source specified in a COPY or INCLUDE statement cannot be located.

**** Copy source xxxxxx is being edited by another user**

The source specified in a COPY or INCLUDE statement was located but it is being accessed by another user.

**** Duplicate line number**

The same Thoroughbred Basic line has been generated twice. Call Thoroughbred Product Support.

**** Duplicate procedure**

The same procedure name has been used twice in a script.

**** Invalid format (format-name)**

The specified format is not defined in Dictionary-IV, or it is corrupt.

**** Invalid link (link-name)**

The specified link is not defined in Dictionary-IV, or it is corrupt.

**** Invalid screen (screen-name)**

The specified screen is not defined in t Dictionary-IV, or it is corrupt.

**** Invalid view (view-name)**

The specified view is not defined in Dictionary-IV, or it is corrupt.

****(link-name) has no text field defined**

There is no text field defined in the script. Specify a text-id for the relevant command.

****(link-name) text field "text-id" is not defined**

The specified text field is not defined in the link.

**** Missing Text File Name**

This message can only occur for links that specify MSORT or TISAM files. Check the TEXT File specification in the Link Definition.

**** No sorts defined for (link-name)**

There are no sorts defined in the specified link.