Thoroughbred® OPENworkshop™ Reference Manual

THOROUGHBRED
software international, inc

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 085588-2638

Document Number: OW8.8.0M101

INTRODUCTION

OPENworkshop is the Thoroughbred Object-oriented Development Environment. It features new concepts that change the way developers build business software.

OPENworkshop offers:

- Much lower development and maintenance costs
- A much more flexible system for users
- The choice of both Character and Graphical interfaces

OPENworkshop is based on simple concepts:

- Objects and attributes
- Wholes and parts
- Classes and members

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

Object Technology

Object Technology began development in the early 1980's. Founded on research designed to maximize the re-usability of software, Object Technology (OT) creates Objects that reflect real-world entities, and Methods that implement the way Objects behave.

As is often the case, this new technology has taken a long time to get to market. Two fundamental problems impeded its deployment in practical business applications: hardware costs and the lack of suitable software infrastructures.

Object Technology demands more memory and processing power than traditional software development environments. The cost differential has been large enough, until recently, for end-users of business applications to avoid OT-based solutions. Today, much better price/performance values are available from hardware suppliers.

True Object Technology is rigid and inflexible when developing Transaction types of applications. For this reason Thoroughbred's OPENworkshop is defined as "Object Oriented". Object Oriented provides the flexibility in developing applications for "real world" business applications, by allowing the Object Rules to be modified and enhanced when needed to satisfy application requirements. Most available OT-based systems are usually Object Oriented because of this flexibility.

Enormous development efforts have been put into operating systems and utilities over the history of commercial computing. They have combined to create a wealth of resources available for application developers to use in their solutions. Early attempts to deliver OT-based development environments ignored these resources in a drive to overturn traditional architectures. Now, mainstream environment developers have found ways to deliver Object Technology while leveraging on the available infrastructures.

As a result, practical OT-based solutions are now deliverable, and we can expect to see significant market growth as the “early adopters” consolidate their positions.

Thoroughbred Mission

Thoroughbred's formal Mission is to provide the most highly portable language development environments for developers creating end-user business applications.

The word “Portable” has been used widely in the software industry, and people attribute many different meanings to it. Thoroughbred itself has two distinct developer requirements in mind when committing to portable language development environments.

Firstly, developers wish to be able to design an application for their chosen markets once, and then implement it on many different hardware and software environments, depending on their own customer's needs and preferences. Thoroughbred‘s mission is to deliver this portability, allowing the developer to maintain only a single application code set.

Secondly, developers have invested heavily, designing applications for the end-user business market, and need to be able to conserve that past investment. Changes in technology, costs, and the competitive environment will drive them to adapt and enhance their applications, but Evolution, rather than Revolution, is the preferred strategy. Thoroughbred takes responsibility for enabling developers to choose this strategy. Ten, or even five years ago, it was possible to deliver the Evolution capability by guaranteeing upward portability in the programming language.

More recently, demands for evolution have become more complex. Evolution requirements include the enhancement of the user interface of their application to support graphical presentation (GUI), or the data environment to encompass Client/Server architectures. Thoroughbred's solution to this level of portability is based on the Dictionary-IV system dictionary. Thoroughbred guarantees that all applications based on Dictionary-IV will operate **without change** in all environments supported by Thoroughbred.

Positioning OPENworkshop

OPENworkshop is an Object-oriented development and run-time environment encompassing Thoroughbred’s three-tier environment.

OPENworkshop Features:

- Data controls the application
- Applications can be developed incrementally
- Relational Integrity is maintained and insured

- Can modify Object Rules
- Global Definition-based
- GUI and Character Applications with same Program Code
- Client/Server or Local Host-based Applications
- Internet Transaction Processing
- Window, UNIX, Linux, or DEC VMS is supported
- Thoroughbred, ORACLE, VMS, and a variety of ODBC databases are supported

OPENworkshop provides the tools to develop mission critical applications for a variety of business requirements and system configurations.

Most of the major relational database vendors have developed or are developing Object-oriented products. In addition to these suppliers a new set of ventures are attempting to develop new Object-oriented products from scratch, using entirely new architectures. OPENworkshop is an evolution of Thoroughbred’s Dictionary-IV, and builds on the portability of previous Thoroughbred development products.

OPENworkshop provides support for enterprise-wide, distributed Object databases as well as local, host-based systems. Thoroughbred has always had a policy to make its development environments open for its developers. As these new products become established, Thoroughbred will enable developers to take advantage of them.

OPENworkshop supports an Object-oriented Programming environment. Built to support Windows clients and UNIX/Linux Servers running either Thoroughbred’s robust database or Relational Databases including ORACLE and most ODBC compliant databases including Informix, Sybase, and SQL server. OPENworkshop enables the application designer to define and use the most appropriate database for the application rather than being limited to a proprietary database or limited third party databases. Each Format definition can have its own database support.

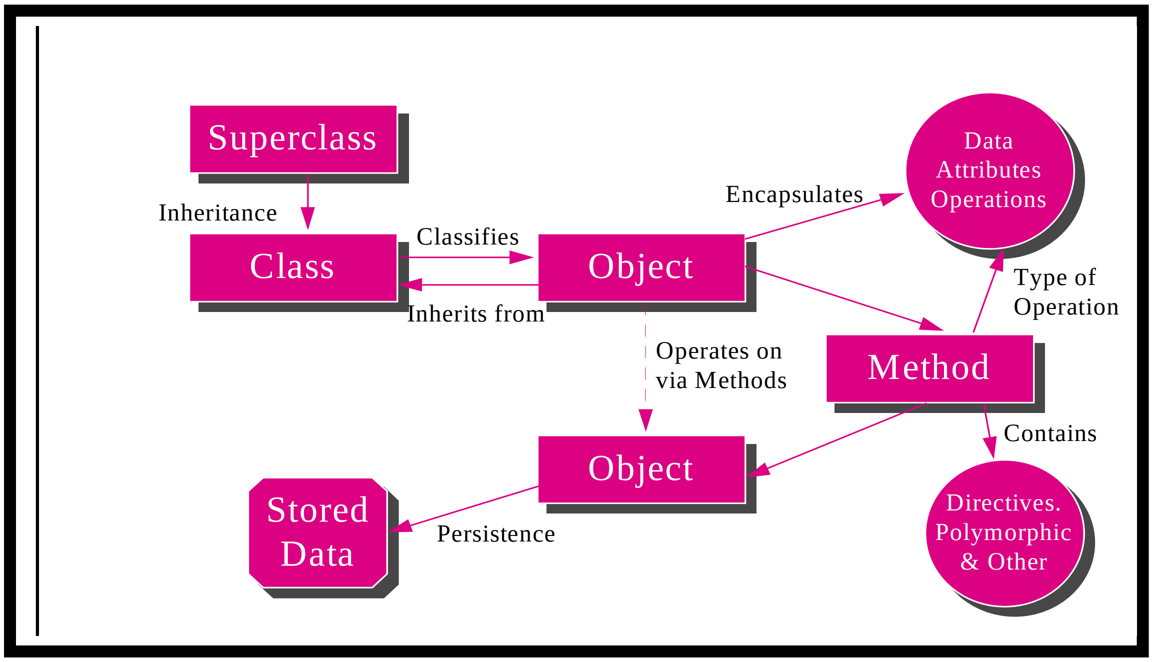
OPENworkshop provides support for developers wishing to use a graphical user interface (GUI) on a client workstation. Thoroughbred's VIP provides a Graphical User Interface to the application without requiring changes to the application’s code.

OPENworkshop is a highly portable Object-oriented development environment for developers creating end-user business applications. It embodies Object Technology and delivers the benefits. It is based on a time proven system engine. It uses and supports components that Thoroughbred has developed and refined for many years, while at the same time introducing many new features.

Deliberately, OPENworkshop is not a break with the past, but a path for *evolution* to the future.

Object Technology Concepts

Object Technology embodies key concepts that combine to deliver its benefits. Extensive technical literature[1] is available on the subject, but the following provides a summary of the concepts and an introduction to how OPENworkshop delivers them.



Object

Is any thing, real or abstract, about which we store data and those operations that manipulate the data.

(James Martin)

OPENworkshop allows the developer to store information that reflects the properties of real world items in Data-names. The operations that may manipulate the data are defined in Methods and are associated with the Data-name.

Class

A collection of objects, which share common attributes and methods.

(Ian Graham)

[1] See bibliography for a small sample

In OPENworkshop Classes define a collection of Objects. Classes themselves are collected into sets with common structures or behavior. For example, all Views in OPENworkshop have similar behavior.

Method

An implementation of an operation. Code that may be executed to perform a requested service. Methods associated with an Object may be structured into one or more programs.

(Object Management Group)

OPENworkshop Methods perform operations on Data-names. Some specific types of operation are identified for particular support by OPENworkshop, including:

- Pre- and Post-Processing Methods prepare an Object for modification by the end-user and validate the data afterwards.
- Insert Methods assist in creating a new Object.
- I/O triggers validate the database and ensure that updates are performed consistently.

Inheritance

The construction of a definition by incremental modification of other definitions.

(Object Management Group)

In OPENworkshop all Classes are created and modified incrementally, and any definition created is interpreted consistently wherever it is referenced. OPENworkshop Formats and Links are used to collect these definitions together.

Encapsulation

Are the results (or act) of hiding the implementation details of an Object from its user?

(James Martin)

An Object may CONNECT to another OPENworkshop Object, and allow it to perform the operations it needs. When it has finished, it may return a value.

It is the combination of these concepts that allows developers to reduce application development and maintenance costs.

Polymorphism

The ability to use the same expression to denote different operations.

(Ian Graham)

An expression or message can operate on Objects of different Classes. This type of re-useable code greatly reduces development and maintenance costs.

Recursive

Objects can send messages to their own Methods recursively or send messages to themselves.

(Ian Graham)

The ability to interrupt an action to undertake another action or subroutine, and then to interrupt this again with the same subroutine and so on. For the Thoroughbred Environment this is equivalent to Public Programs.

Persistence

The property of an object by which its existence transcends time.

(Object Management Group)

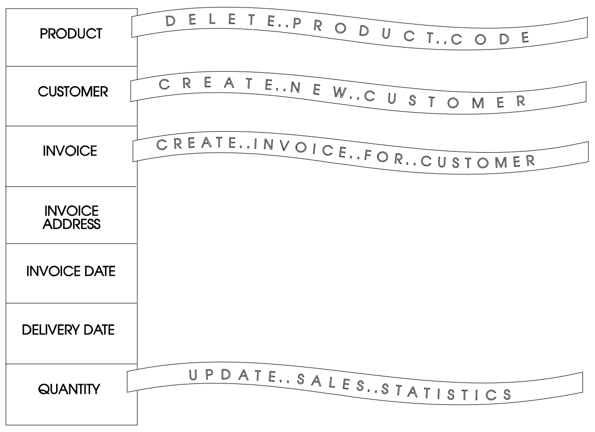
The value of data remains after the Class or Method that created it no longer exists. An example is Data Objects that can be stored in files, which is the ultimate form of persistence.

OPENworkshop System Concepts

Traditional software architectures are hierarchical in design, driven by a number of programs. Typically, users can select a program from a set of choices on a menu. Programs engage in a dialog with the user through data input screens, and may display data in lists and tables. The application program controls both the dialog with the user and the maintenance and updating of data items.

OPENworkshop redefines the relationships between data and program code, and between application designer and application user. Developers using OPENworkshop must design their Data Objects first, and then associate “Methods”, which are independent modules of program code, with data items.

Because individual data elements control their own code, OPENworkshop eliminates any concept of a MAIN program. In place of a control structure that is imposed on the user through menus and escape keys, (all of which must be designed and planned for by the developer), OPENworkshop provides CONNECT directives, pop-up menus and user interface support that allows the user to control the dialogue with the system.



In OPENworkshop the data controls the application code, not the other way around. The system architecture enables the developer to partition the application code into Methods associated with Objects.

Because Objects are then packaged with their Methods, they are easy to re-use. Previous work is immediately available to use when creating other Objects with similar properties. Future modifications are automatically reflected in all places where the Object is used.

These benefits are important during initial application design, but even more important during the much longer maintenance life of an application.

The User Interface

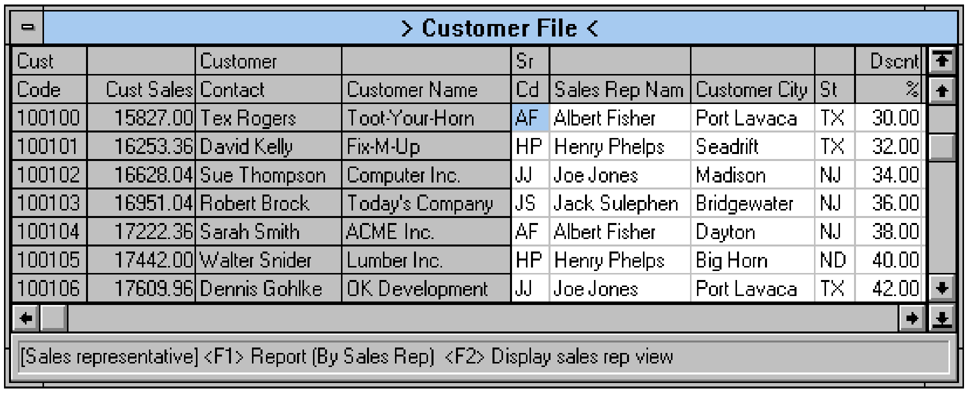
OPENworkshop Presentation Classes (View, Screen, Menu, Report, Query, and Help) are used by developers to implement the user interface for the application. These are described in more detail later, but the OPENworkshop user interface contains some important features that are highlighted below.

Navigating through an OPENworkshop Application

OPENworkshop offers a user interface model that is quite different from traditional menu-controlled, form-oriented data processing systems. It is important that you gain a clear understanding of the flexibility and benefits that OPENworkshop offers the user at an early stage. The sample application delivered with OPENworkshop will give you an opportunity to try it for yourself.

The OPENworkshop user interface begins with the view. The view is used as the primary display while an OPENworkshop application is running. Traditional systems begin with a menu or screen.

As in a spreadsheet, a view presents information in rows and columns. It is a form of presentation that is dense in information, and highly flexible.


> Customer File <

Cust Code	Cust Sales	Customer Contact	Customer Name	Sr Cd	Sales Rep Nam	Customer City	St	Dscnt %
100100	15827.00	Tex Rogers	Toot-Your-Horn	AF	Albert Fisher	Port Lavaca	TX	30.00
100101	16253.36	David Kelly	Fix-M-Up	HP	Henry Phelps	Seadrift	TX	32.00
100102	16628.04	Sue Thompson	Computer Inc.	JJ	Joe Jones	Madison	NJ	34.00
100103	16951.04	Robert Brock	Today's Company	JS	Jack Sulephen	Bridgewater	NJ	36.00
100104	17222.36	Sarah Smith	ACME Inc.	AF	Albert Fisher	Dayton	NJ	38.00
100105	17442.00	Walter Snider	Lumber Inc.	HP	Henry Phelps	Big Horn	ND	40.00
100106	17609.96	Dennis Gohlke	OK Development	JJ	Joe Jones	Port Lavaca	TX	42.00

[Sales representative] <F1> Report (By Sales Rep) <F2> Display sales rep view

The view shown above gives information about customers and their orders. It could be considered to be a starting point for anyone concerned with processing orders.

The view allows users to explore information throughout the system. In the view shown above, select **Report (F1)** while in the Sales Rep column to produce a report of sales by Sales Reps. Alternatively, select **Display Sales (F2)** to display information about Sales by Sales Reps.

Users can chain from view to view almost without restriction. Select **Close (F4)** to return to the previous item.

Developers can make other classes available through select functions. Pop-up menus, screens, reports and queries, and more views can all be associated with select functions for any column.

Many of the above effects can also be achieved through other development environments. The difference is the ease with which the OPENworkshop developers can create this level of flexibility, without losing control of the integrity of the data managed by the application.

The following are some of the rules that an OPENworkshop user can rely upon when working within a view:

- *If you can see it, you can find out more about it.*

  The Customer Sales column gives the total sales to the customer. Move the cursor to that column, select **Invoice Detail** and OPENworkshop opens a view showing details of all invoices.

- *If you can see it, you can change it.*

  The user can modify the data within any of the fields displayed in a view, provided the developer has allowed it. When the data is changed, methods and triggers ensure that all other related files are also modified appropriately. OPENworkshop can control these modifications, ensuring that only valid values are entered. Data validation can be as simple or complex as needed.

- *If you can't see it, it won't take long to find it.*

  With OPENworkshop the developer does not have to carefully predict where a user may wish to go next from any screen or menu option. The environment makes it easy for the developer to open doors to views, and it deals with data consistently, without having to close and open programs.

- *If you have to go do something else, everything will still be here when you come back.*

  Partway through entering a sales order you may need to go and attend to an outstanding invoice. OPENworkshop not only lets you do this, but it preserves the entire context and restores it when you return, without the developer having to make specific arrangements to allow it.

- *You'll wonder how you did without it.*

  A system that begins with the view is unconventional at first. We are too used to the rigid control that menus imply. Use the system for a day or so, and you will see the advantages it offers.

Mix both Character Terminals and Graphical Workstations

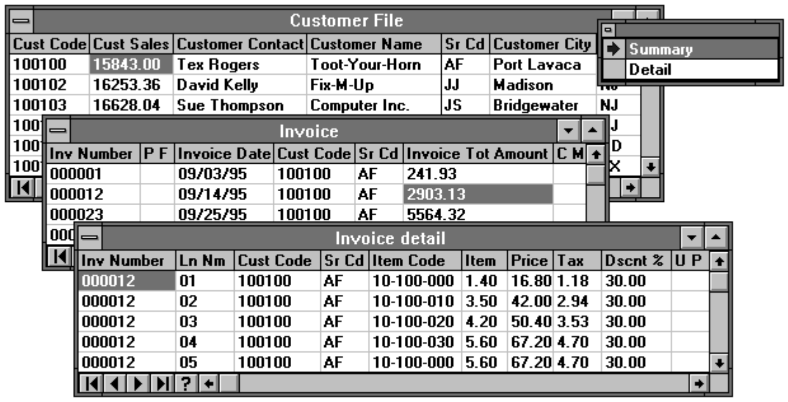
OPENworkshop offers a unique choice for the physical presentation of these Classes. The user may be equipped with a character terminal or a graphical workstation. Graphical workstations are PC-based workstations running a Microsoft operating system.

Also all of these options can be mixed in a single application site, with a single set of code. OPENworkshop manages the communication with the specified user interface device as appropriate.

It would not be uncommon to implement a business system on OPENworkshop with a mixture of character terminals and graphical workstations.

Views

A View presents information in rows and columns. Like a spreadsheet, it is a form of presentation that is dense in information, and highly flexible. People naturally relate to this layout, once they see how flexible it is.



Customer File

Cust Code	Cust Sales	Customer Contact	Customer Name	Sr Cd	Customer City	
100100	15843.00	Tex Rogers	Toot-Your-Horn	AF	Port Lavaca	
100102	16253.36	David Kelly	Fix-M-Up	JJ	Madison	
100103	16628.04	Sue Thompson	Computer Inc.	JS	Bridgewater	NJ
100						J
100						D
100						X

Invoice

Inv Number	P F	Invoice Date	Cust Code	Sr Cd	Invoice Tot Amount	C M
000001		09/03/95	100100	AF	241.93	
000012		09/14/95	100100	AF	2903.13	
000023		09/25/95	100100	AF	5564.32	
000						

Invoice detail

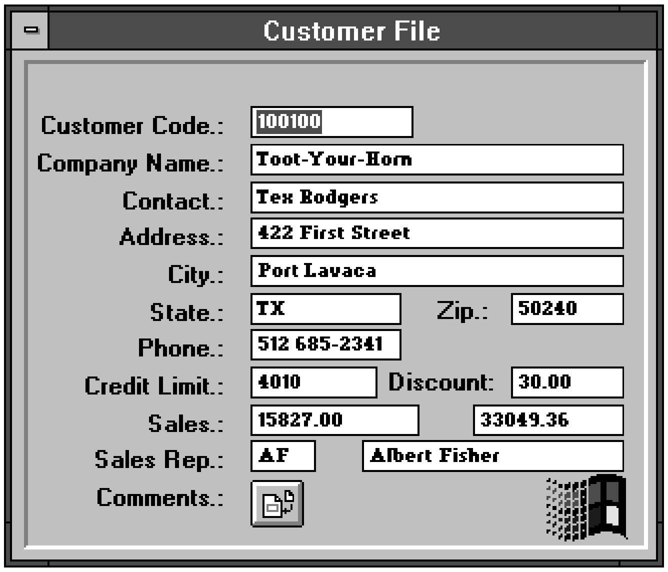
Inv Number	Ln Nm	Cust Code	Sr Cd	Item Code	Item	Price	Tax	Dscnt %	U P
000012	01	100100	AF	10-100-000	1.40	16.80	1.18	30.00	
000012	02	100100	AF	10-100-010	3.50	42.00	2.94	30.00	
000012	03	100100	AF	10-100-020	4.20	50.40	3.53	30.00	
000012	04	100100	AF	10-100-030	5.60	67.20	4.70	30.00	
000012	05	100100	AF	10-100-000	5.60	67.20	4.70	30.00	

The Views shown here give information about customers and invoices.

The Views allow the user to explore information throughout the system. As the user moves around, options for action are presented. If the user is operating a character terminal, options are associated with Function Keys, and the options available are described on the top line of the display. On a graphical workstation, options are displayed as a list, and may be selected using the mouse.

OPENworkshop allows the developer to specify a different list of options for each column. Thus the options can relate directly to the Data Object currently highlighted.

Screens



Customer Code.:	100100		
Company Name.:	Toot-Your-Horn		
Contact.:	Tex Rodgers		
Address.:	422 First Street		
City.:	Port Lavaca		
State.:	TX	Zip.:	50240
Phone.:	512 685-2341		
Credit Limit.:	4010	Discount:	30.00
Sales.:	15827.00		33049.36
Sales Rep.:	AF		Albert Fisher
Comments.:			

In contrast to a View, an OPENworkshop Screen presents information for one instance of a data set at a time, set out as a form. This layout is used to collect or display detailed information where numerous collections of data are shown simultaneously.

A Screen supports the same Pre- and Post-processing options as a View, and therefore allows the user identical flexibility to move to other Classes, modify or lookup data, and return.

Menus

OPENworkshop offers the developer a number of menu types. The simplest option is the classical “list” of options from which to choose.

A Selection Menu allows you to scroll through a list and make a selection. The parameters associated with the selection pass to a Method.

A Matrix Menu displays options in a matrix. The user moves the highlight to the required cell in the matrix, and then selects it. This form of presentation can significantly reduce the number of layers of Menu required in many applications.

Help

The Help system allows the developer to build Help into all levels of the user interface. At each step, Help can be displayed as a simple instruction, or a menu of levels. The Help system also supports a Subject Index, which the user can use to seek help by using keywords.

For Developers with Dictionary-IV Experience

If you are experienced with Thoroughbred Dictionary-IV, you will already be familiar with many of the classes that OPENworkshop provides. Be prepared, however, to revise the way you think about these classes and the way you use them to build a system.

OPENworkshop is an object-oriented development environment. It features new concepts that dramatically change the way developers build business software. It also allows developers to retain their past investments in software based on Thoroughbred programming environments.

OPENworkshop operates under Dictionary-IV. It uses links, formats, views, screens, and other Dictionary-IV features to define and control its operation. It supports Report-IV and Query-IV reports and queries. While OPENworkshop implements extensions to Dictionary-IV that support Object Technology concepts, it continues to work with those previously defined classes.

OPENworkshop is supported by the Script-IV and Thoroughbred Basic programming languages. Much existing code can be retained when an application is transformed into the OPENworkshop environment. The most surprising difference is the amount of code that can be thrown away because it is no longer needed.

OPENworkshop extends the capabilities of Dictionary-IV and adds some new directives to Script-IV. Most existing applications can be adapted to work in OPENworkshop, but some changes will be required.

OPENworkshop offers the benefits of reduced development and maintenance costs, even compared with Dictionary-IV based development. It also offers a much more flexible user interface. Some further benefits include:

- *Pass control anywhere at any time*

  Dictionary-IV provides a hotkey facility that provides access to screens, views, reports, queries, and menus. This facility is greatly enhanced in OPENworkshop by a CONNECT method-type capability. You can provide access to any method from anywhere in the system.

- *Re-entrance*

  The new structure based on objects and methods, (as opposed to programs and data), has significant implications in the area of re-entrance. It removes most barriers and also takes responsibility for management of the run-time context away from the developer. An OPENworkshop application can allow users to create a new invoice while they are processing an existing invoice. The developer has to take no special action.

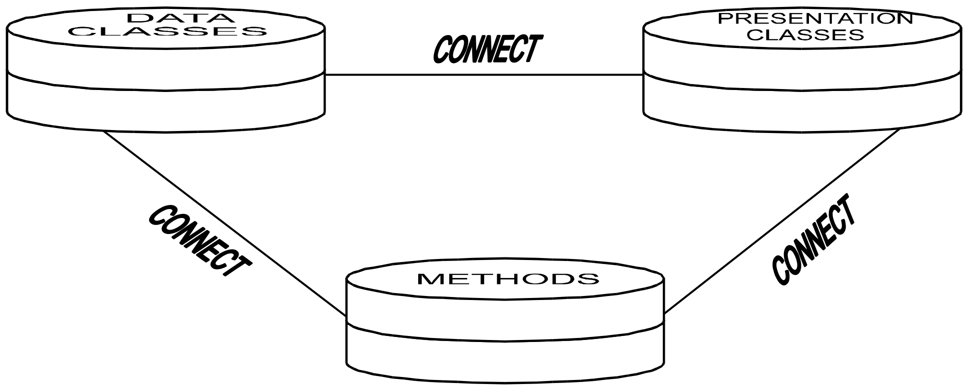
- *Interactive debugger*

  A powerful interactive debugger is provided, together with a global dictionary map and many where-used views that show where data element names are defined and referenced.

- *Structured help*

  The help system has been extended to allow help to be grouped and categorized, and accessed through a Help Topics subsystem.

OPENworkshop Classes and Methods



An OPENworkshop application is built using Data Classes, Presentation Classes, and Methods. CONNECT directives are used to invoke instances of a Class.

Data Classes

Data Classes hold the application data. From the point of view of the application developer the most important Classes of Object are:

Data Element Name	Defines an item of information, together with its attributes. A Data-name definition also specifies Methods to be used whenever the data item is created, displayed, or amended.
Format	Collects together a set of Data-names that will be stored together in a single table in the OPENworkshop table or file system.
Link	Specifies the physical files or tables that will be used to store a Format and associated key indexes. It also specifies default Presentation Classes (View and Screen) to use to display the data and allow it to be edited. Specifies the Trigger Method to be used when any data in the Link is updated to ensure that Referential Integrity is maintained throughout the system.
Library	Collects together all the Classes relevant to an application or subsystem.

Presentation Classes

Presentation Classes display or print information for the application user to read or edit. They also allow the user to select subsequent actions.

View	Displays data items, in spreadsheet form, as a table of rows and columns. Each column represents a different Data-name. Each row represents a record. Where there is more information than can fit in a window, vertical and horizontal scrolling is provided. Views display information from a single Link, and from multiple joined Links combining data items. Views can also display calculated values. Views allow the displayed data to be modified, and rows to be added to or deleted from the file or table being displayed. They also allow users to “drill down” or explore related information.
Screen	Displays data items as a “form”. Allows the data to be modified and records to be added to or deleted from the file or table being displayed.
Message	Displays a message and allows limited user input in response.
Report and Query	Provides output to a printer or terminal as formatted reports containing application data and, if required, calculated values.
Menu	Presents a set of options for the user to select. Menus can be presented as a simple list or as a matrix of options.
Help	Displays context-sensitive Help.

Methods

Methods contain the program code that performs the logic of the application. In an OPENworkshop application all program code is organized into Methods, each being associated with specific user actions or system functions.

View Method

Called whenever a View is preparing a row to display. Ensures that all data items required for display are available, and calculates any values required by the View.

Pre-Processing Method

Called whenever a View or Screen is preparing to allow a user to edit a data item.

Post-Processing Method

Called whenever a View or Screen has completed editing a data item. May be used to provide complex data validation or to modify related data items in the current row, for example.

Insert Method

Called whenever a View is required to add a new row. Can be used to set initialized values or to verify that the addition of a new row should be allowed.

Link Trigger Method

The Link Trigger Method is defined in a Dictionary-IV Link definition as the Link I/O Trigger. Called whenever a record in a file is to be updated. The Trigger Method is responsible for ensuring that the Data-names are being updated according to application rules and that any associated data will also be updated appropriately. Trigger Methods are the means by which OPENworkshop applications ensure that Referential Integrity is preserved throughout the application.

File Suffix Method

Builds up a file suffix. Is used to manage files in directories.

After Read Method

Called whenever a record has been read by a Screen. Prepares the record for display or editing.

Application Method

Implement the Application logic. Users can initiate a Method by selecting the appropriate option from a menu. Methods can also call other Methods. Such Methods may be designed to perform any purpose the application designer needs.

Startup Method

Called when the application begins. The OPENworkshop Start Method is defined at the operator code level.

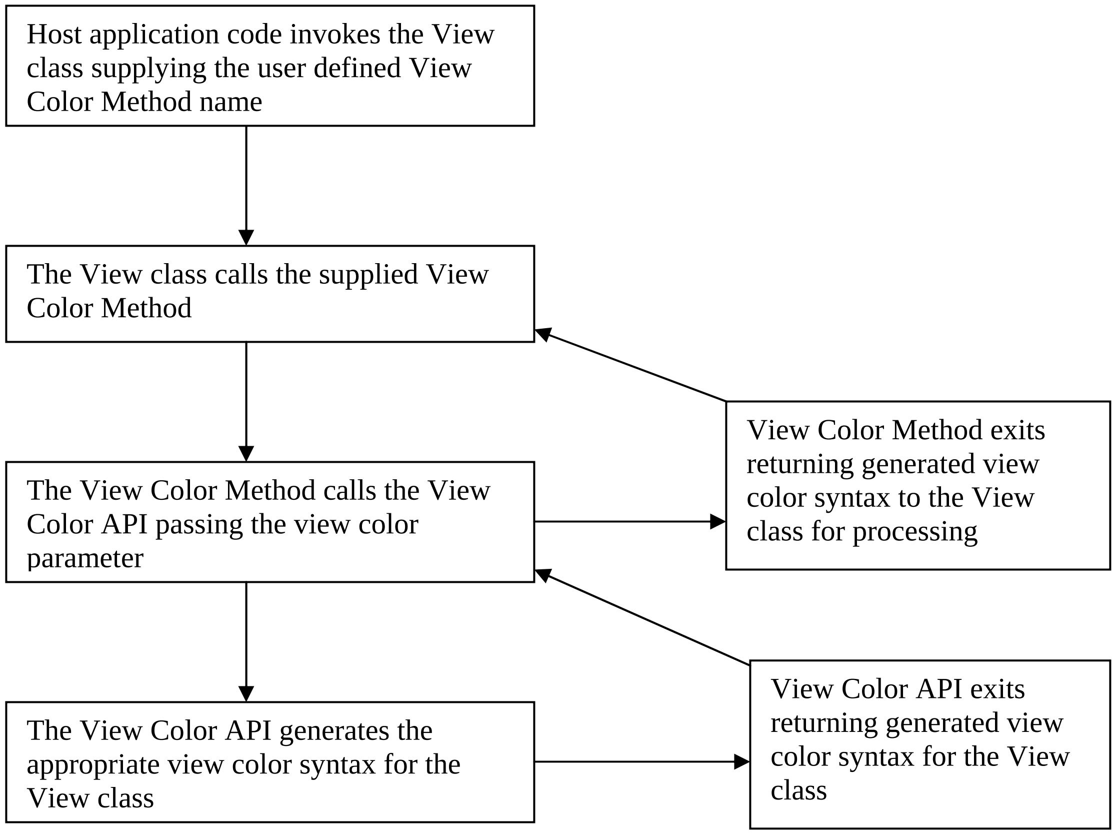
View Color Method

The SetColor Method can be used to dynamically control view colors at runtime. For example, it is possible to have all negative numbers in a column displayed using white text on a red background. Background and foreground colors can be set for a cell, a column, a range of columns, a row or a range of rows.

If you are running with VIP enabled please refer to the VIP for Dictionary-IV manual, Views, Dynamic View Colors.

The user defined View Color Method name is passed to the View class when performing a CONNECT VIEW. The View class will execute the View Color Method when moving off a column or moving off a row. The View Color Method then constructs the appropriate color parameters to be applied to the view by the View class. The View Color Method must pass the color parameters to the View Color API for parsing. The View Color API will interpret the supplied color parameters and generate compact syntax that is in turn processed by the View class.

The following diagram illustrates the process flow when using a View Color Method:


Host application code invokes the View class supplying the user defined View Color Method name
The View class calls the supplied View Color Method
The View Color Method calls the View Color API passing the view color parameter
The View Color API generates the appropriate view color syntax for the View class
View Color Method exits returning generated view color syntax to the View class for processing
View Color API exits returning generated view color syntax for the View class

View Class SetColor Method Name Syntax

`SETCOLOR METHOD=`*`method-name`*

SETCOLOR METHOD Required syntax.

method-name The name of the method to be executed. Pass the View class the View Color Method name in VIEW$[29]. For more information about VIEW$[29] see the OPENworkshop Manual.

View Color Method CALL/ENTER List

```
METHOD V$[ALL],VCOLR$[ALL],LNK$[ALL]
```

`V$[x,y] View Class Array Supplied by the View Class`

[0,0]	(1,1)	Length of column attribute entry.
	(2,1)	Number of view used column.
	(3,1)	Number of view deleted columns.
	(4,1)	Number of view heading rows.
	(5,n)	New column id and column number string.
	(6,1)	New column number.
	(7,1)	Next new column id.
	(8,1)	Next new column number.
	(9,n)	n column id & n column number.
[0,1]	(1,1)	Column window address (column).
	(2,1)	Column window address (row).
	(3,1)	Column width.
	(4,1)	Column dataname number. 00 implies new column.
	(5,1)	New column id.
	(6,n)	New column data (build by view method). V$[0,n](6) will be displayed by the view CLASS when the next row is printed. V$[0,n] Same as V$[0,1]

`VCOLR$[ALL] View Color Array`

[0] API returns "." when no errors else returns one of the following error codes:

"100"	Not enabled
"101"	Supplied parameters invalid
"102"	Supplied function requires supplied color parameters
"103"	Supplied row and column parameters invalid
"104"	Supplied row range invalid
"105"	Supplied column range invalid
"106"	Supplied cell range invalid
"107"	Supplied foreground color invalid
"108"	Supplied background color invalid

The content of [0] is used by the view class to determine how it should process colors. Upon returning from the view color method:

If VCOLR$[0]="." AND VCOLR$[3]<>"" the view class will assume a valid color command string was built by the API and will apply it.

If the above conditions fail the view class will build a command string for this row to apply the default view colors.

[1] Column, row information supplied by the View class

(1,3)	Current column number, 1 based.
(4,3)	Current row number, 1 based.
(7,3)	First data row number, 0 based.
(10,3)	Last data row number, 0 based.

[2] Color parameters built by the View Color Method, passed to the color API.

(1,1)	Mode: "S" Set color parameters supplied in VCOLR$[2] "R" Reset entire view back to default colors
(2,3)	Starting column number, 1 based. "999" to change all columns in the starting row (5,3). "998" for the current column in the view. When VCOLR$[2](1,1)="S" color parameters in bytes 5 through 61 must be supplied. Multiple color parameters are supported. When supplying multiple color parameters, each set must be padded to 30 bytes.
(5,3)	Starting row number, 0 based and include heading rows, must be >=VCOLR$[1](7,3) and <=VCOLR$[1](10,3). "999" to change all rows for the column specified in (2,3). "998" to change the current row.
(8,3)	Ending column to define a range. "999" to change all columns in the ending row. " " no ending column.
(11,3)	Ending row to define a range must, be >=VCOLR$[1](7,3) and <=VCOLR$[1](10,3). "999" to change all rows in the ending col. " " no ending row. **Note:** When both ending column and ending row are " " the starting column and row values are applied to the ending column and row values.
(14,9)	Foreground color, Basic color keyword padded 9 bytes
(23,9)	Background color, Basic color keyword padded 9 bytes
(32,30)	Next set of color parameters

Color values can be supplied using one of two notations:

- BASIC color keywords, keywords are not case sensitive.
- RGB hex notation is supported to provide greater color granularity.

The format is:

#rrggbb

where:

# is required syntax

rr is a red value

gg is a green value

bb is a blue value

For example: #ff00ff

When neither a foreground nor a background color is supplied the view default colors are applied.

VCOLR$[3] SetColor command string built by the API and returned to the View Color Method. This string must be returned to the View class.

LNK$[ALL] - See link array in the OPENworkshop manual.

Example Call View Class

```
METHOD MSGX$[ALL]
```

Call View Class with a View Color Method name.

```
....
PROCEDURE

      DIM VT$[29];
      VT$[1]="OEVCUST",
      VT$[29]="SETCOLOR METHOD=OEVCOLOR";
      CALL "OO3A",VT$[ALL];
      GOTO CUEXIT
```

Example View Color Method (OEVCOLOR):

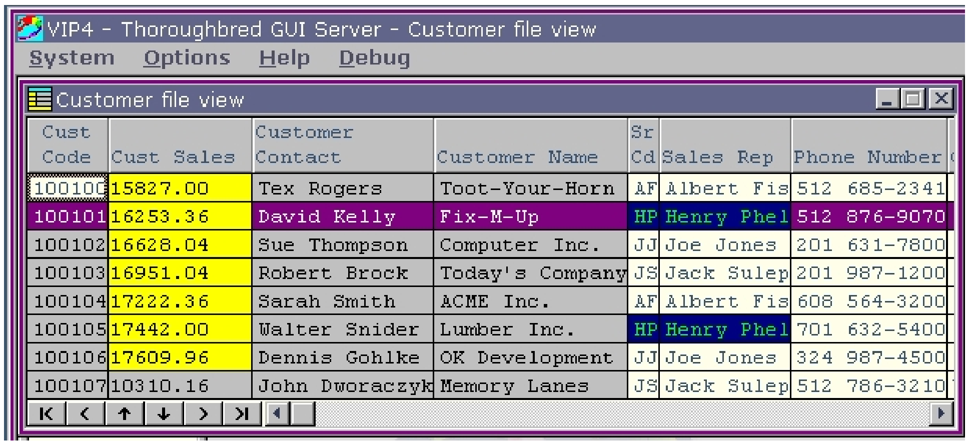
```
METHOD V$[ALL], VCOLR$[ALL],LNK$[ALL]
```

This method (OEVCOLOR) is called by the View Class.

```
....
PROCEDURE
FORMAT INCLUDE #OEFCUST,OPT="NONE";

! By default if VCOLR$[3] does not return to the view class a color command.
!    string, the current row will be initialized to default view colors. It
!    is not necessary to reset row colors back to defaults.


! When CUST-CODE is 100101 display entire row using white on magenta:

      IF #OEFCUST.CUST-CODE = "100101" ! If row is customer code 100101
         VCOLR$[2]=:+                  !    Set color parms:
             "999"+                    !         all cols in this row +
             VCOLR$[1](4,3)+           !         this row number +
             "   "+                    !         no ending col (range) +
             "   "+                    !         no ending row (range) +
             PAD("WHITE",9)+           !         foreground color +
             PAD("MAGENTA",9)          !         background color
      FI;                              ! enidf

! The next examples perform a test on a column basis, regardless of the
! current column. This code is an example of how to determine which column
! in a view contains what data element name. In most real-life cases, this
! is not necessary, the format entry number in the view array is sufficient.
! This is simply an example if this type of logic should be required.


! When CUST-SALES > 15,000 display that column in this row using
!   black on yellow

      IF #OEFCUST.CUST-SALES > 15000   ! If cust sales > 15000
         FX$="#OEFCUST";               !    Set format name
         FOR V=1 TO NEA("V$",2);       !    Loop through data elements
            EN=DEC(V$[0,V](4,1)),      !      Get format entry nbr
            DN$=ATR(FX$,EN,21);        !      Get its data name
            IF CVT(DN$,128)="CUST-SALES"!     If its the right one
               VCOLR$[2]=:+            !          Set color parms:
                   STR(V-1:"000")+     !               this col +
                   VCOLR$[1](4,3)+     !               this row +
                   "   "+              !               no ending col (range) +
                   "   "+              !               no ending row (range) +
                   PAD("BLACK",9)+     !               foreground color +
                   PAD("YELLOW",9),    !               background color
               V=NEA("V$",2)+1         !         Force end of loop
            FI;                        !      endif
         NEXT V                        !    Next data element in format
      FI;                              ! endif

! When SR-CODE=HP print both the SR-CODE and sales rep name (<Ax>) using
! light green on blue. This uses a column range to set the color of two
! consecutive columns.
```

```
      IF #OEFCUST.SR-CODE="HP"          ! If sales rep is H
         FX$="#OEFCUST";                !    Set format name
         FOR V=1 TO NEA("V$",2);        !    Loop through data elements
            EN=DEC(V$[0,V](4,1)),       !      Get format entry nbr
            DN$=ATR(FX$,EN,21);         !      Get its data name
            IF CVT(DN$,128)="SR-CODE"   !      If its the right one
               V0$=V$[0,0],             !         Get col attr string
               EL=ASC(V0$(1,1)),        !         Get len of col attr entry
               NC$=V0$(EL),             !         Get 'new col A<x>' string
               NC=POS("A"=NC$,2);       !         Scan for new col "A"
               IF NC                    !         If found rep name
                  CN=ASC(NC$(NC+1,1))   !            Get rep name col nbr
               FI;                      !         endif
               VCOLR$[2]=:+             !         Set color parms:
                   STR(V-1:"000")+      !              this col +
                   VCOLR$[1](4,3)+      !              this row +
                   STR(CN-1:"000")+     !              ending col (name) +
                   VCOLR$[1](4,3)+      !              this row +
                   PAD("LGREEN",9)+     !              foreground color +
                   PAD("BLUE",9),       !              background color
               V=NEA("V$",2)+1          !         Force end of loop
            FI;                         !      endif
         NEXT V                         !   Next column
      FI;                               ! endif
```

```
* Call API to construct the set color command string required by view class

      IF LEN(VCOLR$[2])>1               ! If have something
         VCOLR$[2]="S"+VCOLR$[2];       !    Set API function: set colors
         CALL "OO3AV01",VCOLR$[ALL],    !    Call API, color array,
                          V$[ALL]       !              view array
      FI;                               ! endif

* Will exit back to view processing with VCOLR$[3] built for the view class.

      GOTO CUEXIT
```



Cust Code	Cust Sales	Customer Contact	Customer Name	Sr Cd	Sales Rep	Phone Number
100100	15827.00	Tex Rogers	Toot-Your-Horn	AF	Albert Fis	512 685-2341
100101	16253.36	David Kelly	Fix-M-Up	HP	Henry Phel	512 876-9070
100102	16628.04	Sue Thompson	Computer Inc.	JJ	Joe Jones	201 631-7800
100103	16951.04	Robert Brock	Today's Company	JS	Jack Sulep	201 987-1200
100104	17222.36	Sarah Smith	ACME Inc.	AF	Albert Fis	608 564-3200
100105	17442.00	Walter Snider	Lumber Inc.	HP	Henry Phel	701 632-5400
100106	17609.96	Dennis Gohlke	OK Development	JJ	Joe Jones	324 987-4500
100107	10310.16	John Dworaczyk	Memory Lanes	JS	Jack Sulep	512 786-3210

Screens - Introduction

Unique to screens is the ability to use default or custom graphical screens. For both graphical screen types, any change made to the corresponding character screen definition will be reflected in the graphical screen. All VIP clients will present a graphical screen based on a common character definition on the host. For better performance, the graphical screen is cached on the client. Each time a screen is invoked the host will compare the last change date and time and language code to the graphical screen cached on the client. If these do not match and a custom screen exists on the host, the custom screen will be downloaded and cached on the client. If these do not match and a custom screen does not exist on the host, a new default screen will be generated and cached on the client.

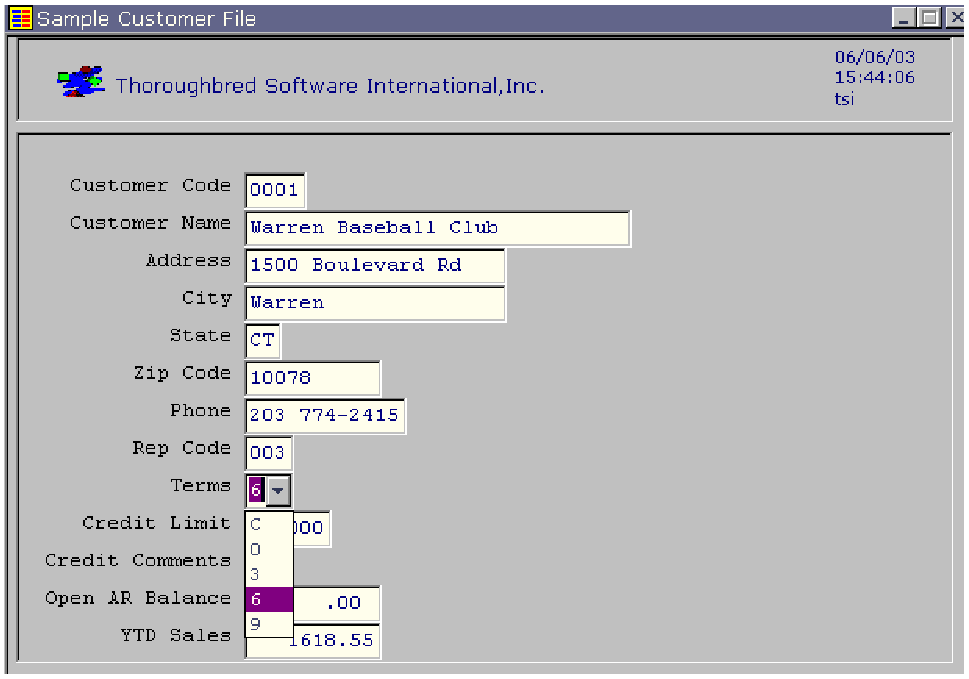
Control Types

Both default and custom screens will create control types based on the data element attributes as defined by the format.

- Standard input fields will be collected using a text box control.
- Calculated (formula) fields will be displayed using a text box control using special foreground and background colors. See Special Field Colors later in this section.
- Security fields will be displayed using a text box control using special foreground and background colors. See Special Field Colors later in this section.
- Text fields will be displayed and edited using a graphical text editing control.
- Valid Value lists will be displayed and selections collected using a ComboBox control. Clicking on the ComboBox button will display a drop down list containing a list of valid values.
- Context Menus are automatically generate for all fields. A right mouse click on the active field will display a context menu. This menu displays options based on the attributes of the data element as defined by the format definition. For more information on data element Context Menus see the Views section of Creating Graphical Presentations.

In addition to the above, custom screens support button controls and image controls. Please see the Screens-Custom in a following section.

The following image is an example of a default graphical screen with a ComboBox control used for processing the Valid Value list defined for the Terms field.


Sample Customer File

Thoroughbred Software International,Inc.

06/06/03
15:44:06
tsi

Customer Code	0001
Customer Name	Warren Baseball Club
Address	1500 Boulevard Rd
City	Warren
State	CT
Zip Code	10078
Phone	203 774-2415
Rep Code	003
Terms	6
Credit Limit	)00
Credit Comments	
Open AR Balance	.00
YTD Sales	1618.55

Special Field Colors

Colors for calculated (formula) fields, security fields, and highlighted (current) field can be set in the WorkStation Manager. These color properties can be applied at run time to both default screens and custom screens. For more information see the WorkStation Manager manual (Special Fields Color tab).

NOTE: This does not apply to Dictionary-IV graphical menus.

Communicating Between Classes

The CONNECT Directive

The CONNECT directive is the most important directive in OPENworkshop. It allows “connections” to be made from one Object or Class to another. Views, Screens, Menus, Help, Reports and Queries can be connected directly between themselves.

The power and flexibility of OPENworkshop is greatly enhanced by allowing intervening Methods to modify the behavior of the Classes and the behavior of the connections between them.

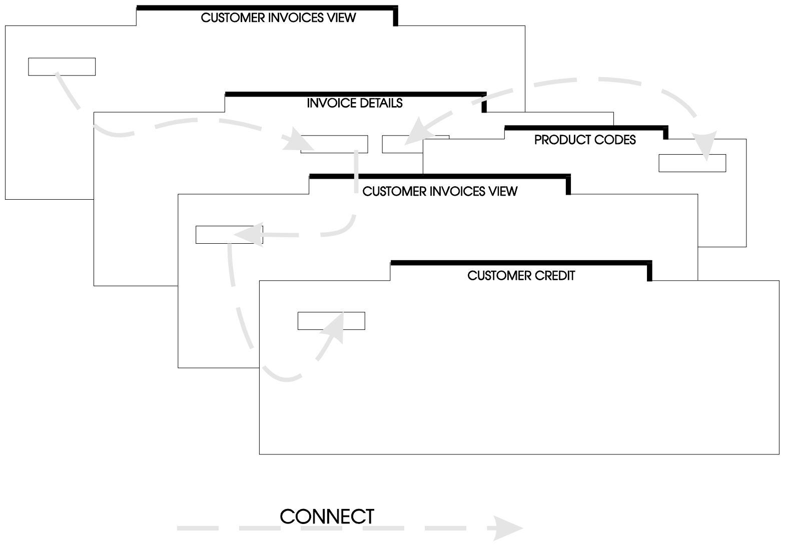
The CONNECT directive is the framework that links OPENworkshop Classes and Methods together. It not only invokes the required Object, it is also the vehicle for passing messages to the invoked Object.

Developers can CONNECT from a View, Screen, Menu, or Method to any of the OPENworkshop Classes or Methods. You can use CONNECT within a Class. For example, a View can use CONNECT to initiate another View, or even to create a second instance of the same View.

Messages carried by the CONNECT directive are used to pass information to the invoked Class or Method. They can be used to communicate information about the Object that the target should work on, or modify the behavior of the target.

CONNECT directives are simple in concept, uncomplicated to define, but extremely powerful in execution.

CONNECT from a View or Screen

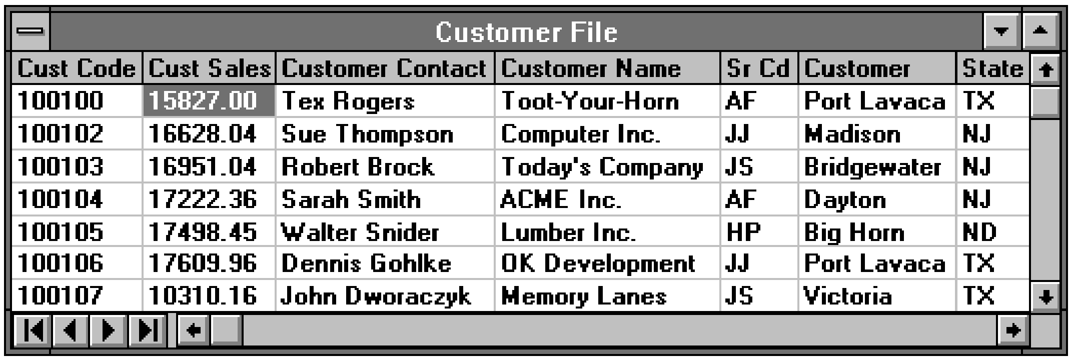


For a View or a Screen, CONNECT directives are placed in the Data-name Pre- and Post-Processing definitions in the associated Dictionary-IV Format. A Pre-Process is executed whenever the Cursor is positioned on the Data-name in a View or a Screen. A Post-Process is executed when a user has completed editing a Data-name.

Note that these CONNECT definitions are associated with a Data-name. In other words, having been specified once, they are in effect wherever the Data-name is used, possibly in many Views and Screens throughout the application. One definition is all you need to create and maintain. What is more, you can guarantee the behavior of the system is consistent for the user, wherever that Data-name is displayed.

CONNECT to a View

The CONNECT directive allows you to define which rows should be placed in the View, and in which order to sort them.


Customer File

Cust Code	Cust Sales	Customer Contact	Customer Name	Sr Cd	Customer	State
100100	15827.00	Tex Rogers	Toot-Your-Horn	AF	Port Lavaca	TX
100102	16628.04	Sue Thompson	Computer Inc.	JJ	Madison	NJ
100103	16951.04	Robert Brock	Today's Company	JS	Bridgewater	NJ
100104	17222.36	Sarah Smith	ACME Inc.	AF	Dayton	NJ
100105	17498.45	Walter Snider	Lumber Inc.	HP	Big Horn	ND
100106	17609.96	Dennis Gohlke	OK Development	JJ	Port Lavaca	TX
100107	10310.16	John Dworaczyk	Memory Lanes	JS	Victoria	TX

Rows in a View can be Selected and Sorted.

Note: From the **F5** Sort Window you can press **F2** on any highlighted sort and the order of that sort will be reversed. Reversing the order of the selected sort will retain the range values properly. VIEW$[30] set to an "R" will cause the view to be presented in reverse order from the selected sort.

The general structure of the CONNECT VIEW directive simple to use and understand, as is shown in the following syntax:

```
CONNECT VIEW "View-name"
[USING Data-name or Expression]
| [USING RANGE FROM Data-name or Expression;
      TO Data-name or Expression;]
   | [SELECT WHEN Expression;]
[SORT BY n]
```

Development Environment

In addition to the focus on usability for the OPENworkshop end-user, Thoroughbred has concentrated on providing a highly flexible and accessible environment for developers. To a great extent this has been achieved by the simple tactic of implementing the developer's interface to the system using OPENworkshop Classes and Methods.

The developer's environment positively encourages an incremental development methodology. All OPENworkshop Classes are easily available to the developer at all times, and can be modified as the application is running. Most changes that the developer can make are effective immediately, so the developer can see the results immediately.

For example, when a Menu is displayed the developer can select “Edit” to edit the definition of that Menu. When the editing is finished, OPENworkshop will re-displays the Menu in its revised form, so the developer can see and check the results immediately. The same facility is provided for View, Screen and Help. Similar capabilities exist for Link, Format, and Data-name.

OPENworkshop also provides the developer with a comprehensive testing and debugging toolkit. Data and control status can be inspected at any time through an online debug system.

A complete cross-referencing system shows the structure and relationships within the system for the developer. These relationships are displayed using OPENworkshop Views, thus providing all the search and selection facilities the developer needs. In addition, the developer can point to any item in the View, and select it to display or edit the OPENworkshop Object directly, further increasing the high degree of accessibility of OPENworkshop Class definitions.

Any environment that is so open to developers' needs to be protected from accidental or over-inquisitive access by end users. OPENworkshop has security protection that disables or enables this access, by passwords and user logins.

Getting Started in OPENworkshop

Thoroughbred makes getting started with OPENworkshop fast and efficient. Most important of all, it will provide the developer with the tools to succeed. As with any new technology there is some learning to do.

OPENworkshop offers the developer strong benefits in productivity, as well as a more flexible user interface. But to gain from these benefits it is essential that developers invest in training. In fact, Thoroughbred is so convinced of this need that it is a *requirement* that any new customer have at least one member of staff attend a training course. Having taken this step, the developer will be well positioned to move forward.

Developers who already have experience with Thoroughbred's Dictionary-IV, and have applications that are based on it, can migrate their applications to OPENworkshop easily. Many Dictionary-IV items can be used directly, and Thoroughbred provides conversion utilities for others.

While experienced Dictionary-IV developers will be at an advantage, OPENworkshop features some fundamental new concepts that must be understood. Therefore Thoroughbred requires that they too take part in training before beginning to develop in OPENworkshop.

OPENworkshop runs on most popular operating systems. Once installed on the selected hardware and operating system, an example application is available for developers to use as a starting point for their subsequent development activities.

Developers will find that it is easy and quick to create a prototype skeleton of their intended application. In fact this is the best way to become familiar with the environment. After this reflect on the initial design decisions made for the prototype. Then, re-specify the Classes before you start to develop the first real application. The OPENworkshop facilities positively encourage this incremental development style.

OPENworkshop gives you tangible, demonstrable results very quickly. It is an ideal environment to use if your business needs to customize applications for customers, or to develop new applications for customer requirements. You can show your customer the prototype as it is developed, and amend it step by step with them.

Bibliography

Object Analysis and Design Comparison of Methods

Author: Object Management Group
Publisher: John Wiley and Sons

Principles of Object Oriented Analysis and Design

Author: James Martin
Publisher: Prentice-Hall

Object Oriented Methods

Author: Ian Graham
Publisher: Addison-Wesley

Object Lifecycles: Modeling the World in States

Author: Sally Shlaer & Stephen J. Mellor
Publisher: Prentice-Hall

OPENworkshop Summary

Object Oriented

- Data controls the application
- Applications can be developed incrementally
- Relational Integrity Maintained
- Can Modify Object Rules
- Global Definition Based

Object Classes

Data Classes

Data Name

- Defines Item attributes
- Specifies Methods used when data is created, displayed or amended

Format

- Defines the Data item, and how it is stored

Link

- Specifies Files and Indexes
- Specifies Default Presentation Class
- Specifies Trigger Method for updates
- Insures data and Referential Integrity

Library

- Catalogues the relevant Classes to an application or subsystem

Presentation Classes

View - It is more than a browse

- Browse - Displays Data as Rows and Columns
- Multiple Link joins
- Calculated values
- Drill-downs
- Modify Data

Screen

- Displays Data as a Form

Message

- Displays a message with user response possible

Menu

- Catalogs options for user selection including list and matrix

Help

- Context sensitive help text

Report

- Custom and systems report library
- Programmable report designer supporting
  Report design and formatting
  Calculated and prompt entered data

Query

- SQL Query/Report generator
- Point and click user interface
- GWW connection to spreadsheets

Methods

View Method

- Called when a View prepares a row to display
- Ensures that data items are available, and calculates values required by the View

Pre-Processing Method

- Called when a View or Screen allows data entry or modification

Post-Processing Method

- Called when a View or Screen completes editing a Data item
- Data validation

Insert Method

- Called when a View or Screen adds a new row
- Default values and verification of new data

Link Trigger Method

- Called when a record is being updated
- Ensures that Data-names are being updated according to the application rules
- Ensures Referential Integrity

After Read Method

- Called when a record has been read by a screen, and prepares for editing

Application Method

- Implements application logic and rules

CONNECT

Links the Classes and Methods

- Invokes the required Object
- Passes messages to the Object
- Pre or Post Process
- Initiate additional views, Screens, Menus, Queries, Reports, Messaging, Methods

Re-Entrant

- When invoked the current environment is preserved, and restored upon completion

Object Concepts

Inheritance

- Obtain Characteristics from a Super class

Polymorphism

- Same expression or message can operate on Objects of different Classes

Recursive

- Objects can send messages to their Methods recursively, or to themselves

Persistence

- Created Object transcends time and its creator

Three Tier Compliance

Presentation

- Graphical (GUI) - with VIP
- Character - with same application
- HTML - with VIP

Rules

- OPENworkshop
- Database
- Thoroughbred
- Oracle
- ODBC

Environments Supported

- UNIX/Linux
- VMS
- Windows

CONVERTING FROM DICTIONARY-IV

OPENworkshop classes are based on Dictionary-IV structures. OPENworkshop methods can be written in either Script-IV or Thoroughbred Basic. Developers with applications that use Dictionary-IV are in a great position to move them to OPENworkshop.

Some changes will be required. This section outlines a recommended approach to moving an application to OPENworkshop, and details the necessary changes. The focus of this document is on code developed based on the Solution-IV source code models and development standards.

First, the benefits and advantages of the OPENworkshop development environment over Dictionary-IV are reviewed. Next, the process of moving your application code to OPENworkshop is detailed.

Differences between Dictionary-IV and OPENworkshop

A paradigm shift occurs when a developer moves from a 3GL/4GL based development environment to an object-oriented environment. This shift occurs in moving the developer's focus from the programs to the actual data itself. Rather than designing a program, which performs a series of procedures and functions on data files, the focus moves to the data, files themselves and interactions between these files. Many programs in OPENworkshop are tied directly to the individual data element itself at the format level.

Moving to OPENworkshop

The following subsections describe changes that must be made to existing specifications and code when moving to OPENworkshop. For more information on how to change existing features, please refer to the following sections.

Formats	No changes are required to any format definition you have already developed. You will probably wish to take advantage of the additional facilities available, particularly to add pre-processing and post-processing functions.
Links	No changes are required to any link definition you have already developed. You will probably wish to take advantage of the additional facilities available, particularly to add pre-processing and post-processing functions.
Menus	OPENworkshop allows you to use pop-up menus. You will probably wish to convert many of your menus to OPENworkshop pop-up menus. **RUN"OO"** and select the **Utilities** option. For more information see the OPENworkshop Main menu.
Views	Existing Dictionary-IV views are automatically converted to OPENworkshop Views. You will probably decide to modify your existing views to take advantage of the greater flexibility of OPENworkshop views.

Screens	No changes are required to any screen definition you have already developed. Screen definitions that are not already windowed must be converted to windowed screens. For more information see the Dictionary-IV Administrator Guide (Dictionary-IV Supplemental Utilities).
Reports	No change is required for existing reports.
Queries	No change is required for existing queries.
Help	No changes are required to any help definition you have already developed. Help definitions that are not already windowed must be converted to windowed help. For more information see the Dictionary-IV Administrator Guide (Dictionary-IV Supplemental Utilities).
Scripts	These will need to be changed to reflect OPENworkshop requirements, as described in the following sections.
Thoroughbred Basic programs	These will need to be changed to reflect OPENworkshop requirements, as described in the following sections.

The process of moving to OPENworkshop basically involves cleaning up certain types of undisciplined code. This requires removing the following types of code:

- Code that performs manual window manipulation.
- PRINT @ commands
- INPUT commands that do not use the dictionary
- WINDOW commands
- RUN statements

The process of changing this type of code is explained in detail below and has been divided into two phases, Phase I and Phase II. For more information, please refer to the following two sections.

Phase I Changes

The following changes are performed in Phase I and are required when moving to OPENworkshop:

Creating OPENworkshop Menus

OPENworkshop gives you the capability of using pop-up menus rather than the full screen menus as in Dictionary-IV. The OO program can be run using the command:

```
RUN"OO"
```

To convert your Dictionary-IV menus to OPENworkshop list box menus select **Utilities** then select **Build List Box Menus**. Enter the 2-character Library name or names (separated by commas) or enter the 2-character Library name and menu name or names (separated by commas).

```
[SH:][L1,L2,Ln]
```

or

```
[SH:][LLM1,LLM2,LLMn]
```

Where:

SH is the sub-heading
Ln is a specific library name
LLMn is a specific library and menu name.

These menus are converted while honoring any existing VIP menu headings thus creating a different List Box menu for each menu heading

The system creates help modules matching the screen menu name and overwrites help modules with like names. It creates multiple help names for screen menus containing sub-headings. A menu help module will be created containing the sub-headings and another one containing the selections that are subordinate to each sub-heading. The help module names will be:

```
LL    LLMM, LLMM0, LLMM1, LLMMn
LLM1  LLM1, LLM10, LLM11, LLMMn
```

Windowed Screens

Two types of screens exist in Dictionary-IV – standard (non-windowed) screens and windowed screens. When running under OPENworkshop it is necessary to convert any standard screens to windowed screens first. One reason is that when running under the OPENworkshop environment, you are never actually in the main window (WINDOW 0). Also, each method cleans up all windows created by the method and all files opened by the method. It does this by saving the window list and data file channel list when the method is initiated and deleting all windows and closing all files not on these lists when the method terminates.

As an added benefit, windowed character-based screens may be defined which use custom colors as well as a variety of graphic characters.

All screens in your application can be converted to windowed screens using the Window Conversion Utilities found on the Dictionary-IV UTMENU11 menu Dictionary-IV Supplemental Utilities menu), then selecting Convert Screens Into Window Format. For more information see the Dictionary-IV Administrator Guide.

Windowed Help

Because of the issues mentioned above regarding windowed screens, all help should be converted to windowed help as well. Use the Dictionary-IV UTMENU11 menu (Dictionary-IV Supplemental Utilities menu), and then select Convert Help Text Into Window Format. For more information see the Dictionary-IV Administrator Guide.

Programs

Programs in OPENworkshop are called methods. It is important to understand that all methods in OPENworkshop are actually public programs. The following paragraphs outline the differences between Dictionary-IV scripts and OPENworkshop methods.

Script methods (type S) are similar to primary scripts (type 1) in Dictionary-IV. Most menu options initiate type S methods. These are the most common methods used in OPENworkshop development.

Thoroughbred Basic methods (type M) are similar to Thoroughbred Basic programs in Dictionary-IV. See the next section for converting Thoroughbred Basic programs to type M methods.

Continuation scripts (type 2) differ in OPENworkshop only in that they are now called as public programs. When the OPENworkshop compiler encounters a RUN program-name command that specifies a type 2 script, it will automatically generate a CALL to the script followed by an exit to the OPENworkshop menu system. This works fine as long as your continuation script always returns to the menu.

However, if your continuation script re-runs your primary script, you can quickly run out of memory because of recursive calls. To fix this problem, see the section on Changing RUNs below.

Overlay scripts (type 3), continuation scripts (type 2), and public scripts (type 6) work in the same way as under Dictionary-IV.

NOTE: There is one additional issue with multiple program levels pertaining to the graphical user interface (GUI). When a PRINT SCREEN is processed, the OPENworkshop environment keeps track of the public program level at which the PRINT SCREEN occurred. If a PRINT SCREEN occurs at a lower level (more deeply nested level), the environment assumes the program has been called recursively and it creates a new instance of this screen. A new instance is created in both character mode and under the GUI. This can result in extra objects being created.

To avoid this problem, make sure that your OPEN SCREEN and PRINT SCREEN are in the same program as your PRINT SCREEN DATA and INPUT SCREEN DATA. Do not put them in an overlay (public) or a primary script that chains to a continuation script.

BEGINs, Thoroughbred Basic Public Programs, Hard-Coded Channels

Although not a problem for script developers, any Thoroughbred Basic program that does a BEGIN must be changed. These Thoroughbred Basic programs should be converted to type M methods. The process for doing this is as follows:

- First, change the script type from B (Thoroughbred Basic) to an M type method.
- Second, change the beginning of the method to read as follows:

  ```
  METHOD A$,B$,C$ (or whatever variables are on ENTER list)
  (documentation can go here without the need for an
   exclamation (!) point.)
  .... (signifies the start of the actual program code)
  ```

- Third, remove any general SET ESC and SET ERR logic at the beginning of the program. All methods generate their own escape and error logic automatically. Finally, to exit the method, change your exit to the following:

  ```
  GO TO CUEXIT
  ```

  CUEXIT is a standard routine, which will close all channels opened by the program, delete any windows, which were created, and set the precision back.

- Fourth, remove any hard-coded channels that are used. Instead of opening the file on a hard-coded channel, use the UNT function as follows:

  ```
  C3=UNT;
  OPEN (C3) "<file-name>"
  ```

- Last, recompile all methods using the OPENworkshop Source-IV compiler.

  Note: Any formats you use can be included with an FN declaration immediately following the word METHOD. This will include the format with the OPT="NONE" option.

Changing RUNs

Because there is no main program in OPENworkshop, every method may be thought of as a self-contained public program, it is illegal to perform a RUN program-name command from anywhere in your application. At first, this sounds like an arbitrary restriction, but looking closer, we see that these RUN commands can be easily replaced. This is shown in several different examples.

First, consider the example of a period end update process. Here, you have a primary script (type 1) that chains to subsequent continuation (type 2) scripts by using a RUN directive. In most cases, these will not have to be changed at all, other than changing the type 1 script to a type S method.

The OPENworkshop script compiler actually translates these RUNs into a CALL to an ENTER-less public program automatically. An ENTER-less public is a public program with no ENTER statement. When you call a public in this way, all 4GL and 3GL variables are passed by reference into the public program. In addition, any channels you have open are kept open. Essentially, the entire environment is made available to the called program. When the called program terminates, the calling program automatically exits. This all works perfectly as long as you don't chain through more programs than you have available memory to load all of these programs simultaneously.

Next, consider the same example, only the script we are chaining to has a different environment area and is a type 1 or type U script. There are two different approaches, which can be taken with these situations.

- First, the scripts you are chaining through can each be changed into a type 6 public script with an ENTER statement containing no variables. Then, the calling program simply calls each of these publics in succession.

- Alternately, you can change the scripts you are chaining through to type S methods. Each RUN would then be replaced with the following code:

  ```
  CLOSE ALL (this closes all open channels)
  CONNECT METHOD "chain to prog name"
  ```

Next, consider a typical data entry program. Here a primary (type 1) script RUNs a series of continuation (type 2) scripts depending on what the user selects. Typically, the header portion is handled by one script, the lines by one script and the summary screen by another script. Remember that the script compiler translates each of these RUNs into a CALL. Obviously, these must be changed or you will run out of memory after entering only a few transactions.

Again, the change here is relatively simple. Each of the RUNs in the continuation scripts is changed to set a flag and do a TERMINATE instead of a run as follows:

```
LET #TAFINST1.INST-LINE-CONTROL = "1"
(1 for header, 2 for lines, 3 for summary)
TERMINATE
```

Then, in the lead script a routine is used to control which program should be called as follows:

```
LET #TAFINST1.INST-LINE-CONTROL = "1"
LET STATUS = ""
DO LOOP UNTIL #TAFINST1.INST-LINE-CONTROL = " "
LET P$ = "ARPIDAT" + #TAFINST1.INST-LINE-CONTROL
CALL P$
ENDLOOP
```

The reason the above code contains a CALL P$ instead of the original RUN P$ is that the script compiler will automatically TERMINATE after a RUN. Thus, our loop variable would never be checked, and it would automatically exit as soon as one transaction was entered.

One function that comes in handy is TCB(13). This function will return the public program level at which you are currently running. For example:

```
IF TCB(13) THEN
   TERMINATE
ELSE
   RUN "ARPCUST0"
ENDIF
```

Changing RUN "ID" to TERMINATE

Our next example of a RUN is where we are returning to the menu. Doing a TERMINATE in OPENworkshop will return you to the calling program if you are in a script or method that was called by another script. Once you are back to the original method initiated from the OPENworkshop menu, a TERMINATE will return you to the menu just as it does in Dictionary-IV. All RUN "ID" statements should be changed to TERMINATE.

Changing RUN "IRPCA0" to CONNECT REPORT

This code was used in Dictionary-IV to chain from a script to a Report-IV report. Again, the RUN is not allowed. All RUN "IRPCA0" statements should be changed to use the CONNECT REPORT directive as follows:

```
CONNECT REPORT "report name"
```

In addition, often a program was RUN in the T lines at the conclusion of a report. These termination lines should be removed and the script that drives the report should simply CONNECT REPORT to each report in the sequence of reports to print. At the end of the CONNECT REPORT sequence, the update program can be initiated using a CONNECT METHOD.

Saving Format Changes in Report-IV

In order to make Report-IV extensible, Report-IV will attempt to put back all formats to their initial value when the report was started. This may cause problems if you actually want to change a global format data element. If you want to change a format, after setting the format value, you must assign the bracket variable to the format. For example, to change a variable in #TAFINST1, do the following:

```
LET #TAFINST1.INST-PRINTER-ID = "LP"
LET ]TAFINST1$ = #TAFINST1
```

This sample code will save the data in format #TAFINST1 that you have changed.

Changing RUN "IDVIEW" to CONNECT VIEW

IDVIEW is used in Dictionary-IV to chain from a script to a multi-record maintenance view. Since RUNs are not allowed in OPENworkshop, these statements should be changed to use the CONNECT VIEW directive as follows:

```
CONNECT VIEW "view name"
```

Removing WINDOW DELETEs

A WINDOW DELETE directive was often required in Dictionary-IV to clear a windowed screen. Under OPENworkshop, because of the way the environment cleans up when exiting a method, these WINDOW DELETEs are simply not needed. Further, because of the recursive nature of OPENworkshop, you could actually run Invoice Entry from Invoice Entry. In the second instance of Invoice Entry, the window names are not the same as the screen names and a WINDOW DELETE directive will cause serious problems.

WINDOW DELETE ALL and PRINT 'WC'

As mentioned before, since you are never in the main window (window 0) while running an OPENworkshop application, both of these directives should be avoided.

WINDOW SELECT and WINDOW SELECT (NOUPDATE)

As mentioned previously, an OPENworkshop method cleans up all windows created when the method terminates. Any WINDOW SELECT (NOUPDATE) directives you are using should be removed. Otherwise, when the method exits, the environment will not be aware of anything displayed in a window with the NOUPDATE option, and thus will not be able to clean it up. You will experience display problems when you return to the menu.

Also, WINDOW SELECT ("window name"), even without the NOUPDATE option, should be avoided for the same reason as WINDOW DELETE above. That is, the window name will no longer be the same as the screen name beyond the first instance of the program.

PRINT @ (C,L)

Although the PRINT @ command is still supported by OPENworkshop, this code is not supported when using GUI screens. This is because there is no one-to-one correspondence between a Thoroughbred Basic @ position and the pixel locations on a GUI screen. The data you are printing should be put into a screen formula and the PRINT @ should be replaced with:

```
PRINT SCREEN screen-name FORMULAS
```

INPUT @ (C,L)

As with PRINT @, the INPUT @ should be replaced with

```
INPUT SCREEN screen-name DATA-NAME LIST
```

PRINT 'CE'

The 'CE' mnemonic that clears from the cursor position to the end of the screen can cause problems in two areas.

- A GUI screen has no way of dealing with this mnemonic or any other mnemonic, which prints data or manipulates the screen presentation.

- The screen color attributes will be lost for the area of the screen, which is cleared if running in character mode.

Instead, you should use the PRINT SCREEN DATA and PRINT SCREEN DATA CLEAR directives to display and clear screen information.

Phase II Changes

Phase II changes are aimed at helping you reduce the amount of procedural code required to perform various functions in your application.

CONNECT VIEW versus PRINT VIEW

Changing your PRINT VIEWs to CONNECT VIEWs provides a significant increase in flexibility to your application. A CONNECT VIEW can be made to look like a PRINT VIEW with the added benefit that the user can change the view on the fly without changing the underlying code or recompiling any programs. In addition to changing the columns defined in the view, the location and size of the view may also be changed.

CONNECT SCREEN versus INPUT SCREEN

As mentioned previously, OPENworkshop provides several major classes that can reduce the amount of procedural code in your application substantially. By restructuring your data entry programs so they are built around CONNECT SCREEN instead of INPUT SCREEN, you can eliminate a lot of duplicate code. The reason is that this code is tied to the format data element name to which it applies instead of existing in many different programs that operate on this particular data file.

CONNECT VIEW versus INPUT SCREEN LINE OFFSET

In the same way as CONNECT SCREEN, CONNECT VIEW can eliminate a great deal of complex code that was built for header/line entry using INPUT SCREEN LINE OFFSET. All of the flow control code for line item entry is built into the CONNECT VIEW class and can be eliminated from your application. In addition, the I/O triggers contain the code to update the individual files and the INSERT METHOD command handles the line insert logic. This greatly simplifies the line entry process.

Inquiry Programs

Inquiry programs can be re-implemented eliminating virtually all of the program code while at the same time adding functionality. Again, this is due to the power of CONNECT. The inquiry can begin with a masterfile view such as a customer view, which allows selection of a particular customer. From here a pop-up menu can be displayed which shows various inquiry options. From each inquiry option, a CONNECT VIEW can be displayed with further access options for each of these records.

Using the Graphical User Interface

Much of what you need to do to use the graphical user interface is covered in the previous sections on converting to OPENworkshop. The following list summarizes the rules that must be followed when verifying that your application will run with the GUI:

- Verify that you are not doing any direct Thoroughbred Basic PRINT @'s or INPUT @'s in your code.
- Avoid WINDOW DELETE, WINDOW CREATE, and WINDOW SELECT directives.
- Convert all screens and help text to windowed format if using standard format.
- Make sure OPEN SCREENs and all PRINT SCREENs are at the same public program level.

ACCESS TO DEVELOPER'S FACILITIES

Menus

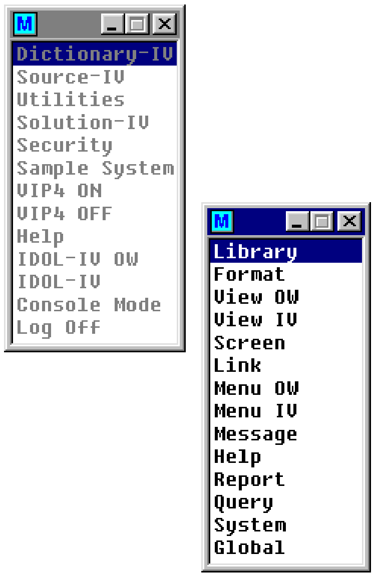
OPENworkshop provides programmers with facilities to help speed the development process. These facilities are designed to encourage a prototyping, incremental approach to development.

System classes are all highly accessible while an application is running so that they can be inspected and modified at will. Powerful tracing facilities and where-used views make it easy to trace the recent history of processing and control in the application under development.

To help learn the wide-ranging facilities, OPENworkshop's context-sensitive help system prompts the developer whenever required.

Dictionary-IV Menu

When you first log on to OPENworkshop, a short menu is displayed. Select the option Dictionary-IV to display a menu of classes that you can work with, as displayed here.



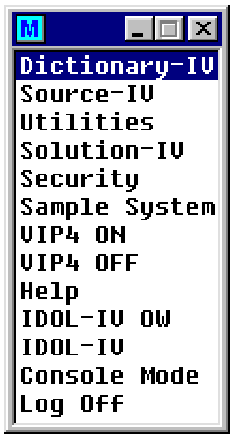
The Dictionary-IV Menu provides access to all classes in the environment, either by selecting FORMAT from the menu, or indirectly by first selecting a library, and then formats within the library.

The Dictionary-IV Menu, shown here, is available from the main OPENworkshop Menu and also from the **Ctrl-P** hotkey, which can be used at any time during operation of an application.

Access to these facilities is protected, ensuring that only users with developer status can access them.

RUN”OO”/Ctrl-P Hotkey

The Dictionary-IV Menu and other utilities are available at any time to developers by running the OO program. **RUN”OO”** from Thoroughbred Basic Console mode or **OO** from any Dictionary-IV menu selection field.

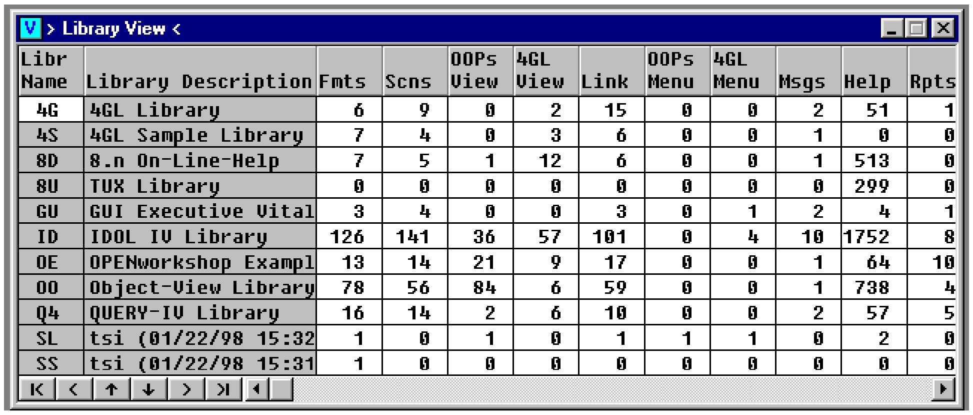


Try it during your exploration of the OPENworkshop system. Select **Close (F4)** to return to your previous position.

Note: Do not try DEVELOPER ON/OFF from the System Utilities until you have reviewed the Developer Status section of this manual.. The facilities for developers are controlled by this switch and will become unavailable to you. For more information see the Dictionary-IV Administrator Guide.

The Library View

The first option of the Dictionary-IV Menu displays the Library View, a typical example of which is shown below. This view provides a summary of all the classes and objects in the system. Select **Library** from the menu to display the following view.

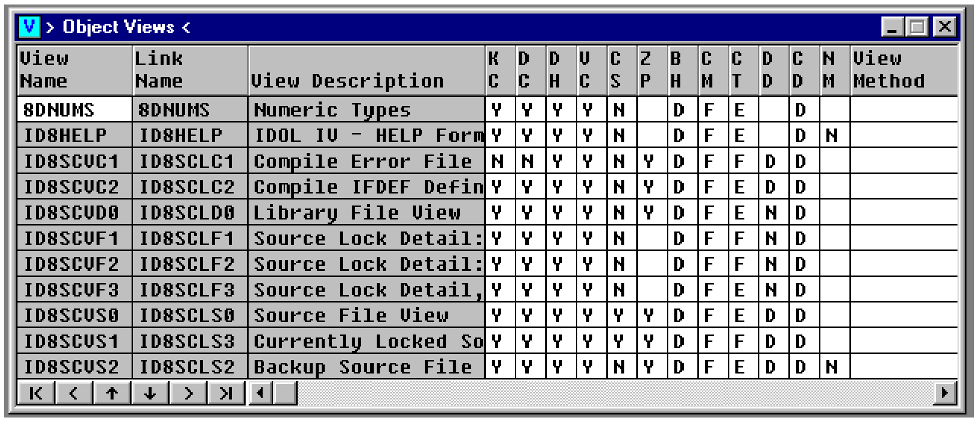


Libr Name	Library Description	Fmts	Scns	OOPs View	4GL View	Link	OOPs Menu	4GL Menu	Msgs	Help	Rpts
4G	4GL Library	6	9	0	2	15	0	0	2	51	1
4S	4GL Sample Library	7	4	0	3	6	0	0	1	0	0
8D	8.n On-Line-Help	7	5	1	12	6	0	0	1	513	0
8U	TUX Library	0	0	0	0	0	0	0	0	299	0
GU	GUI Executive Vital	3	4	0	0	3	0	1	2	4	1
ID	IDOL IV Library	126	141	36	57	101	0	4	10	1752	8
OE	OPENworkshop Exampl	13	14	21	9	17	0	0	1	64	10
OO	Object-View Library	78	56	84	6	59	0	0	1	738	4
Q4	QUERY-IV Library	16	14	2	6	10	0	0	2	57	5
SL	tsi (01/22/98 15:32	1	0	1	0	1	1	1	0	2	0
SS	tsi (01/22/98 15:31	1	0	0	0	0	0	0	0	0	0

More than just displaying the list, the Library View provides access to all the objects in the system. Try this for yourself. Move the highlight cursor to, the Fmts column and the OE row, which is highlighted in the illustration above then, press **Enter**. A list of formats from the OE library will be displayed.

Access Related Objects

In many cases, objects are related to others. A view uses a link. The view may have an associated view method and help.



> Object Views <

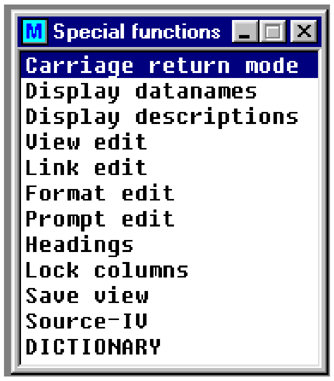
View Name	Link Name	View Description	K C	D C	D H	V C	C S	Z P	B H	C M	C T	D D	C D	N M	View Method
8DNUMS	8DNUMS	Numeric Types	Y	Y	Y	Y	N		D	F	E		D		
ID8HELP	ID8HELP	IDOL IV - HELP Form	Y	Y	Y	Y	N		D	F	E		D	N	
ID8SCVC1	ID8SCLC1	Compile Error File	N	N	Y	Y	N	Y	D	F	F	D	D		
ID8SCVC2	ID8SCLC2	Compile IFDEF Defin	Y	Y	Y	Y	N	Y	D	F	E	D	D		
ID8SCVD0	ID8SCLD0	Library File View	Y	Y	Y	Y	N	Y	D	F	E	N	D		
ID8SCVF1	ID8SCLF1	Source Lock Detail:	Y	Y	Y	Y	N		D	F	F	N	D		
ID8SCVF2	ID8SCLF2	Source Lock Detail:	Y	Y	Y	Y	N		D	F	F	N	D		
ID8SCVF3	ID8SCLF3	Source Lock Detail,	Y	Y	Y	Y	N		D	F	E	N	D		
ID8SCVS0	ID8SCLS0	Source File View	Y	Y	Y	Y	Y	Y	D	F	E	D	D		
ID8SCVS1	ID8SCLS3	Currently Locked So	Y	Y	Y	Y	Y	Y	D	F	F	D	D		
ID8SCVS2	ID8SCLS2	Backup Source File	Y	Y	Y	Y	N	Y	D	F	E	D	D	N	

Wherever you are displaying the Object Views, all of these related objects are available. Simply move the cursor to the relevant field and select an option.

Select **Close (F4)** to return to the Library View.

Special Functions

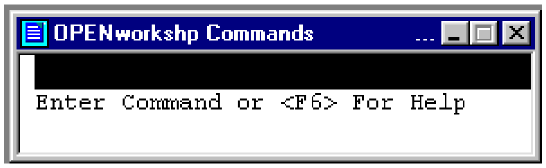
In views and screens the **Special functions (F7)** option provides access to a menu of maintenance functions for developers. The specific list of available functions may change from place to place, as you move through the system.



Select **Special functions** while the Library Header is displayed to see a typical example of the special functions available while in a view.

F10 Key

In a menu or help, the **F10** key presents a command line, allowing you to connect to Dictionary-IV maintenance and a set of utilities. Access to the **F10** command line can be blocked by disabling this feature in the Installation Information screen. The following tables show the commands that may be entered:



```
Enter Command or <F6> For Help
```

Press **F6** and select **Dictionary Commands**.

Command	Argument	Example
FORMAT	format-name	[FORMAT OEFCUST] or [F OEFCUST]
GLOBAL		[GLOBAL] or [GLO]
HELP	help-name	[HELP OEHELP] or [H OEHELP]
LIBRARY	library-name	[LIBRARY OE] or [L OE]
LINK	link-name	[LINK OELCUST] or [LIN OELCUST]
MENU	pop-up-menu-name	[MENU OEM01] or [M OEM01]
MENUS	screen-menu-name	[MENUS OEMS01] -- no shorthand
MESSAGE	message-dict-name	[MESSAGE OEMSGS] or [MES OEMSGS]
METHOD	method-name [msg]	[METHOD OEGENTD] or [MET OEGENTD]
QUERY	query-name	[QUERY OEQRY1] or [Q OEQRY1]
REPORT	report-name	[REPORT OERCUST] or [R OERCUST]
SCREEN	screen-name	[SCREEN OESCUST] or [S OESCUST]
SOL4		[SOL4] or [SO]
SOURCE	lib.source	[SOURCE OE.OEM1] or [SOU OE.OEM1]
SYSVARS		[SYSVARS] or [SY]
UTILITY		[UTILITY] or [U]
VIEW	view-name	[VIEW OEVCUST] or [V OEVCUST]

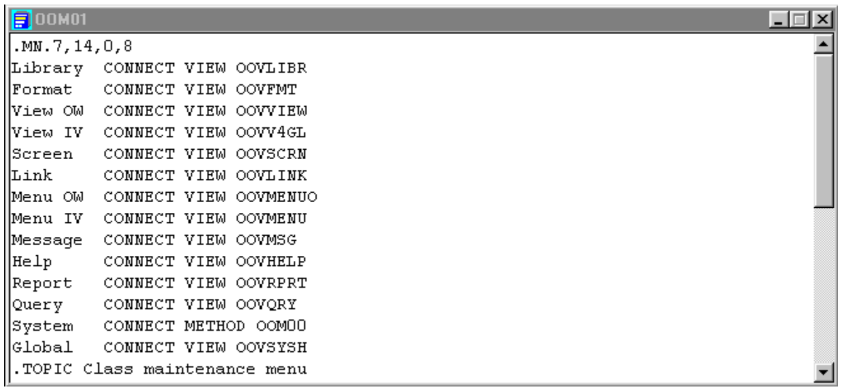
Press **F6** and select **Utility Commands**.

Command	Command description
/	Display unix directory.
A	Print disk directory.
B	Build fileset list.
D	Rename programs or files.
E	Erase programs or files.
F	Display file information.
G	Ghost task communication.
I	Hex file dump.
J	Basic defined directories.
L	List disk directory.
M	Display task allocation.
O	Define files.
R	Update Object Libraries.
S	Transfer programs or files.
T	Transfer of expand programs or files.
U	Unix Utilities

F1 Key

You can view and maintain the definition controlling a menu by pressing **F1** while the menu is displayed. The following figures show the result of pressing **F1** while the standard Dictionary-IV menu is displayed:

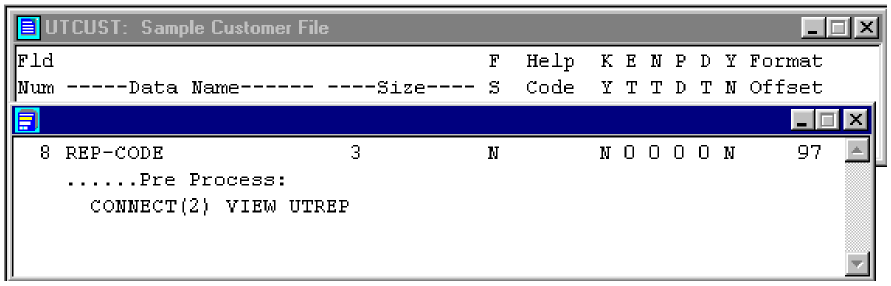
Menu Definition



```
.MN.7,14,0,8
Library  CONNECT VIEW OOVLIBR
Format   CONNECT VIEW OOVFMT
View OW  CONNECT VIEW OOVVIEW
View IV  CONNECT VIEW OOVV4GL
Screen   CONNECT VIEW OOVSCRN
Link     CONNECT VIEW OOVLINK
Menu OW  CONNECT VIEW OOVMENUO
Menu IV  CONNECT VIEW OOVMENU
Message  CONNECT VIEW OOVMSG
Help     CONNECT VIEW OOVHELP
Report   CONNECT VIEW OOVRPRT
Query    CONNECT VIEW OOVQRY
System   CONNECT METHOD OOM00
Global   CONNECT VIEW OOVSYSH
.TOPIC Class maintenance menu
```

F11 Key

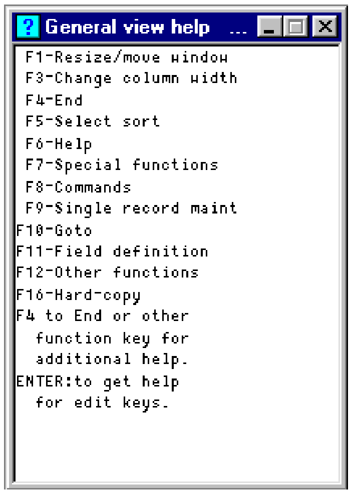
Whenever the cursor is in a data element name field of a screen or a view you can press the **F11** key to obtain the definition of the data element name. You can also maintain the data element name at this point through the **F11** key.



```
Fld                                 F  Help  K E N P D Y Format
Num -----Data Name------ ----Size---- S  Code  Y T T D T N Offset

  8 REP-CODE              3           N        N O O O O N     97
    ......Pre Process:
      CONNECT(2) VIEW UTREP
```

Help

Whenever you are in doubt, select **Help (F6)** to tell you what your options are. Help changes as you move from place to place through the system.



```
 F1-Resize/move window
 F3-Change column width
 F4-End
 F5-Select sort
 F6-Help
 F7-Special functions
 F8-Commands
 F9-Single record maint
F10-Goto
F11-Field definition
F12-Other functions
F16-Hard-copy
F4 to End or other
  function key for
  additional help.
ENTER:to get help
  for edit keys.
```

Summary of Function Keys and Access Routes

Key	**Menu**	**View**	**Screen**	**Help**
F1	Edit	Edit	Edit	Edit
F2	-----	Edit	-----	-----
F3	Print Screen	Print Screen	-----	Print Screen
F4	End (Close)	End (Close)	End (Close)	End (Close)
F5	-----	Sort	Sort	-----
F6	Help	Help	Help	Help
F7	------	Special Functions	-----	Special Functions
F8	Move	File Maintenance Commands	-----	Move
F9	-----	Display Screen	-----	-----
F10	Command	Go to	Go to	Command
F11	-----	Edit Data Element	Edit Data Element	Edit Data Element
F12	-----	Other Functions	-----	-----
F13	-----	-----	-----	Debug
F14	-----	-----	-----	-----
F15	-----	-----	-----	-----
F16	Print	Print	Print	Print

A developer can redefine the meaning of any of these keys by assigning a CONNECT directive to that key.

OPENWORKSHOP CLASSES

This section provides reference material on the following OPENworkshop classes:

- **Presentation Classes**

  Help
  Menu
  Message
  Query
  Report
  Screen
  View

- **Data Classes**

  Library
  data element name
  Format
  Link

- **Directives**

  CONNECT Directive
  Other Directives

Presentation Classes

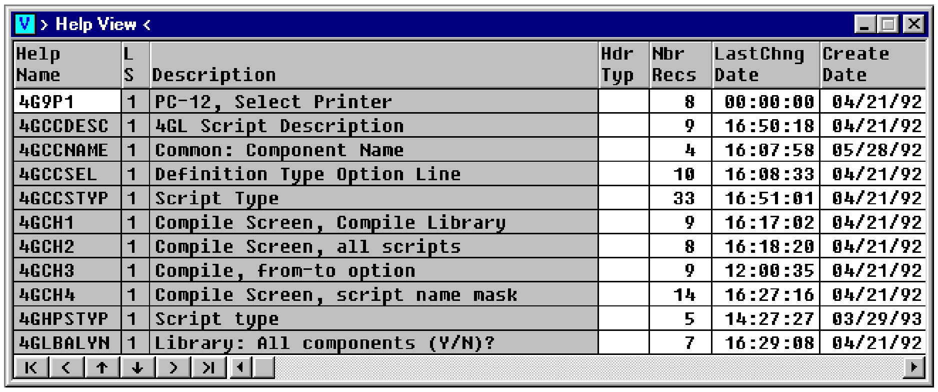
Help

The help subsystem lets you provide context-sensitive, structured help throughout an OPENworkshop application. You can specify help messages to display on an application, class or data element name level. You can present help in a number of different forms.

OPENworkshop displays a help message whenever a user selects **Help (F6)**. Other help messages are displayed as prompts by OPENworkshop applications. As a developer you can define the size and position of the window in which the help is displayed. In VIP size and position confirms to Windows standards. There are two alternative strategies for developing the text of help messages:

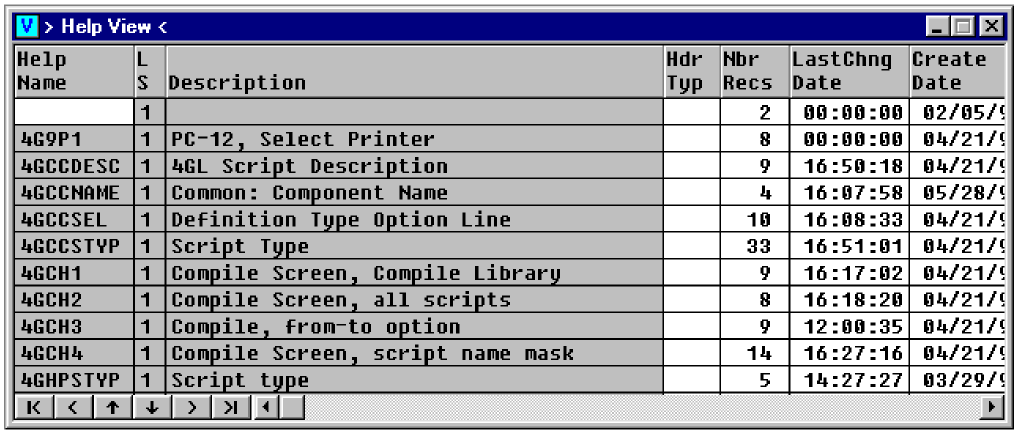
- Since OPENworkshop allows you to create help whenever none is defined, it is possible to wait until the application has been created then build the help as the application is exercised. For help messages that will be displayed in response to the user selecting **Help** this strategy will often serve to make your help more relevant and accurate in shorter time.

- The alternative is to create help through the Help Header, shown below. This method must be used to create indexed or function key help.


> Help View <

Help Name	LS	Description	Hdr Typ	Nbr Recs	LastChng Date	Create Date
4G9P1	1	PC-12, Select Printer		8	00:00:00	04/21/92
4GCCDESC	1	4GL Script Description		9	16:50:18	04/21/92
4GCCNAME	1	Common: Component Name		4	16:07:58	05/28/92
4GCCSEL	1	Definition Type Option Line		10	16:08:33	04/21/92
4GCCSTYP	1	Script Type		33	16:51:01	04/21/92
4GCH1	1	Compile Screen, Compile Library		9	16:17:02	04/21/92
4GCH2	1	Compile Screen, all scripts		8	16:18:20	04/21/92
4GCH3	1	Compile, from-to option		9	12:00:35	04/21/92
4GCH4	1	Compile Screen, script name mask		14	16:27:16	04/21/92
4GHPSTYP	1	Script type		5	14:27:27	03/29/93
4GLBALYN	1	Library: All components (Y/N)?		7	16:29:08	04/21/92

To add a new help item select **Line Insert** in the Header View. OPENworkshop creates a new row.


> Help View <

Help Name	LS	Description	Hdr Typ	Nbr Recs	LastChng Date	Create Date
	1			2	00:00:00	02/05/9
4G9P1	1	PC-12, Select Printer		8	00:00:00	04/21/9
4GCCDESC	1	4GL Script Description		9	16:50:18	04/21/9
4GCCNAME	1	Common: Component Name		4	16:07:58	05/28/9
4GCCSEL	1	Definition Type Option Line		10	16:08:33	04/21/9
4GCCSTYP	1	Script Type		33	16:51:01	04/21/9
4GCH1	1	Compile Screen, Compile Library		9	16:17:02	04/21/9
4GCH2	1	Compile Screen, all scripts		8	16:18:20	04/21/9
4GCH3	1	Compile, from-to option		9	12:00:35	04/21/9
4GCH4	1	Compile Screen, script name mask		14	16:27:16	04/21/9
4GHPSTYP	1	Script type		5	14:27:27	03/29/9

Enter a name, description and other required attributes, as shown in the table below. Then, with the highlight on the Help Name, select **Edit (F1)** to create the help text using the help text editor.

Column	**Attribute**
Help Name	Two character Library name followed by up to 6 character Help name
LS	Language Suffix. Defaults to 1. Select a different suffix for different languages where you are using multi-language support. See Dictionary-IV Administrator Guide for more details.
Description	Up to 40 character description

Hdr Typ	Header Type.
	
	C = Centered
	L = Left aligned
	R = Right aligned
	space = no header
	
	OPENworkshop uses the Description as the Header.

Other columns	Maintained by OPENworkshop

Help Text Editor

The help text editor allows you to create and edit the contents of a help text file, and also control where the window is displayed on the screen. In VIP size and position confirms to Windows standards. The following table provides an overview of the facilities provided by the editor. See the on-line help within OPENworkshop or review the Dictionary-IV Reference Manual for more details. Below is a table of help editor for character functions:

Key	Function	Description
F1	Split Line	Splits the current line into two lines at the cursor position. Characters after the current cursor are moved to the following line.
F2	Join Line	Joins two lines eliminating any blanks at the end of the first line and the beginning of the second.
F3	Spell Check	Performs a spelling check on the help file contents.
F4	End	Ends the editing session, giving you the choice of saving or discarding the changes made.
F5	Format Text	Justifies text from the current line to the end of the paragraph.
F6	Help	Displays help for the help file editor.
F7	Special Functions	A menu of special functions as follows:
	Change Tabs	Sets tab positions for the file.
	Resize/Move Window	Moves and resizes the window in which help is displayed.
	Edit Another Document	Allows you to view or edit another document, returning the current document on completion.
	Color Settings	Allows color highlights to be added to help.
	Edit Mode Options	Allows the on/off status of a number of flags to be changed. These flags control text mode vs graphics mode, right margin alignment, automatic hyphenations, character attributes.
	Change Window Title	Allows the title of the help message to be changed.
	Select date	Inserts the current date into the file being edited.
	Extended Help	Presents help on how to design help text.
	Window Move	Allows the position of the help window to be changed using tab and up/down keys.
F8	Search/Replace	Allows a search or search and replace throughout the text file
F9	Expand Window	Expands the window to full screen size.
F10	GoTo	Searches the document for a pattern and places the cursor at that position.

Key	Function	Description
F11	Margin	Sets the left margin to the current cursor position.
F12	Copy	Copies the contents of another document to the current position in the help text.
F13	Cut/Paste	Allows text to be cut from one place and pasted elsewhere within the file.
F14	Undo	Undoes the last change made to the file. Multiple uses successively undo previous changes.
F15	Character/Graphics Mode	Switches editing between editing characters and editing graphics characters.
F16	Print	Prints the help message text.

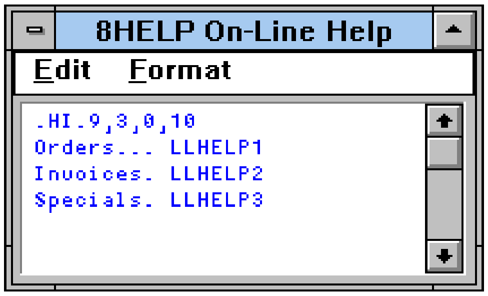
OPENworkshop offers a choice of types of help. See the beginning of this subsection for examples of their appearance. The type is specified to OPENworkshop by the first line of the help text.

Simple Help

Displays a simple text message when **Help (F6)** is selected. Any help text file that does not begin with one of the reserved help type specifiers (see the following) is assumed to be simple help.

To define a simple help file select **Line Insert** in the Help Header, specify the attributes required, then select **Edit (F1)**. Create the text. Adjust the position and size of the help window as required, and save the help file.

Indexed Help



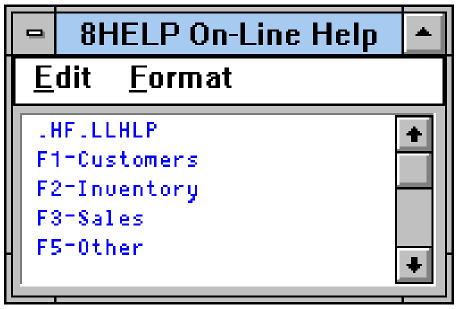
```
.HI.9,3,0,10
Orders... LLHELP1
Invoices. LLHELP2
Specials. LLHELP3
```

Shows a list of help subjects that the user can select when this indexed help is displayed. After the operator has selected from the list OPENworkshop displays the referenced help file.

Indexed help is specified by the characters .HI. starting at the first row, first column. You must then specify the window size and position for the help. (In VIP size and position conforms to Windows standards.) In this example, the help is to be displayed in a window that is 9 characters wide, 3 characters high, positioned at column 0, row 10.

To define a help index file select **Line Insert** in the Help Header, specify the attributes required, then with the highlight on the name field select **Edit (F1)** for the help index file. Create the text as indicated in the model shown above and save it. Then create a help item for each of the options specified in the indexed help list. Typically, each of these items will be simple help, but they may be any type.

Function Key Help



```
.HF.LLHLP
F1-Customers
F2-Inventory
F3-Sales
F5-Other
```

Displays a help message for a number of function keys, determined by the contents of the function key help file. When the user presses a function key while this help message is displayed, a further help message is shown, selected by appending the two digit function key value (01, 02, etc.) to the help name prefix specified in the first line of the file.

In this example, pressing **F3** would display a help module named LLHLP03. Function key help is specified by the characters .HF. starting at the first line and the first column of the help file.

To define a function key help file press **Line Insert** in the Help Header view, specify a help name, then with the highlight on the Help Name select **Edit (F1)** for the help text file. Create the text as indicated in the model shown. Adjust the position and size of the help window as required, and save the help file. (In VIP size and position conforms to Windows standards.) Then create a help item for each of the function keys specified in the function key help file to contain the required help text. Typically, each of these items will be simple help, but they may be any type.

String Substitution

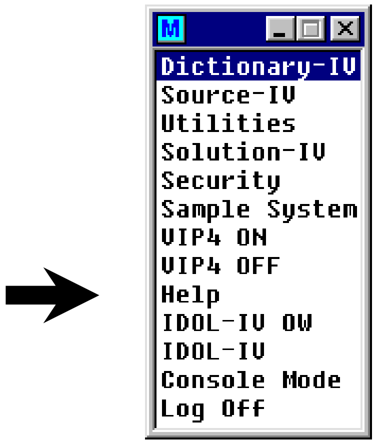
The help subsystem allows the application developer to substitute strings in a help message under the control of the application. To take advantage of this facility, use the CONNECT HELP directive and supply substitution parameters in the HELP$ string array, which is described in the **HELP$** section of this manual.

Help Topics Subsystem

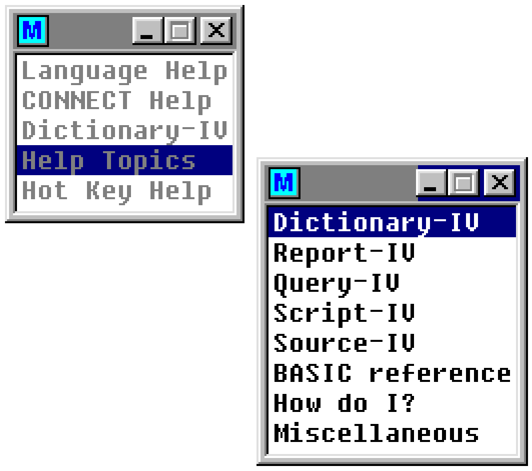
Complementing the context-sensitive help system, OPENworkshop provides an indexed help subsystem to let users find help on any topic.

Using Help Topics

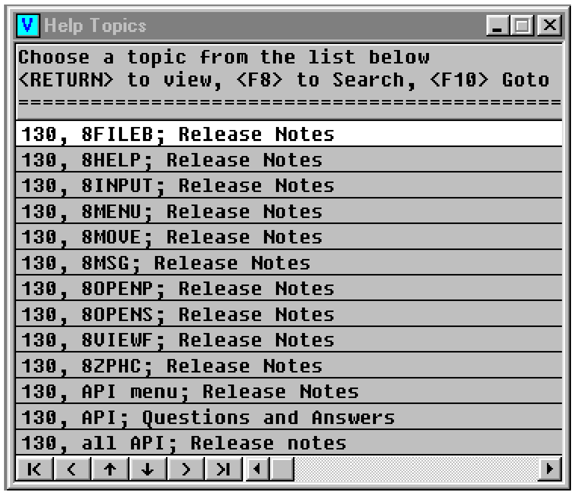
From the Main menu select **Help**.



A menu of major topics is presented for the user to select from.

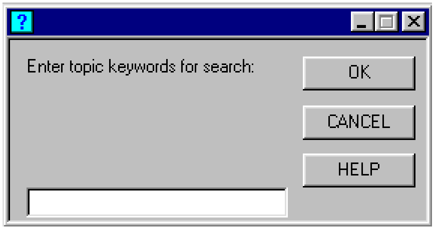


Select the category required, and the Help Topics subsystem opens a view of all available help for the selected category.



```
Choose a topic from the list below
<RETURN> to view, <F8> to Search, <F10> Goto
=============================================
130, 8FILEB; Release Notes
130, 8HELP; Release Notes
130, 8INPUT; Release Notes
130, 8MENU; Release Notes
130, 8MOVE; Release Notes
130, 8MSG; Release Notes
130, 8OPENP; Release Notes
130, 8OPENS; Release Notes
130, 8VIEWF; Release Notes
130, 8ZPHC; Release Notes
130, API menu; Release Notes
130, API; Questions and Answers
130, all API; Release notes
```

Select **Search (F8)** to search for items that contain one or more keywords.



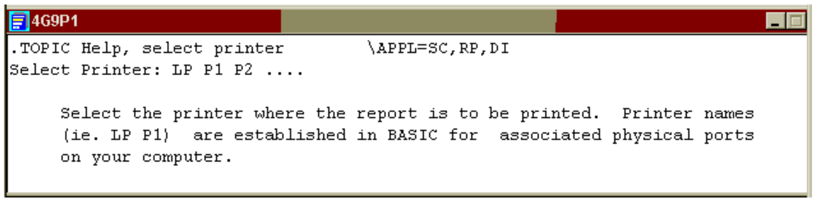
At the prompt, enter either a single text string or multiple items separated by commas. OPENworkshop searches for all items in the current category that contain any of the keywords entered, i.e., the keyword search does a logical OR between multiple keywords.

Select **GoTo (F10)** to go to an entry in the list that commences with a specified character string.

The list contains both help items and menus. Once the required item is located in the list, select it to display the item. If the chosen item is help, OPENworkshop simply displays the help text. If the chosen item is a menu, OPENworkshop displays the menu and allows the user to either select from that menu or end the display.

Creating Help Topics Items

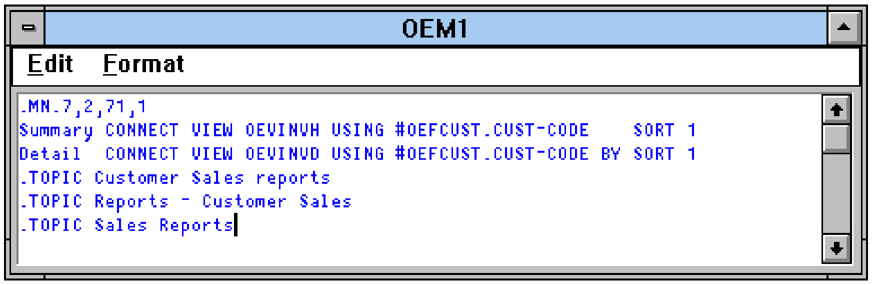
The developer can make help items or menus available to the Help Topics subsystem simply by adding a .TOPIC declaration to its definition.



```
.TOPIC Help, select printer        \APPL=SC,RP,DI
Select Printer: LP P1 P2 ....

     Select the printer where the report is to be printed.  Printer names
     (ie. LP P1)  are established in BASIC for  associated physical ports
     on your computer.
```

The help shown above contains a Help Topic declaration. The text, Customer Code, will be added to the topics available. The category into which this topic is to be placed is defined in the \APPL segment of the declaration; this declaration will be in the OE category. If no \APPL statement is made, the item will be placed in the Miscellaneous (ZZ) category.

NOTE: If the first line of the help definition contains a help type specification, then .TOPIC must be at the end of a help definition after the Help text. See the example below.

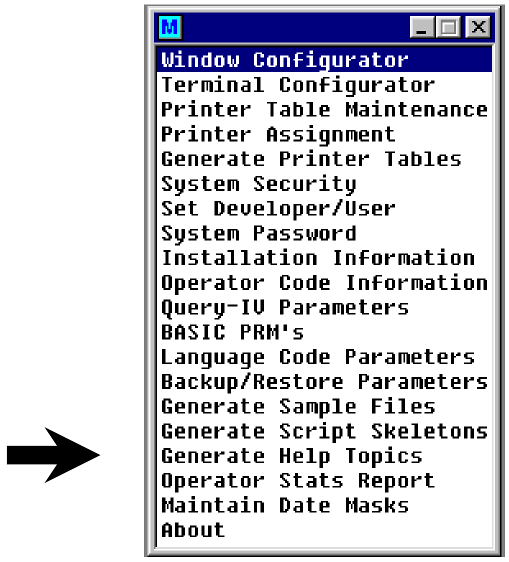


```
.MN.7,2,71,1
Summary CONNECT VIEW OEVINVH USING #OEFCUST.CUST-CODE    SORT 1
Detail  CONNECT VIEW OEVINVD USING #OEFCUST.CUST-CODE BY SORT 1
.TOPIC Customer Sales reports
.TOPIC Reports - Customer Sales
.TOPIC Sales Reports
```

The menu shown above contains multiple Help Topic declarations, enabling the item to be searched for in various ways.

Generating the Help Topics Index

The Help Topics index must be generated before the Help Topics subsystem can be used. Use the Control Menu/System Administration menu to display the menu below.

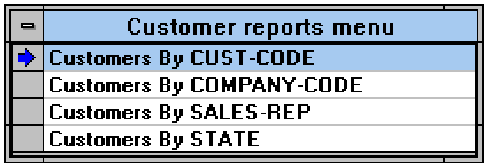


OPENworkshop requests the names of the libraries to be added to the index. If an index already exists, you are given the choice of appending to the existing index or replacing the existing index. For more information see the Dictionary-IV Administrator Guide.

Menu

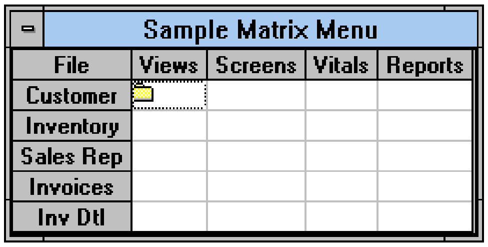
A menu allows the user to choose an action from a number of options. OPENworkshop supports three different forms of menus: a simple list, a matrix menu and a selection menu.

List Menu Sample



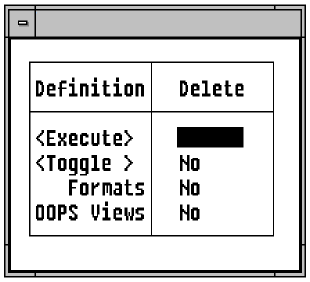
In a list menu on a character terminal move the cursor up and down using the **Up Arrow** and **Down Arrow** keys. Alternatively you can press the first character of the option you require, and the lightbar will move to the next line starting with that character. In a graphical user interface, point and double click on the item.

Matrix Menu Sample



A matrix menu allows you to choose from a matrix of options. On a character terminal use the arrow keys to move around the columns and rows, and the **Enter** key to select the required option. On a graphical workstation, point and double click to select the option.

Selection Menu Sample

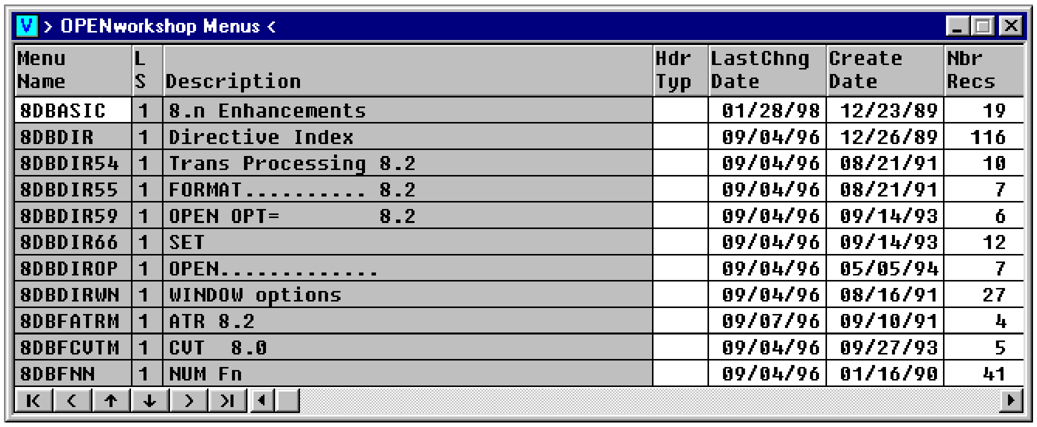


A selection menu is used to pass selection parameters to an operation. The example on the previous page shows a menu that will execute a Delete function when the **Enter** key is pressed. Each of the parameters listed below the cursor can be toggled by moving to the cell and selecting it. Alternatively, select **Toggle** to toggle all the parameters at once, when you are ready, select **Execute**.

The developer is able to control the position and appearance of the list, matrix, or selection menu on the screen, and the groups of users who may see and select the menu as whole or individual options on the menu.

Maintaining Menus

To maintain menus move to the Menu Header view, as shown below:



Menu Name	LS	Description	Hdr Typ	LastChng Date	Create Date	Nbr Recs
8DBASIC	1	8.n Enhancements		01/28/98	12/23/89	19
8DBDIR	1	Directive Index		09/04/96	12/26/89	116
8DBDIR54	1	Trans Processing 8.2		09/04/96	08/21/91	10
8DBDIR55	1	FORMAT........... 8.2		09/04/96	08/21/91	7
8DBDIR59	1	OPEN OPT= 8.2		09/04/96	09/14/93	6
8DBDIR66	1	SET		09/04/96	09/14/93	12
8DBDIROP	1	OPEN.............		09/04/96	05/05/94	7
8DBDIRWN	1	WINDOW options		09/04/96	08/16/91	27
8DBFATRM	1	ATR 8.2		09/07/96	09/10/91	4
8DBFCVTM	1	CVT 8.0		09/04/96	09/27/93	5
8DBFNN	1	NUM Fn		09/04/96	01/16/90	41

To edit an existing menu, move to the row and select **Edit (F1)**. To create a new menu, select **Line Insert**, enter a name and attributes (see below) for the menu and then with the highlight on the name field select **Edit (F1)** to edit the new definition.

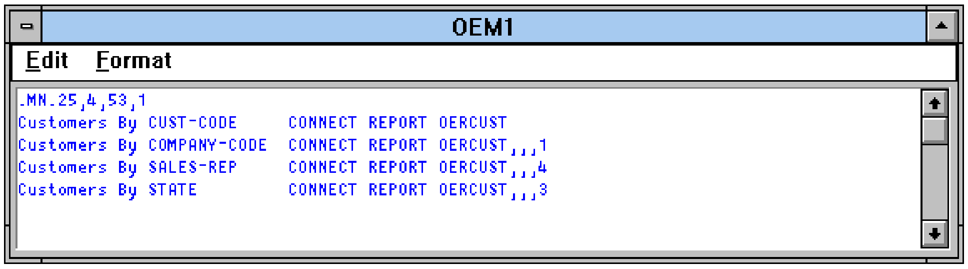
Column	**Attribute**
Menu Name	Two character library name followed by up to 6 character help name.
LS	Language suffix. Defaults to 1. Select a different suffix for different languages where you are using multi-language support. See the Dictionary-IV Administrator Guide for more details.
Description	Up to 40-character description.

Hdr Typ	Header Type:
	
	C = Centered
	L = Left aligned
	R = Right aligned
	space = no header
	
	OPENworkshop uses the description as the Header.

Other columns	Maintained by OPENworkshop

A file containing the menu definition is opened. You can edit the file using the text editor. The definitions for each of the three sample menus shown at the beginning of this section are shown in the following. The type of menu required is specified in the first line of the menu definition.

List Menu

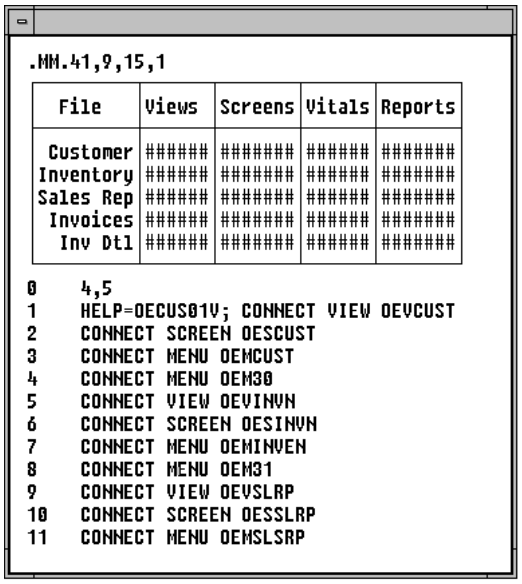


```
.MN.25,4,53,1
Customers By CUST-CODE     CONNECT REPORT OERCUST
Customers By COMPANY-CODE  CONNECT REPORT OERCUST,,,1
Customers By SALES-REP     CONNECT REPORT OERCUST,,,4
Customers By STATE         CONNECT REPORT OERCUST,,,3
```

In the first line, .MN. specifies this as a list menu, to be displayed in a window that is 25 characters wide, 4 lines high, and positioned at column 53 and row 1.

Each line of the menu is defined in the following lines of the definition. The first 25 characters are displayed in the window. The directive to be processed when the option is selected starts at character position 25+2. If the definition defines more options than the height of the window, scroll within the window.

Matrix Menu



```
.MM.41,9,15,1
┌────────────┬──────┬───────┬──────┬───────┐
│    File    │Views │Screens│Vitals│Reports│
│            │      │       │      │       │
│   Customer │######│#######│######│#######│
│  Inventory │######│#######│######│#######│
│  Sales Rep │######│#######│######│#######│
│   Invoices │######│#######│######│#######│
│    Inv Dtl │######│#######│######│#######│
└────────────┴──────┴───────┴──────┴───────┘
0     4,5
1     HELP=OECUS01V; CONNECT VIEW OEVCUST
2     CONNECT SCREEN OESCUST
3     CONNECT MENU OEMCUST
4     CONNECT MENU OEM30
5     CONNECT VIEW OEVINVN
6     CONNECT SCREEN OESINVN
7     CONNECT MENU OEMINVEN
8     CONNECT MENU OEM31
9     CONNECT VIEW OEVSLRP
10    CONNECT SCREEN OESSLRP
11    CONNECT MENU OEMSLSRP
```

As with the list menu, the first line defines the type of menu and the size and position of the window. The example also shows how to specify a help file that will be available when the menu is displayed.

The next 9 lines, including the line graphics, in this example define the layout of the menu on the screen. Line graphics define the outline border. Other text is displayed as typed, except fields filled with a # (pound sign).

These fields define where the cursor can move to in the displayed matrix menu.

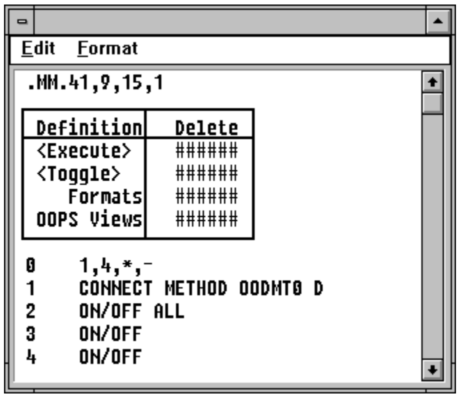
You must specify a regular matrix, i.e., no gaps can be left. However, different columns do not all have to be specified at the same width. Choose widths that suit the column headings.

The line numbered 0 that follows the screen representation specifies the number 4 columns and 5 rows within the matrix menu options matrix. Each row and column combination offers an option to the user, defined in the following lines of the menu definition.

The following lines specify the directives to be executed for each cell in the matrix. Lines are numbered by counting the columns row-by-row. The line numbered 1 specifies the directives to be executed for row 1 and column 1. Line 2 specifies the option for row 1 and column 2. Line 5 specifies row 2 and column 1. See the following table for cell numbering in this example.

Vitals	**Views**	**Screens**	**Vitals**	**Reports**
Customer	1	2	3	4
Inventory	5	6	7	8
Sales Rep	9	10	11	12
Invoices	13	14	15	16
Inv Dtl	17	18	19	20

Selection Menu



```
.MM.41,9,15,1

 Definition   Delete
 <Execute>    ######
 <Toggle>     ######
    Formats   ######
 OOPS Views   ######

0    1,4,*,-
1    CONNECT METHOD OODMT0 D
2    ON/OFF ALL
3    ON/OFF
4    ON/OFF
```

A selection menu is a type of matrix menu, and is specified as a matrix menu in line 1 of the definition. The screen layout and fields are also specified as in a matrix menu. Note that more than one column of options can be specified if required.

The line after the screen definition specifies the number of columns and rows, followed by the characters to be passed for the ON and OFF state for each option, as follows:

0 *Cols,Rows,PassOn,PassOff,DisplayOn,DisplayOff,DefaultValue*

Cols	number of columns of options to be displayed in the matrix.
Rows	number of rows of options to display.
PassOn	single character to be passed with directive when option is set ON. The default is **Y**.
PassOff	single character to be passed with directive when option is set OFF. The default is **N**.
DisplayOn	string to display in matrix when option is set ON. The default is **Yes**.
DisplayOff	string to display in matrix when option is set OFF. The default is **No**.
DefaultValue	default state for options. The default is **No**.

As with the matrix menu the succeeding lines numbered 1 and above, specify the directives to be executed when an option is selected. In this example the first option is designed to be used to execute the OODMT0 method, passing message D, while the other options toggle the values of parameters.

Lines 2, 3 and 4 of the example above contain the ON/OFF directive. They direct the menu to toggle the value of a parameter between ON and OFF when the user selects the option. ON/OFF ALL on line 2 toggles all parameters in the menu.

The current values of these parameters are passed to the called method through string array MSG1$[ALL].

MSG1$[1]	Message from CONNECT directive (in this example the string **D**)	
MSG1$[2]	(1,1)	Number of parameters
	(2,1)	The PassOn character (in this example "*")
	(3,1)	The PassOff character (in this example "-")
	(4,n)	The parameter values, in the order R1C1, R1C2, ... R2C1 .. to the end.

Help in Menus

OPENworkshop allows you to define the text to be displayed when a user selects **Help** while a menu is displayed. A help file may be defined for the entire menu, and also for individual menu lines.

To specify the help text for the entire menu, append **\HELP=HELPNAME** to the first line of the menu definition. To specify the help file for individual lines, insert **\HELP=HELPNAME** at the beginning of the line.

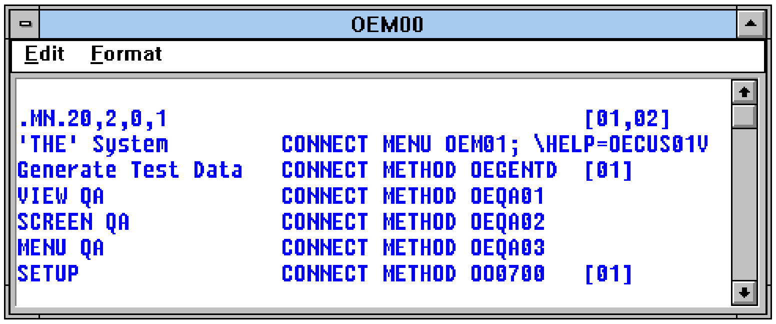
Note: **HELPNAME** is the name of your help object.

Menus can also be added to the Help Topics subsystem index. For more information, see the Help Topic subsystem section earlier in this section.

Security in Menus

OPENworkshop security controls are different from those in Dictionary-IV. Users may be assigned to one or more security groups, and access to menu options or whole menus can be limited to members of selected groups.

Security groups are identified by a number. See the Security section of this manual for an explanation of how groups are managed.



```
.MN.20,2,0,1                                      [01,02]
'THE' System          CONNECT MENU OEM01; \HELP=OECUS01V
Generate Test Data    CONNECT METHOD OEGENTD   [01]
VIEW QA               CONNECT METHOD OEQA01
SCREEN QA             CONNECT METHOD OEQA02
MENU QA               CONNECT METHOD OEQA03
SETUP                 CONNECT METHOD 000700    [01]
```

To specify which security groups may access an entire menu add the security group list to the first line of the menu definition:

```
[mm,nn...]
```

where mm and nn are the security groups given access. No other users will be able to display the menu. Any number of groups may be specified.

To limit access to a line on a menu add the security group list to the relevant menu lines. Users who are not members of these groups will not be able to see or select these lines.

In the sample menu above, members of security groups 01 and 02 may access the menu, but only members of group 01 will be able to see or access the Generate Test Data or Setup options.

Where security is defined on a per-line basis, OPENworkshop will not display menu lines that are not available to the current user. To avoid displaying a blank line, OPENworkshop suppresses these lines, resulting in a vertically shortened menu display. If you wish to ensure that such a menu is displayed so that the bottom line of the menu remains in the same place, append the T character to the security group list. For example:

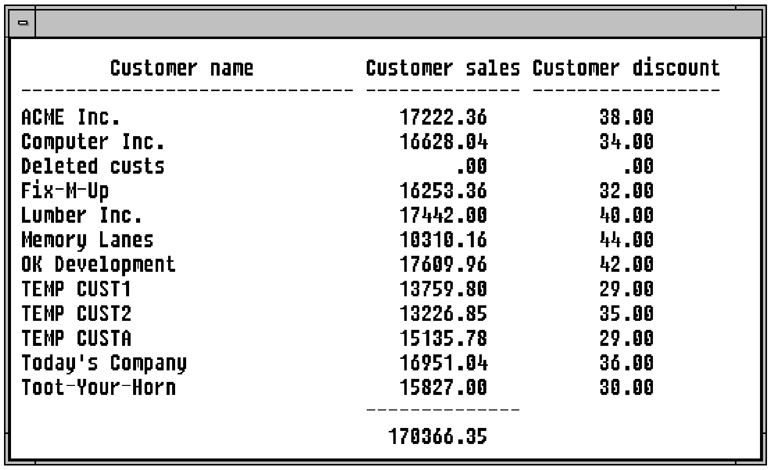
```
[01,02,T]
```

on a menu line allows groups 01 and 02 to see the line. All other users will see only the remaining menu lines, but in this case the window will be displayed starting one line lower on the screen.

NOTE: Security on a per-line basis is not available for selection menus.

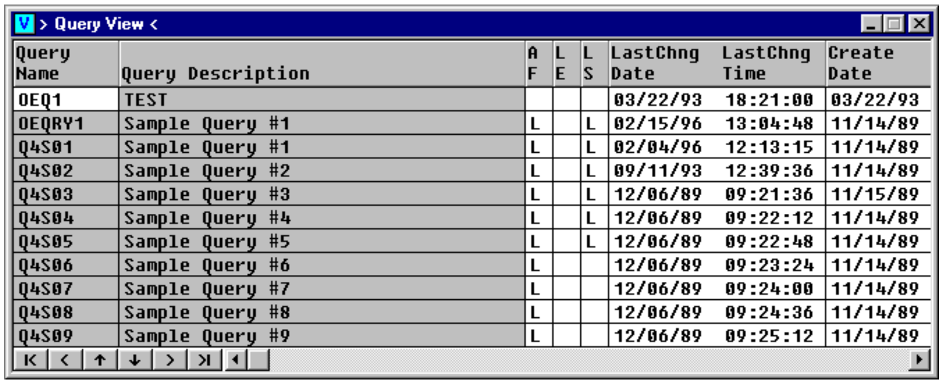
Query

A query creates a result that can be listed on the screen or printed to a file or printer. A sample query follows:



```
        Customer name            Customer sales Customer discount
------------------------------ -------------- -----------------
ACME Inc.                           17222.36          38.00
Computer Inc.                       16628.04          34.00
Deleted custs                            .00            .00
Fix-M-Up                            16253.36          32.00
Lumber Inc.                         17442.00          40.00
Memory Lanes                        10310.16          44.00
OK Development                      17609.96          42.00
TEMP CUST1                          13759.80          29.00
TEMP CUST2                          13226.85          35.00
TEMP CUSTA                          15135.78          29.00
Today's Company                     16951.04          36.00
Toot-Your-Horn                      15827.00          30.00
                               --------------
                                   178366.35
```

Select the Query Header view to create or maintain queries:



Query Name	Query Description	A F	L E	L S	LastChng Date	LastChng Time	Create Date
OEQ1	TEST				03/22/93	18:21:00	03/22/93
OEQRY1	Sample Query #1	L		L	02/15/96	13:04:48	11/14/89
Q4S01	Sample Query #1	L		L	02/04/96	12:13:15	11/14/89
Q4S02	Sample Query #2	L		L	09/11/93	12:39:36	11/14/89
Q4S03	Sample Query #3	L		L	12/06/89	09:21:36	11/15/89
Q4S04	Sample Query #4	L		L	12/06/89	09:22:12	11/14/89
Q4S05	Sample Query #5	L		L	12/06/89	09:22:48	11/14/89
Q4S06	Sample Query #6	L			12/06/89	09:23:24	11/14/89
Q4S07	Sample Query #7	L			12/06/89	09:24:00	11/14/89
Q4S08	Sample Query #8	L			12/06/89	09:24:36	11/14/89
Q4S09	Sample Query #9	L			12/06/89	09:25:12	11/14/89

To edit an existing query move to the row and select **Edit (F1).** To create a new query, select **Line Insert**, type a name and description for the menu and then select **Edit** to edit the new definition. To delete a query, select **Line Delete**.

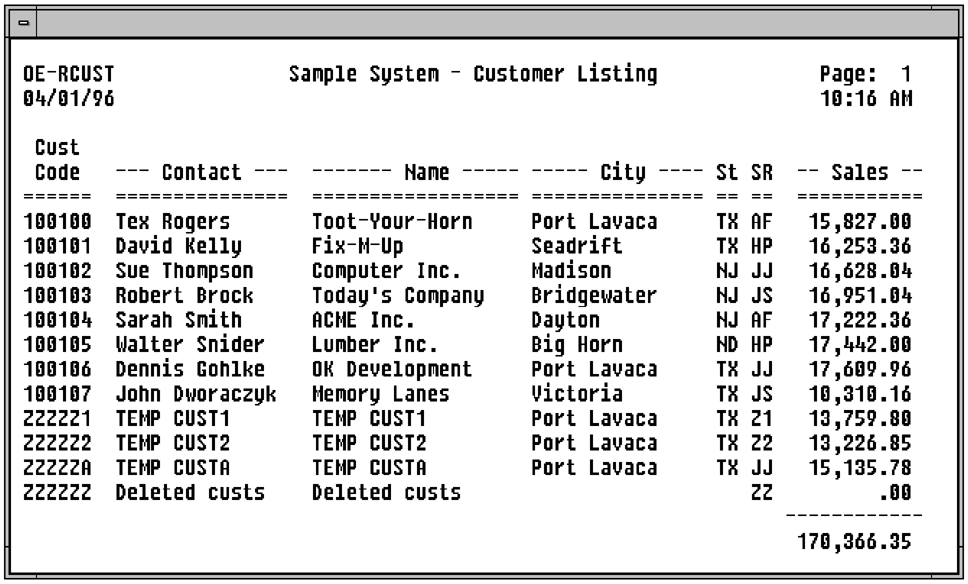
Queries are executed using the CONNECT QUERY directive.

For more information on queries, see the Query-IV Reference Manual.

Information can be passed to Query-IV through the QUERY$ string array. See the QUERY$ section of this manual.

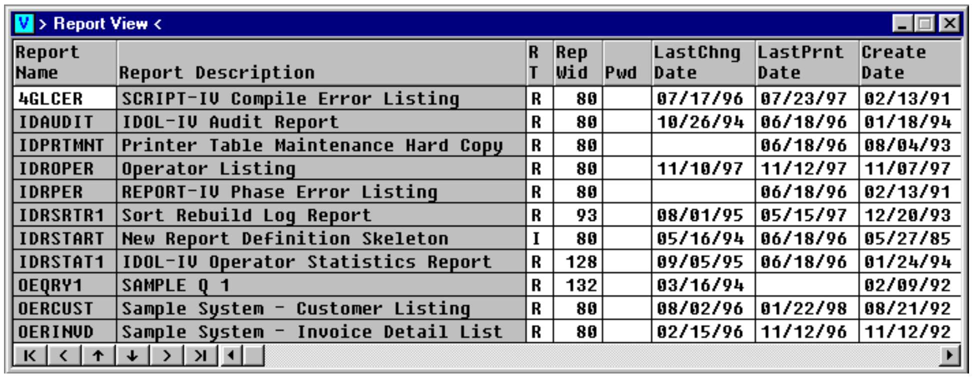
Report

Creates a report that can be listed on the screen or printed to a file or printer. A sample report follows:



```
OE-RCUST                Sample System - Customer Listing                Page:  1
04/01/96                                                                10:16 AM

 Cust
 Code   --- Contact ---  ------- Name -----  ----- City ----  St SR   -- Sales --
======  ===============  ==================  ===============  == ==  ===========
100100  Tex Rogers       Toot-Your-Horn      Port Lavaca      TX AF    15,827.00
100101  David Kelly      Fix-M-Up            Seadrift         TX HP    16,253.36
100102  Sue Thompson     Computer Inc.       Madison          NJ JJ    16,628.04
100103  Robert Brock     Today's Company     Bridgewater      NJ JS    16,951.04
100104  Sarah Smith      ACME Inc.           Dayton           NJ AF    17,222.36
100105  Walter Snider    Lumber Inc.         Big Horn         ND HP    17,442.00
100106  Dennis Gohlke    OK Development      Port Lavaca      TX JJ    17,609.96
100107  John Dworaczyk   Memory Lanes        Victoria         TX JS    10,310.16
ZZZZZ1  TEMP CUST1       TEMP CUST1          Port Lavaca      TX Z1    13,759.80
ZZZZZ2  TEMP CUST2       TEMP CUST2          Port Lavaca      TX Z2    13,226.85
ZZZZZA  TEMP CUSTA       TEMP CUSTA          Port Lavaca      TX JJ    15,135.78
ZZZZZZ  Deleted custs    Deleted custs                           ZZ          .00
                                                                     ------------
                                                                      170,366.35
```

Select the Report Header view to create or maintain reports:



> Report View <

Report Name	Report Description	R T	Rep Wid	Pwd	LastChng Date	LastPrnt Date	Create Date
4GLCER	SCRIPT-IV Compile Error Listing	R	80		07/17/96	07/23/97	02/13/91
IDAUDIT	IDOL-IV Audit Report	R	80		10/26/94	06/18/96	01/18/94
IDPRTMNT	Printer Table Maintenance Hard Copy	R	80			06/18/96	08/04/93
IDROPER	Operator Listing	R	80		11/10/97	11/12/97	11/07/97
IDRPER	REPORT-IV Phase Error Listing	R	80			06/18/96	02/13/91
IDRSRTR1	Sort Rebuild Log Report	R	93		08/01/95	05/15/97	12/20/93
IDRSTART	New Report Definition Skeleton	I	80		05/16/94	06/18/96	05/27/85
IDRSTAT1	IDOL-IV Operator Statistics Report	R	128		09/05/95	06/18/96	01/24/94
OEQRY1	SAMPLE Q 1	R	132		03/16/94		02/09/92
OERCUST	Sample System - Customer Listing	R	80		08/02/96	01/22/98	08/21/92
OERINVD	Sample System - Invoice Detail List	R	80		02/15/96	11/12/96	11/12/92

To edit an existing report move to the row and select **Edit (F1)**. To create a new report select **Line Insert**, type a name and description for the menu and then select **Edit** to edit the new definition. To delete a report select **Line Delete**.

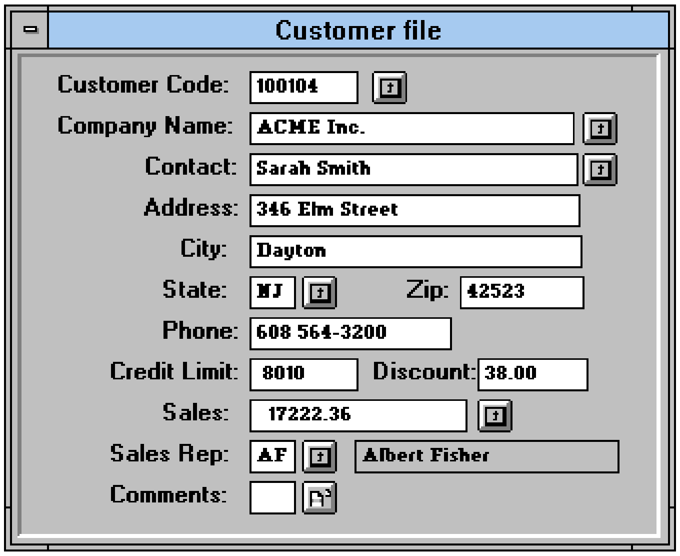
Reports are executed using the CONNECT REPORT directive.

For more information on reports, see the Report-IV Reference Manual.

Information can be passed to Report-IV through the REPORT$ string array. See the REPORT$ section of this manual.

Screen

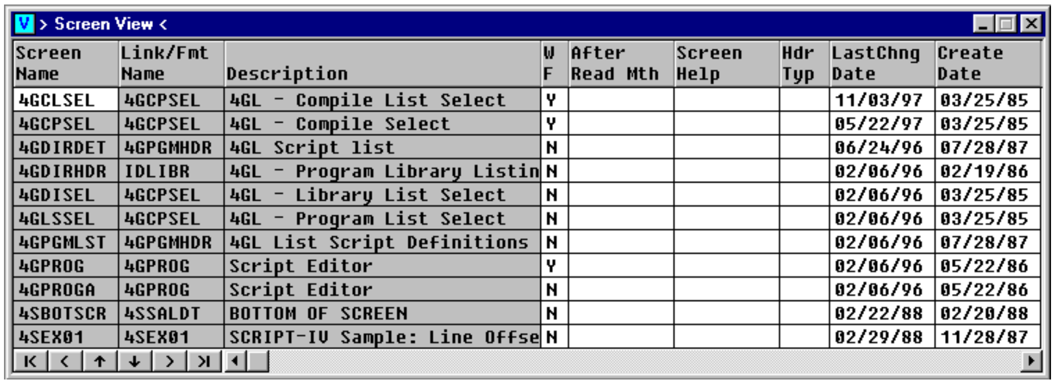
A screen displays data in a character or graphic format (if VIP is installed) and may also allow data to be input. A screen displays information from a single record in the underlying file, as shown in the example below.



Customer file

Customer Code:	100104
Company Name:	ACME Inc.
Contact:	Sarah Smith
Address:	346 Elm Street
City:	Dayton
State:	NJ
Zip:	42523
Phone:	608 564-3200
Credit Limit:	8010
Discount:	38.00
Sales:	17222.36
Sales Rep:	AF Albert Fisher
Comments:	

Select the Screen Header view to maintain or add screens:



> Screen View <

Screen Name	Link/Fmt Name	Description	W F	After Read Mth	Screen Help	Hdr Typ	LastChng Date	Create Date
4GCLSEL	4GCPSEL	4GL - Compile List Select	Y				11/03/97	03/25/85
4GCPSEL	4GCPSEL	4GL - Compile Select	Y				05/22/97	03/25/85
4GDIRDET	4GPGMHDR	4GL Script list	N				06/24/96	07/28/87
4GDIRHDR	IDLIBR	4GL - Program Library Listin	N				02/06/96	02/19/86
4GDISEL	4GCPSEL	4GL - Library List Select	N				02/06/96	03/25/85
4GLSSEL	4GCPSEL	4GL - Program List Select	N				02/06/96	03/25/85
4GPGMLST	4GPGMHDR	4GL List Script Definitions	N				02/06/96	07/28/87
4GPROG	4GPROG	Script Editor	Y				02/06/96	05/22/86
4GPROGA	4GPROG	Script Editor	N				02/06/96	05/22/86
4SBOTSCR	4SSALDT	BOTTOM OF SCREEN	N				02/22/88	02/20/88
4SEX01	4SEX01	SCRIPT-IV Sample: Line Offse	N				02/29/88	11/28/87

OPENworkshop maintains the following objects and attributes associated with a screen:

Column	**Attribute**
Link/Format Name	The link or format used by the screen to locate the required data for the screen.
Description	A text description of the screen and its purpose.
After Read Method	After data has been read from a file and before it is displayed to the user in a screen, an optional after read method can be used to prepare the data for display. This method may be used, for example, to prepare calculated fields. See the description of after read methods for more information.

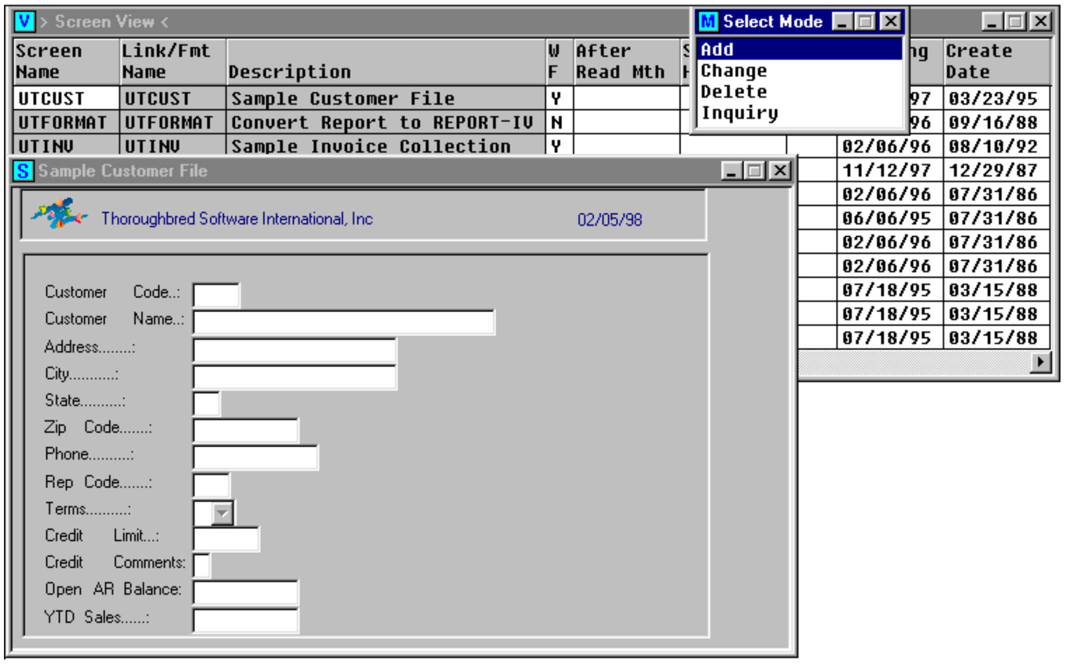
Column	Attribute
Screen Help	Specifies the help file that should be used to offer context-sensitive help for this screen.

Hdr Type	Header Type.
	
	C = Centered
	L = Left aligned
	R = Right aligned
	space = no header
	
	OPENworkshop uses the description as the header.

Other Columns	Maintained by OPENworkshop

To edit an existing screen definition put the cursor in the row that contains the definition and select **Edit (F1)**. To change the name of, or to copy an existing screen, type the new name and choose the options for the subsequent menus, which will display. To create a definition of a new screen select **Line Insert**, type a name for the new screen, then select **Edit** to edit the new definition.

The following display was the result of selecting **Edit** for the OESCUST screen:



```
Screen    Link/Fmt                               W  After      S     ng  Create
Name      Name      Description                  F  Read Mth   H         Date
UTCUST    UTCUST    Sample Customer File         Y                   97  03/23/95
UTFORMAT  UTFORMAT  Convert Report to REPORT-IV  N                   96  09/16/88
UTINV     UTINV     Sample Invoice Collection    Y             02/06/96  08/10/92
                                                               11/12/97  12/29/87
                                                               02/06/96  07/31/86
                                                               06/06/95  07/31/86
                                                               02/06/96  07/31/86
                                                               02/06/96  07/31/86
                                                               07/18/95  03/15/88
                                                               07/18/95  03/15/88
                                                               07/18/95  03/15/88
```

02/05/98

Customer Code..:
Customer Name..:
Address........:
City...........:
State..........:
Zip Code.......:
Phone..........:
Rep Code.......:
Terms..........:
Credit Limit...:
Credit Comments:
Open AR Balance:
YTD Sales......:

OPENworkshop displays the current screen definition, consisting of a template background and a number of data fields that display and collect data values.

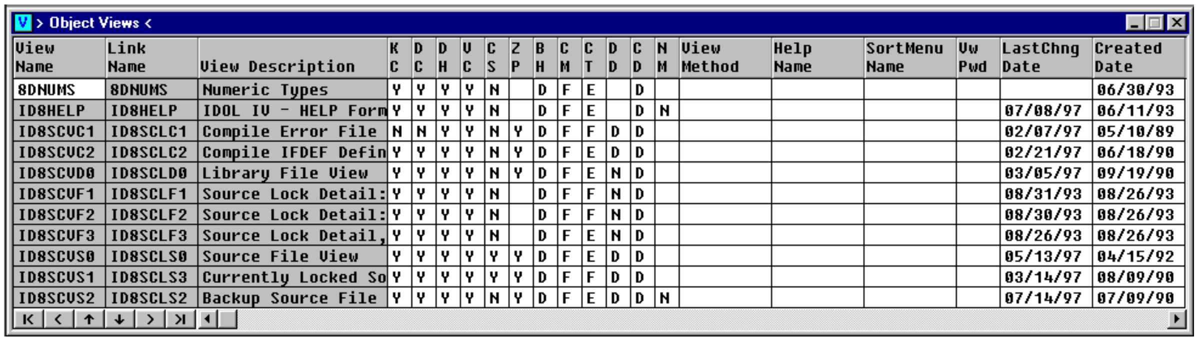
The following table provides an overview of the functions available for changing a screen definition in character mode. For more information, see on-line help or the Dictionary-IV Reference Manual. For more information about changing a graphic screen see the VIP for Dictionary-IV Reference Manual.

Below is a list of screen editor functions:

Key	**Function**	**Description**
F1	Preview Code Characters	Displays the screen with all of the special code characters converted.
F2	Delete data field	Deletes the data field from the screen, but does not affect the underlying format definition.
F3	Move data field	Moves the field within the Screen.
F4	End	Ends the editing session, giving you the choice of saving or discarding the changes made.
F5	Data field	Creates a data field in the screen definition. The field can display data from the underlying format or may be a calculated field for display only.
F6	Help	Displays help for the help file editor.
F7	Special Functions	A menu of special functions as follows:
F8	Print window	Prints the screen definition.
F10	Display data elements	Displays a list of data elements from the current format.
F15	Edit mode options	Switches editing between editing characters and editing graphics characters.
F16	Print	Prints the screen.

View

A view displays data from one or more files in rows and columns. If the developer has allowed it, the user can edit data by selecting the required cell and typing. The following example shows the Customer File view from the sample application supplied with OPENworkshop.


> Object Views <

View Name	Link Name	View Description	K C	D C	D H	V C	C S	Z P	B H	C M	C T	D D	C D	N M	View Method	Help Name	SortMenu Name	Vw Pwd	LastChng Date	Created Date
8DNUMS	8DNUMS	Numeric Types	Y	Y	Y	Y	N		D	F	E		D							06/30/93
ID8HELP	ID8HELP	IDOL IV - HELP Form	Y	Y	Y	Y	N		D	F	E		D	N					07/08/97	06/11/93
ID8SCVC1	ID8SCLC1	Compile Error File	N	N	Y	Y	N	Y	D	F	F	D	D						02/07/97	05/10/89
ID8SCVC2	ID8SCLC2	Compile IFDEF Defin	Y	Y	Y	Y	N	Y	D	F	E	D	D						02/21/97	06/18/90
ID8SCVD0	ID8SCLD0	Library File View	Y	Y	Y	Y	N	Y	D	F	E	N	D						03/05/97	09/19/90
ID8SCVF1	ID8SCLF1	Source Lock Detail:	Y	Y	Y	Y	N		D	F	F	N	D						08/31/93	08/26/93
ID8SCVF2	ID8SCLF2	Source Lock Detail:	Y	Y	Y	Y	N		D	F	F	N	D						08/30/93	08/26/93
ID8SCVF3	ID8SCLF3	Source Lock Detail,	Y	Y	Y	Y	N		D	F	E	N	D						08/26/93	08/26/93
ID8SCVS0	ID8SCLS0	Source File View	Y	Y	Y	Y	Y	Y	D	F	E	D	D						05/13/97	04/15/92
ID8SCVS1	ID8SCLS3	Currently Locked So	Y	Y	Y	Y	Y	Y	D	F	F	D	D						03/14/97	08/09/90
ID8SCVS2	ID8SCLS2	Backup Source File	Y	Y	Y	Y	N	Y	D	F	E	D	D	N					07/14/97	07/09/90

Selecting Actions

As the user moves from cell to cell in the view the actions available to the user change, depending on the pre-process directives defined in the relevant data element names.

Special Prompts are used to display available selections for the column. For more information, see Special Prompts later in this section.

On a character terminal this list is displayed on line 0 of the visual display, and options are chosen by the user pressing a function key.

On a graphical workstation special prompts are displayed in the status bar of the GUI Server window. Press the function keys or click the right mouse button for a table view of available CONNECTS.

Moving Around The View on a Character Terminal

While in a view the user can move horizontally from cell to cell. The shaded display area to the left of the vertical bar contains fixed columns. To the right of the bar, columns can be scrolled horizontally. Pressing the **Tab** key when the cursor is in the right-most column, scrolls to the right. Pressing **Back Tab** when in the column immediately to the right of the vertical bar scrolls to the left. Using the **Left Arrow** and **Right Arrow** keys moves the cursor left or right through the columns that are visible.

The **Home** key provides four levels of operation, depending on where you are in the view. At the first press the cursor is moved to the first column of the current line. At the second consecutive press it moves to the first cell on the current display. At the third consecutive press it moves to the first cell of the entire view. A fourth press goes to the end of the file, i.e., the last record of the view.

Vertical movement through the view can be done a line at a time using the **Up Arrow** and **Down Arrow** keys. **Page Up** and **Page Down** move the display by the number of lines in the window.

Changing a View

Certain attributes of a view can be modified while the view is displayed. All of the facilities shown in this section are available to developers. Users are prevented from using facilities marked below as developers.

Change Sort Sequence

Select **Sort (F5)** to display a list of the sort sequences available. When you have selected a sequence the view is redisplayed using that sequence.

Delete Column (Developers)

Press **Delete** to delete the current column from a view.

Insert Column (Developers)

Press **Insert** to insert a new column in the view, or to recover a deleted column. OPENworkshop displays a list of data element names for selection. Highlight the required name and press **Enter** to insert before, or press **right arrow** to insert after, the current column. If you wish to add a column that is not in the list provided, select a blank name (initially A, after A has been used B etc.). You will be asked to supply the link and data element name for the data to be displayed in the new column. The key to the file containing the proposed data element name must be in the current view.

Delete Row

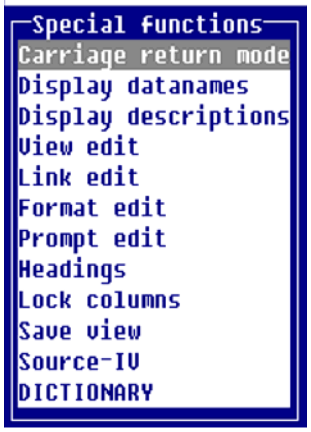
Press **Line Delete** to delete the current row. The data will be removed from the data file.

Insert Row

Press **Line Insert** to insert a new line before the current line.

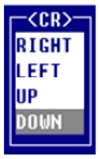
F7 Special Functions

Select **F7 Special functions** and the system displays the following:



Carriage return mode

Controls the direction the lightbar will move after field editing. Valid directions are right, left, up, and down.



Display datanames

Turns the data name display feature on and off. When on, the following information about the current column appears at the bottom of the view:

- Data element name.
- Maximum entry length of the field.
- Current width of the column.
- Carriage return direction mode (Enter key).

CUST-CODE	Data Name
(4)	Current column width
(8)	Maximum field entry length
<CR>=D	Carriage return mode

Display descriptions

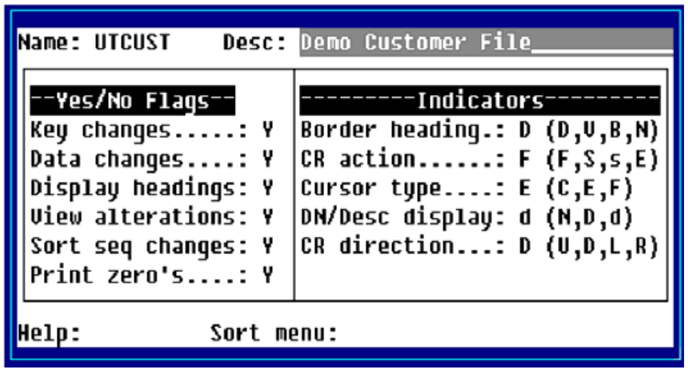
Turns the description display feature on and off. When on, the following information about the current column appears at the bottom of the view.

Data element description (defined in format).

Customer Code	Data element description.

View edit

Allows modifications to the view specifications. Each specification that can be modified will have **Help** available (**F6**). If access is denied to this function, see your system administrator for additional explanation.



```
Name: UTCUST    Desc: Demo Customer File

--Yes/No Flags--          ---------Indicators---------
Key changes.....: Y       Border heading.: D (D,V,B,N)
Data changes....: Y       CR action......: F (F,S,s,E)
Display headings: Y       Cursor type....: E (C,E,F)
View alterations: Y       DN/Desc display: d (N,D,d)
Sort seq changes: Y       CR direction...: D (U,D,L,R)
Print zero's....: Y

Help:         Sort menu:
```

Key changes

Controls the ability to add, delete, and modify records and to use **F8** CHANGE, DELETE, COPY, and MOVE commands in a view. Select one of the following:

Y	Allows changes to the key fields in the view.
N	Does not allow you to change the key fields.

Data changes

Controls the ability to add, delete, and modify records and to use **F8** CHANGE, DELETE, COPY, and MOVE commands in a view. Select one of the following:

Y	Allows changes to the non-key fields in the view.
N	Does not allow you to change the non-key fields.

Display headings

Turns on or off the view headings. Select one of the following:

Y	Displays the heading lines.
N	Does not display the heading lines.

View alterations
Controls the ability to delete or insert columns or to save a view. Select one of the following:

Y	Allows view specification changes.
N	Does not allow view specification changes.

Sort seq changes
Turn on or off the ability to use **F5** Sort to select a different sort sequence. Select one of the following:

Y	Allows view sort changes.
N	Does not allow view sort changes.

Print zeros
Display/print zeros or blanks for the numeric value zero. Select one of the following:

Y	Prints zeros (0) when the numeric data is zero (0).
N	Prints blanks when the numeric data is zero (0).

Border heading
Contains view window heading specifications. The headings always display in the top border of the view window. Select one of the following:

D	The view description is centered in the top border.
V	The eight-character view name is displayed in the left-most top border.
B	Both the eight-character view name and description are displayed in the top border.
N	No headings display in the top border.

CR Action
Defines the action of the **Enter** key while in the view. Select one of the following:

F	Enter the normal field edit mode.
S	Enter screen maintenance.
s	Enter screen maintenance and return to View after record is edited.
E	Exit the view.

Cursor type
Defines the view lightbar specifications. Select one of the following:

C	One character cursor marks the column/row intersection.
E	Full field lightbar marks the column/row intersection.
F	The full row is marked with the lightbar.

DN/Desc display
Displays the data names in the bottom border of the view flag. Select one of the following:

N	Do not display the data name or description.
D	Display the data name.
d	Display the data name description.

CR direction
Defines the direction of the lightbar movement after pressing the **Enter** key to modify a field. Select one of the following:

U	The lightbar goes up to the prior row.
D	The lightbar goes down to the next row.
L	The lightbar goes left to the previous column.
R	The lightbar goes right to the next column.

Help
Contains the name of the help module, which provides the text, to use for the view under the application help option.

Sort Menu
Contains the name of a menu to be used in place of the default selection pop-up menu. This menu can be more application specific than the default sort selection menu.

Example:

In a customer file you may decide to restrict usage of a sort on sales. You may create a custom sort menu that does not include this sales sort.

When defining a sort menu, you must return a valid sort number with each selection contained in the developer-supplied menu.

For example:

```
.MN.13,4,65,1
Customer Name      1
Salesrep           2
State              3
Zip Code           4
```

It also allows you to present the data in a more readable display.

View Method
Contains the name of a developer supplied method that will be called each time a view row is to be displayed. The method will build all new column values that have been defined by new columns A<x> through Z<x>. The view method is specified in the view header.

Link edit

Displays the Link Definition for this View. You may edit this definition and sorts. For more information see Link Definition in the Dictionary-IV Developer Guide.

If the system denies you access to this function see your system administrator for assistance.

Format edit

Displays the Format Definition for this View. You may edit this definition. For more information see Format Definition in the Dictionary-IV Developer Guide.

If the system denies you access to this function see your system administrator for assistance.

Prompt edit

Modifies the help text containing view/screen prompts that were referenced by VP[n] or SP[n]. For more information, see Format Definition in the Dictionary-IV Developer Guide.

Headings

Activates the heading edit mode. If headings have not been defined for a view, one blank line will be inserted and the entire view is shifted down. Once in heading edit mode, the **Arrow** and **Tab** keys can be used to select the desired column. You may type the desired heading values. Use the **Line Insert** key to add as many heading lines as desired.

To remove a heading line press **Line Delete**. Press **F4** to exit heading mode.

You may make the column width larger than the field length defined in the format. This does not allow you to change the length of the data entered in the column, but it does allow you to increase the width of the heading.

For example: If you have a 2-character State column, you may press **F7**, select Column width, and increase the size to 5. This still only allows 2-character data entry, but allows you a 5-character heading. You may now type the heading **State**.

Lock columns

Locks and unlocks columns to the left of and including the column on which the lightbar is located. A vertical line designates a locked column. These columns remain stationary when scrolling in the view.

If two columns were locked, the vertical line would be placed to the right of the Customer Name column. These two columns then remain stationary when scrolling in the view.

Save view

Saves the view. The save process preserves all view attributes including deleted and locked columns, reduced field sizes, sort pattern, and any view size. There is no prompt for a view name or confirmation of changes.

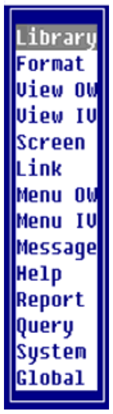
Source-IV

Displays the Source-IV menu and allows you to make selections. This function is only available if the environment is defined as an OPENworkshop Development environment.

For more information see the Source-IV and OPENworkshop reference manuals.

DICTIONARY

Displays the following menu:

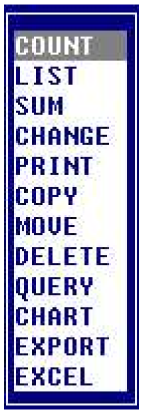


This function is only available if the environment is defined as an OPENworkshop development environment.

For more information about the Class Definitions see the Dictionary-IV Developer Guide.

F8 View Commands

Select **F8** while in a view to obtain a menu of commands that can be used on the view. These commands perform frequently required operations, saving the developer from the need to implement them.



Security

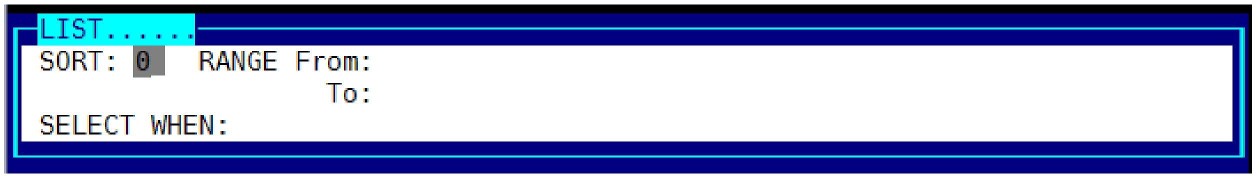
The developer can prevent access to specific commands by setting appropriate security groups in the default **F8** command menu OOM2.

Custom **F8** Command menus can be created in order to provide limited **F8** functionality when the user mode is set to USERS. This can be done at anytime by creating an OPENworkshop startup method and setting the global format #IDSV.VIEW-USR-CMD set to the name of the **F8** Command window to be displayed from a View. This is useful if you do not want your users to be able to access **F8** commands like COPY, MOVE, DELETE, SUM, COUNT. If #IDSV.VIEW-USR-CMD is not set the default menu OOM2 will be used.

For convenience a modified version of OOM2 named OOM2USR is distributed with OPENworkshop. This version of the command window limits options to LIST, PRINT, QUERY, CHART, EXPORT, and EXCEL.

SELECT

Many of the **F8** commands make use of the standard OPENworkshop SELECT window:



```
LIST......
SORT: 0   RANGE From:
                 To:
SELECT WHEN:
```

SORT: The Link sort number by which the requested command will be sequenced. **F2** will allow a valid sort definition to be selected from a pop-up menu.

RANGE From: The selected command will include the data starting with the RANGE From key value.

RANGE To: The selected command will include the data ending with the RANGE To key value.

SELECT WHEN: Select conditional expression. The conditional expression may contain any relational operators allowed in the IDOL-IV Script language.

F2 will allow valid data names to be selected from a pop-up menu.

F3 will allow the select statement to be edited in the Text Editor.

Any date mask valid for NTD() is valid in the SELECT statement with date type 8 fields.

Example select statements:

```
#FORMAT.DN1 = “XYZ”
 #FORMAT.DN2  > 10 AND #FORMAT.DN3 < 6
 CDN - #FORMAT.DUE-DATE  < 30
 #FORMAT.DUE-DATE(dd) = “01”
 #FORMAT.DUE-DATE(Mon) = “Jan”
 #FORMAT.DUE-DATE(Dy)=”Fri”
 #FORMAT.DUE-DATE(YYYY) = NTD(CDN,”YYYY”)
```

A secure SELECT WHEN clause provides the developer with the ability to control what records are displayed in the view while preventing the user from changing or seeing the secured SELECT WHEN. A secure SELECT WHEN is never displayed in the **F8** Selection. The user retains the ability to enter a SELECT WHEN clause. The user defined SELECT WHEN will be appended to the "hidden" secure SELECT WHEN clause.

To hide the SELECT clause from the user, enable the Secure Select When option by setting VIEW$[13](8,1)=”Y”.

For more information on the VIEW$[ALL] array, please see the online help: CONNECT HELP, CONNECT VIEW, VIEW MSG array.

Commands

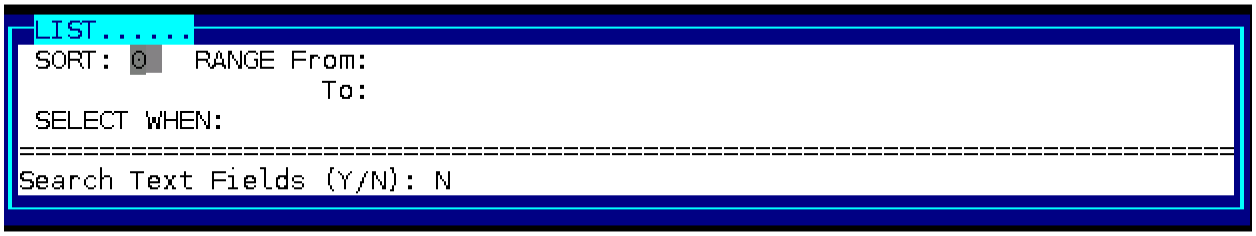
COUNT

Counts the rows in the view. This command allows the user to select a range of records and a SELECT WHEN rule to control what is counted.

LIST

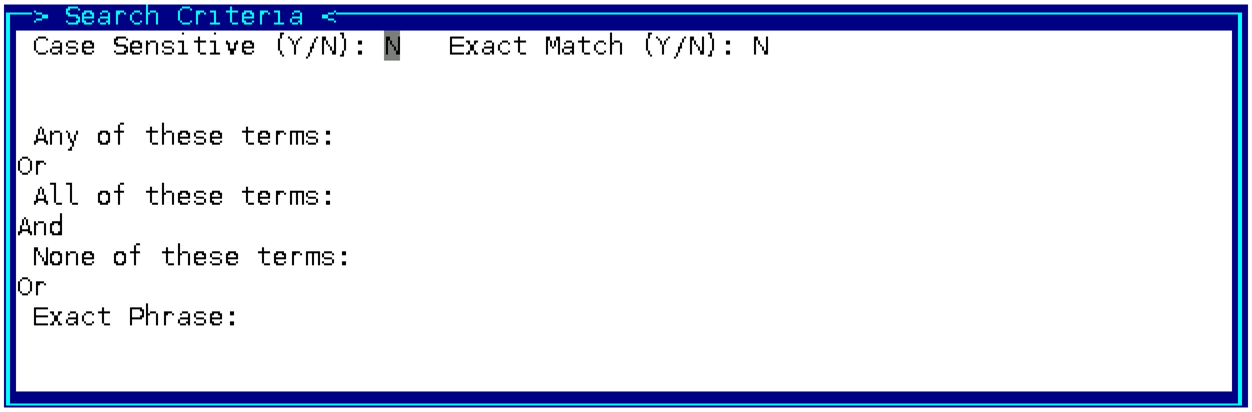
Re-lists rows in the view. The command allows the user to select a new sort sequence, a range of records and a SELECT WHEN rule to control what is listed. This command is useful when you temporarily want to restrict the number of rows displayed.

The LIST option also supports text field search rules to further refine the selection results. Only those records meeting all the selection criteria including the text field search will be included in the results. When the LIST option is selected the following SELECT window is displayed. This is only available with the LIST option.



```
LIST......
 SORT: 0   RANGE From:
                    To:
 SELECT WHEN:
==============================================================================
Search Text Fields (Y/N): N
```

Search Text Fields (Y/N): By default this option is set to “N”. Enter “Y” to enter text field search criteria.



```
-> Search Criteria <-
 Case Sensitive (Y/N): N   Exact Match (Y/N): N


 Any of these terms:
Or
 All of these terms:
And
 None of these terms:
Or
 Exact Phrase:
```

Case Sensitive (Y/N): By default the case sensitive option is not enabled. For example searching for the term "View" would return a result for all of the following words: "View", "VIEW", and "view". Enable the Case Sensitive option to return a result only for the term "VIEW", or only for the term "View", or only for the term "view".

Exact Match (Y/N): By default the exact match option is not enabled. Results are returned if the search term is found in any form within the text. It is equivalent to the wildcard *term*. For example searching for the term "View" will return a result for all of the following words: "View", "VIEW$", "view-name", and "8VIEWF". When the exact match option is enabled only the words "View" and "VIEW" will return a result.

Any of these terms: This search option uses the OR relational operator.

All of these terms: This search option uses the AND relational operator.

None of these terms: This search option uses the NOT relational operator.

Exact Phrase: This search option will only return a result when the exact phrase is matched. When searching for a phrase, do not specify All, Any, or Not terms

SUM

Sums values in a column. This command allows the columns to be summed to be restricted using a range and a SELECT WHEN rule.

CHANGE

Performs a bulk change on the data in a view, changing all rows that fit within the specified range and SELECT WHEN rule. As an option, the user is able to request confirmation on a row-by-row basis.

PRINT

Prints the rows in a view. The rows to be printed can be restricted using a range and SELECT WHEN rule. The sort sequence in which the rows are to be printed can be specified.

COPY

Copies data from the current view to another file. The rows to be copied can be restricted using a range and a SELECT WHEN rule. The user is able to ask to approve each row, or approve overwriting of existing rows in the file being copied to.

This command is a useful way to copy data from one file to another.

MOVE

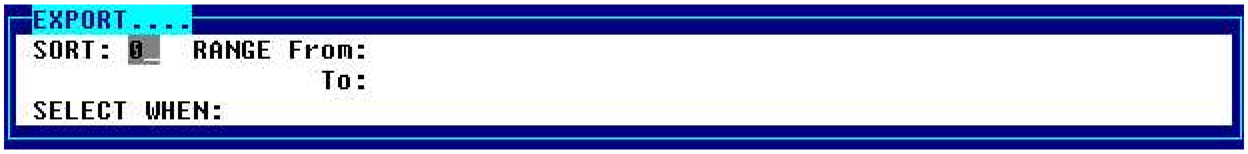
Moves data from the current view to another file. The same controls as the Copy command are provided, but moved data is deleted from the source view. Therefore, use this command with care.

DELETE

Deletes rows from a view and its underlying file. A range and the SELECT WHEN rule can restrict what is deleted. The user can ask to approve deletion of each row.

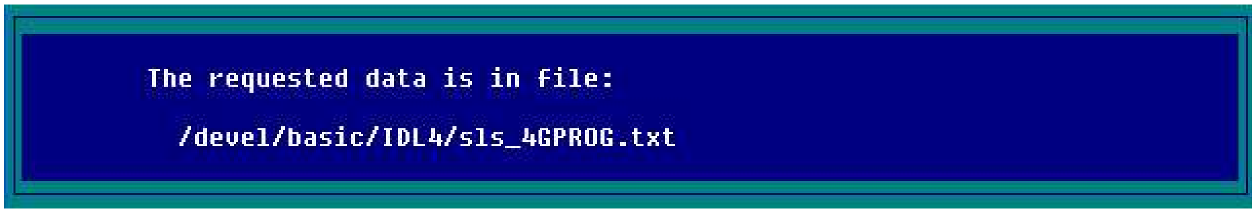
EXPORT

This option exports the data currently displayed in the current View to a .txt file. The file is created in the default data file directory defined by #IDSV.DISK.NBR-DATAFILES.



```
EXPORT....
SORT: 0_  RANGE From:
                  To:
SELECT WHEN:
```

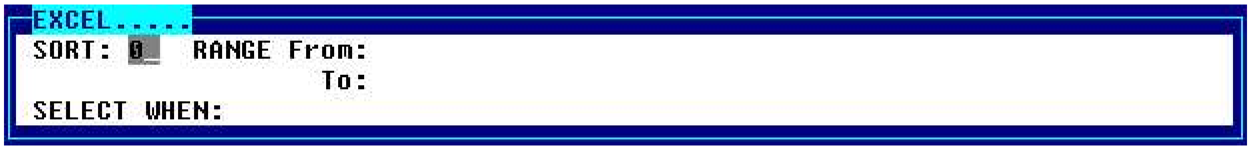
When you have selected the range press Enter and you will see the following window. Note SORT0 is the default sort.



```
The requested data is in file:

  /devel/basic/IDL4/sls_4GPROG.txt
```

EXCEL

This option exports the data displayed in the current View to an Excel spreadsheet on the Windows workstation using Thoroughbred's Gateway for Windows product. The data to be exported can be refined by specifying a SORT, RANGE From/To, and a SELECT WHEN clause.



```
EXCEL.....
SORT: 0_   RANGE From:
                  To:
SELECT WHEN:
```

Excel will be launched and the spreadsheet will be populated with data from the View based on the SORT, RANGE From/To, and SELECT WHEN clause.

A new Excel workbook is created each time Excel is selected for output. To switch between workbooks select the appropriate Microsoft Excel Book in the task bar. Or from the Excel menubar select Window and then select the appropriate Book (window).

NOTE: A secure SELECT WHEN clause in VIEW$[7] provides the developer with the ability to control what records are displayed in the view while preventing the user from changing or seeing the secured SELECT WHEN. A secure SELECT WHEN is never displayed in the **F8** Selection.

The user retains the ability to enter a SELECT WHEN clause. The user defined SELECT WHEN will be appended to the "hidden" secure SELECT WHEN clause.

The secure select clause is enabled by setting VIEW$[13](8,1)="Y".

3GL Example:

```
     DIM VIEW$[13];
     VIEW$[1]="OEVCUST",
     VIEW$[7]="SELECT WHEN #OEFCUST.SR-CODE ="+QUO+"HP"+QUO,
     VIEW$[13]="       Y";
     CALL "OO3A",VIEW$[ALL]
```

4GL Example:

```
     DIM VIEW$[13]
     LET VIEW$[1]="OEVCUST",
         VIEW$[7]="SELECT WHEN #OEFCUST.SR-CODE ="+QUO+"HP"+QUO,
         VIEW$[13]="       Y"
     CONNECT VIEW "OEVCUST"
```

For more information on the VIEW$[ALL] array, please see the online help: CONNECT HELP, CONNECT VIEW, VIEW MSG array.

F8 Functionality for USERS

Create a custom **F8** Command menu in order to provide limited **F8** functionality when the user mode is set to USERS.

This can be done at anytime by creating an OPENworkshop startup method and setting the global format #IDSV.VIEW-USR-CMD set to the name of the **F8** Command window to be displayed from a View. This is useful if you do not want your users to be able to access **F8** commands like COPY, MOVE, DELETE, SUM, COUNT.

If #IDSV.VIEW-USR-CMD is not set the default menu OOM2 will be used.

For convenience a modified version of OOM2 named OOM2USR is distributed with OPENworkshop. This version of the command window limits options to LIST, PRINT, QUERY, CHART, EXPORT, and EXCEL.

Single Record Maintenance (F9)

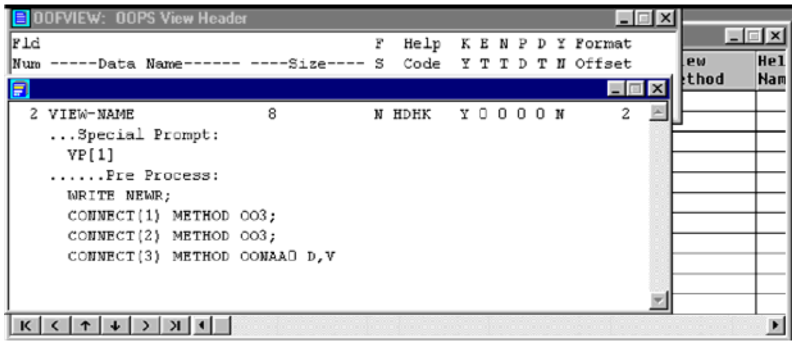
Select **Single Record Maintenance (F9)** to display a screen showing the current row in more detail. The screen that will be displayed will normally be defined in the link definition. If a screen was not specified in the link definition, a default screen will be generated.

Press **F9** from a screen to display a view starting with the current record. The view that will be displayed will normally be defined in the link definition. If a view was not specified in the link definition, a screen will be displayed that allows a link name to be entered. The view will be created from the link name entered.

Format Editor (F11)

The data element name definition is displayed within a text editor that can be edited using the editor. Attributes are shown in the first line of the file, and succeeding lines display directives that define Valid Values, messages, pre- and post-processing directives and other definitions for the data element name.

Select **F11** to display the following window:

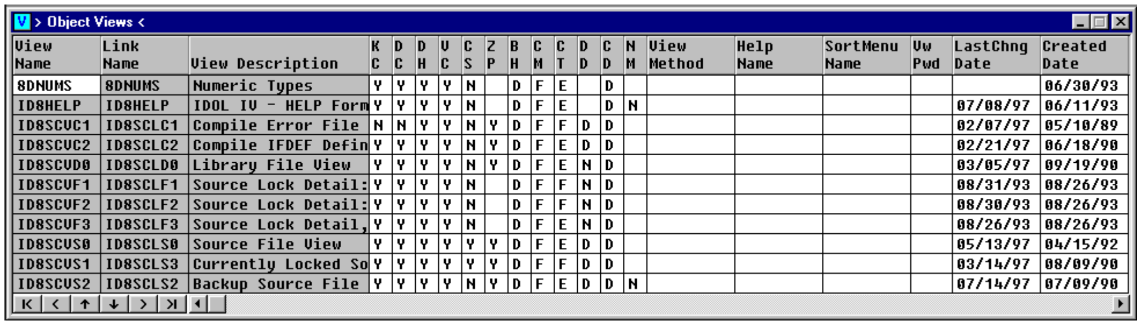


```
Fld                                        F  Help   K E N P D Y Format
Num -----Data Name------ ----Size----      S  Code   Y T T D T N Offset
  2 VIEW-NAME              8               N HDHK    Y 0 0 0 0 N      2
    ...Special Prompt:
      VP[1]
    ......Pre Process:
      WRITE NEWR;
      CONNECT(1) METHOD 003;
      CONNECT(2) METHOD 003;
      CONNECT(3) METHOD OONAAD D,V
```

This definition contains directives organized into sections. The editor assists in creating headers for these sections, and handles indentation.

Create/Maintain View

To create a new view or maintain existing views move to the View Header, as shown below:



View Name	Link Name	View Description	K C	D C	D H	V C	C S	Z P	B H	C M	C T	D D	C D	N M	View Method	Help Name	SortMenu Name	Vw Pwd	LastChng Date	Created Date
8DNUMS	8DNUMS	Numeric Types	Y	Y	Y	Y	N		D	F	E		D							06/30/93
ID8HELP	ID8HELP	IDOL IV - HELP Form	Y	Y	Y	Y	N		D	F	E		D	N					07/08/97	06/11/93
ID8SCVC1	ID8SCLC1	Compile Error File	N	N	Y	Y	N	Y	D	F	F	D	D						02/07/97	05/10/89
ID8SCVC2	ID8SCLC2	Compile IFDEF Defin	Y	Y	Y	Y	N	Y	D	F	E	D	D						02/21/97	06/18/90
ID8SCVD0	ID8SCLD0	Library File View	Y	Y	Y	Y	N	Y	D	F	E	N	D						03/05/97	09/19/90
ID8SCVF1	ID8SCLF1	Source Lock Detail:	Y	Y	Y	Y	N		D	F	F	N	D						08/31/93	08/26/93
ID8SCVF2	ID8SCLF2	Source Lock Detail:	Y	Y	Y	Y	N		D	F	F	N	D						08/30/93	08/26/93
ID8SCVF3	ID8SCLF3	Source Lock Detail,	Y	Y	Y	Y	N		D	F	E	N	D						08/26/93	08/26/93
ID8SCVS0	ID8SCLS0	Source File View	Y	Y	Y	Y	Y	Y	D	F	E	D	D						05/13/97	04/15/92
ID8SCVS1	ID8SCLS3	Currently Locked So	Y	Y	Y	Y	Y	Y	D	F	F	D	D						03/14/97	08/09/90
ID8SCVS2	ID8SCLS2	Backup Source File	Y	Y	Y	Y	N	Y	D	F	E	D	D	N					07/14/97	07/09/90

To edit an existing View definition, position the cursor to the required row and select **Edit (F1).** To change the name of, or to copy an existing view, type the new name and choose the option required from the succeeding menu. To create a definition of a new view select **Line Insert**, type a name for the new screen, then with the highlight on the name field select **Edit (F1)** to edit the new definition.

OPENworkshop maintains the following objects and attributes for a view:

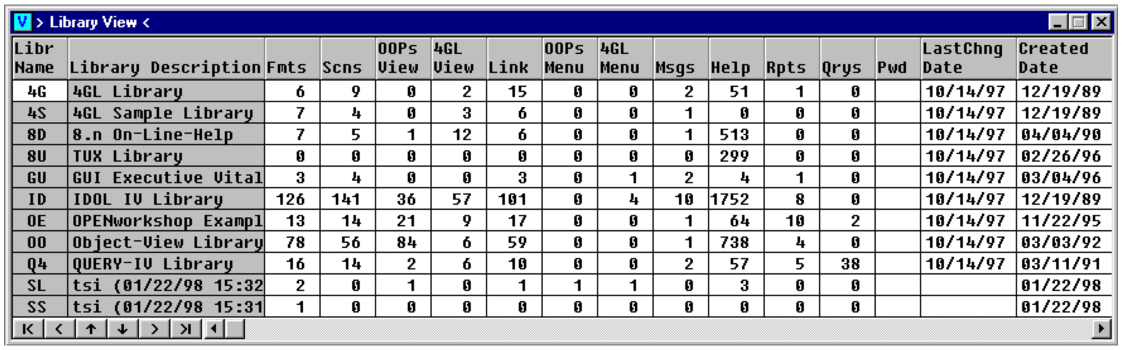
Column	**Attribute**
Link Name	The link used by the view to locate the required data for the view.
KC	Key Change. Value Y allows the key data to be changed by the user. N prevents change.
DC	Data Change. Value Y allows non-key data in the view to be changed by the user. N prevents change.
DH	Display Headings. Value Y causes the view heading (column headings) to be displayed. N switches them off.
VC	View Changes. Value Y allows the specification of the view to be changed by the user. N prevents change.
CS	Change Sort. Value Y allows the sort used for data displayed in the view to be changed. N prevents change.
ZP	Zero Print. Where a data item is zero is can be displayed in the view as a "0" character (ZP = Y), or as a blank (ZP = N).
BH	Border Heading. An eight-character view name is displayed in the left-most top border (BH = V), the view description is centered in the top border (BH = D), Both the eight character view name and description are displayed in the top border (BH = B), or No headings display in the top border (BH = N).
CM	CR Mode. Controls the behavior of the view when the user presses **Enter**. Value F: enter Field Edit. S: enter associated Screen. E: exit and return the key value for the current row to the caller. See description of the VIEW$ array. See the VIEW$ section of this manual.
CT	Cursor Type. Value C displays a single-character cursor. E: highlight the current field. F: highlight to full row.
DD	Display Data Element Name. The name is displayed at the bottom left corner of the border. D: display the data element name. d: display the data element name description. N: none.
CD	CR Direction. After user presses **Enter**, move to the next cell D: Down, L: Left, R: Right, U: Up.
View Method	Called whenever a view has read data for a row and is preparing a row for display. The view method ensures that all required data items, including any calculated columns, are available. See the OPENworkshop Methods section of this manual for more information about view methods.
Help Name	Specifies the help file that should be used to offer context-sensitive help for this view.
Sort Menu Name	Name of a menu that may be used to help the user select a sort sequence in which to display the view.
View Password	If security requirements mean that a password should be used to be able to display the view the required password is specified in this field.

Data Classes

Library

A library collects all of the classes relevant to an application or subsystem. A library name is two alphanumeric characters, which are used to preface all names of classes in the library.

You can explore the library by moving the cursor from cell to cell in the Library Header, an example of which is shown below:



> Library View <

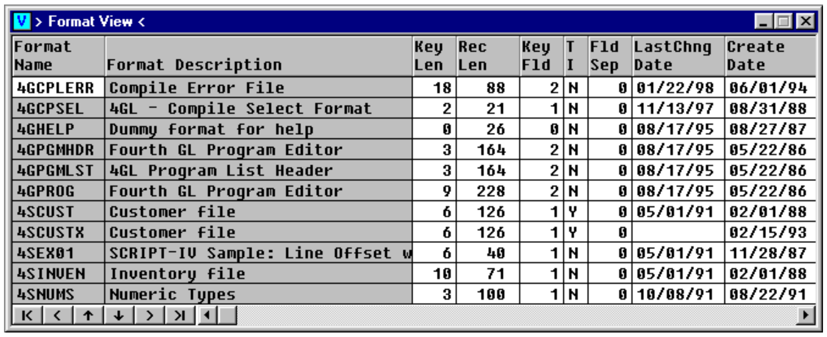
Libr Name	Library Description	Fmts	Scns	OOPs View	4GL View	Link	OOPs Menu	4GL Menu	Msgs	Help	Rpts	Qrys	Pwd	LastChng Date	Created Date
4G	4GL Library	6	9	0	2	15	0	0	2	51	1	0		10/14/97	12/19/89
4S	4GL Sample Library	7	4	0	3	6	0	0	1	0	0	0		10/14/97	12/19/89
8D	8.n On-Line-Help	7	5	1	12	6	0	0	1	513	0	0		10/14/97	04/04/90
8U	TUX Library	0	0	0	0	0	0	0	0	299	0	0		10/14/97	02/26/96
GU	GUI Executive Vital	3	4	0	0	3	0	1	2	4	1	0		10/14/97	03/04/96
ID	IDOL IV Library	126	141	36	57	101	0	4	10	1752	8	0		10/14/97	12/19/89
OE	OPENworkshop Exampl	13	14	21	9	17	0	0	1	64	10	2		10/14/97	11/22/95
OO	Object-View Library	78	56	84	6	59	0	0	1	738	4	0		10/14/97	03/03/92
Q4	QUERY-IV Library	16	14	2	6	10	0	0	2	57	5	38		10/14/97	03/11/91
SL	tsi (01/22/98 15:32	2	0	1	0	1	1	1	0	3	0	0			01/22/98
SS	tsi (01/22/98 15:31	1	0	0	0	0	0	0	0	0	0	0			01/22/98

Select **View Library Formats** to display the Format Header view, which lists all formats defined for the library. **View All Formats** displays all formats known to the system. **Print** prints some or all the formats in the library.

Format

A format collects a set of data element names that will be stored in a single file in the OPENworkshop file system or that make a logical group for other processing reasons.

To maintain an existing format or to create a new one move to the Format Header view as shown below:

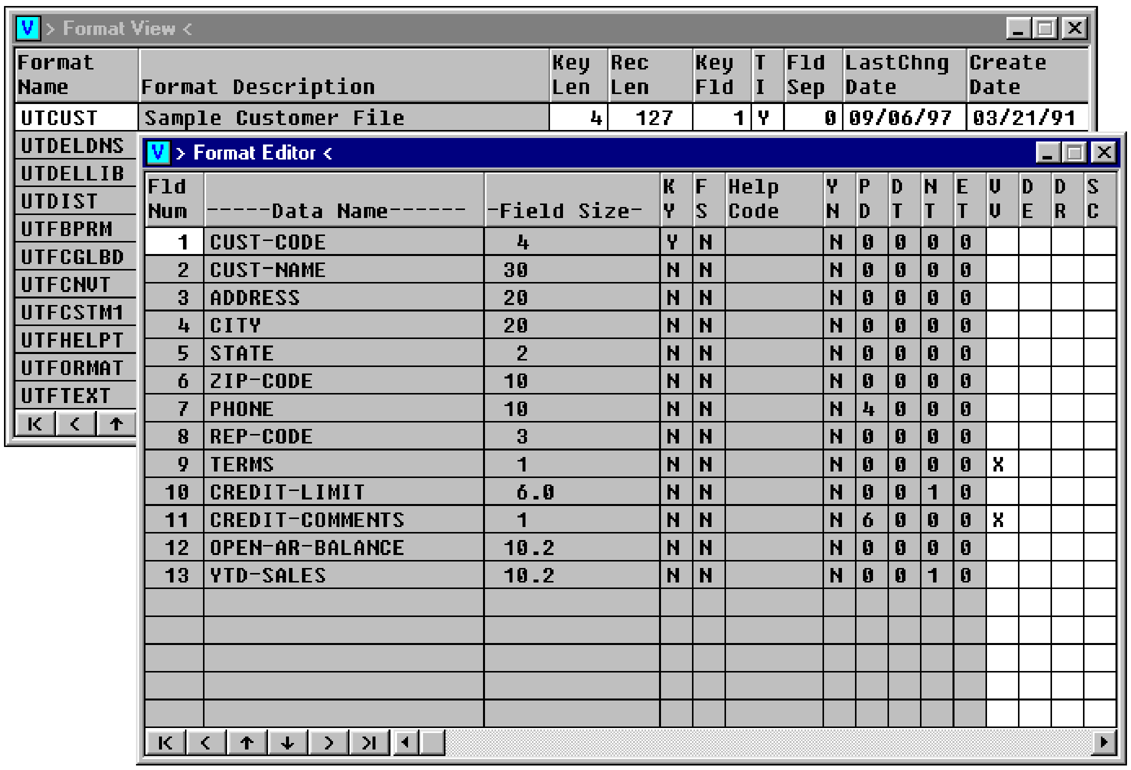


> Format View <

Format Name	Format Description	Key Len	Rec Len	Key Fld	T I	Fld Sep	LastChng Date	Create Date
4GCPLERR	Compile Error File	18	88	2	N	0	01/22/98	06/01/94
4GCPSEL	4GL - Compile Select Format	2	21	1	N	0	11/13/97	08/31/88
4GHELP	Dummy format for help	0	26	0	N	0	08/17/95	08/27/87
4GPGMHDR	Fourth GL Program Editor	3	164	2	N	0	08/17/95	05/22/86
4GPGMLST	4GL Program List Header	3	164	2	N	0	08/17/95	05/22/86
4GPROG	Fourth GL Program Editor	9	228	2	N	0	08/17/95	05/22/86
4SCUST	Customer file	6	126	1	Y	0	05/01/91	02/01/88
4SCUSTX	Customer file	6	126	1	Y	0		02/15/93
4SEX01	SCRIPT-IV Sample: Line Offset w	6	40	1	N	0	05/01/91	11/28/87
4SINVEN	Inventory file	10	71	1	N	0	05/01/91	02/01/88
4SNUMS	Numeric Types	3	100	1	N	0	10/08/91	08/22/91

Select **Line Insert** to add a new format. OPENworkshop creates a new record and waits for you to enter the format name and description. To delete a format select **Line Delete**.

To modify a format definition, select **Edit (F1)** to display the Format Maintenance screen:



Format Name	Format Description	Key Len	Rec Len	Key Fld	T I	Fld Sep	LastChng Date	Create Date
UTCUST	Sample Customer File	4	127	1	Y	0	09/06/97	03/21/91
UTDELDNS								
UTDELLIB								
UTDIST								
UTFBPRM								
UTFCGLBD								
UTFCNVT								
UTFCSTM1								
UTFHELPT								
UTFORMAT								
UTFTEXT								

Fld Num	-----Data Name------	-Field Size-	K Y	F S	Help Code	Y N	P D	D T	N T	E T	V V	D E	D R	S C
1	CUST-CODE	4	Y	N		N	0	0	0	0				
2	CUST-NAME	30	N	N		N	0	0	0	0				
3	ADDRESS	20	N	N		N	0	0	0	0				
4	CITY	20	N	N		N	0	0	0	0				
5	STATE	2	N	N		N	0	0	0	0				
6	ZIP-CODE	10	N	N		N	0	0	0	0				
7	PHONE	10	N	N		N	4	0	0	0				
8	REP-CODE	3	N	N		N	0	0	0	0				
9	TERMS	1	N	N		N	0	0	0	0	X			
10	CREDIT-LIMIT	6.0	N	N		N	0	0	1	0				
11	CREDIT-COMMENTS	1	N	N		N	6	0	0	0	X			
12	OPEN-AR-BALANCE	10.2	N	N		N	0	0	0	0				
13	YTD-SALES	10.2	N	N		N	0	0	1	0				

To change an existing data element name move the cursor to the required data element name using **Up Arrow** and **Down Arrow** keys, then select **Edit (F1).** To insert a new data element name move to the next blank space and type the required name. Data element names can be reordered in the format by selecting **Move**.

For more information on maintaining data element names within a format, see the following subsection. The Dictionary-IV Developer Guide also provides extensive information.

Data Element Names

A data element name defines an item of information, together with its attributes. A data element name definition also specifies methods to be used whenever the data item is created, displayed, or amended. The attributes, directives, and methods associated with a data element name are defined in a format.

A data element name defines attributes of the data item, such as length, data type, padding, etc. In addition, it defines directives that control the dynamic behavior of the application when the data element name is used. Please see on-line help or the Dictionary-IV Developer Guide for an explanation of features not covered in this section.

Valid Values

As in Dictionary-IV, OPENworkshop will validate user input to a field against values in a list, a range, a text field, or a lookup file. For more information, see the Dictionary-IV Developer Guide.

See the section on Other Directives later in this section for the full syntax available for Valid Values.

In addition to capabilities provided in Dictionary-IV, OPENworkshop supports the following directives:

LOOKUP VIEW

Compares the input with key values in the specified view. If the value is present it is accepted, otherwise the view is opened and a valid value can be selected by scrolling in the view. For more information see the section on LOOKUP VIEW later in this section.

LOOKUP SCREEN

Specifies a screen that should be displayed when the cursor is in data entry mode and **Select Screen (F9)** is selected. Typically this screen will be used to display full information on the item. For more information see the section on LOOKUP SCREEN later in this section.

Default Values (Preset Values)

Defines a default value that is displayed when the data element name is accessed.

Delete Record

You can specify that a record may only be deleted if the data element name contains a specified value or condition.

Security

OPENworkshop extends the security facilities of earlier releases of Dictionary-IV. Users may belong to one or more security groups. A data element name can be specified only to allow access to members of one or more security groups.

To specify that a data element name should apply security group controls, enter the group or groups allowed access, surrounded by square brackets, in the security field.

Example

```
[02,31]
```

Specifies that only members of security groups 02 and 31 may access the information in this data element name. This level of security overrides all others described in the Dictionary-IV Reference Manual, but can be used in conjunction with the previously released features as well.

Special Prompts

OPENworkshop displays prompts to users as they move between data element names in a screen or view. In a graphical view the prompt is displayed in the help panel of the view. On a character terminal the prompt is always displayed on the first line of the screen.

These prompts are maintained in the dictionary as a help definition, all special prompts for a format are grouped in a single help text definition. Each prompt takes one line in the help definition, and prompts are indexed by line number. The help definition that contains these prompts has the same name as the format.

The special prompts field of the data element name definition contains the index to the prompt required for this data element name. Each data element name may have a view prompt and a screen prompt.

If a screen prompt is specified, it is displayed whenever the cursor is in the data element name in data entry mode, i.e., in a screen or after data entry mode has been entered in a view.

If a view prompt is specified, it is displayed whenever the cursor is on the data element name in cursor mode. Additionally, if no screen prompt is specified, the view prompt is displayed when in data entry mode.

In the Special Prompts or Message field of a data element name

```
VP[1];
SP[6]
```

specifies VP[1] of the Special Prompts help for the relevant format that contains the view prompt and line 6 contains the screen prompt. The Special Prompts help file can be edited by selecting that option when editing the data element name definition.

Note: For more information, see the section on Defining special prompts later in this section.

Pre-Process

OPENworkshop evaluates and executes any pre-process directives after a data element name has been read and before data is input in a view or screen. Specifically, the pre-process is activated:

- In a view, when the operator presses any alphanumeric or punctuation key or presses **Enter** to choose data entry mode.
- In a screen, when the cursor moves to a field.

These directives can be used to prepare the data element name for display, or to offer the user choices in the functions to perform. Note that a view method can also be specified for a view to prepare a row for display after a row has been read. For more information, see the OPENworkshop Classes section of this manual.

In addition to facilities offered by Dictionary-IV, OPENworkshop supports the following directives:

INSERT METHOD

Specifies a method to be used to prepare a new row in a view after it has been created using **Line Insert**. An insert method can only be specified in the first data element name of the primary key of a format. This directive is available in OPENworkshop only. For more information, see the OPENworkshop Methods section of this manual.

CONNECT

Passes control to a method, view, screen, help, menu, report or query. In most cases, the CONNECT directive specifies a function key that the user must press to make the connection. This directive is available in OPENworkshop and Dictionary-IV.

The following directives are only executed by a view, and only after a CONNECT has been selected and before it is executed.

WRITE MODR

Rewrites a modified row in a view before executing a CONNECT. This directive is available in OPENworkshop and Dictionary-IV.

WRITE NEWR

Writes the row when a new row has been inserted in a view before executing a CONNECT. This directive is available in OPENworkshop and Dictionary-IV.

INDENTV

Forces the next view to be displayed indented relative to the current view. This directive is available in OPENworkshop and Dictionary-IV.

*GOTO **

Scrolls the current row to the first row of a view before any CONNECT is executed. This directive is available in OPENworkshop and Dictionary-IV.

Post-Process

OPENworkshop evaluates and executes post-process directives in a view or screen when the user exits from data entry mode:

- In a view, when the user is in data entry mode and presses **Enter**, or a cursor movement key.
- In a screen, when the cursor moves out of the field.

In addition to the facilities offered by Dictionary-IV, OPENworkshop supports the following directive:

CONNECT METHOD

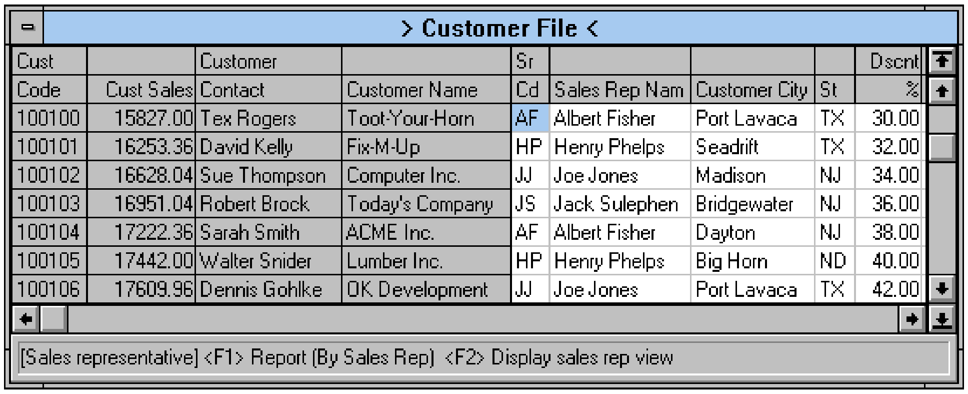
Executes the specified method after data input is complete in a screen or a view. If the data element name is displayed in a view, the method may return post-processing directives that will be honored by the view before control is returned to the user. Typically these directives update the displayed row. For more information see the section on Method Return Directives later in this section.

Audit

OPENworkshop retains the audit capabilities of Dictionary-IV, allowing an audit trail of data changes to be maintained. For more information, see the Dictionary-IV Developer Guide.

Example Data Element Name Definitions

The first set of examples is based on the following view, which is taken from the sample application shipped with OPENworkshop.

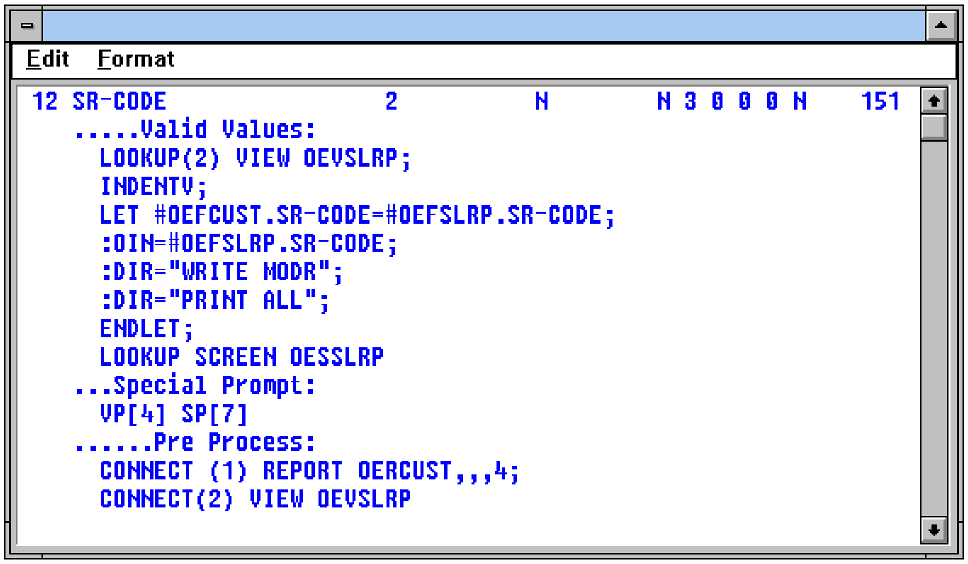


> Customer File <

Cust Code	Cust Sales	Customer Contact	Customer Name	Sr Cd	Sales Rep Nam	Customer City	St	Dscnt %
100100	15827.00	Tex Rogers	Toot-Your-Horn	AF	Albert Fisher	Port Lavaca	TX	30.00
100101	16253.36	David Kelly	Fix-M-Up	HP	Henry Phelps	Seadrift	TX	32.00
100102	16628.04	Sue Thompson	Computer Inc.	JJ	Joe Jones	Madison	NJ	34.00
100103	16951.04	Robert Brock	Today's Company	JS	Jack Sulephen	Bridgewater	NJ	36.00
100104	17222.36	Sarah Smith	ACME Inc.	AF	Albert Fisher	Dayton	NJ	38.00
100105	17442.00	Walter Snider	Lumber Inc.	HP	Henry Phelps	Big Horn	ND	40.00
100106	17609.96	Dennis Gohlke	OK Development	JJ	Joe Jones	Port Lavaca	TX	42.00

[Sales representative] <F1> Report (By Sales Rep) <F2> Display sales rep view

The following example shows the data element name definition for the Sales Rep column.

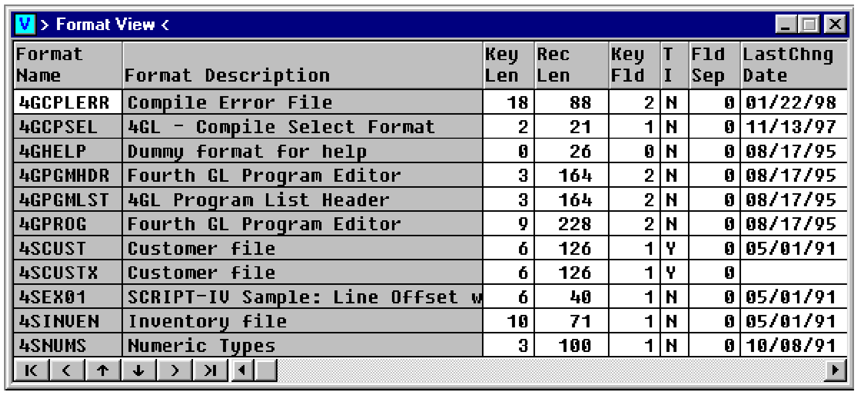


```
12 SR-CODE                2          N        N 3 0 0 0 N    151
   .....Valid Values:
     LOOKUP(2) VIEW OEVSLRP;
     INDENTV;
     LET #OEFCUST.SR-CODE=#OEFSLRP.SR-CODE;
     :OIN=#OEFSLRP.SR-CODE;
     :DIR="WRITE MODR";
     :DIR="PRINT ALL";
     ENDLET;
     LOOKUP SCREEN OESSLRP
   ...Special Prompt:
     VP[4] SP[7]
   ......Pre Process:
     CONNECT (1) REPORT OERCUST,,,4;
     CONNECT(2) VIEW OEVSLRP
```

The Valid Values section validates that data entered is a correct and current code then, if so, accepts the data and reprints the row. Pre-process directives allow **F1** and **F2** to trigger a report and connect to a view.

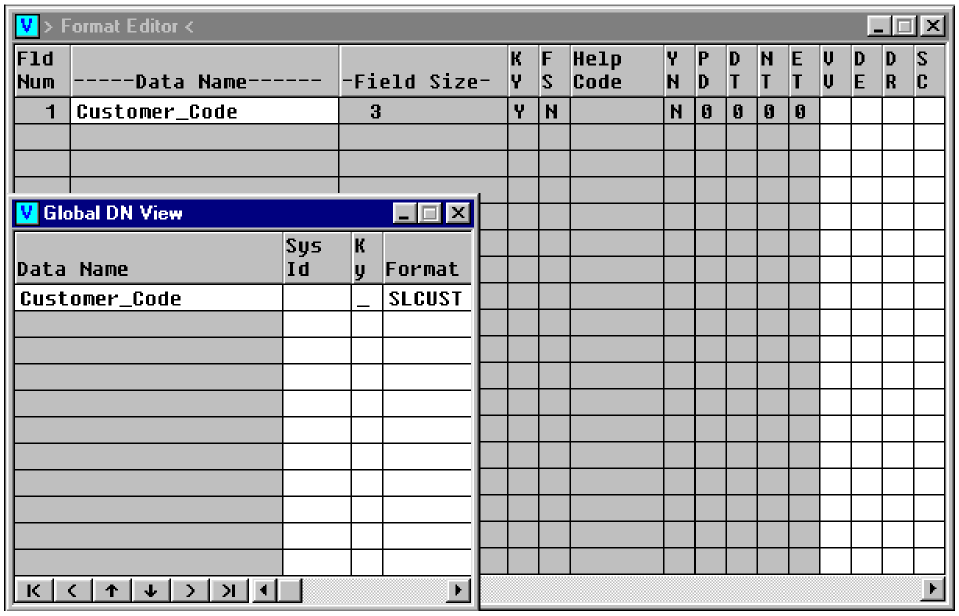
Defining and Maintaining Data Element Names

OPENworkshop provides two routes for data element name maintenance. The Format Maintenance screen, shown below, is used to create the data element name. See the Format section earlier in this manual for how to access the Format Maintenance screen.



> Format View <

Format Name	Format Description	Key Len	Rec Len	Key Fld	T I	Fld Sep	LastChng Date
4GCPLERR	Compile Error File	18	88	2	N	0	01/22/98
4GCPSEL	4GL - Compile Select Format	2	21	1	N	0	11/13/97
4GHELP	Dummy format for help	0	26	0	N	0	08/17/95
4GPGMHDR	Fourth GL Program Editor	3	164	2	N	0	08/17/95
4GPGMLST	4GL Program List Header	3	164	2	N	0	08/17/95
4GPROG	Fourth GL Program Editor	9	228	2	N	0	08/17/95
4SCUST	Customer file	6	126	1	Y	0	05/01/91
4SCUSTX	Customer file	6	126	1	Y	0	
4SEX01	SCRIPT-IV Sample: Line Offset w	6	40	1	N	0	05/01/91
4SINVEN	Inventory file	10	71	1	N	0	05/01/91
4SNUMS	Numeric Types	3	100	1	N	0	10/08/91



> Format Editor <

Fld Num	-----Data Name------	-Field Size-	K Y	F S	Help Code	Y N	P D	D T	N T	E T	V V	D E	D R	S C
1	Customer_Code	3	Y	N		N	0	0	0	0				

Global DN View

Data Name	Sys Id	K y	Format
Customer_Code		_	SLCUST

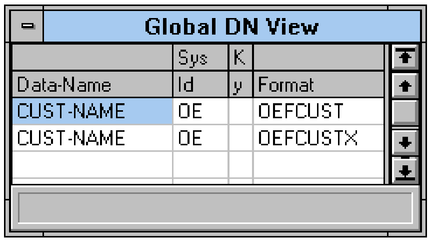
The cursor is positioned at the first data element name field. Type the first data element name required to commence the definition.

If the data element name you supply has already been used OPENworkshop will display a view that allows you to examine the existing definition and use it. You can also press **F8** to display a view of all defined data element names. Data element names can be selected from this view.

Options available when pressing **F2**:

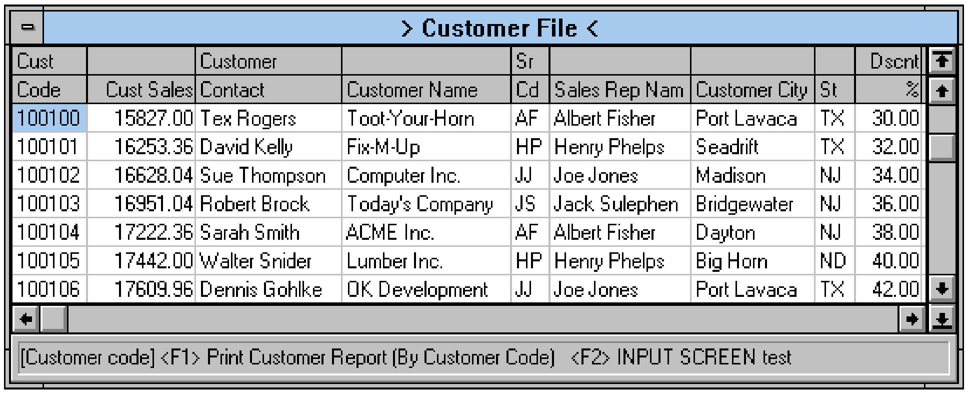
Display Data Name Detail	Shows detail of the data element name the cursor is located on.
Global System Id VIEW	Displays the Global System Definition. For more information, see the Global Cross Reference System section of this manual.
Dataname Where Used (ALL)	Shows the data element name used in the system where format is defined in the global dictionary.
Dataname VIEW all systems	Shows a view of all data element names for all systems.

Either select the existing definition or, if this is a new data element name, enter new attributes and directives in the fields provided. For more information on the attributes available for data element name definitions, see the Dictionary-IV Developer Guide.



Data-Name	Sys Id	Ky	Format
CUST-NAME	OE		OEFCUST
CUST-NAME	OE		OEFCUSTX

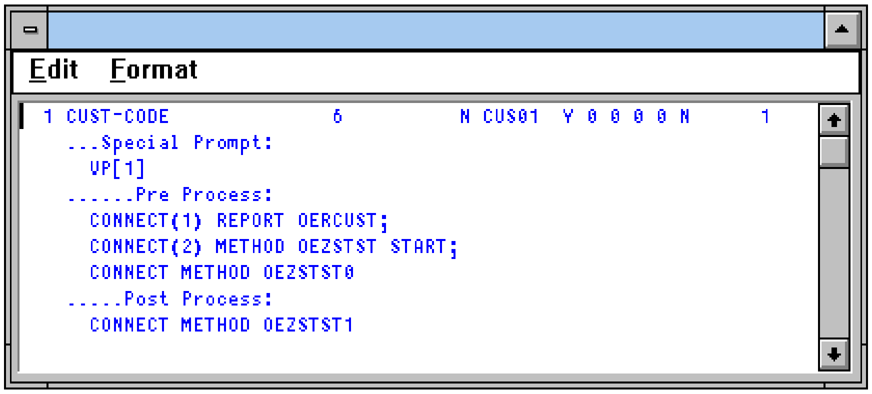
To maintain an existing data element name definition you can use the screen shown above, but OPENworkshop provides a convenient alternative. This option is available whenever the cursor is in the required data element name field in a view or screen. For example, in the following view select **Data Name Edit (F11)**.



Cust Code	Cust Sales	Customer Contact	Customer Name	Sr Cd	Sales Rep Nam	Customer City	St	Dscnt %
100100	15827.00	Tex Rogers	Toot-Your-Horn	AF	Albert Fisher	Port Lavaca	TX	30.00
100101	16253.36	David Kelly	Fix-M-Up	HP	Henry Phelps	Seadrift	TX	32.00
100102	16628.04	Sue Thompson	Computer Inc.	JJ	Joe Jones	Madison	NJ	34.00
100103	16951.04	Robert Brock	Today's Company	JS	Jack Sulephen	Bridgewater	NJ	36.00
100104	17222.36	Sarah Smith	ACME Inc.	AF	Albert Fisher	Dayton	NJ	38.00
100105	17442.00	Walter Snider	Lumber Inc.	HP	Henry Phelps	Big Horn	ND	40.00
100106	17609.96	Dennis Gohlke	OK Development	JJ	Joe Jones	Port Lavaca	TX	42.00

[Customer code] <F1> Print Customer Report (By Customer Code)   <F2> INPUT SCREEN test

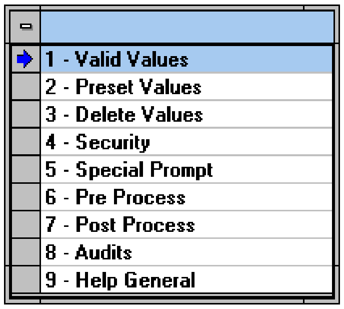
The data element name definition is displayed within a text editor that can be edited using the editor. Attributes are shown in the first line of the file, and succeeding lines display directives that define Valid Values, messages, pre- and post-processing directives and other definitions for the data element name.



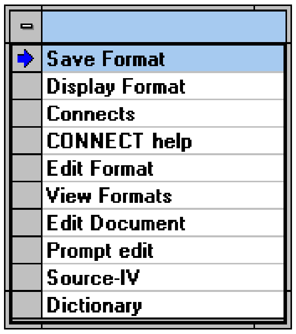
```
1 CUST-CODE           6          N CUS01  Y 0 0 0 0 N      1
  ...Special Prompt:
    VP[1]
  ......Pre Process:
    CONNECT(1) REPORT OERCUST;
    CONNECT(2) METHOD OEZSTST START;
    CONNECT METHOD OEZSTST0
  .....Post Process:
    CONNECT METHOD OEZSTST1
```

This definition contains directives organized into sections. The editor assists in creating headers for these sections, and handles indentation.

If you wish to add a new section into a data element name definition, position the cursor at the point you wish to insert the new section, then select **Insert Section (F11)**. The resulting menu, shown here, is used to select the type of section required.

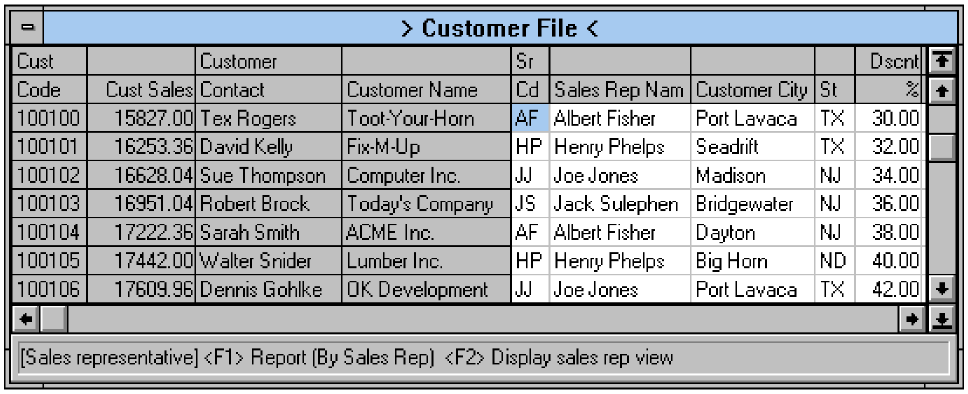


While you are editing a data element name definition a number of useful functions are available. Select **Special Functions (F7)** to display this menu.



Function	**Description**
Save Format	Saves the data element name definition as it currently is defined. This must be used, rather than **Close,** to save your work.
Display Format	Displays the definition of the entire format within the editor. However, no changes may be saved after this mode has been entered.
Connects	Displays only the CONNECT statements in the data element name definition, hiding other details.
Connect Help	Displays extensive help on the use of directives in data element names.
Edit Format	Enters format edit. See the Format subsection in this section.
View Formats	Displays and enters the Format Header view.
Edit Document	Opens the named document in edit mode.
Prompt Edit	Opens the special prompt file for the data element name in edit mode, allowing the prompts to be maintained. See the example following this table.
Source-IV	Enters Source-IV
Dictionary	Enters the main OPENworkshop dictionary menu.

The following figure shows a sample view with the special prompt displayed for the Sales Representative Code column:

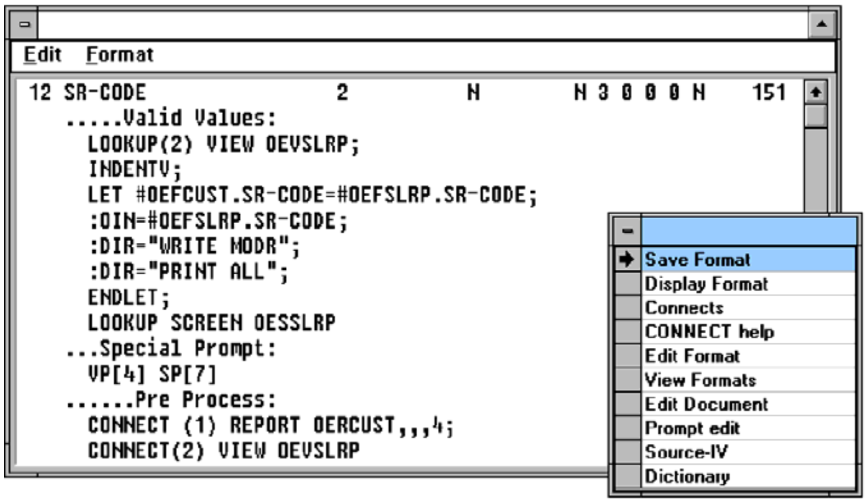


> Customer File <

Cust Code	Cust Sales	Customer Contact	Customer Name	Sr Cd	Sales Rep Nam	Customer City	St	Dscnt %
100100	15827.00	Tex Rogers	Toot-Your-Horn	AF	Albert Fisher	Port Lavaca	TX	30.00
100101	16253.36	David Kelly	Fix-M-Up	HP	Henry Phelps	Seadrift	TX	32.00
100102	16628.04	Sue Thompson	Computer Inc.	JJ	Joe Jones	Madison	NJ	34.00
100103	16951.04	Robert Brock	Today's Company	JS	Jack Sulephen	Bridgewater	NJ	36.00
100104	17222.36	Sarah Smith	ACME Inc.	AF	Albert Fisher	Dayton	NJ	38.00
100105	17442.00	Walter Snider	Lumber Inc.	HP	Henry Phelps	Big Horn	ND	40.00
100106	17609.96	Dennis Gohlke	OK Development	JJ	Joe Jones	Port Lavaca	TX	42.00

[Sales representative] <F1> Report (By Sales Rep) <F2> Display sales rep view

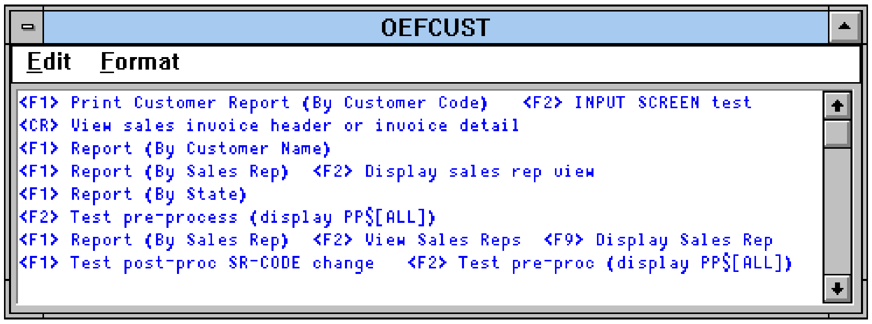
The next figure shows the data element name definition for that column, which was displayed by selecting **Data Name Edit (F11)**.



```
12 SR-CODE              2         N         N 3 0 0 0 N    151
   .....Valid Values:
     LOOKUP(2) VIEW OEVSLRP;
     INDENTV;
     LET #OEFCUST.SR-CODE=#OEFSLRP.SR-CODE;
     :OIN=#OEFSLRP.SR-CODE;
     :DIR="WRITE MODR";
     :DIR="PRINT ALL";
     ENDLET;
     LOOKUP SCREEN OESSLRP
   ...Special Prompt:
     VP[4] SP[7]
   ......Pre Process:
     CONNECT (1) REPORT OERCUST,,,4;
     CONNECT(2) VIEW OEVSLRP
```

Defining Special Prompts

Two prompts are used in the previous example for the data element name. They are described as VP[4] and SP[7]. The text of these prompts is defined in the file, which is accessed through Prompt Edit on the **Special Functions (F7)** menu. This file, shown below, collects together all the special prompts defined for all data element names within the parent format.



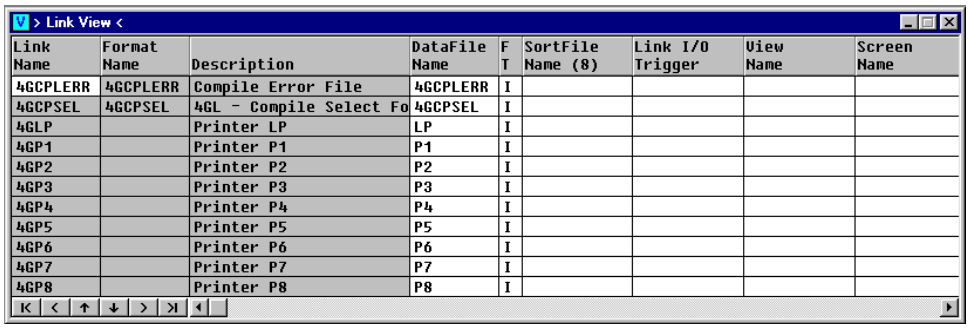
```
<F1> Print Customer Report (By Customer Code)   <F2> INPUT SCREEN test
<CR> View sales invoice header or invoice detail
<F1> Report (By Customer Name)
<F1> Report (By Sales Rep)  <F2> Display sales rep view
<F1> Report (By State)
<F2> Test pre-process (display PP$[ALL])
<F1> Report (By Sales Rep)  <F2> View Sales Reps  <F9> Display Sales Rep
<F1> Test post-proc SR-CODE change   <F2> Test pre-proc (display PP$[ALL])
```

VP[4] instructs OPENworkshop to use the 4th prompt when displaying the data element name in a view, and SP[7] specifies that the 7th prompt is to be used when in edit mode of that data element name.

Link

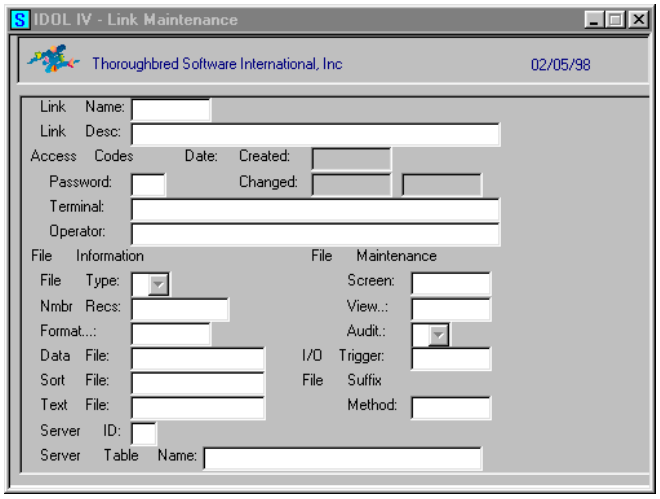
A link is a Dictionary-IV interface to a data file and associated dictionary definitions. It may be used in scripts, reports, queries, and database maintenance. It specifies default presentation classes (view and screen) to use to display the data and allow it to be edited. The link also specifies any trigger method to be used to ensure that referential integrity is maintained throughout the system when data referenced in the link is updated.

To add or modify a link definition move to the Link Header view for the application.



Link Name	Format Name	Description	DataFile Name	F T	SortFile Name (8)	Link I/O Trigger	View Name	Screen Name
4GCPLERR	4GCPLERR	Compile Error File	4GCPLERR	I				
4GCPSEL	4GCPSEL	4GL - Compile Select Fo	4GCPSEL	I				
4GLP		Printer LP	LP	I				
4GP1		Printer P1	P1	I				
4GP2		Printer P2	P2	I				
4GP3		Printer P3	P3	I				
4GP4		Printer P4	P4	I				
4GP5		Printer P5	P5	I				
4GP6		Printer P6	P6	I				
4GP7		Printer P7	P7	I				
4GP8		Printer P8	P8	I				

Select **Line Insert** to add a new link. OPENworkshop creates a new record and waits for you to enter the link name and description. To delete a link select **Line Delete**.



To modify a link definition, select **Edit** to display the screen for link maintenance.

Field	**Attribute**
Access Codes	Allows access controls to be set for the data file controlled by this link. Access can be restricted to specific terminals or operator codes, and may be password protected.
File Type	Type of file.

Field	**Attribute**
Nmbr Recs	Number of records to create. When 0 is entered, an automatically expanding file is created.
Format	The format-name defines the data items referenced by the link
Data File	The name of the file containing the subject data.
Sort File	The name of the file containing the sort keys and index.
Text File	The name of the file containing text fields, if required.
Screen	The name of the default screen to use when **F9** is pressed in a view. If no default screen is specified, OPENworkshop creates and displays a default screen whenever **F9** is pressed in a view.
View	The name of the default view to use when displaying data from the link.
Audit	Enable or disable audits.
I/O Trigger	I/O trigger methods are responsible for controlling referential integrity in an OPENworkshop application. They are executed, when defined in a link, whenever the data file referenced by the link is about to be updated. Trigger methods ensure that any update is consistently applied across all files that rely on data values in the link. For more information, see the OPENworkshop Methods section of this manual..
File Suffix Method	A file suffix method generates a file suffix to be applied to file names to ensure that files are handled unambiguously within the file system. For more information, see the OPENworkshop Methods section of this manual.

For more information on link attributes, see the Dictionary-IV Developer Guide.

CONNECT Directives

The CONNECT directive can be used:

- From a menu
- From a data element name pre-process
- From a data element name post-process
- From within any script method

The general form of the CONNECT directive is:

```
CONNECT [(function-key-number)] CLASS class-name [message]
```

Example

```
CONNECT(2) METHOD OEZSTST START
```

This example connects to the OEZSTRT method when **F2** is pressed. The START message string is passed to the method.

Note: Where a graphical user interface is employed, the option is displayed in a pop-up menu when you click the right mouse button, as well as being associated with the function key.

The following rules apply to CONNECT directives:

- The optional function-key-number is only valid when the CONNECT directive is called from a data element name pre-process. It defines the function key that the user must press to cause the activation of this CONNECT directive.
- The optional message is not valid for CONNECT HELP, CONNECT MENU, CONNECT SCREEN, and CONNECT VIEW. It is only available for CONNECT METHOD, CONNECT REPORT, and CONNECT QUERY.
- Only methods have the ability to pass string arrays to connected classes. These arrays can be used to control the behavior of the connected class. If it is important to exercise this level of control you should first connect to a method, then use the method to set the required parameters in an array before calling the required class.

Data Element Name Pre-process CONNECT Directives

The data element name pre-process CONNECT directives are evaluated when the cursor is placed in a view or screen on the data element name, before the user has entered any data. Typically the pre-process facility is used to allow the user to decide the next action. Screen prompts and view prompts, displayed on the first line of the screen, provide the user with information on options.

In a data element name pre-process a CONNECT statement specifies the actions to be taken. If the CONNECT statement includes a function-key-number, the directive is executed when that function key is pressed. If no function key number is supplied, or if 0 is specified, the directive is executed automatically when the view or screen field enters data entry mode.

The following CONNECT directives may be used in a data element name pre-process.

```
CONNECT[(function-key-number)] HELP help-name
```

Connects to the help subsystem, which will display the named help message. When the connected help is closed control will be returned to this pre-process.

```
CONNECT[(function-key-number)] MENU menu-name
```

Connects to the menu subsystem and displays the named menu. When the connected menu is closed control will be returned to this pre-process.

```
CONNECT[(function-key-number)] METHOD method-name
   [message]
```

Passes control to the named method. When the method exits then control will be returned to this pre-process. Methods may pass back method return directives, to be honored by the pre-process. See the information on pre-process in the Data element names subsection earlier in this section.

```
CONNECT[(function-key-number)] METHOD method-name
   [PP=xx]
```

xx is a two-character value which will be received by the system data element name #IDSV.PASSED-PARMS and by the Script system variable MENU-PARMS. This data element name can be used to control the behavior of the method.

```
CONNECT[(function-key-number)] QUERY query-name
   [,parameter[,parameter]...]
```

Passes control to Query-IV to execute the named query. When the query terminates, control will be returned to this pre-process. Parameters may be supplied to Query-IV in a comma-separated string. For more information on parameters, refer to information on the QUERY$ string array in the QUERY$ section of this manual.

```
CONNECT[(function-key-number)] REPORT report-name
   [,parameter[,parameter]...]
```

Passes control to Report-IV to execute the named report. When the report terminates, control will be returned to this pre-process. Parameters may be supplied to Report-IV in a comma-separated string. For more information on parameters, refer to information on the REPORT$ string array in the REPORT$ section of this manual.

```
CONNECT[(function-key-number)] SCREEN screen-name
```

Connects to the named screen. When the connected screen is closed, control returns to this pre-process.

```
CONNECT[(function-key-number)] VIEW view-name
[USING data element name or expression; |
USING RANGE FROM data element name or expression; TO data element name or
expression;]
[SELECT WHEN expression;]
[SORT [BY] n]
```

Connects to the named view. When the connected view is closed control is returned to this pre-process.

The USING and USING RANGE clauses specify one or more data element name values or expressions that, when evaluated, restrict the key range in the connected view to select records to be displayed.

The SELECT WHEN clause is used when the required selection criterion is to be based on non-key field(s).

The optional SORT BY clause specifies which sort definition will be used in the connected view.

If an expression has embedded blanks then it must be terminated by a semicolon.

Script Method CONNECT Directives

The script method CONNECT directives are evaluated when invoked. Methods may pass parameters and messages through the CONNECT directive using string arrays. For a description of the relevant array see the section on String Arrays. The script method must dimension these arrays before they are used.

The following CONNECT directives can be used in a script method.

`CONNECT HELP help-name`

Connects to the help subsystem, which will display the named help message. When the connected help is closed control will be returned to the original method.

The method may supply parameters and messages to control the behavior of the help subsystem by building string array HELP$. See the HELP$ section of this manual.

`CONNECT MENU menu-name`

Connects to the menu subsystem and displays the named menu. When the menu is closed, control will be returned.

The method may supply parameters and messages to control the behavior of the Menu subsystem by building string array MENU$. See the MENU$ section of this manual.

`CONNECT METHOD "method-name" ["message"]`

Passes control to the named method. When the method exits, control will be returned to the original method.

The method may supply parameters and messages to control the behavior of the called method by building the MSG1$ string array. OPENworkshop will place the optional message in MSG1$[1]. See the information on application methods in the OPENworkshop Methods section of this manual.

`CONNECT QUERY query-name[,parameter[,parameter]...]`

Passes control to Query-IV to execute the named query. When the query terminates, control will be returned. Parameters may be supplied to Query-IV in a comma-separated string. For more information on parameters, see the information on the QUERY$ string array in the QUERY$ section of this manual.

Alternately, the method may supply parameters and messages to control the behavior of the Query-IV subsystem by directly building the QUERY$ string array.

`CONNECT REPORT report-name[,parameter[,parameter]...]`

Passes control to Report-IV to execute the named report. When the report terminates, control will be returned. Parameters may be supplied to Report-IV in a comma-separated string. For more information on parameters, refer to the REPORT$ string array section of this manual.

Alternately, the method may supply parameters and messages to control the behavior of the Report-IV subsystem by directly building the REPORT$ string array.

`CONNECT SCREEN screen-name`

Connects to the named screen. When the screen is closed, control returns to the method.

The method may supply parameters and messages to control the behavior of the called screen by building the SCREEN$ string array. For more information on parameters, refer to the SCREEN$ string array section of this manual.

```
CONNECT VIEW view-name
```

Connects to the named view. When the view is closed, control is returned to the method.

Optional USING, USING RANGE and SELECT WHEN clauses may be used to specify one or more data element name values or expressions that, when evaluated, should be used as the key in the connected view to select records to be displayed.

The optional SORT clause specifies the sort index that should be used when record selection is carried out in the connected view.

Unlike a pre-process CONNECT, when these optional functions are invoked from a script method they must be placed in a string within string array VIEW$. For more information on parameters, see the VIEW$ string array section of this manual.

If an expression has embedded blanks then it must be terminated by a semicolon.

Other CONNECT Directives

The CONNECT directive may also be initiated from a data element name post-process or menu. In these cases, the directive behaves as described for data element name pre-process CONNECT directives, except that the function key number option is not available and the following exceptions apply:

CONNECT	**From Data Element Name Post-Process**	**From Menu**
HELP	As data element name pre-process	As data element name pre-process
MENU	As data element name pre-process	As data element name pre-process
METHOD	Method return directives will not be evaluated	Method return directives will not be evaluated. If the menu is a selection type menu then selection parameters will be passed to the called method in the MSG$[2] string array.
QUERY	As data element name pre-process	As data element name pre-process
REPORT	As data element name pre-process	As data element name pre-process
SCREEN	As data element name pre-process	As data element name pre-process
VIEW	As data element name pre-process	As data element name pre-process

Other Directives

This section describes further directives that are available to the OPENworkshop developer.

LOOKUP SCREEN

The LOOKUP SCREEN directive may be used within a data element name Valid Values field to specify a screen definition to be used when the user selects **Lookup Screen (F9)** in a view or a screen. LOOKUP SCREEN enables the default screen defined in the link to be overridden.

Typically, the called screen will be used to supply full details about an object that is displayed or referenced by the data item displayed. The normal use of LOOKUP SCREEN when in a screen or a view is to switch from a screen to a view or a view to a screen for a specific data record. LOOKUP SCREEN allows the lookup to be extended to connect to any screen in the application.

From a view, the field must be in entry mode before the LOOKUP SCREEN will be honored.

The syntax for LOOKUP SCREEN is:

```
LOOKUP SCREEN screen-name
```

LOOKUP VIEW

LOOKUP VIEW may be specified within a data element name Valid Values field to assist in validation of data input by the user. On completion of input in a data element name within a view or a screen where LOOKUP VIEW has been specified, OPENworkshop will compare the received data with key values in the specified view. If there is a match the field is accepted. If there is no match, then OPENworkshop connects to the view, scrolling the rows to the nearest match. The user is then able to move to the desired row and select **OK (Enter)** to select the record.

Note: The maximum length of the full lookup syntax is 256 bytes.

Syntax is:

```
LOOKUP[(function-key-number)] VIEW view-name
[ERMFIRST;] [SPACEOK] [USING data element name or expression;]
[LET expression; or directive; ENDLET]
```

- If ERMFIRST is specified a brief error message:

  ```
  The value entered is not contained in the lookup file
  ```

  Displays before the connect to the view is made. **Close (F4)** returns to user input, allowing the field to be re-entered. Any other response connects to the view.

- If the function key number is used, when in input mode the view will be displayed when the function key is pressed.

- If SPACEOK is specified then blank entries will be accepted.

- The optional USING clause may be used to restrict the number of rows displayed in the connected view.

- A variable :OIN is provided, containing the value entered by the operator. This may be used to construct the key. :OIN can also be assigned a value by the LET / ENDLET directive.

- Variable :DIR may be assigned method return directive value(s). These directives will be executed on return from the lookup view.

Example

```
LOOKUP(2) VIEW OEVINV0 USING #OEFPARM.DEPT + :OIN;
LET #OEFINVD.INV-LPRICE = #OEFINVH.ITEM-PRICE;
#OEFINVD.ITEM-CODE = #OEFINVH.ITEM-CODE;
:DIR="PRINT ROW"; :OIN=#OEFINVH.ITEM-CODE; ENDLET
```

Method Return Directives

Method return directives provide a means for methods to communicate directives or status flags back to a screen or view that invoked the method. Return directives are only evaluated by screens and views.

The method must place the directive(s) in string array MSG1$[0] before returning. If more than one directive is to be returned they must be separated by semicolons.

- Print the help specified by the help-code.

  ```
  ERROR [help-code]
  ```

- Exit the screen or view. Does the same as a **Close (F4).**

  ```
  EXIT
  ```

- Move the cursor to the specified column number (for a view), field number (for a screen) or data element name.

  ```
  GOTO FIELD Field-number | Data Element Name
  ```

- Move to the row with the specified key value. Only meaningful to a view.

  ```
  GOTO KEY Key-value | "Key-value"
  ```

- Print the data record in the view or screen.

  ```
  PRINT [ROW]
  ```

- Write the current record to file.

  ```
  WRITE
  ```

- Specifies the normal completion of connected method.

  ```
  . (period)
  ```

- Determine the number of rows to scroll up/down.

  ```
  SCROLL UP | DOWN
  ```

Examples:

```
SCROLL DOWN 10
SCROLL UP 10
SCROLL DOWN 10; GOTO FIELD data-name
```

- This allows you to scroll to the right and left of a locked column, leaving the cursor on the current column.

```
TAB | TAB BACK
```

Examples:

```
TAB 5
TAB BACK 5
```

- Signifies the method did not complete normally.

```
[Any other value]
```

Example:

```
IF MSG$[1]<>""              ! If line number generated
   MSG$[0]="PRINT ROW;"+    ! Set return
      "GOTO FIELD 5"        ! directives.
ELSE                        ! else
   MSG$[0]="ERROR OEERM01"  ! Set error return.
FI;                         ! endif
```

Thoroughbred Basic Method CONNECT Equivalents

Thoroughbred Basic methods may also invoke the equivalent of CONNECT directives. Behavior and requirements are similar to those of script methods.

The CONNECT directive is not valid in Thoroughbred Basic, and the Thoroughbred Basic Environment will not automatically create and dimension the string arrays used to communicate parameters and messages. The Thoroughbred Basic method must dimension the required arrays and CALL a program as follows:

CONNECT	**CALL**	**ARRAYS**
HELP	004, 001Z	HELP$
MENU	001, 001Z	MENU$
QUERY	00R	QUERY$
REPORT	00R	REPORT$
SCREEN	002A	SCREEN$
VIEW	003A	VIEW$

See on-line help for a description of the CALL lists for these calls.

OPENworkshop Methods

The ability to write methods, and to connect to methods from OPENworkshop classes, is the key to the flexibility and productivity associated with working in OPENworkshop.

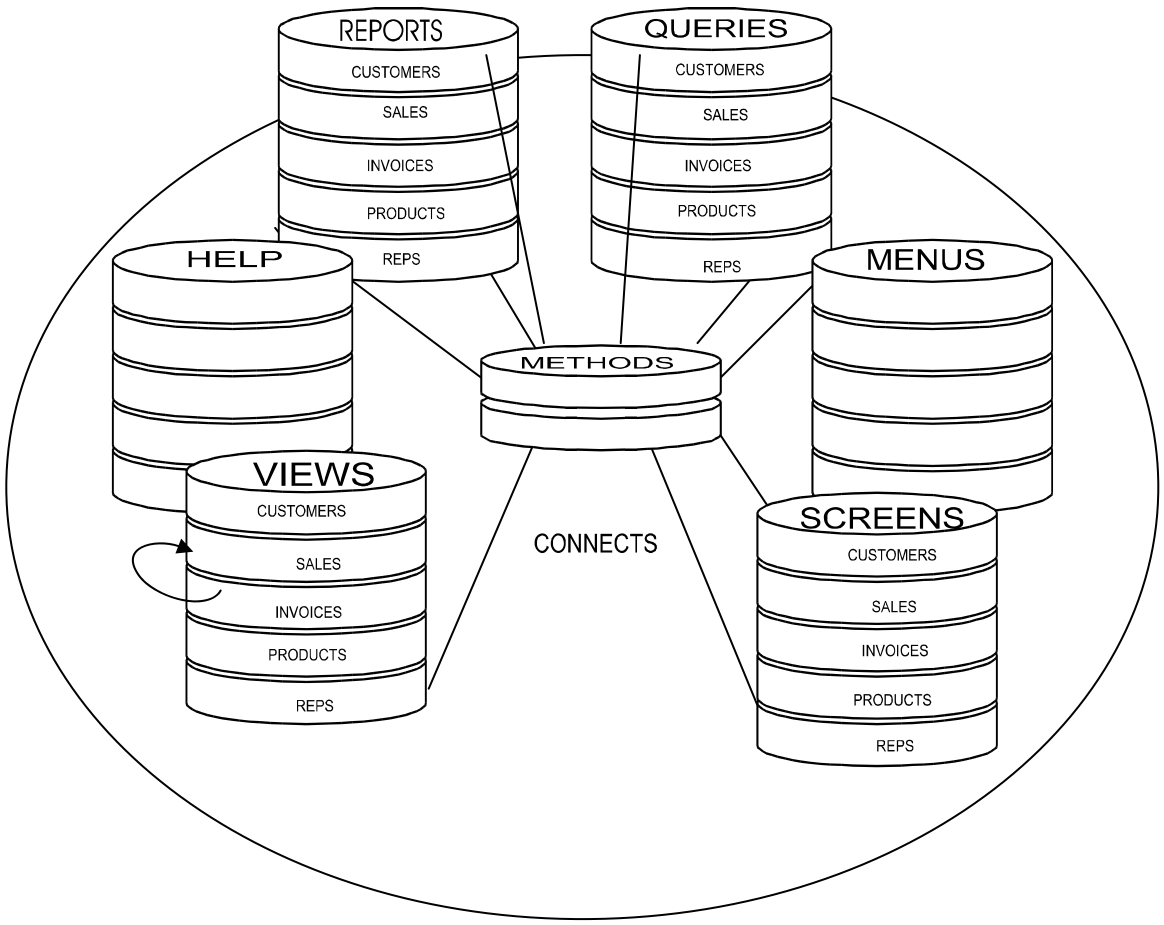
Methods may be written in the Script-IV or Thoroughbred Basic languages, but Script-IV should be used whenever possible. The Script-IV environment automates many of the processes necessary for a reliable and consistent application.

Methods may be initiated through a number of different routes. Methods can be connected from a menu, screen, or view. Methods can be passed parameters and messages. These are made available to the method in string arrays, described in the String Arrays section of this manual.. Methods written in Script-IV will automatically have access to these arrays. Methods written in Thoroughbred Basic must declare them.

Methods can connect to all OPENworkshop classes and to other methods.

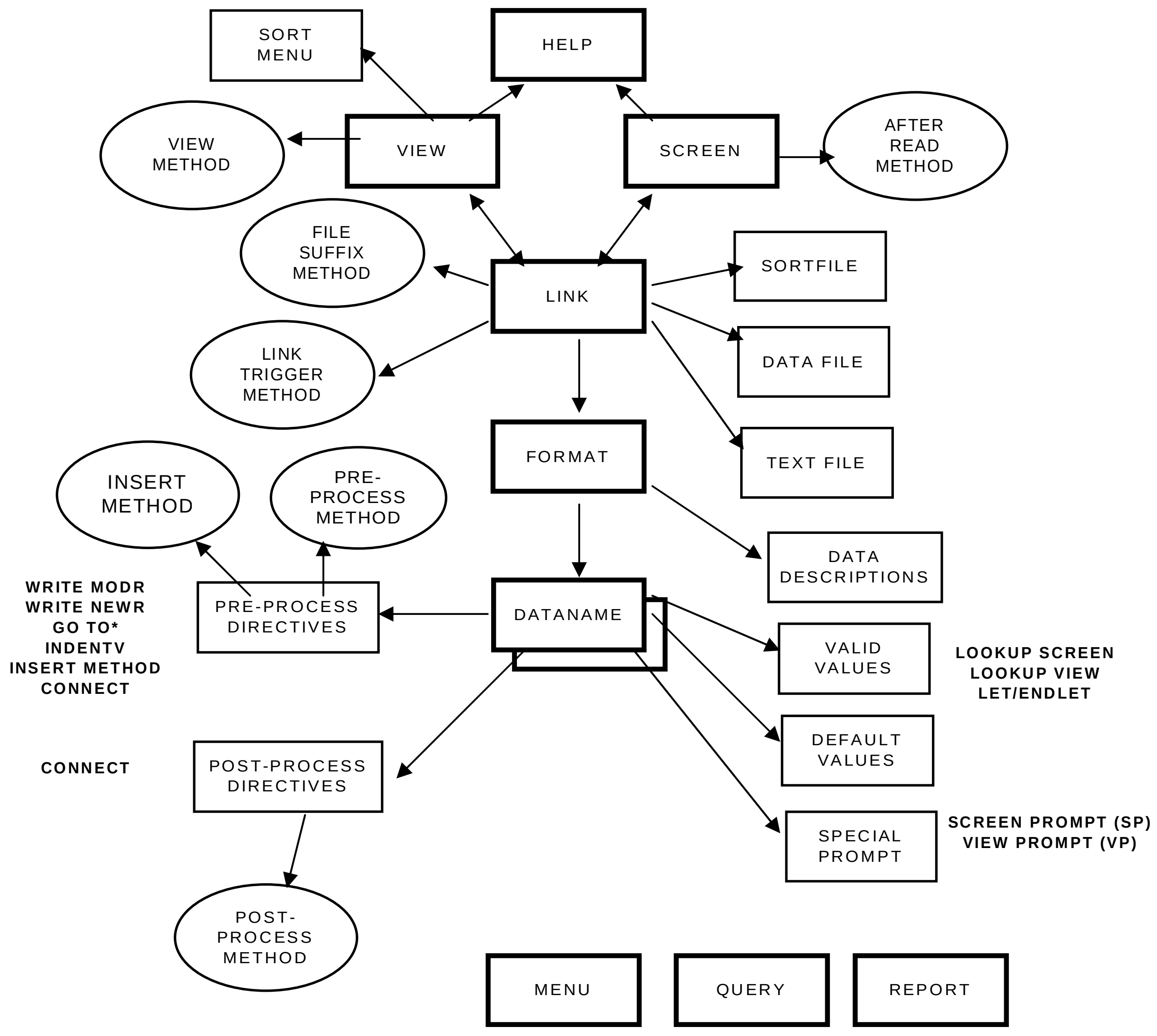
Methods can return information to their caller. Most frequently this is accomplished using method return directives, but in some cases information may also be returned through the string arrays. Methods may also communicate information through global variables.

For information about method creation see the Creating Script Methods section later in this section.



Types of Methods

OPENworkshop recognizes a number of different types of methods, each type being designed for a specific purpose, and each being specified at different locations within Dictionary-IV.



The diagram above indicates the types of methods available and the ways they can be initiated.

Data Element Name Pre-process Method

A data element name pre-process method is activated by a CONNECT METHOD directive defined in the pre-process field of a data element name definition. It is invoked when a view or screen that contains the data element name is preparing to allow the user to edit a data item defined by the data element name.

The method may take any action that is necessary for the application, but is usually used for one of the following purposes:

- To perform an application function requested by the user pressing a function key. For example, Process Invoice.

- To set up the environment prior to performing a connect to another class. For example, calculate a key value before passing control on to a view.

On entry, a method that is invoked through a pre-process CONNECT METHOD directive will receive four string arrays:

Pre-process creates array:	PP$[ALL]	SA$[ALL]	FA$[ALL]	LNK$[ALL]
Passed to script method as:	MSG1$[ALL]	MSG2$[ALL]	MSG3$[ALL]	MSG4$[ALL]

On exit, control will be returned to the calling screen or view. The method is able to return status information and directives to the caller through method return directives. The method must place these directives in MSG1$[0] prior to completion.

For more information refer to the Method Return Directives section in this manual. For information about how and where to define methods see the Creating Script Methods section later in this section.

Example:

```
* This Script-IV method is invoked when a user wishes to add or change
* invoice data by using a screen for data input.
*
LN OELINVD
SN OESINVD
FN OEFINVD

MAINLINE
                              ! Get the Global data for the invoice
   FORMAT INCLUDE #OEFINVD, OPT="NONE"
   LET OEFINVD = #OEFINVD     ! Copy to local format
                              ! (For an explanation of Global vs Local
                              ! Formats see the Local versus global
                              ! formats section later in this manual)
   OPEN SCREEN OESINVD
   PRINT SCREEN OESINVD
   PRINT SCREEN OESINVD DATA  ! Output current data values
   INPUT SCREEN OESINVD       ! Input the revised data values
   LET #OEFINVD = OEFINVD     ! Copy new data to Global
   LET MSG1$[0] = "PRINT ROW" ! Re-print the row in the View
                              ! on exit.
```

Data Element Name Post-process Method

A data element name post-process method is activated by a CONNECT METHOD directive defined in the post-process field of a data element name definition. It is invoked when a view or screen that contains the data element name has completed editing a data item defined by the data element name.

The method may take any action that is necessary for the application, but is usually used for one of the following purposes:

- To perform an application function needed as a result of the completed data entry. For example, to populate fields in the current row that depends on this input field.

- To set up the environment after the caller has accepted data, for example, to set global variables to new values.

On entry, a method that is invoked through a post-process CONNECT METHOD directive will receive four string arrays:

Post-process creates array:	PP$[ALL]	SA$[ALL]	FA$[ALL]	LNK$[ALL]
Passed to script method as:	MSG1$[ALL]	MSG2$[ALL]	MSG3$[ALL]	MSG4$[ALL]

On exit, control will be returned to the calling screen or view. The method is able to return status information and directives to the caller through method return directives. The method must place these directives in MSG1$[0] prior to completion.

For more information refer to the Method Return Directives section in this manual.

Example:

```
*   This Script-IV method is invoked when the quantity of an inventory
*   item has been entered. The item discount, price, and tax are then
*   calculated and reprinted in the view.

LN OELINVN
FN OEFINVD
```

```
MAINLINE

   PRECISION 2
   FORMAT INCLUDE #OEFINVN, OPT="NONE"      ! Inventory file.
   LET OEFINVD = FMD("#OEFINVD"),           ! Make local copy of
   OEFINVN = #OEFINVN                       ! Invoice details and
                                            ! Inventory details.
   IF OEFINVD.ITEM-CODE <> OEFINVN.ITEM-CODE THEN
      OPEN LINK OELINVN                     ! Not the required
                                            ! inventory item.
      LET OEFINVN.ITEM-CODE = OEFINVD.ITEM-CODE
      READ OELINVN                          ! Get the required
                                            ! inventory item.
      LET FMD("#OEFINVN")=OEFINVN           ! Put it in the global.
   ENDIF
   LET Q = NUM(MSG1$[2]),                   ! The new quantity
                                            ! from the operator.
   P = (OEFINVN.ITEM-PRICE * OEFINVD.INV-DISC)/100,
   OEFINVD.INV-QTY = Q,
   OEFINVD.INV-LPRICE = OEFINVN.ITEM-PRICE - P,
   OEFINVD.INV-LEXTEN = OEFINVD.INV-LPRICE * Q,
   OEFINVD.INV-LTAX = OEFINVD.INV-LEXTEN * TAX-RATE,
   MSG1$[0]="PRINT ROW; GOTO FIELD 11"      ! Return directives.
   LET FMD("#OEFINVD") = OEFINVD            ! Replace the global
                                            ! with new values.
```

File Suffix Method

A file suffix method is used to obtain a correct file suffix for a link to open a data file. It is typically required in applications that use a file suffix to differentiate between different parts of the application data. For example, a file suffix is sometimes used to differentiate between different companies within a single accounting application. For more information on file suffixes, see the Dictionary-IV Administrator Guide.

The name of the file suffix method is specified in the link definition.

On entry the method will receive string array MSG1$[ALL], with MSG1$[1] containing the link-name to be opened.

Link creates array:	MSG1$[ALL]
Passed to script method as:	MSG1$[ALL]

The method should place the file suffix in global object #IDSV.FILE-SUFFIX. MSG$[0] should be set to an appropriate method return directive status flag.

For more information refer to the Method Return Directives section of this manual.

Example:

```
PROCEDURE
   FORMAT INCLUDE #OEFCMPNY,OPT="NONE"
   FORMAT INCLUDE #IDSV,OPT="NONE"
   IF MSG1$[1] = "OELCUST " THEN
      LET #IDSV.FILE-SUFFIX = #OEFCMPNY.COMPANY-CODE
   ENDIF
   LET MSG1$[0] = "."
```

Insert Method

An insert method is activated when an operator performs a **Line Insert** in a view that has an insert method specified. The insert method is specified by using an INSERT METHOD directive in the pre-processing section of a data element name definition in a format. The data element name must be a key field of the underlying format.

The insert method is responsible for checking that insertion of a new record is acceptable, and for building any required default data not already specified in the data element name definitions contained in the view.

On entry the method will receive four string arrays:

View creates array:	PP$[ALL]	SA$[ALL]	FA$[ALL]	LNK$[ALL]
Passed to script method as:	MSG1$[ALL]	MSG2$[ALL]	MSG3$[ALL]	MSG4$[ALL]

On exit control will be returned to the calling view. The method is able to return status information and directives to the caller through method return directives. The method must place these directives in MSG1$[0] prior to completion.

For more information refer to the Method Return Directives section of this manual.

OPENworkshop coordinates the actions of insert methods and view methods when both are defined for a view. When a view is first displayed, and when rows are displayed in a view after scrolling operations, any specified view method is invoked. When an operator selects **Line Insert** a fresh row is created in the view and the view method is applied to that row. The insert method is then invoked and processed for the new row.

```
*   This Thoroughbred Basic Method is activated when an operator inserts
*   a new line into an invoice. It sets up default details and generates
*   the new invoice line number.

METHOD MSG$[ALL],SA$[ALL],FA$[ALL]
FN OEFCUST,OEFINVH,OEFINVD
. . . .

PROCEDURE
   IF CVT(#OEFINVH.INV-NUM,128)=""      ! If invoice header is space
      MSG$[0]="ERROR OEERM00";          ! Set error return.
      EXIT                              ! Get out.
   FI;                                  ! endif
   #OEFINVD = #OEFINVH,                 ! Set detail to common
                                        ! DNs from header.
   #OEFINVD.INV-DISC =                  ! Set invoice line
   #OEFCUST.CUST-DISC;                  ! item discount.
   CALL "OO71",MSG$[ALL],SA$[ALL],      ! Get next line number from
      FA$[ALL];                         ! sequence number generator
   IF MSG$[1]<>""                       ! If line number generated
      MSG$[0]="PRINT ROW;"+             ! Set return
         "GOTO FIELD 5"                 ! directives.
   ELSE                                 ! else
      MSG$[0]="ERROR OEERM01"           ! Set error return.
   FI;                                  ! endif
   EXIT                                 ! Get out.
```

Trigger Method

A trigger method is called whenever an UPDATE command is executed on a link where a trigger method is defined. An update is executed whenever data in a view or a screen is changed, and also whenever an UPDATE command is encountered in a script method. The existence of a trigger method is declared in the link definition. For more information, see the UPDATE command section of this manual.

Trigger methods are a corner stone of the referential integrity facilities of an OPENworkshop application. Trigger methods are responsible for ensuring that all relevant files are consistently maintained according to application rules.

All links can have a trigger method if required. If a trigger method updates data in a link other than the one for which it is defined, the trigger method for that link will also be executed. This powerful mechanism allows an update to occur in all related files.

NOTE: Trigger methods must be written in Script-IV.

On entry the method will receive two string arrays:

Link creates array:	LNK$[ALL]	KT$[ALL]
Passed to script method as:	MSG1$[ALL]	MSG2$[ALL]

On completion the method must issue a Return Directive in MSG1$[0].

Trigger Methods can allow or not allow file I/O requests based on the return directive.

For more information, see the Method Return Directives section in this manual.

Example:

```
!  This method performs the update rules for the Invoice Header file.
!  Part of the requirement is to modify data in the invoice detail, and
!  this method simply does an UPDATE, causing a trigger for the invoice
!  detail file to be executed.

LN OELINVD                           ! Define invoice detail link.
FN OEFINVH                           ! Define invoice header frmt.
. . . .

INVHUPDATE
   FORMAT INCLUDE #OEGF, OPT="NONE"  ! Include global
                                     ! flags definition.
   INCLUDE OE.OEIFCK                 ! Validate I/O function
   LET OEFINVH=MSG2$[0],             ! Set new: invoice header rec.
      CC$=OEFINVH.CUST-CODE,           ! customer code.
      SR$=OEFINVH.SR-CODE,           ! sales rep code.
      OEFINVH=FMD("#OEFINVH"),         ! Set old: invoice header rec.
         FUNC$=MSG1$[0]                ! I/O function.

   IF CC$ <> OEFINVH.CUST-CODE OR    ! If invoice customer code or
      SR$ <> OEFINVH.SR-CODE OR        ! sales rep changed or
      FUNC$="D" THEN                   ! invoice header delete
      IF FUNC$="D" THEN              ! If header being deleted
         LET #OEGF.INVHC="X"         ! Set invoice hdr delete flag.
      ENDIF                          ! endif
      OPEN LINK OELINVD              ! Open invoice detail link.
      UPDATE OELINVD FUNC$           ! Update all detail records for
         USING KEY RANGE             ! the invoice.
         FROM OEFINVH.INV-NUM        ! (The trigger method for the
         TO OEFINVH.INV-NUM          ! details in OELINVD runs
         PROCESSING IS INVDUPDATE    ! for each record updated)

      LET #OEGF.INVHC=""             ! Clear invoice hdr delete flag.
   ENDIF                             ! endif
   LET MSG1$[0]="."                  ! Set ok return status.

INVDUPDATE                           ! Processing For Invoice Detail
                                     ! Update.
   LET OEFINVD.CUST-CODE = CC$,      ! Update: customer code.
      OEFINVD.SR-CODE = SR$          ! sales rep code.
```

View Method

A view method is invoked when a view is preparing a row for display, having read the row data from a file. Views may contain data from multiple links and formats, and may also contain calculated columns. The view method is responsible for ensuring that all column values are available for display by the view or for preventing a row from being displayed.

On entry the method will receive three string arrays.

View creates array:	V$[ALL]	VM$[ALL]	LNK$[ALL]
Passed to script method as:	MSG1$[ALL]	MSG2$[ALL]	MSG3$[ALL]

OPENworkshop executes the view method once for each row displayed in the view.

The VM$ array is provided as a working storage area for use by the developer, and may contain any data required by the method. The first time the view method is executed when preparing a view for display the VM$[ALL] array will not be dimensioned. If the array is to be used, the method must dimension the array and place values in it. OPENworkshop will retain the array when the method exits, and then returned to it on subsequent view method calls. When the calling view is closed by the operator the VM$ array is deleted from memory.

To avoid conflicts the View Method should maintain a unique channel list for reads performed by the method. Because the View Method can be invoked by any number of Scripts or Methods and at various levels of recursion, the channel context can vary.

On exit from the view method, control will be returned to the calling view. The method is able to return status information and directives to the caller through method return directives. The method must place these directives in MSG1$[0] prior to completion. Setting LNK$[0] to any value other than "." before exiting will suppress the display of the row.

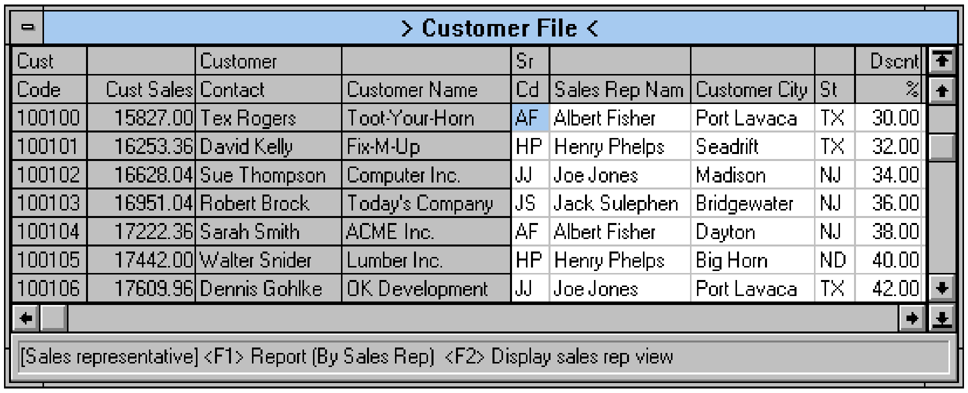
Trigger methods can allow or disallow file I/O requests based on the Return Directive.

For more information refer to the Method Return Directives section in this manual.

Example:

```
!  This method prepares the Customer File view, an example of which
!  is shown below. In this view, the Sales Rep Name column is looked
!  up from the Sales Rep file by the view method. Because this is the
!  first column that was added to the view it is referred to as column
!  A. This is an example of a Thoroughbred Basic method.

METHOD V$[ALL], VM$[ALL], LNK$[ALL]
FN OEFCUST, OEFSLRP
. . . .
```

```
PROCEDURE
   V0$ = V$[0,0],              ! Get view info entry.
   EL = ASC(V0$(1,1)),         ! Set view attribute entry size.
   NC$ = V0$(EL);              ! Get new column definition string.
   IF NEA("VM$",1)=0           ! If the View Method array not built
      DIM VM$(1);              !   Dim array.
      CH=UNT;                  !   Get next available channel
      OPEN (CH) "OESLRP"       !   Open sales rep file
      VM$[1]=STR(CH)           !   Save channel for next call
   ELSE                        ! Else
      CH=NUM(VM$[1])           !   Get sales rep channel previously
                               !   opened by this method
   FI;                         ! endif
   SAV$ = #OEFSLRP,            ! Save contents of current sales rep
                               ! record. This is necessary if the
                               ! contents of the sales rep record are
                               ! being returned to a class that may
                               ! change the contents of the sales rep
                               ! file based on current contents of
                               ! sales rep data record.

   SR$=CVT(#OEFCUST.SR-CODE,128),    ! Set customer sales rep code.
   I=POS("A"=NC$,2);                 ! Find new column id "A".
   IF I                              ! If found
      I=ASC(NC$(I+1,1));             ! Set column number.
      GOSUB GETSLSRP                 ! Go get sales rep name.
   FI;                               ! endif
   EXIT                              ! Get out.

GETSLSRP                             ! Get Sales Rep Name For
                                     ! Customer View.
   V$[0,I]=V$[0,I](1,EL);            ! Chop last sales rep name
                                     ! from column entry.
   IF STL(SR$)                       ! If cus sales rep code not null
      CLEAR ERC;                     ! Clear error code and
      READ (CH,                      ! Read sales rep
      KEY=#OEFCUST.SR-CODE,          ! file using customer
         ERC=1) #OEFSLRP;            ! sales rep code.
      IF ERC=0                          ! If no error
      V$[0,I]=V$[0,I](1,EL)+         ! Append sales rep name to
         #OEFSLRP.SR-NAME            ! column attribute string.
      FI;                               ! endif
      #OEFSLRP=SAV$                  ! Restore sales rep data rcrd.
   FI;                               ! endif
   RETURN                            ! Return.
```


> Customer File <

Cust Code	Cust Sales	Customer Contact	Customer Name	Sr Cd	Sales Rep Nam	Customer City	St	Dscnt %
100100	15827.00	Tex Rogers	Toot-Your-Horn	AF	Albert Fisher	Port Lavaca	TX	30.00
100101	16253.36	David Kelly	Fix-M-Up	HP	Henry Phelps	Seadrift	TX	32.00
100102	16628.04	Sue Thompson	Computer Inc.	JJ	Joe Jones	Madison	NJ	34.00
100103	16951.04	Robert Brock	Today's Company	JS	Jack Sulephen	Bridgewater	NJ	36.00
100104	17222.36	Sarah Smith	ACME Inc.	AF	Albert Fisher	Dayton	NJ	38.00
100105	17442.00	Walter Snider	Lumber Inc.	HP	Henry Phelps	Big Horn	ND	40.00
100106	17609.96	Dennis Gohlke	OK Development	JJ	Joe Jones	Port Lavaca	TX	42.00

[Sales representative] <F1> Report (By Sales Rep) <F2> Display sales rep view

After Read Method

An after read method is initiated by a screen when a record has been read and before it is displayed for editing.

The method may take any action that is necessary for the application but is usually used for one of the following purposes:

- To ensure that all relevant data is available and calculated prior to the screen displaying the data.
- To deal with potential error conditions, such as RECORD BUSY, when the read returns.

On entry the method will receive four string arrays:

Screen creates array:	ARM$[ALL]	SA$[ALL]	FA$[ALL]	LNK$[ALL]
Passed to script method as:	MSG1$[ALL]	MSG2$[ALL]	MSG3$[ALL]	MSG4$[ALL]

On exit control will be returned to the calling screen. The method is able to return status information and directives to the caller through method return directives. The method must place these directives in MSG1$[0] prior to completion.

For more information refer to the Method Return Directives section in this manual.

In addition to the standard return directives, the following directives are also available:

"" Continue processing record.

"**.**" Continue processing record.

SKIP Reject record, print **record not found** message.

SKIP-NOMSG Reject record, do not print the message.

OPENworkshop automatically adds **;SKIP-NOMSG** to any ERROR or EXIT return directives used.

The WRITE directive is not allowed in inquiry or logical screen entry modes. In processing a WRITE return directive the calling screen automatically re-EXTRACTs the record unless the SKIP or SKIP-NOMSG commands are used.

Example:

```
*  This Thoroughbred Basic method is associated with a screen that
*  displays customer information. Part of the information is derived
*  from the Sales Rep file. The after read method positions on screen
*  to the correct place in the Sales Rep name file.

*  The Customer Masterfile screen contains a formula that prints the
*  Sales Rep's name. This method reads the sales rep record into the
*  format #OEFSLRP. If the sales rep is not found, a return directive
*  is used to move the cursor to the sales rep code field if the mode
*  allows record edits.

METHOD MSG1$[ALL], MSG2$[ALL], MSG3$[ALL], MSG4$[ALL]
FN OEFCUST, OEFSLRP
. . . .

PROCEDURE
   MSG1$[0]=".",              ! Default return value to ok.
   MD$=MSG1$[2](1,1),         ! Get maintenance mode.
   KF=NUM(MSG1$[2](5,1));     ! Get key found flag.
   IF KF AND POS(MD$="ACDI")  ! If key found & add/chg/del/inq mode
      GOSUB GETSREP           ! Process sales rep code.
   FI;                        ! endif
   ]FMT$="";                  ! Don't delete formats included by
                              ! the FN directive
   GOTO CUEXIT                ! Cleanup and exit.

GETSREP                             ! READ SALESREP RECORD.
   CH=UNT;                          ! Get available channel number.
   OPEN (CH) "OESLRP";              ! Open code file.
   CLEAR ERC;                       ! Clear error control.
   SET ERC 0;
   READ (CH,KEY=#OEFCUST.SR-CODE,   ! Read
      ERC=1) #OEFSLRP;              ! sales rep file.

   IF ERC                                 ! If error reading sales rep
      IF ERR=11                           ! If not found
         #OEFSLRP.SR-NAME="*Not found*";  ! Set text.
         IF POS(MD$="AC")                 ! If add or change modes
            PRINT 'RB',;                  ! Ring bell.
            MSG1$[0]="GOTO FIELD SR-CODE" ! Start at sales rep field.
         FI                               ! endif
      ELSE                                ! else
         IF ERR=0                         ! If record busy
            #OEFSLRP.SR-NAME="*Busy*"     ! Set text.
         ELSE                             ! else
            MSG1$[0]="ERROR OESLERR"      ! Process unexpected error.
         FI                               ! endif
      FI                                  ! endif
   FI;                                    ! endif
   RETURN                                 ! done
```

Application Method

An application method, called through a CONNECT METHOD directive via a menu or a script, implements any general application logic required by the developer.

On entry the method will receive string array MSG1$[ALL], where:

MSG1$[0]	Defined by developer
MSG1$[1]	"Message"
MSG1$[2..n]	Defined by developer

Example:

```
Set Terminal Date  --------- CONNECT METHOD OOUTIMET
Set System Date    --------- CONNECT METHOD OOUTIMES
```

```
*  This application method is invoked when connected from a menu or
*  application program. MSG1$[1] will contain either a "T" or "S"
*  depending on which value is passed from the CONNECT directive.

METHOD MSG1$[ALL]

PROCEDURE
   IF MSG1$[1]="T"
      "Set Terminal Date" logic
   ELSE
      IF MSG1$[1]="S"
      "Set System Date" logic
      FI
   FI
```

Creating Script Methods

New Script-IV Facilities

The Script-IV language features and facilities are largely based on the Script-IV language, but some differences do apply. For the convenience of developers who have Script-IV experience, these differences are outlined below. The Associated Systems section of this manual provides a full guide to migrating existing Dictionary-IV applications to OPENworkshop. For more information, see the Script-IV Language Reference.

- OPENworkshop introduces a new UPDATE directive to the Script-IV language. UPDATE operates essentially as a CHANGE directive, except that it causes a trigger method associated with the link to be executed. For more information, see the UPDATE command section later in this manual.

- OPENworkshop no longer supports Type 1 or Pre/Post Processing scripts. However, any existing scripts of these types can very easily be changed to methods.

- While continuation scripts, overlays and public scripts may still be used, the way that OPENworkshop encourages code to be developed in small, independent methods makes them unlikely to be needed. The data environment and behavior on termination of continuation scripts and overlays is different from that of the Dictionary-IV environment. For more information, see the Script execution environment section later in this manual.

- OPENworkshop introduces local and global formats. For more information, see the local versus global formats section later in this manual.

- A comprehensive debugging environment.

- Full cross-referencing between all classes, methods and where they are used.

Source-IV

Thoroughbred Source-IV is a source code development and control system. In addition to library management and a context sensitive editor, Source-IV offers version control and the ability to recover any previous version of your source. This manual assumes that you will be using Source-IV for your development. For more information, see the Source-IV Reference Manual.

Once created, scripts must be compiled before they can be run.

How to structure a script method

Scripts are divided into two sections, the Data Environment section and the Procedures section.

The **Data Environment** section is located at the beginning of the script method before any procedures and is optional. It is used to declare the dictionary definitions such as format, screen, view, and link to be used by the script. The section is only necessary if Dictionary-IV definitions are going to be used by the script.

The **Procedures** section consists of one or more independent procedures containing Script-IV commands.

Readability

Readability can affect long-term maintenance costs and affect the overall profitability of a software product. Script-IV is a self-documenting language designed for readability and easy maintenance. The commands and clauses are written in a simple, descriptive language and parameters are always identified by specific phrases. In addition, comments or remarks may also be included in a script to enhance the readability. This section describes several ways to enhance the overall readability of a script.

However, some software developers prefer a terse, less wordy language and are willing to sacrifice some readability in exchange for shorter commands. The Script-IV language is flexible enough to allow for this preference without detracting from the overall understanding of the commands. This is accomplished, in part, by optional syntax elements that do not affect the command function and are available only as readability aids.

Given the choice between readability and terseness, it is very important that you select your style and be consistent in the use of the various readability options. Some thought should also be given to the design and organization of procedures in your scripts. The increase in productivity offered by Script-IV can be enhanced by spending additional time in the analysis and design phase of product development. This will not only benefit development during the implementation but will contribute to the readability and long-term maintenance of your scripts.

Comments

Comment or remark lines can be included in a script and are not compiled. Any line that contains an * (asterisk) in column 1 is treated as a comment line. You can use comment lines as dividers to separate or set off a group of procedures or other segments within a script. For example:

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

Optional Syntax

This refers to optional syntax elements that do not affect the command function and are purely available as readability aids. Three common ones are: IS, ARE, and PROCESS. For example, the following two clauses perform the same function:

```
MISSING KEY PROCESS IS
   TOTALS-PROCEDURE

MISSING KEY TOTALS-PROCEDURE
```

Exception: Readability options, such as IS or ARE, must be used to separate two expressions in a script. For example, the first line below causes a compile error, while the second does not:

```
READ LMINDD USING KEY SORT 1 "5"

READ LMINDD USING KEY SORT 1 IS "5"
```

Optional Punctuation

The **;** (semicolon) and **.** (period) can both be used to enhance command readability by marking the end of a command. The script compiler ignores these two punctuation marks.

The period can be used to show the end of a command in the same way it shows the end of a sentence. This helps to identify the beginning and end of commands, especially when a command is indented or broken up on several lines.

```
CHANGE APCKREG USING KEY CHK-NO
   MISSING KEY PROCESS IS ADD-CHECK
   PROCESSING IS WRITE-CHECK.
PRINT SCREEN APCHECKS. INPUT SCREEN
      APCHECKS.
```

The semicolon can be used to separate commands that appear together on a single line, or to separate clauses within a command.

```
OPEN SCREEN POSCN1; OPEN SCREEN
POSCN2; OPEN SCREEN POSCN3; ADD
CUSFIL USING KEY CUS-CODE; DUPLICATE
KEY IS CHANGE-CUS; ERROR IS
ERROR-LOG.
```

Again, the important thing is to be consistent in your usage.

Optional Line Spacing

Blank lines can be used to separate logical segments of your script. You may want to separate procedures or groups of procedures from each other, or even separate individual commands. Blank lines are not compiled, and since scripts are compressed before being stored on disk, blank lines do not require storage space.

Optional Indention

Although commands and their clauses can be strung together one after the other without placing the beginning of each command on a new line, this would make a very tight and hard-to-follow script. It is a good idea to indent commands to highlight clauses and reflect any hierarchy in processing. If you develop and follow your own standards for indenting, it can greatly strengthen the structure of your scripts and aid readability.

We recommend that you start the beginning of each command on a new line. You may also want to use standard indention for commands such as the following:

```
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
```

You can also use a combination of indention and punctuation.

The following 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 following example uses the same command and demonstrates how structuring a script can improve 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
```

Data Resources in Script-IV Methods

Script-IV methods have a wealth of data resources to satisfy all the needs of developers. In addition to the Dictionary-IV resources, local 4GL data element names and 3GL variables can also be utilized.

Compile-time Definitions

Compile-time definitions contain data that needs to be resolved when the script is compiled. Compile-time definitions include all definition names specified in data declarations. The data declarations include all formats, screens, views, links, and local data element names. These definitions must be declared before they can be used in a script.

Each data declaration consists of a command followed by a definition name. The command must begin in the left-most column of the line and the definition name must be indented at least one tab stop on the same line. More than one declaration is allowed per line if the definition names are separated by commas or spaces. For more information on the different data declaration commands, see the Script-IV Language section of this manual.

Example:

```
FN LCFORMAT, LMCUSFL
SN LMSCUSFL
```

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

A script name is a compile-time definition when used in the INCLUDE command:

Definition	**Command**
Script Name	DN
Format	FORMAT INCLUDE

When used in script commands, these compile-time names must not be delimited by quotes. They cannot be parameterized or passed to a command in a data element name or variable. They must be hard-coded in your script, for example:

```
OPEN SCREEN LMSCUSFL
```

For more information on data declaration commands, see the descriptions of the DN, FN, LA, LN, SN, and VN commands in the Script-IV Language Reference.

Run-time Definitions

Run-time definitions contain only data resolved at execution time. Run-time definitions include all program, on-line help, and message dictionary definitions, and all 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

These definition names are used in script commands without being declared and can be used in string constants, variables, data element names, or expressions. They can be soft-coded in your script; for example, OPEN MESSAGES "ARMSGS" or OPEN MESSAGES MESSAGE-LIST.

Data Element Names in Script-IV

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

Data element 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 defined in the dictionary or script. 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.

Types of Data Element Names

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

- **Format data element names**

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

  Format data element names can have additional capabilities over local data element names. You can specify many different data types such as string, integer, decimal, date, phone number, social security number, yes/no, or text field, as well as specifying automatic controls on the entry and editing of the data such as prompts, help, valid entries, default values, etc.

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

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

- **Link alias data element names**

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

-

Local data element names

Data element names can be defined in a script using the DN data declaration command. These local data element names do not reside in the dictionary but are handled much like a format data element name. They are not qualified by a format name or other name, and therefore must be unique among any other local data element names or data element 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.

Logical versus Physical Formats

A format is used to describe a structure of data within a data record, usually via a link.

In scripts, a format can be used independently of a data file or link. In this usage, it is referred to as a *logical* format and can function like a data element 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. Logical formats are a powerful feature of Script-IV.

For example, INPUT SCREEN enables an operator to enter data into a format or part of a format, CALL can pass a format to a script. LET can assign values or other format names to a format string. For more information on how formats operate in these commands, see the Script-IV Language Reference.

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

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

Formats for Examples 1 and 2:

Format ONE	Format TWO	Format ABC
CUS-CODE	SLS CODE	CUS-CODE
CUS-NMAE	SLS-NAME	CUS-NAME
CUS-ADDRESS		CUS-ADDRESS
SLS-CODE		SLS-CODE
		SLS-CODE

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 are displayed and manipulated independently of each other.

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 element 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.

Example 1 illustrates a multiple screen design. Example 2 illustrates single screen design using a logical format.

Local versus Global Formats

OPENworkshop allows the developer to create global or local working copies of a Dictionary-IV format. A global instance of a format is created in memory when it is invoked, and remains in memory until the class or method that invoked it terminates (or deletes it). A global instance of a format is available to all other classes and methods within the OPENworkshop environment.

By contrast a local instance of a format, while it remains in memory until the method that invoked it terminates (or deletes it), is not available to any other classes or methods.

To specify that you wish to create or reference a global instance of a format, precede the format name with a # (pound sign). For example, contrast:

```
LET #OEFINVD.INV-DISC = DISCOUNT
LET OEFINVD.INV-DISC = DISCOUNT
```

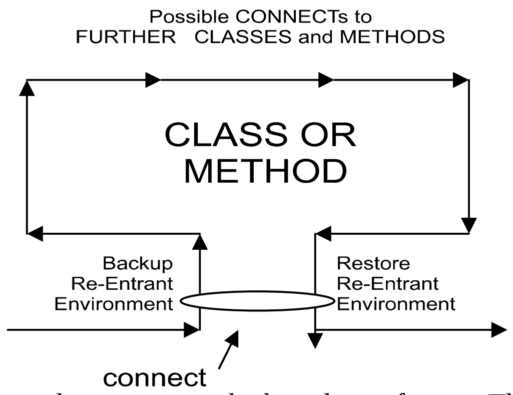
The first example assigns a value to a global instance of format OEFINVD, while the second assigns the value to a local instance.

The Importance of Global Formats in OPENworkshop

The ease with which developers can connect between classes and methods while maintaining the context of current data records is largely due to the existence of global formats. All OPENworkshop views and screens create a global instance of the format on which they are based.

Because the instance of the format is global, any methods that are invoked using the CONNECT directive (either directly from the view or screen or via any chain of other views, screens and menus) are available to those methods. In this way, methods are able to derive the current data context with ease simply by reading the current contents of the #FORMAT.

Handling of Global Formats


Possible CONNECTs to
FURTHER CLASSES and METHODS
CLASS OR METHOD
Backup Re-Entrant Environment
Restore Re-Entrant Environment
connect

All OPENworkshop views and screens must be based on a format. The format name is specified in the definition of the view or screen in Dictionary-IV. Views and screens may, additionally, refer to other formats through joins specified in their definition. On entry to a view or a screen

OPENworkshop creates a global instance of the format on which it is based, and any joined formats (having first made a backup copy if formats were included prior to entry).

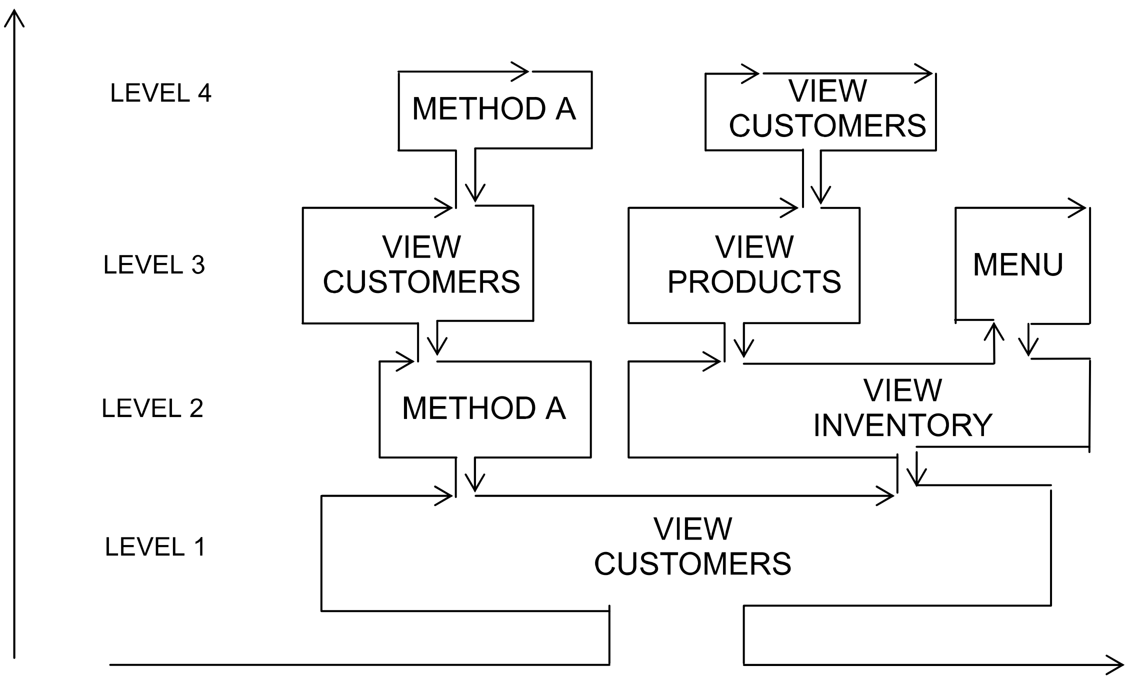
When a CONNECT directive is executed, the global format is populated with the contents of the record currently active, i.e., the row at the cursor position in the case of a view.

This instance of the global format is available to all other classes or methods invoked from this time until either it is replaced by another instance of the global (this happens, for example, when view OEVCUST is invoked while another instance of the same view is open) or the view or screen that created this instance terminates. When a view or screen terminates, the instance of the global format that it created is deleted and the backup copy that it created is restored.

This behavior enables you to create re-entrant applications.

The Control structure section later in this manual discusses how OPENworkshop manages classes and methods through a structure of levels; each CONNECT, explicit or implicit, causes OPENworkshop to move to the next higher level.

Scope of Global Variables



The diagram above depicts how any instance of a global format is available to all higher level classes or methods.

How to Use Global Formats in a Script Method

If you wish to access the contents of an existing global format from within a script method, you must first declare the structure of the format, using the statement:

```
FORMAT INCLUDE #format-name OPT="NONE"
```

This statement, placed in the Procedure Section of the method, loads the attribute table of the format into memory for the run-time environment to use.

If you wish to modify the contents of a global format you must first make a backup copy of its current contents, and must restore the pre-existing copy of the global format before you exit. This discipline ensures that the re-entrance capability of your application is preserved.

If you wish to create an instance of a global format for other classes or methods to access, and have that global format persist after your method has terminated, create it using FORMAT INCLUDE, and set global variable]FMT$ to null before the method terminates. This step prevents OPENworkshop from deleting the global formats included in your method when it terminates. See the information on environment tables in the Variables section later in this manual.

A Warning About Assignments with Global Formats

An assignment of one format to another, such as

```
LET OEFCUST1 = OEFINV
```

Normally compares the two formats and moves the contents of all data element names that are present in both formats from the format in the right hand side to the format in the left hand side of the statement. This is not true for any assignment that has a global format on one side and a local format on the other side. In such cases, OPENworkshop assumes that both formats are compatible and simply moves the data, i.e., simply moves the string.

Example:

```
LET #OEFCUST1 = OEFCUST
```

Moves the data from the local format to the global format. An assignment between two dissimilar formats, such as:

```
LET #OEFCUST1 = OEFINV
```

is likely to lead to unexpected and undesirable results.

I/O and Formats

Script methods perform I/O using directives such as READ, ADD and CHANGE. These always use a link to determine the file to be used, and the memory-resident format definition to be used.

OPENworkshop script methods always use a local format for these I/O functions. It is not possible, for example, to read to a global instance of a format. Instead, you must read to the local format, and then assign the data to the global.

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.

Variables

Variables are data elements that contain values. The value contained in a variable can change during execution. Script-IV recognizes numeric, string, and system variables. You can create numeric and string variables. System variables are numeric or string variables that Script-IV defines for you.

Data Element Names versus Variables

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

Variables are not dictionary-based. They provide an alternative way of holding and manipulating data and do not have the requirements that are built into data element name definitions. However, these very characteristics can defeat a primary purpose of using the system dictionary for data definition: to obtain data independence. Much productivity provided by fourth generation languages is tied to data independence, and this can be lost if variables are not used wisely.

Numeric Variables

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 numeric variables you define.

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

Elastic String Variables

Elastic string variables contain string values. These values are any alphanumeric value; lengths of strings can range from 0 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 string variables, see the Thoroughbred Basic Reference Manual. To establish control of naming conventions for variables, see the descriptions of the .LONGVAR and .SHORTVAR commands in 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 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

Restrictions on Variables

You cannot define a Script-IV reserved word as a variable name. For a list of reserved words, see the Script-IV Language Reference.

OPENworkshop defines a number of string arrays, for which the meanings are reserved. See the String Arrays section of this manual.

Environment Tables

OPENworkshop maintains a number of environment tables to help it manage the run-time data environments and to deliver re-entrance:

]FMT$	**Format Name Table**
]WIN$	**Windows Name Table**
]PUB$	**Resident Public Programs Name Table**
]CH$	**Current List of Open Channels**
]PREC	**Precision**
]PFX$	**OPEN files using a full system path**

On entry to any Script-IV or Thoroughbred Basic Method OPENworkshop saves the current contents of these tables before passing control to the method. On exit from the method it restores the saved status.

You may override this behavior if you wish any changes your method has made to persist. Use the following statements with care:

```
LET ]PUB$=""
```

This statement prevents restoration of the Public Programs Table, leaving any resident public programs you have added in place.

```
LET ]FMT$=""
```

Prevents deletion of the format table(s) you have INCLUDED, leaving added global formats in place.

```
LET ]WIN$=""
```

Leaves any windows that you have opened in place.

```
LET ]CH$=""
```

Prevents open file channels from being closed.

```
LET ]PREC=PRC
```

Prevents precision from being reset to its original setting.

LET]PFX$="*file-name*"

Lets you create a path for OPENing the data file. CONNECT VIEW, CONNECT SCREEN, and OPEN LINK will call the FILE-SUFFIX Method before attempting to open the data-file defined for the Link. The data-file name will be appended to this path and the system will attempt to OPEN this file using the full path. If an error occurs, the normal error processing will take place

Note: The FILE-SUFFIX method now executes for an OPEN LINK *link-name* regardless of whether or not an ampersand (@) has been specified in the *data-file-name*.

Procedures Section

The Procedures Section of a Script-IV method consists of one or more independent procedures containing script commands. These procedures are the main body of the script.

Procedures

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

- identifies the body of commands as a unique procedure within the script.
- must be different from all other procedures in the script and must not conflict with any keywords.

- must begin in column 1, the left-most column of the screen, and must appear on a line by itself.
- can be from 1 through 64 characters long, must not contain space characters, and 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.

Main Procedure:

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 and returns to the previous level. If there are no commands in the main procedure, the script will automatically terminate and return without processing further procedures.

Script-IV Commands

The Script-IV commands are the core of the Script-IV language. Commands are generally grouped together 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 and descriptive of its purpose.

The Script-IV Language Reference provides a detailed description of the Script-IV commands.

Thoroughbred Basic Functions

The OPENworkshop environment additionally allows Script-IV methods to incorporate Thoroughbred Basic functions. These are particularly valuable in manipulation of date and string variables.

A full description of Thoroughbred Basic capabilities can be found in the Thoroughbred Basic Reference Manual.

Types of Scripts

The Dictionary-IV environment supports a number of different types of scripts, each designed to be used under different conditions. While OPENworkshop continues to support most of these types, the vast majority of applications will only need to use Type S, Script Methods.

For the sake of completeness, particularly for those readers interested in migrating a 4GL application to the OPENworkshop environment, the other supported types are described in this section. Scripts written in Dictionary-IV can easily be converted to OPENworkshop methods by changing the type 1 primary scripts to OPENworkshop type S methods. The continuation, overlay and public scripts do not need to have their type changed and, when recompiled, will work properly with the new type S scripts.

You must specify the script type when the script is defined. You can select one of several different types: method, continuation, overlay and public.

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

Script Method

The script method (type S script) is used as a starting point for processing, within the context of this discussion. You can execute a script method, using the CONNECT directive, from:

- An OPENworkshop menu, view or screen using the CONNECT METHOD directive.
- Another script method using the CONNECT METHOD command.
- A 3GL program or Thoroughbred Basic Console Mode, using the RUN PUBLIC "program-name" directive.

This script automatically creates a backup copy of all re-entrant data. When it terminates, the backup data environment is restored and execution returns to the class or method that invoked it.

Continuation Script

This script serves as the continuation of a script method or another continuation script. You can execute this script using the RUN script-name command. This script keeps all files open, retains the values contained in variables, replaces any other program in memory, accepts data from a script method or continuation script, and passes data to certain other script types.

Unlike the Dictionary-IV environment, continuation scripts will return to the caller of its parent primary script on termination.

The Script-IV data environment is shared with the parent script. It must be declared at the beginning of 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.

Overlay Script

This script serves as an overlay to a script method, 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 script method, continuation, or overlay script using the RUN OVERLAY script-name command. The overlay script can accept and return the entire environment between it and its 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 at the beginning of 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.

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 3GL and 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 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.

Diagram of Control Relationships Among Scripts

The following diagram indicates the control relationships between the different types of scripts:

Script Execution Environment

Data Environment

Type of Script	**Data Environment**	
	Script-IV	**3GL**
Method	Initial data declaration. All values are initialized.	All values are initialized.
Continuation	Data declarations must match the parent script.	All values from the executing script are automatically available.
Overlay	Data declarations must match the parent script.	All values are independent of the executing script.
Public Script	Data declarations are independent of the executing script. All values are independent of executing script except those passed and returned.	All values are independent of the executing script except those passed and returned.

Control Structure

Script and Thoroughbred Basic methods are able to connect to other methods, including themselves. Each connect creates a new (higher) level within the OPENworkshop control structure.

Method A connects to Method B, which in turn connects to a further instance of Method A. They both terminate, then Method A connects to a further instance of Method A, and so on.

UPDATE Command

With OPENworkshop the UPDATE command is supported by the Script-IV language. The command combines capabilities of ADD, DELETE and CHANGE commands.

UPDATE is syntactically similar to the CHANGE command. The main difference is a new function string. This string indicates whether the UPDATE command is to "A" - ADD, "U" - CHANGE, or "D" - DELETE the specified record(s).

UPDATE also supports two new areas of Script functionality:

- Link I/O Trigger Processing
- Format-based Delete Values Processing and Audit Processing

The UPDATE command supports Transaction Processing. It creates a transaction to include all I/O triggers that are executed prior to the initial UPDATE being completed. If any of the triggers fail, the UPDATE will rollback any Basic I/O that has taken place. If the triggers pass, the initial UPDATE is completed and all data within the transaction is committed to the database. By default Transaction Processing is enabled. It can be disabled by adding the following entry to the IPLINPUT file:

```
PRM NOTRANS
```

For a summary of functionality see the Comparison of Functionalities chart below:

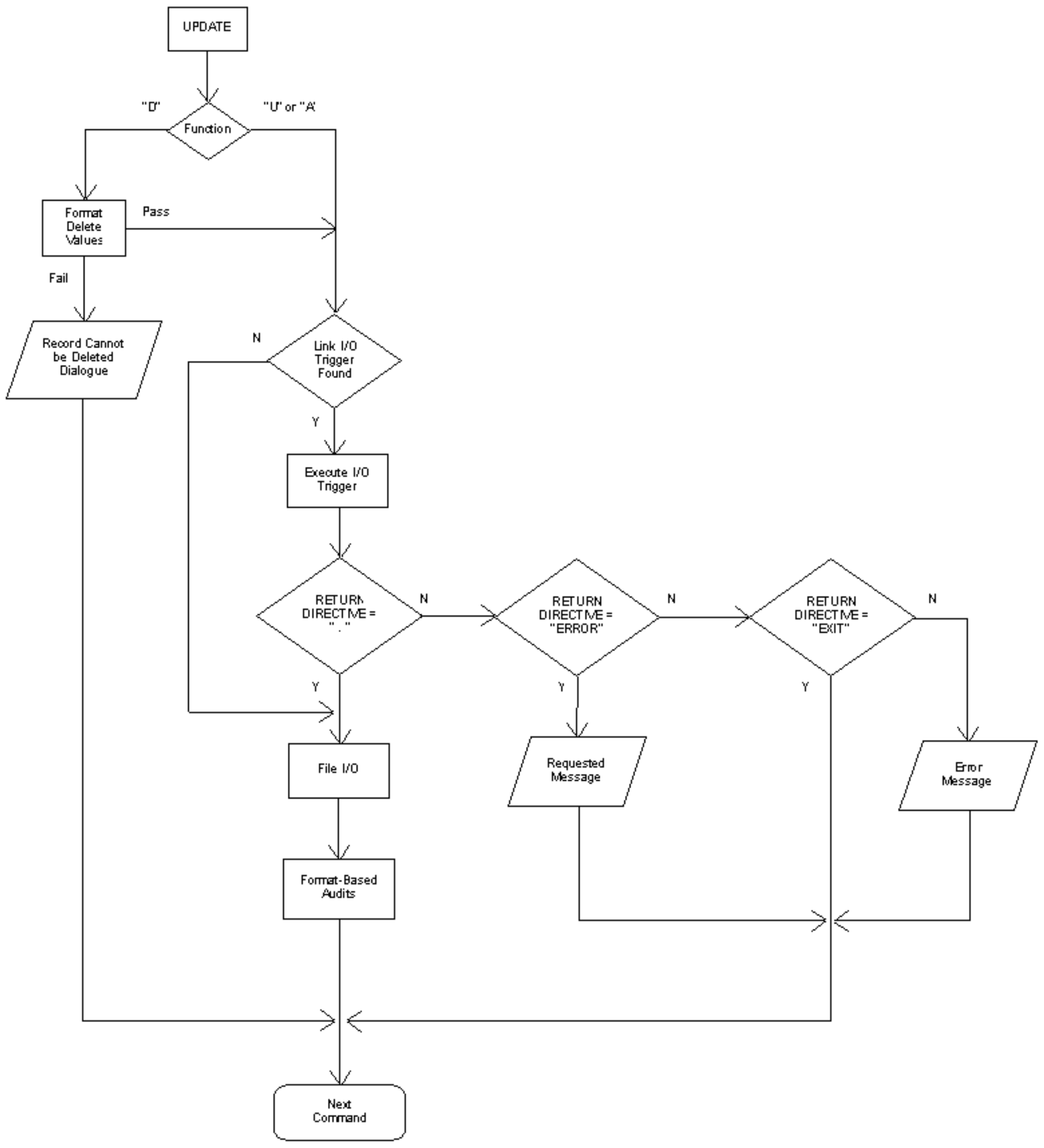
	Format Delete Values	**Format-Based Audit**	**Link I/O Trigger**
Single/Multi Record Maintenance	Yes	Yes	No
CONNECT SCREEN/VIEW	Yes	Yes	Yes
Script ADD	N/A	No	No
Script CHANGE	N/A	No	No
Script DELETE	No	No	No
Script UPDATE	Yes	Yes	Yes

Syntax

```
UPDATE link-name function [USING key-file-access] [index-file-access]
 [MISSING KEY [PROCESS IS] procedure]
 [BUSY [PROCESS IS] procedure]
 [END [PROCESS IS] procedure]
 [ERROR [PROCESS IS] procedure]
 [SELECT [WHEN] condition] [PROCESSING [IS] procedure]
 [TEXT "text-id" [WINDOW window-options]]
 [RETRY [IS] retry-code-string]

function =  "A" - Add record.  Record range/access not valid.
            "U" - Change record.    Record range/access valid
            "D" - Delete record.  Record range/access not valid.
```

Examples

```
UPDATE OELINVH "U"            ! Change sales rep and/or
   USING KEY SORT 1           ! customer key for all
   RANGE FROM OLDK$           ! invoice records
      TO OLDK$                ! using old
   PROCESSING IS INVHUPDATE   ! customer key.
```



Example of Script-IV Method (this example is on your system)

```
LN OELINVH, OELINVN, OELCUST, OELSLRP
FN OEFINVD INVDTL

FORMAT INCLUDE #OEGF, OPT="NONE"       ! Include global flags defn.
FORMAT INCLUDE #OEFINVN,               ! Include
   OPT="NONE"                          !  inventory fmt.

LET FUNC$=MSG1$[0],                    ! Set: I/O function
   CUSTC$=#OEGF.CUSTC,                 !  Cust code changed flag
   SRC$=#OEGF.SRC,                     !  Sales rep code changed flag
   INVENC$=#OEGF.INVENC,               !  Inventory code changed flag
   INVHD$=#OEGF.INVHC                  !  Invoice header delete flag
```

```
IF POS(FUNC$="AUD")=0 OR              ! If not add/chng./del. or
      CUSTC$="X" OR                   !    customer code changed or
      SRC$="X" OR                     !    sales rep code changed or
      INVENC$="X" THEN                !    inventory code changed
   LET MSG1$[0]="."                   !  Then set ok return status.
   TERMINATE                          !  Get out.
ENDIF                                 ! endif

IF #OEFINVH.INV-PRINTED-FLAG =        ! If invoice printed
      "P" THEN
   LET MSG1$[0]="ERROR OEERM01"       !  Print no mods allowed msg.
   TERMINATE                          !  Get out.
ENDIF                                 ! endif

IF INVHD$="X" THEN                    ! If invoice header change
   PRINT @(0,0),MSG2$[1],             !  Show line items.
ENDIF                                 ! endif

LET OEFINVD=MSG2$[0],                 ! Set new invoice detail record
   DNEW  = OEFINVD.INV-LEXTEN,        !         line extension amount
   TNEW  = OEFINVD.INV-LTAX,          !         line tax amount
   QNEW  = OEFINVD.INV-QTY,           !         quantity
   SRNEW$ = OEFINVD.SR-CODE,          !         sales rep code
   CCNEW$ = OEFINVD.CUST-CODE,        !         customer code
   ICNEW$ = OEFINVD.ITEM-CODE,        !         inventory item code
   OEFINVD= FMD("#OEFINVD"),          ! Set old invoice detail record
   DOLD  = OEFINVD.INV-LEXTEN,        !         line extension amount
   TOLD  = OEFINVD.INV-LTAX,          !         line tax amount
   QOLD  = OEFINVD.INV-QTY,           !         quantity
   SROLD$ = OEFINVD.SR-CODE,          !         sales rep code
   CCOLD$ = OEFINVD.CUST-CODE,        !         customer code
   ICOLD$ = OEFINVD.ITEM-CODE         !         inventory item code

IF FUNC$="U" THEN                     ! If update
   LET DELTD = DNEW - DOLD,           !  Set delta: dollars.
   DELTQ = QNEW - QOLD,               !             quantity.
   DELTT = TNEW - TOLD                !             tax.
ELSE                                  ! else
   LET DELTD = OEFINVD.INV-LEXTEN,    !  Set line item: dollars
      DELTQ = OEFINVD.INV-QTY,        !                 quantity
      DELTT = OEFINVD.INV-LTAX        !                 tax
ENDIF            ! endif

IF CVT(ICOLD$,128)="" THEN            ! If old inventory item = space
   LET ICOLD$=ICNEW$                  !  Set it to new item
ENDIF                                 ! endif

IF FUNC$="D" THEN                     ! If delete being done
   LET DELTD =: * (-1),               !  Reverse delta
      DELTQ =: * (-1),                !    value
      DELTT =: * (-1)                 !      sign
ENDIF            ! endif
```

```
IF DELTD OR DELTQ OR                   ! If dollars or qty or
      SROLD$SRNEW$ OR                  !     sales rep code or
      CCOLD$CCNEW$ OR                  !     customer code or
      ICOLD$ICNEW$ THEN                !     item code changed

   IF (DELTD OR DELTQ) AND             !  If dollars or qty changed &
      INVHD$"X" THEN                   !      invoice header not being
                                       !      deleted
         OPEN LINK OELINVH             !    Open invoice header link
         LET OEFINVH.INV-NUM =         !    Set invoice header
            OEFINVD.INV-NUM            !      invoice number.
         UPDATE OELINVH "U"            !    Update invoice
            PROCESSING IS INVHUPDATE   !      header amount.
         CLOSE LINK OELINVH            !    Close invoice header link
   ENDIF                               !  endif

   IF DELTD OR DELTQ OR                !  If dollars or qty or
         CCOLD$CCNEW$ THEN             !        customer code changed
      OPEN LINK OELCUST                !    Open customer file link
      LET OEFCUST.CUST-CODE =          !    Set customer code
         CCOLD$                        !      to be updated.
      UPDATE OELCUST "U"               !    Update old customer
         PROCESSING IS CUSTUPDATE      !      record
         BUSY PROCESS CUST-BUSY        !    Update format if busy

      IF CCOLD$  CCNEW$ THEN           !    If cust code was changed
         LET OEFCUST.CUST-CODE =       !      Set new customer
            CCNEW$                     !        to be updated
         UPDATE OELCUST "U"            !      Update new
            PROCESSING IS              !        customer
               CUSTUPDATE1             !          record
      ENDIF                            !    endif

      CLOSE LINK OELCUST               !    Close customer file link
   ENDIF                               !  endif

   IF DELTD OR DELTQ OR                !  If dollars or qty or
         SROLD$SRNEW$ THEN             !      sales rep code changed
      OPEN LINK OELSLRP                !    Open sales rep file link.
      LET OEFSLRP.SR-CODE =            !    Set sales rep code
         SROLD$                        !      to be updated.
      UPDATE OELSLRP "U"               !    Update old sales
         PROCESSING IS SLSRPUPDATE     !      rep record.

      IF SROLD$  SRNEW$ THEN           !    If sales rep code changed
         LET OEFSLRP.SR-CODE =         !      Set new sales rep code
            SRNEW$                     !        to be updated.
         UPDATE OELSLRP "U"            !      Update new
            PROCESSING IS              !        sales rep
               SLSRPUPDATE1            !          record.
      ENDIF                            !    endif

      CLOSE LINK OELSLRP               !    Close sales rep file link
   ENDIF                               !  endif
   IF DELTD OR DELTQ OR               !  If dollars or qty or
         ICOLD$ICNEW$ THEN            !      inventory code changed
      OPEN LINK OELINVN               !    Open inventory file link.
      LET OEFINVN.ITEM-CODE =         !    Set old inventory code
         ICOLD$                       !      to be updated.
      UPDATE OELINVN "U"              !    Update old
         PROCESSING IS INVENUPDATE    !      inventory record.

      IF ICOLD$  ICNEW$ THEN          !    If inventory code changed
         LET OEFINVN.ITEM-CODE =      !      Set new inventory code
            ICNEW$                    !        to be updated.
         UPDATE OELINVN "U"           !      Update new
            PROCESSING IS             !        inventory
               INVENUPDATE1           !          record.
      ENDIF                           !    endif

      CLOSE LINK OELINVN              !    Close inventory file link.
   ENDIF                              !  endif

ENDIF                                 ! endif

LET MSG1$[0]="."                      ! Set ok return status.

INVHUPDATE                            ! Invoice header update
   LET OEFINVH.INV-AMOUNT=: + DELTD +
      DELTT

CUST-BUSY                             !
   LET OEFCUST = #OEFCUST             ! Update the
   DO CUSTUPDATE                      !  data record
      LET #OEFCUST = OEFCUST          !    format.
      LET RETRY = "C"                 ! Continue with next directive

CUSTUPDATE                            ! Old
   LET OEFCUST.CUST-SALES=: + DELTD   !  customer
                                      !    total
   IF CCOLD$CCNEW$ THEN               !      sales
      LET OEFCUST.CUST-SALES=: - DNEW !        update.
   ENDIF                              !

CUSTUPDATE1                           ! New customer
   LET OEFCUST.CUST-SALES=: + DNEW    !  sales update.

SLSRPUPDATE                           ! Old sales
   LET OEFSLRP.SR-SALES=: + DELTD     !  rep record
   IF SROLD$SRNEW$ THEN               !    sales
      LET OEFSLRP.SR-SALES=: - DNEW   !      update.
   ENDIF                              !

SLSRPUPDATE1                          ! New sales rep record
   LET OEFSLRP.SR-SALES=: + DNEW      !  sales update.
```

```
INVENUPDATE                                ! Old inventory
   LET OEFINVN.ITEM-SALES=: + DELTD        !  record
   IF ICOLD$ICNEW$ THEN                    !    sales
      LET OEFINVN.ITEM-SALES=: -DNEW       !      update.
   ENDIF                                   !

INVENUPDATE1                               ! New inventory
   LET OEFINVN.ITEM-SALES=: + DNEW         !  record sales update.
```

Creating Thoroughbred Basic Methods

Methods may be written in Thoroughbred Basic. Thoroughbred Basic capabilities are described in the Thoroughbred Basic Reference Manual.

Note: Thoroughbred Basic Methods must be maintained in Source-IV.

The scope of this section is limited to an explanation of factors that affect the use of Thoroughbred Basic in OPENworkshop.

Thoroughbred Basic Methods in Source-IV

When creating a Thoroughbred Basic method in Source-IV, specify it as type M.

CONNECT in Thoroughbred Basic Methods

The CONNECT directive is not valid in Thoroughbred Basic, and the Thoroughbred Basic environment will not automatically create and dimension the string arrays used to communicate parameters and messages. The Thoroughbred Basic method must dimension the required arrays and CALL a program as follows:

CONNECT	**CALL**	**ARRAYS**
HELP	004, 001Z	HELP$[4]
MENU	001, 001Z	MENU$[3]
QUERY	00R	QUERY$[11]
REPORT	00R	REPORT$[11]
SCREEN	002A	SCREEN$[17]
VIEW	003A	VIEW$[19]

Please see the on-line help for a description of the CALL lists for these calls.

Thoroughbred Basic Language Restrictions

The following directives must not be used in OPENworkshop:

```
BEGIN
CLEAR
CLOSE
END
FORMAT DELETE ALL
PRINT 'WC', or 'WO'
```

NOTE: System format #IDSV is automatically included.

If a Thoroughbred Basic method needs to open a channel, the UNT function must be used to obtain the next available channel.

CUEXIT is a standard routine, which will close all channels opened by the program, delete any windows, which were created, and set the precision back. For more information, see the Associated Systems section of this manual.

ASSOCIATED SYSTEMS

This section provides information about systems and facilities that can be utilized by OPENworkshop applications:

- Testing and Debugging
- The Cross Reference system
- Gateway for Windows
- Security
- Controlling access to Developer facilities

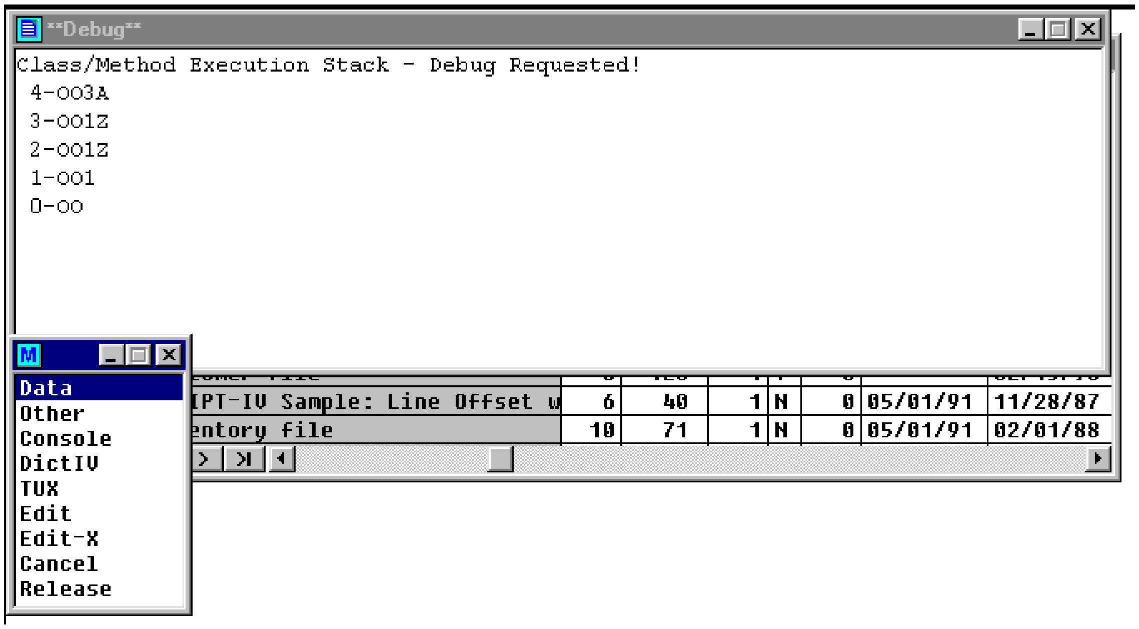
Testing and Debugging Environment

OPENworkshop provides testing and debugging facilities. System activity can be traced and trapped, and data values investigated using a set of run-time utilities. To make the debugging environment available, ensure that the IPLINPUT file contains the statement:

```
PRM DEBUG=OOZ00
```

Activating the Debugging Environment

At any time during development or testing of an OPENworkshop application with graphical user interface (GUI) active, press **Ctrl-B** to invoke the debugging environment. In windows, the following message will be displayed:



```
Class/Method Execution Stack - Debug Requested!
 4-003A
 3-001Z
 2-001Z
 1-001
 0-00
```

```
IPT-IV Sample: Line Offset w   6   40   1 N   0 05/01/91 11/28/87
entory file                   10   71   1 N   0 05/01/91 02/01/88
```

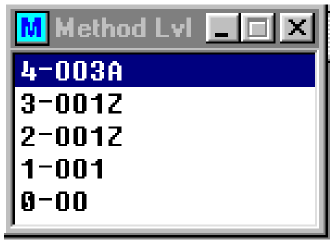
The highest number displayed provides the interrupted level. In the example above, the number is **9**.

OPENworkshop displays the current stack of classes and methods in progress. Active classes and methods are displayed with the highest numbers being the most recently invoked. A short menu displays the options available at this stage:

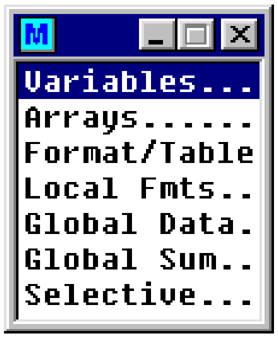
Option	Action
Data	Allows the developer to investigate the current data environment.
Other	See the Other debug functions section later in this manual.
Console	Enters the Thoroughbred Basic Environment.
DictIV	Presents the Dictionary-IV Menu.
TUX	Enters Thoroughbred's TUX utility.
Edit	Enters Source-IV for the program that is interrupted.
Edit-X	Enters Source-IV for general program editing.
Cancel	Returns to executing the current program.
Release	Exits the Thoroughbred Environment.

Data

When you select **Data**, the method levels display:

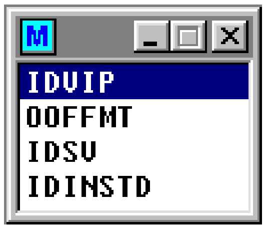


Select the method level and the following displays:

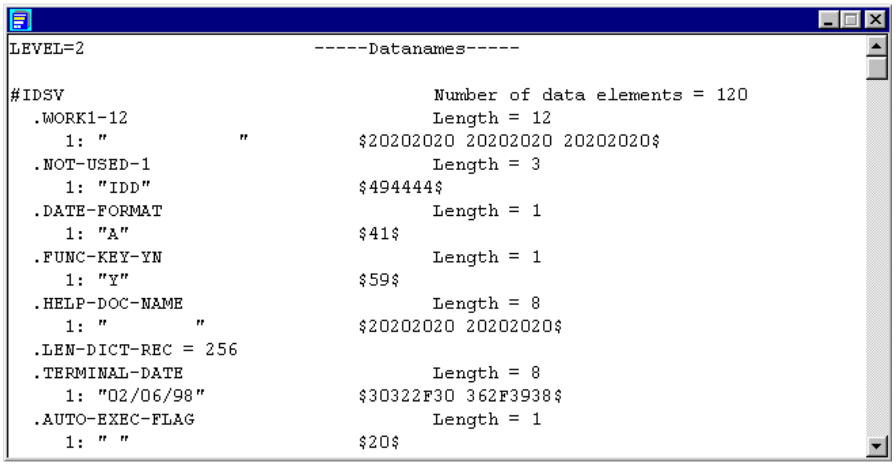


Option	Action
Variables	Displays the status of all variables in the current environment.
Arrays	Displays the status of all arrays in the current environment.
Format/Table	Displays global formats.
Local Fmts	Displays Script formats containing links.
Global Data	Displays the status of all global data.
Global Sum	Displays size information on global data.
Selective	Allows a selection of data types to be displayed.

The following diagram shows a window displayed after selecting **Format/Table** from the menu.



Make your selection and the system displays the following:



```
LEVEL=2                      -----Datanames-----

#IDSV                                   Number of data elements = 120
  .WORK1-12                             Length = 12
     1: "            "           $20202020 20202020 20202020$
  .NOT-USED-1                           Length = 3
     1: "IDD"                    $494444$
  .DATE-FORMAT                          Length = 1
     1: "A"                      $41$
  .FUNC-KEY-YN                          Length = 1
     1: "Y"                      $59$
  .HELP-DOC-NAME                        Length = 8
     1: "        "               $20202020 20202020$
  .LEN-DICT-REC = 256
  .TERMINAL-DATE                        Length = 8
     1: "02/06/98"               $30322F30 362F3938$
  .AUTO-EXEC-FLAG                       Length = 1
     1: " "                      $20$
```

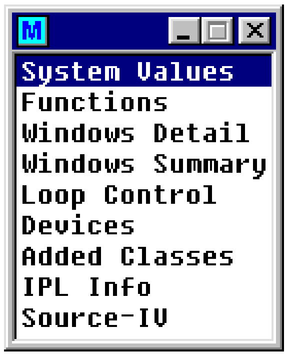
The window is displayed in text editing mode, allowing you to page through the file, or enter a search for values or data element names.

This example displays data element name information for Level 2 and the contents of the global data element name #IDSV. To investigate the data element name values for a different level, move the cursor to the level displayed, for example, over the 2 in the example above, type in the level you want to check, and select **End (F4)**.

To search for a data element name or any other string in the file, select **Search (F10)**, provide the name to be searched for, preceded by a " (quote) and select **OK (Enter)**.

Other

When you select **Other** the system displays the following menu:



The options shown above give you access to the items as indicated in the menu and below. As with the data environment, you can review the values at different levels by modifying the level indicator and selecting **End (F4)**.

Item	**Information**
System Values	Values of all system variables in the environment.
Functions	Location and use of functions.
Windows Detail	Detailed status of all windows in the environment.
Windows Summary	A summary of the above.
Loop Control	Status of any loop control statements that are active, e.g., WHILE, GOSUB, and so on.
Devices	Channel assignments for files, printers, terminals, and other device control status information.
Added Classes	Lists all classes or methods that have been added to the environment.
IPL Info	Current status of IPLINPUT parameters.
Source-IV	Displays the Source-IV menu. (For more information see the Source-IV Reference Manual.

Breakpoints

Breakpoints can easily be added to methods to trap any combination of program path or condition. A convenient way to create a breakpoint is to edit the method to insert code that tests for the required condition, then executes a statement of the kind:

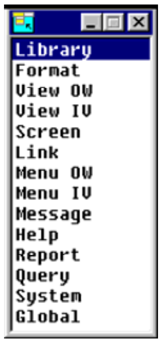
```
INPUT "Breakpoint Reached",A$
```

When this message appears, press **Ctrl-B** to enter the debugging environment. On exit from the debugging environment, the interrupted method continues execution.

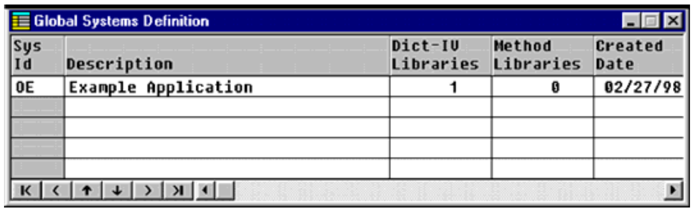
Global Cross Reference System

The **Global** option on the Dictionary-IV menu allows you to access the OPENworkshop global cross-reference system. To use the system you first create cross-reference tables containing the references you need, then view them. A new set of tables can be created whenever you wish.

Select **1-Edit** from the Development Menu or press **F1** from any Dictionary-IV menu. The system displays the Class pop-up window:

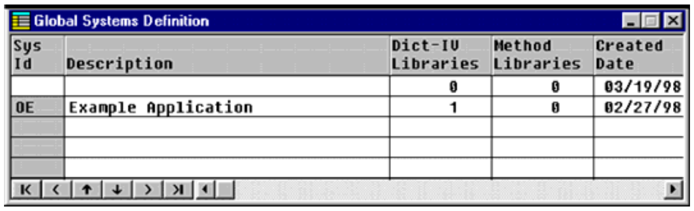


Select **8-Global**. When you first enter this system you must create the cross-references you require. The system displays the following:



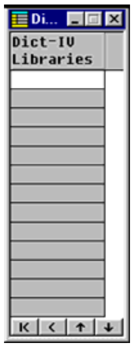
Sys Id	Description	Dict-IV Libraries	Method Libraries	Created Date
OE	Example Application	1	0	02/27/98

Select **Line Insert** to add a new line to the Global Systems Definition, enter the System ID and description.



Sys Id	Description	Dict-IV Libraries	Method Libraries	Created Date
		0	0	03/19/98
OE	Example Application	1	0	02/27/98

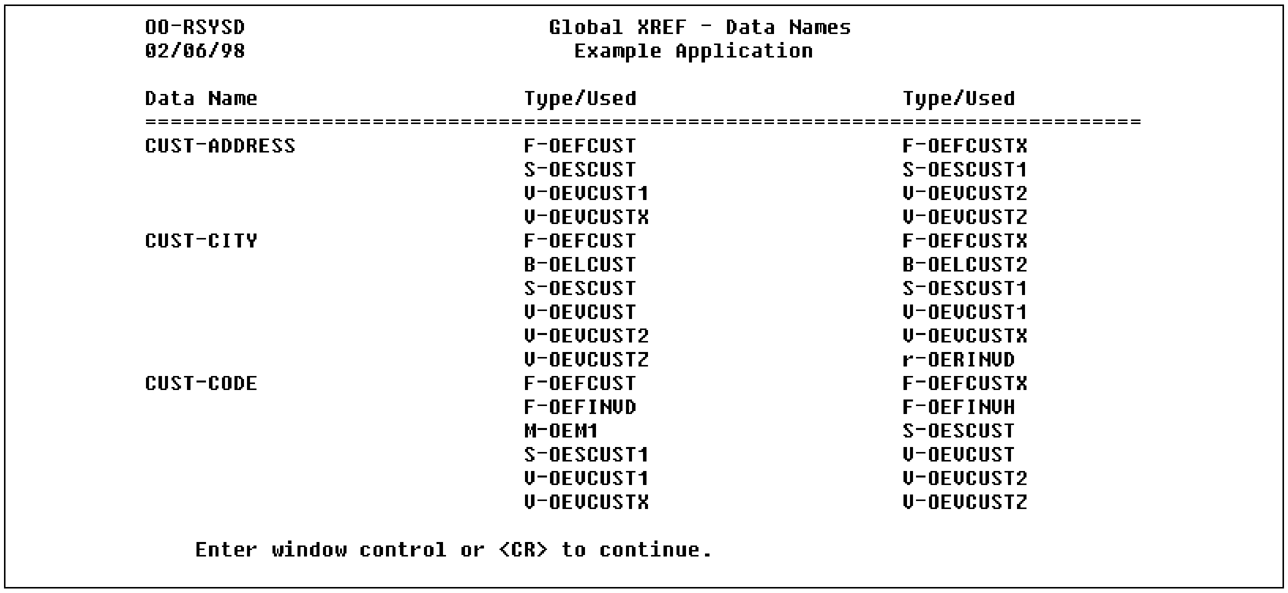
In the Dictionary-IV Libraries column press the **Enter** key and type the names of the libraries you wish to add to the list shown. Note that more than one library can be specified in this list. Enter the Library. Press **F2** to display the Library View.



Press the **Tab** key. Press the **Enter** key. Use the same procedure to add the Method libraries column, press **Tab** key, and then press the **Enter** key to add the Method libraries.

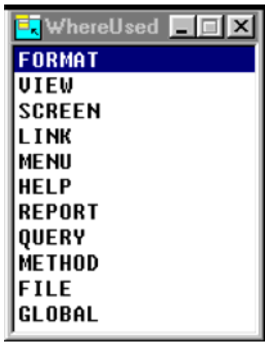
Move the cursor to the System ID column and select the **Create (F5)** option to build the cross-references.

You may also press **F16** to create a report:

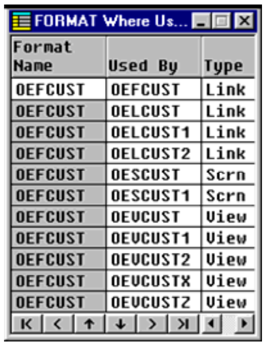


```
OO-RSYSD                        Global XREF - Data Names
02/06/98                          Example Application

Data Name                     Type/Used                     Type/Used
===============================================================================
CUST-ADDRESS                  F-OEFCUST                     F-OEFCUSTX
                              S-OESCUST                     S-OESCUST1
                              V-OEVCUST1                    V-OEVCUST2
                              V-OEVCUSTX                    V-OEVCUSTZ
CUST-CITY                     F-OEFCUST                     F-OEFCUSTX
                              B-OELCUST                     B-OELCUST2
                              S-OESCUST                     S-OESCUST1
                              V-OEVCUST                     V-OEVCUST1
                              V-OEVCUST2                    V-OEVCUSTX
                              V-OEVCUSTZ                    r-OERINVD
CUST-CODE                     F-OEFCUST                     F-OEFCUSTX
                              F-OEFINVD                     F-OEFINVH
                              M-OEM1                        S-OESCUST
                              S-OESCUST1                    V-OEVCUST
                              V-OEVCUST1                    V-OEVCUST2
                              V-OEVCUSTX                    V-OEVCUSTZ

    Enter window control or <CR> to continue.
```

Once the cross-references are built, press **F1** in the **Sys Id** column to explore the cross-reference information that is available through the following WhereUsed menu:



Each option allows you to select a class or other set of system components to review. For example, the FORMAT option displays all formats in the system and shows where they are used. Select **FORMAT** to produce the FORMAT Where Used View, for example:

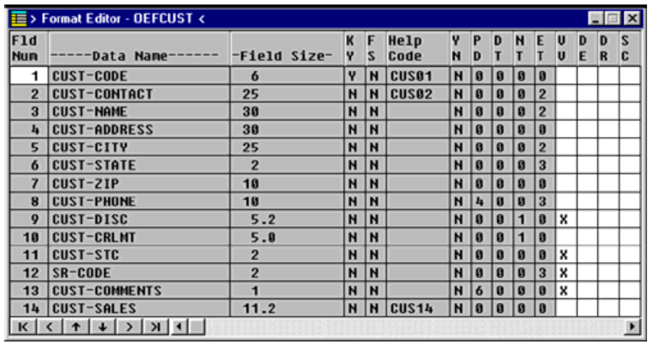


FORMAT Where Us...

Format Name	Used By	Type
OEFCUST	OEFCUST	Link
OEFCUST	OELCUST	Link
OEFCUST	OELCUST1	Link
OEFCUST	OELCUST2	Link
OEFCUST	OESCUST	Scrn
OEFCUST	OESCUST1	Scrn
OEFCUST	OEVCUST	View
OEFCUST	OEVCUST1	View
OEFCUST	OEVCUST2	View
OEFCUST	OEVCUSTX	View
OEFCUST	OEVCUSTZ	View

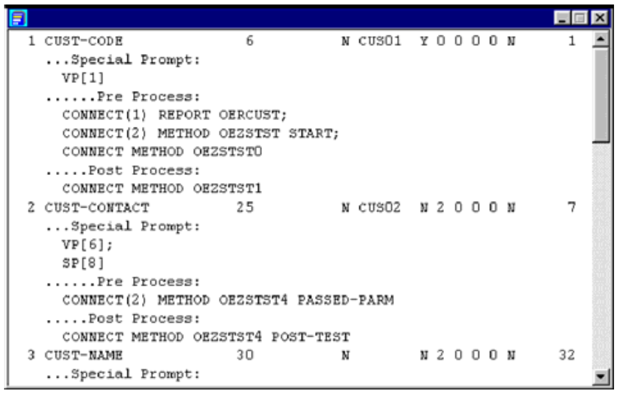
This view shows where every Format in the selected system is used and the type of system component that uses it. Note that all of the classes themselves can be displayed or edited from this where-used view. Move the cursor to the required item and select **Edit (F1)** or **Display (F2)**.

F1 to Edit



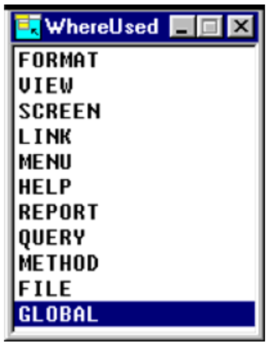
Fld Num	-----Data Name------	-Field Size-	K Y	F S	Help Code	Y N	P D	D T	N T	E T	V V	D E	D R	S C
1	CUST-CODE	6	Y	N	CUS01	N	0	0	0	0				
2	CUST-CONTACT	25	N	N	CUS02	N	0	0	0	2				
3	CUST-NAME	30	N	N		N	0	0	0	2				
4	CUST-ADDRESS	30	N	N		N	0	0	0	0				
5	CUST-CITY	25	N	N		N	0	0	0	2				
6	CUST-STATE	2	N	N		N	0	0	0	3				
7	CUST-ZIP	10	N	N		N	0	0	0	0				
8	CUST-PHONE	10	N	N		N	4	0	0	3				
9	CUST-DISC	5.2	N	N		N	0	0	1	0	X			
10	CUST-CRLMT	5.0	N	N		N	0	0	1	0				
11	CUST-STC	2	N	N		N	0	0	0	0	X			
12	SR-CODE	2	N	N		N	0	0	0	3	X			
13	CUST-COMMENTS	1	N	N		N	6	0	0	0	X			
14	CUST-SALES	11.2	N	N	CUS14	N	0	0	0	0				

F2 to Display

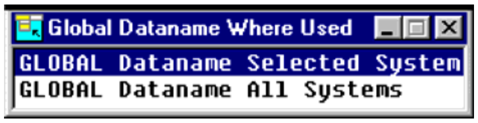


```
 1 CUST-CODE              6           N CUS01  Y 0 0 0 0 N       1
   ...Special Prompt:
     VP[1]
   ......Pre Process:
     CONNECT(1) REPORT OERCUST;
     CONNECT(2) METHOD OEZSTST START;
     CONNECT METHOD OEZSTST0
   .....Post Process:
     CONNECT METHOD OEZSTST1
 2 CUST-CONTACT          25           N CUS02  N 2 0 0 0 N       7
   ...Special Prompt:
     VP[6];
     SP[8]
   ......Pre Process:
     CONNECT(2) METHOD OEZSTST4 PASSED-PARM
   .....Post Process:
     CONNECT METHOD OEZSTST4 POST-TEST
 3 CUST-NAME             30           N        N 2 0 0 0 N      32
   ...Special Prompt:
```

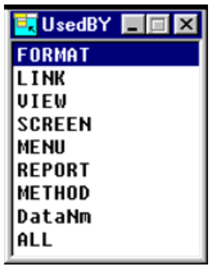
The **GLOBAL** option of the WhereUsed menu shows where data element names are set or used:



Select **GLOBAL** to display the Global Datename Where Used menu:

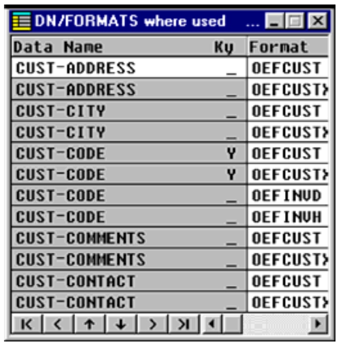

Global Datename Where Used
GLOBAL Dataname Selected System
GLOBAL Dataname All Systems

Selecting the highlighted option displays the following menu:



As before, you can review or change the definitions of items from these views. These cross-reference views are extremely flexible and powerful as can be seen from the following examples.

The **Used By Format** option shows which formats contain a definition of which data element names throughout the application.

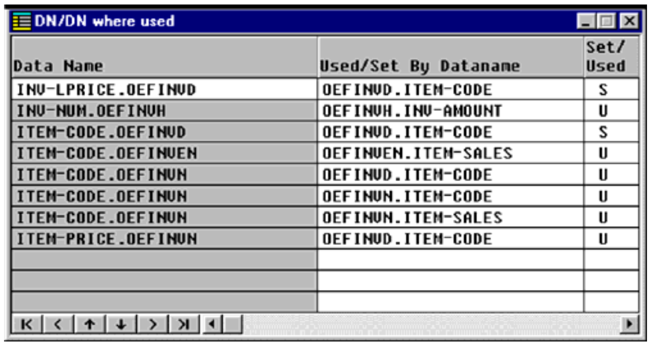


DN/FORMATS where used ...

Data Name	Ky	Format
CUST-ADDRESS	_	OEFCUST
CUST-ADDRESS	_	OEFCUST›
CUST-CITY	_	OEFCUST
CUST-CITY	_	OEFCUST›
CUST-CODE	Y	OEFCUST
CUST-CODE	Y	OEFCUST›
CUST-CODE	_	OEFINVD
CUST-CODE	_	OEFINVH
CUST-COMMENTS	_	OEFCUST
CUST-COMMENTS	_	OEFCUST›
CUST-CONTACT	_	OEFCUST
CUST-CONTACT	_	OEFCUST›

The **Used By Method** option shows all data element names that are used by methods, and whether the method simply read the data name or may also set its value.

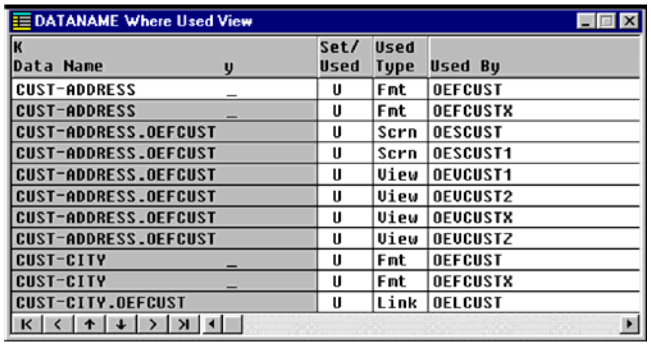
The **Used By Data Name** option shows data element names that are set or used from other data element names. Typically these uses are to be found in data element name pre-process or post-process definitions. The normal order of presentation of the format name and data element name in the left hand column of this display is reversed, so that data element names are listed in alphabetically sorted order.



DN/DN where used

Data Name	Used/Set By Dataname	Set/ Used
INV-LPRICE.OEFINVD	OEFINVD.ITEM-CODE	S
INV-NUM.OEFINVH	OEFINVH.INV-AMOUNT	U
ITEM-CODE.OEFINVD	OEFINVD.ITEM-CODE	S
ITEM-CODE.OEFINVEN	OEFINVEN.ITEM-SALES	U
ITEM-CODE.OEFINVN	OEFINVD.ITEM-CODE	U
ITEM-CODE.OEFINVN	OEFINVN.ITEM-CODE	U
ITEM-CODE.OEFINVN	OEFINVN.ITEM-SALES	U
ITEM-PRICE.OEFINVN	OEFINVD.ITEM-CODE	U

The **Used By All** option shows all uses of data element names by all classes and methods in the application.


DATANAME Where Used View

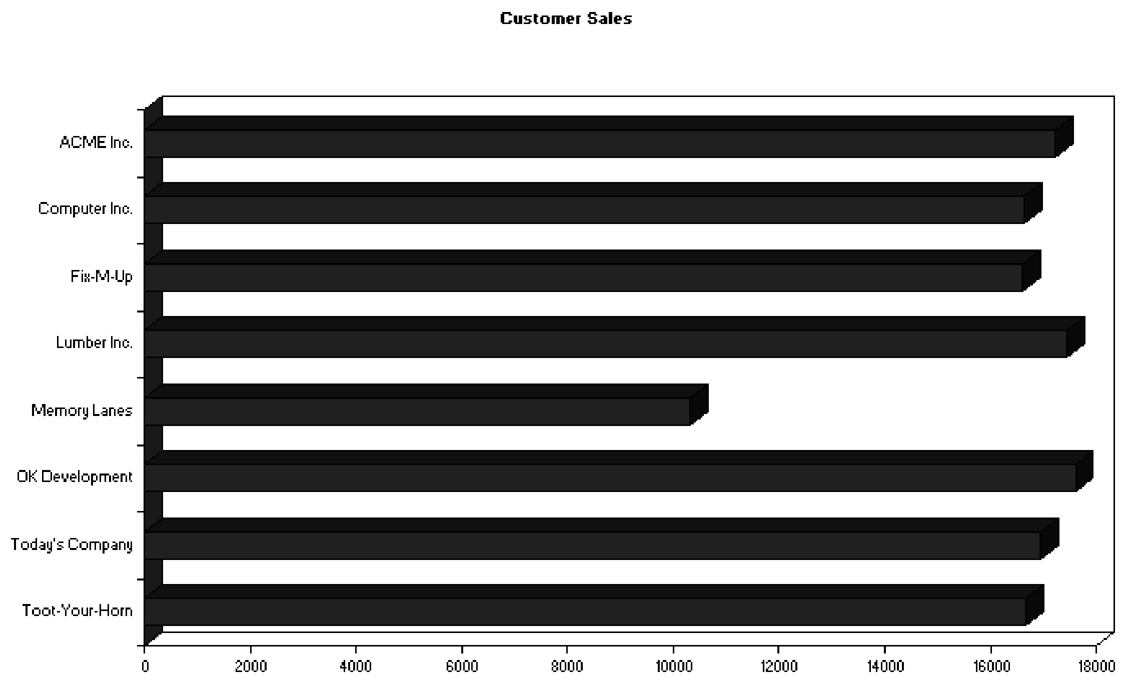
K Data Name	y	Set/ Used	Used Type	Used By
CUST-ADDRESS	_	U	Fmt	OEFCUST
CUST-ADDRESS	_	U	Fmt	OEFCUSTX
CUST-ADDRESS.OEFCUST		U	Scrn	OESCUST
CUST-ADDRESS.OEFCUST		U	Scrn	OESCUST1
CUST-ADDRESS.OEFCUST		U	View	OEVCUST1
CUST-ADDRESS.OEFCUST		U	View	OEVCUST2
CUST-ADDRESS.OEFCUST		U	View	OEVCUSTX
CUST-ADDRESS.OEFCUST		U	View	OEVCUSTZ
CUST-CITY	_	U	Fmt	OEFCUST
CUST-CITY	_	U	Fmt	OEFCUSTX
CUST-CITY.OEFCUST		U	Link	OELCUST

Communication with Gateway for Windows

Gateway for Windows enables a host application to communicate directly with any application software running in the Microsoft Windows environment. It uses DDE communications to transport data between the software running in the Windows environment and the host application. OPENworkshop applications are able to communicate through Gateway for Windows.

You can communicate data through Gateway for Windows using Report-IV, Query-IV, or a method. For more information on Gateway for Windows, see the Gateway for Windows Reference Manual.

The following chart was produced by Microsoft Excel, based on the output from a Report-IV report.



The report that produced the chart follows:

```
ENTRY-SECTION

I111  DIM M$[5];
:     R5 = 0
!                           Start Excel running

I112  M$[1] = "I3";
:     M$[2] = "EXCEL";
:     M$[3] = "SYSTEM";
:     CALL "GWWCOM", M$[ALL]
!                           Set Excel column width

I113  M$[1] = "E";
:     M$[5] = "[SELECT(" + QUO + "C1" + QUO + ")]"
:           + "[COLUMN.WIDTH(20)]"
:           + "[SELECT(" + QUO + "R1C1" + QUO + ")]";
:     CALL "GWWCOM", M$[ALL]
!                           Select sheet 1

I121  M$[1] = "P";
:     M$[3] = "SHEET1"
!                           Termination. Draw chart of the top 8.

T911  M$[1] = "E";
:     M$[3] = "SYSTEM";
:     M$[5] = "[WORKBOOK.INSERT(2)]"
:           + "[CHART.WIZARD(TRUE," + QUO + "SHEET1!R1C1:R8C2" + QUO
:           + ",10,1,2,,,2," + QUO + "Customer Sales" + QUO
:           + "," + QUO + QUO + "," + QUO + QUO + "," + QUO + QUO
:           + ",1,0)]";
:     CALL "GWWCOM", M$[ALL]
!                           Terminate DDE, leave Excel running

T921  M$[1] = "T";
:     CALL "GWWCOM", M$[ALL]

FILE-SECTION

LN    OELCUST SORT BY SORT2
:     SELECT WHEN CUST-CODE LIKE "100*"

CONTROL-SECTION

REPORT-SECTION

!                           For each row send customer name and
!                           sales to Excel worksheet columns 1
!                           and 2.

DC    R5 = R5 + 1;
:     R5$ = STR(R5)

DC    M$[4] = "|R" + R5$ + "C1|"
:           + "|R" + R5$ + "C2|"

DC    M$[5] = "|" + CUST-NAME + "|"
:           + "|" + STR(CUST-SALES) + "|"

DC    CALL "GWWCOM", M$[ALL]
```

Security

The OPENworkshop security system was designed to be a logical layer of security that exists on top of an operating system's security. It is based on UNIX concepts, but accommodations have been made to allow the logical security layer to function with other operating systems such as Windows, and Open VMS.

The primary features are:

- Group level access to menus and selections from menus.
- Group level access to data file links.
- Group level access to fields within a data file.
- Group level access to record levels.
- Drill down views of user IDs and groups that quickly show users within group and groups of which a user is part.
- Prior versions of security that show the delta from one version to the next.
- Reports showing group to user and user to group association.
- Initialization and regeneration of logical security layer from user ID and group files after modifications.

Scope

Security locks may be specified for links, menus and data element names. This is done by specifying a list of groups allowed access. Users who are not part of these groups are denied access. The format of the specification is:

```
[group-nbr1,group-nbr2, . . . ,group-nbrn]
```

Example:

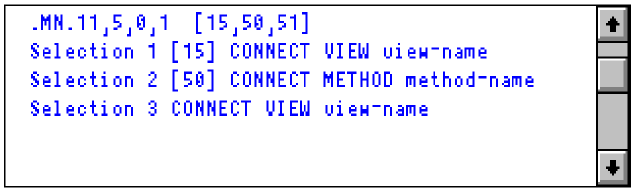
```
[15,25,50,51]
```

In the above example, a menu, link or data element name containing this specification allow access to users who are part of groups 15, 25, 50 or 51. Users who are not part of groups 15, 25, 50, or 51 are denied access.

Note: Menus, links and data element names that do not contain a group security lock are accessible by all groups.

Menus

The security lock specification can be placed in the menu header or on a MENU selection.



```
.MN.11,5,0,1  [15,50,51]
Selection 1 [15] CONNECT VIEW view-name
Selection 2 [50] CONNECT METHOD method-name
Selection 3 CONNECT VIEW view-name
```

Assuming the above menu, the following is true:

- Only users in group 15, 50 or 51 have access to the menu.
- Only users in group 15 can execute selection 1 from the menu.
- Only users in group 50 can execute selection 2 from the menu.
- Any user allowed access to the menu could access selection 3.

Links

The security lock specification can be placed in the link terminal access codes. The group numbers can now contain an R for Read-only access when using CONNECT SCREEN and CONNECT VIEW. CONNECT SCREEN will be forced into Inquiry mode and CONNECT VIEW will run in PRINT VIEW mode.

Example 1:

```
0,T0,T1 [15,50R,51]
```

Users in group 15 or 51 would have access regardless of terminal, whereas group 50 will have Read-Only access in CONNECT SCREEN (Inquiry Mode) and CONNECT VIEW (PRINT VIEW mode).

Example 2:

```
1,T0,T1 [15,50,51]
```

Only users in group 15, 50 or 51 are allowed access when logged in from a terminal other than T0 or T1.

Example 3:

```
[15,50,51]
```

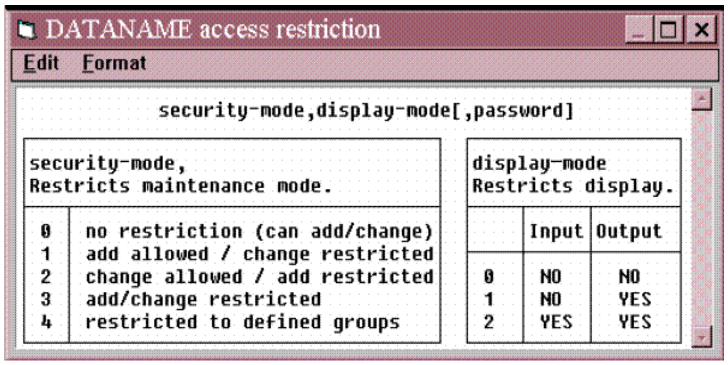
Only users in group 15, 50 or 51 are allowed access regardless of their login terminal.

NOTE: For additional explanation of the terminal access code security option, see the Dictionary-IV Developer Guide. Remember, the group security option shown above can only be specified in the terminal access field.

Data Element Name

Data element name access restrictions are defined in the security attribute field of the data element name definition. Groups not having access through the group security option will be restricted by this specification. The restriction is made up of the following three parts:

```
security-mode,display-mode[,password]
```



security-mode,display-mode[,password]

security-mode, Restricts maintenance mode.	
0	no restriction (can add/change)
1	add allowed / change restricted
2	change allowed / add restricted
3	add/change restricted
4	restricted to defined groups

display-mode Restricts display.

	Input	Output
0	NO	NO
1	NO	YES
2	YES	YES

[,password] overrides security mode but not display mode. If specified, it is required regardless of other security settings. An asterisk (*) can be specified as the password and indicates the password is to be the first 3 characters of the data element name's content.

The security lock specification can be placed in the Data Name security attribute field.

Example 1:

```
3,2 [15,50,51]
```

Users in group 15, 50 or 51 are allowed entry and display access to the data element name. All other users are denied add, change and display access.

Example 2:

```
1,1 [15,50,51]
```

Users in group 15, 50 or 51 have entry and display access to the data element name. All other users are denied change and display access when not adding a new record.

Example 3:

```
[15,50,51]
```

All users have entry and display access to the data element name. This is a useless security specification because no security restriction is specified for users other than 15, 50 or 51.

Record Level

Record Level security allows the access of data records to be restricted to specified group levels. This is accomplished by defining a 2-character field in the desired format with a security code of 4. For more information see Format Definition in the Dictionary-IV Developer Guide.

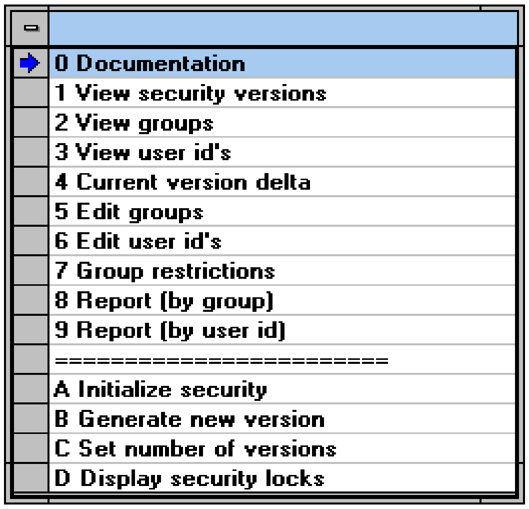
When the value of this field matches a record level restriction key, only the groups defined for the key are allowed to access the data record. This record level security applies to OPENworkshop, Dictionary-IV, and Report-IV

Administration of the Security System

Principles of operation

OPENworkshop security permissions are maintained in tables that define the relationships between user IDs and security group numbers. These tables are used at run time to determine access permissions. The tables are created initially by copying data from the security files in the UNIX operating system.

The OPENworkshop main menu contains the item SECURITY, which is accessible to developers. When that item is selected, the menu below is displayed.



```
0 Documentation
1 View security versions
2 View groups
3 View user id's
4 Current version delta
5 Edit groups
6 Edit user id's
7 Group restrictions
8 Report (by group)
9 Report (by user id)
========================
A Initialize security
B Generate new version
C Set number of versions
D Display security locks
```

Creating security tables

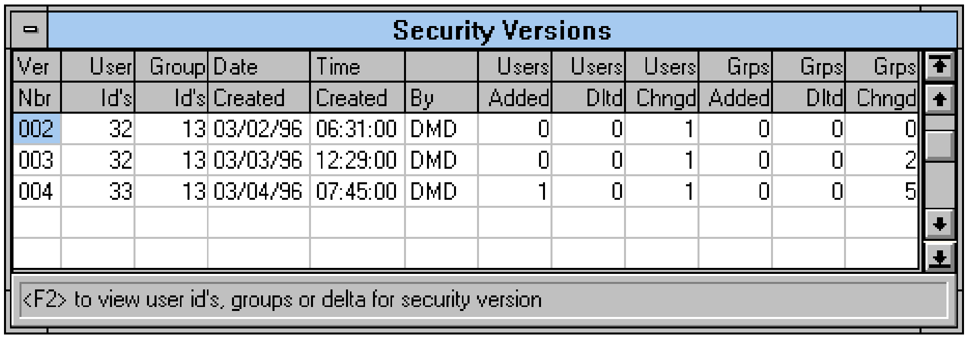
After initial installation, OPENworkshop does not have the security system enabled. To enable security you must:

- Select option A: Initialize security. This option removes any OPENworkshop security files.
- Select option B: Generate new version. This option creates a new version 1 set of security tables and populates them with data taken from /etc/passwd and /etc/group.
- Select option C: Set number of versions. This option controls the amount of history of changes to the security permissions that is maintained by OPENworkshop.
- Ensure the system user ID and group files reflect the desired user/group relationships for the installation. Options 5 and 6 (Edit groups and Edit user ID's) can be used to establish desired relationships in the operating system tables. If not, edit user IDs and groups to the required values, then use option B again to create a new security version. Option B can be used as many times as needed to generate new security versions.

If you are using a UNIX operating system, you must have root permissions to modify the group or user ID file.

Reviewing security permissions

Select option 1 from the security menu to view security versions.



Security Versions

Ver	User	Group	Date	Time		Users	Users	Users	Grps	Grps	Grps
Nbr	Id's	Id's	Created	Created	By	Added	Dltd	Chngd	Added	Dltd	Chngd
002	32	13	03/02/96	06:31:00	DMD	0	0	1	0	0	0
003	32	13	03/03/96	12:29:00	DMD	0	0	1	0	0	2
004	33	13	03/04/96	07:45:00	DMD	1	0	1	0	0	5

<F2> to view user id's, groups or delta for security version

This view allows the administrator to review the changes made between versions, and the current access permissions for groups and the members of those groups.

Group restrictions

Data element name access permissions can be further qualified by using the Group Restrictions option from the security menu. See the Data element name section earlier in this manual for further information on permissions.

Displaying security locks

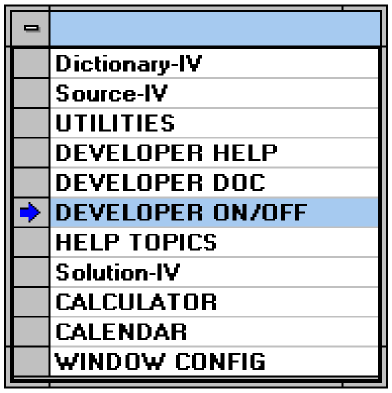
The security locks that have been defined by the developer for an application can be displayed and reviewed using option D from the security menu. Enter a list of one or more libraries to be reported.

This listing shows all security group statements that have been applied to menus, links and data element names in the selected libraries:

```
---- MENUS ---- (OO)

MENU:  OOM0    Main Menu
   Selection:  SOURCE-IV        CONNECT MENU OOM02        [1]
   Selection:  SECURITY         CONNECT MENU OOM05        [1]
MENU:  OOM05   Security Menu                             [O,1]
MENU:  OOM0A   Developer/User menu                       [1]
MENU:  OOMU0A  Sample Matrix Menu                        [50,51]
   Selection:  9       [1] CONNECT VIEW OEVSLSRP
   Selection:  10      [1] CONNECT SCREEN OESSLSRP
   Selection:  11      [1] CONNECT MENU OEMSLSRP
MENU:  OOMU10  Desk Services Menu
   Selection:  DICTIONARY IV         CONNECT MENU OOM01       [1,T]
   Selection:  SOURCE-IV             CONNECT METHOD OOS       [1,T]
   Selection:  DEVELOPER HELP        CONNECT HELP OOM0B       [1,T]
   Selection:  DEVELOPER DOC         CONNECT VIEW DCVDOCA     [1,T]
   Selection:  DEVELOPER ON/OFF      CONNECT METHOD OO0700    [1,T]
MENU:  OOUNX1  Unix command listing
   Selection:  basename..   basename string [ suffix ]\HELP=8UHbasen
   Selection:  cal.......   cal [[ month ] year ]\HELP=8UHcal
   Selection:  calendar..   calendar [ - ]\HELP=8UHcalen
   Selection:  cat.......   cat [ option ] file\HELP=8UHcat
```

Developer Status

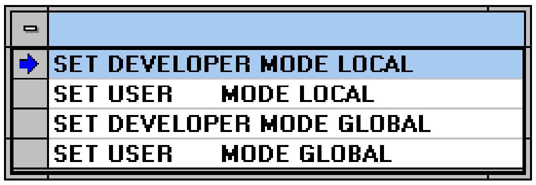
During development of an OPENworkshop application it is very convenient to be able to access and change class definitions quickly. It is also essential that these definitions can be protected from change by other users. This control is provided by the Developer Status.



Developer Status is controlled by an option on the hotkey (**Ctrl-P**) menu displayed above. Access to this item is controlled by menu level security. The user must be a member of security group [01] to gain access to this menu option.

When Developer Status is enabled for a user, that user has access through **F1** to edit views, screens, menus and help; through **F8** to move views, screens, menus and help; and through **F11** to edit data element names.

Selecting **DEVELOPER ON/OFF** displays the menu below. The Developer Status can be set to DEVELOPER or USER for individual logins or for all logins to an application.



The options are described in the following table:

Mode	**Effect**
SET DEVELOPER MODE LOCAL	Gives the current user access to classes controlled by developer mode. This takes place immediately.
SET USER MODE LOCAL	Takes away access to developer mode from the current user.
SET DEVELOPER MODE GLOBAL	Gives developer mode access to all users who have the appropriate security group access.
SET USER MODE GLOBAL	Takes away developer mode from all users. This action takes place with effect from the next time any user logs on.

STRING ARRAYS

String arrays are used throughout OPENworkshop to pass information to methods and to return information to classes that call methods.

Usually these arrays are created automatically for the developer by the class that connects to the method. However, where a Thoroughbred Basic method is connecting to another method (Thoroughbred Basic or script) the caller is responsible for creating and dimensioning the array before the CONNECT can be invoked.

The standard string arrays used in OPENworkshop are explained in detail in the following pages. Each explanation contains the following:

Purpose	The purpose of the array together with any limitations on its use.
Used By	The type of method that uses the information.
Contents	Required entries and their meanings. Where there is a default value for an array entry, it is indicated by an underscore.

The string arrays described in this section are presented in alphabetic order and include the following:

ARM$
FA$
HELP$
KT$
LNK$
MENU$
PP$
QUERY$
REPORT$
SA$
SCREEN$
V$
VIEW$

ARM$

Purpose Passes data and messages between a screen and an after read method.

Used By A screen when a record has been read and before it is displayed for data entry.

Contents

ARM$[0]	On return from the method the entry contains return directives:	
	Null or "."	Continue processing record
	SKIP	Reject record, print "record not found" message
	SKIP-NOMSG	Reject record, don't print message
ARM$[1]	Data Record Read/Extracted	
ARM$[2]	(1,1)	Maintenance mode (A,C,D,I,F,f)
	(2,1)	File type (D,S,I,C)
	(3,1)	Forward (F) / Backward (B) read flag
	(4,1)	Record Extract flag (R,E)
	(5,1)	Key Value Not Found flag (1=found, 0=not found)
	(6,1)	Term-key (binary)
ARM$[3]	Key/Record number used for Read / Extract	

FA$

Purpose Contains the attributes of the format in use by the caller. This array enables the called method to derive the current data context and allows the context to be restored when control returns to the calling object.

The called method must ensure that this array is not modified before it is returned to the caller, or unpredictable behavior may result.

Used By CONNECT METHOD

The calling class creates FA$[ALL]. The method receives string array MSG3$[ALL]

Contents

FA$[0]	Format name.	
FA$[1]	Format header.	
FA$[2]	(1,1)	Format attribute table entry length (ASC 1 byte).
	(2,1)	Number of entries in format table.
	(3,1)	Data name table entry length (ASC 1 byte).
	(4,2)	Format data record length (Binary 2 bytes).
FA$[3]	Format attribute table.	
FA$[4]	Audit string.	
FA$[5]	Pre-processing procedure string.	
FA$[6]	Special prompt string.	
FA$[7]	Preset value string.	
FA$[8]	Valid value string.	
FA$[9]	Delete value string.	
FA$[10]	Security value string.	
FA$[11]	Post process procedure string.	
FA$[12]	IOLIST	
FA$[13]	Field separator elimination table.	
	Note: The field separator elimination table can be used in combination with the RTD function (see the Thoroughbred Basic Reference Manual). This will remove field separators and pad variable length fields. If the format definition does not match the data record, the RTD function will generate ERR=1.	
FA$[14]	Data name table	
	(1,1)	Length of a data name entry.
	(2,1)	Number of data names.
	(3,n)	List of data names.
FA$[15]	Local counters for OO70.	

FA$[16]	File channel numbers.	
	(1,2)	Data file channel number.
	(3,2)	Sort file channel number.
FA$[17]	Data name description table in the current language.	
	Note: When no data name description tables are defined, the data name table will be returned in place of the data name description table, FA$[14]. If the data name description table does not exist in the current language, starting with ENglish, the system will search for and return the first data name description table found.	
	(1,1)	Length of a data name description entry.
	(2,1)	Number of data name descriptions.
	(3,n)	List of data name descriptions.

HELP$

Purpose Passes information to the help subsystem when CONNECT HELP is invoked. It is only necessary to pass HELP$ if the caller wishes to override the defaults or pass substitute parameters. It is not possible to pass HELP$ from a data name or menu.

Used By CONNECT HELP from a method.

Contents

HELP$[0]	Help control data:		
	(1,1) =	"C" -	Display help, allow edit option, pop help window upon termination and return to previous window.
		"X" -	Display help, pop help window upon termination and return to previous window.
		"x" -	Display help and leave the help window displayed and selected.
	(2,1) =	" " -	(space) No heading.
		"C" -	Centered heading.
		"R" -	Right justified heading.
		"L" -	Left justified heading.
		Note: The help description will be used as the displayed heading.	
HELP$[1]	Help code.		
HELP$[2]	Substitute parameters. The help subsystem will substitute values into a help message when the help message is appropriately defined and the substitute parameters are passed through this array element. The syntax for HELP$[2] is: **/*1/Value1;*2/Value2,....**		
	/	is the parameter delimiter (slash character is usual).	
	*1	is any character pair, being the first character string to search for and replace.	
	Value1	is the first replacement string.	
	;	to separate multiple substitution parameters.	
	*2	is any character pair, being the second character string to search for and replace.	
	Value2	is the second replacement string.	
	Note: Any number of replacements may be specified by continuing the pattern.		

KT$

Purpose Passed to trigger methods to communicate information about the new record and changes in sort keys following an update to a data file referenced in a link.

Used By CONNECT METHOD

The calling class creates KT$[ALL]. The method receives string array MSG2$[ALL].

Contents

KT$[0]	New record (format data) string, i.e., the entire contents of the format relating to the file, packed into a string.
KT$[1]	Old primary key (SORT0).
KT$[2]	New primary key (SORT0).
KT$[3]	Old secondary key (SORT1).
KT$[4]	New secondary key (SORT1).
KT$[2n+1]	Old secondary key (SORTn).
KT$[2n+2]	New secondary key (SORTn).

NOTE: When a trigger method is invoked the caller must also supply LNK$, which is used to return directives to the caller. See the description of LNK$ in this section and the OPENworkshop Methods section of this manual.

LNK$

Purpose Contains the attributes of the link in use by the caller. This array enables the called method to derive the current context and allows the data context to be restored when control returns to the calling object.

Except for entry LNK$[0] the called method must ensure that this array is not modified before it is returned to the caller, or unpredictable behavior may result.

Used By CONNECT METHOD

Contents

LNK$[0]	Link message.	
	On entry, contains one of the following:	
	"A"	Add.
	"U"	Update.
	"D"	Delete.
	On return must contain one of the following:	
	"."	Normal exit Requested I/O will be completed.
	Null	Error occurred. Requested I/O will not be completed and an "I/O Not Completed" message will be displayed.
	"ERROR [help-code]"	Requested I/O will not be completed. The message specified by the help code will be displayed and an error will be returned to the caller. If help code is not specified no error message will be displayed and an error condition will be returned to the caller.
	"EXIT"	No error message will be displayed and the requested I/O will not be completed by the OPENworkshop general I/O process. However, it is assumed that the I/O trigger did the I/O and no abnormal condition is assumed.
LNK$[1]	Link name.	
LNK$[2]	Link header.	
LNK$[4]	Terminal access codes.	
LNK$[5]	Operator access codes.	
LNK$[6]	Data file name (after suffix substitution).	
LNK$[7]	Sort file name (after suffix substitution).	
LNK$[8]	Text file name (after suffix substitution).	
LNK$[9]	(1,1)	Primary key length including preset parts.
	(2,n)	Primary key data name numbers not including preset parts.

LNK$[10]	Data name delete value list.	
LNK$[11]	File type:	
	I	Indexed.
	S	Sort.
	D	Direct.
	C	CISAM
LNK$[12]	(1,1)	Number of sort defs.
	(2,1)	Length of sort defs entry.
	(3,n)	Sort definitions.
LNK$[13]	Sort definition table.	
LNK$[14]	Primary key prefix.	
LNK$[15]	Primary key suffix.	
LNK$[16]	Generated link I/O program name.	
LNK$[17]	Mandatory data name number table.	
LNK$[18]	Preprocess table:	
	(1,1)	Data name number.
	(2,1)	Function key value.
LNK$[19]	Audit data name number table.	
	(1,1)	Data name number.
	(2,1)	Type audit.
	(3,8)	Audit file name.
	(11,n)	Next data name audit info.
LNK$[20]	Last audit ADD key.	
LNK$[21]	Generated I/O program.	
LNK$[22]	List of text field IDs.	
LNK$[23]	Data file name (before suffix substitution).	
LNK$[24]	Sort file name (before suffix substitution).	
LNK$[25]	Text file name (before suffix substitution).	
LNK$[26]	Operator supplied sorts.	

MENU$

Purpose Passes information to the menu subsystem when CONNECT MENU is invoked. It is only necessary to pass MENU$ if the caller wishes to override the defaults.

Used By CONNECT MENU from a method.

Contents

MENU$[0]	Menu control data:		
	(1,1)	"C"	Clear menu window and reselect window prior to menu display.
		"X"	Leave menu window displayed and selected upon exit.
		"x"	Display menu but do not allow user to make a selection. Leave menu window displayed and selected upon exit.
	(2,1)	" "	(Space) No heading.
		"C"	Centered heading.
		"R"	Right justified heading.
		"L"	Left justified heading.
		Note: The menu description will be used as the displayed heading.	
MENU$[1]	Menu name.		
MENU$[2]	Menu return value. Return values can only be processed by methods calling CONNECT MENU and are ignored on return to a data name or menu. The value is obtained from the second column of the menu definition, and will be set to null if the user pressed **F4**.		

PP$

Purpose Contains information about the format in use by the caller and the position of the current data name in a screen or view.

Except for PP$[0] and PP$[2] the called method must ensure that this array is not modified before it is returned to the caller, or unpredictable behavior may result.

Used By CONNECT METHOD

Contents

PP$[0]	Input:	Message from method.
	Output:	Return directive.
	"." =	Normal termination.
	null =	Abnormal termination.
PP$[1]	Data record being processed, concatenated into a single string.	
PP$[2]	User input (current field). Used as operator input upon return from post-process methods only.	
PP$[3]	Column/field number.	
PP$[4]	Format data name number.	
PP$[5]	Column/field attributes:	
	(1,1)	column (binary, counting from 0).
	(2,1)	row (binary, counting from 0).
	(3,1)	field size (binary, counting from 0).
	(4,1)	data name number (binary, counting from 0).
PP$[6]	#format-name.	

QUERY$

Purpose Provides controls over the queries to be run, output destination and other operational parameters for Query-IV.

Used By CONNECT QUERY from a method.

Contents

QUERY$[0]	Return status:	
	"Q"	On input indicates a query is being selected.
	"."	On output indicates normal termination.
QUERY$[1]	Query library name. A two-character library name.	
QUERY$[2]	From Query.	
QUERY$[3]	To Query.	
QUERY$[4]	Mask for query range.	
QUERY$[5]	Sort number.	
QUERY$[6]	Query device:	
	"Y"	Select printer prompt.
	null	Ask hard copy question.
	"N"	Terminal output only.
	"LP"	Printer name.
	"Pn"	Printer name.
	"OP"	Use operator default printer.
	"SP"	Use last printer opened by operator.
	"CH:nn"	Use channel nn for output.
	"/file-name"	Create or overwrite output to this file name. Garbage may exist after output text.
	"CR"	Query is connected from another query.
	[1]	Query name (LLQQQQ).
	[2]	X$ not returned.
	[3]	H$ returned to calling query.
	[4]	T$ returned to calling query.

QUERY$[7]	Type of output:	
	"P"	PRINT
	space	PRINT
	"W"	WRITE
	"R"	WRITERECORD
	"N"	No output
QUERY$[8]	Query mode:	
	"D"	Print detail lines only. Header and footer lines will not be printed.
	" -"	Normal printing will be done.
QUERY$[9]	Method to execute prior to printing each query detail line.	
QUERY$[10]	Window name for a user-supplied window.	

When CONNECT QUERY is called from a menu or data name the parameters are supplied in a comma-separated string. The details are identical to those for REPORT$, which follows this page. Refer to REPORT$ for an explanation.

REPORT$

Purpose Provides controls over the reports to be run, output destination and other operational parameters for Report-IV.

Used By CONNECT REPORT from a method.

Contents

REPORT$[0]	Return status:	
	"R"	On input indicates a report is being selected.
	"."	On output indicates normal termination.
REPORT$[1]	Report library name. A two-character library name.	
REPORT$[2]	From Report.	
REPORT$[3]	To Report.	
REPORT$[4]	Mask for report range.	
REPORT$[5]	Sort number.	
REPORT$[6]	Report device:	
	"Y"	Select printer prompt.
	null	Ask hard copy question.
	"N"	Terminal output only.
	"LP"	Printer name.
	"Pn"	Printer name.
	"OP"	Use operator default printer.
	"SP"	Use last printer opened by operator.
	"CH:nn"	Use channel nn for output.
	"/file-name"	Create or overwrite output to this file name. Garbage may exist after output text.
	"CR"	Report is connected from another report.
	[1]	Report name (LLRRRR).
	[2]	X$ not returned.
	[3]	H$ returned to calling report.
	[4]	T$ returned to calling report.

REPORT$[7]	Type of output:	
	"P"	PRINT
	space	PRINT
	"W"	WRITE
	"R"	WRITERECORD
	"N"	No output
REPORT$[8]	Report mode:	
	"D"	Print detail lines only (D, CTnn, STnn, CBnn). Header and footer lines will not be printed.
	" "	Normal printing will be done.
REPORT$[9]	Method to execute prior to printing each report detail line.	
REPORT$[10]	Window name for a user-supplied window.	

The parameters described above are made available to the report through X$, as shown in the table below. Within the report definition, say, in the ENTRY SECTION, you can use Thoroughbred Basic code to evaluate X$ as required.

For example, to determine the sort number from REPORT$[5], evaluate X$(49,2).

REPORT[XX]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
X$(m,n)	(5,2)	(7,6)	(72,6)	(78,6)	(49,2)	(68,2)	(65,1)	(64,1)	(51,12)

When CONNECT REPORT is called from a menu or data name the parameters are supplied in a comma-separated string.

For example:

```
CONNECT REPORT OERNAME,,,2,N
```

Executes the report RNAME from library OE, using sort number 2 and putting the result straight to the terminal. This directive will populate REPORT$ as follows:

REPORT$	**Contents**	**Comment**
[1]	OE	Library name
[2]	RNAME	From report name
[3]	RNAME	To report name
[4]	??????	Mask for report names range
[5]	2	Sort number
[6]	N	Report device

SA$

Purpose Contains the attributes of the screen or view in use by the caller. This array enables the called method to derive the current display context and allows the context to be restored when control returns to the calling object.

The called method must ensure that this array is not modified before it is returned to the caller, or unpredictable behavior may result.

Used By CONNECT METHOD

Contents

SA$[0]	View/Screen name.	
SA$[1]	View/Screen header record.	
SA$[2]	(1,1)	Screen attribute table entry length (ASC 1 byte).
	(2,1)	Number of entries in screen table (ASC 1 byte).
SA$[3]	Screen attribute table:	
	(1,1)	Screen position (ASC 1 byte).
	(2,1)	Screen line (ASC 1 byte).
	(3,1)	Screen entry length (ASC 1 byte).
	(4,1)	Fixed attribute entry number (ASC 1 byte).
	(5,y)	Next screen attribute entry.
SA$[4]	Screen/view formulas:	
	Note on formula storage: If a formula is entered, the fixed attribute entry number above is set to zero and an entry is built in entry [4]. The screen position and line from the attribute is used to locate the formula.	
	(1,2)	Entry length.
	(3,1)	Screen position (ASC 1 byte).
	(4,1)	Screen line (ASC 1 byte).
	(5,1)	Short formula text size 256 bytes (ASC 1 byte).
	or	
	(5,1)	00 Long formula text flag.
	(6,2)	Long formula text size.
	(x,1)	Mask size (ASC 1 byte).
	(x+1,1)	Short formula Thoroughbred Basic statement size 256 bytes (ASC 1 byte).
	or	
	(x+1,1)	00 Long Thoroughbred Basic statement flag.

	(x+2,2)	Long Thoroughbred Basic statement.	
	(xy,z)	Formula text.	
	(x+y+z,m)	Screen mask.	
	(x+y+z+m,b)	Thoroughbred Basic statement.	
SA$[5]	View headings when view specified.		
SA$[6]	Link header when view specified.		
SA$[7]	Pointers to numeric masks.		
SA$[8]	Numeric masks.		
SA$[9]	Printable screen when screen specified.		
SA$[10]	Screen input operations table, 8INPUT interface:		
	(1,1)	Last field processed.	
	(2,1)	B	Pre-process.
		A	Post-process.
		H	Help requested. (When first character of help code = ".").
		F	Formula field. (When FUNC$(3,1) = " ").
	(3,3)	Screen field offset to continue after pre/post process:	
		0	Last field processed.
		1	Next field (default when exit to pre/post process).
		n	Last field plus n.
		-1	Previous field.
		-n	Last field minus n.
	(6,1)	Field column (ASC).	
	(7,1)	Field line (ASC).	
	(8,1)	Field length (ASC).	
	(9,1)	Format entry number (ASC).	
	(10,1)	Post process term-key.	
	(11,1)	Type of input:	
		" "	Input all data element types.
		"K"	Input only KEY type data elements. MUST SET SPARM$[10](12,1)=" ".
		"N"	Input all non-key fields. MUST SET SPARM$[10](12,1)=" ".
	(12,1)	First time flag:	
		" "	First time flag.
		"X"	Re-entry.

	(13,1)	Not used.	
	(14,1)	Line number offset.	
	(15,1)	Offset count limit.	
	(16,1)	Replace/Print data in DATA$ [Y or N].	
	(17,1)	Get occurrence window home position in next two bytes [Y or N].	
	(18,1)	Occurrence window home column (binary).	
	(19,1)	Occurrence window home line (binary).	
	(20,8)	Screen window name (if blank use screen name).	
	(28,8)	Screen window name actually created (different from SPARM$[10](20,8) if a window with that name already exists).	
SA$[11]	8INPUT screen input data element list (format entry numbers).		
SA$[12]	8INPUT pre processing data element list (format entry numbers).		
SA$[13]	8INPUT post processing data element list (format entry numbers).		
SA$[14]	Not used.		
SA$[15]	Not used.		
SA$[16]	Not used.		
SA$[17]	8TEXTF key of data record for corresponding text record. If null, the primary key value will be constructed based on all fields in the format.		
SA$[18]	Not used.		
SA$[19]	Link file definition table.		
SA$[20]	Not used.		
SA$[21]	8PRINT data element list (format entry numbers).		
SA$[22]	Text field operations table:		
	(1,2)	Text file channel number (binary).	
	(3,1)	Use valid value defaults for window (Y/N).	
	(4,1)	Border type:	
		"N"	None.
		"R"	Reverse video line graphic.
		"C"	Character.
		"G"	Line graphic.
	(5,1)	Text field ID.	
	(6,1)	Number of lines in window (ASC 1 byte).	
	(7,1)	Number of characters per line in window (ASC 1 byte).	
	(8,1)	Line 1 of window (ASC 1 byte).	

	(9,1)	Column 1 of window (ASC 1 byte).
	(10,8)	Text field window name. If none supplied, window name will be ".TEXT."+sequential number.
SA$[23]	List of CTL values indicating any field input terminated with one of the CTL values, will be treated as a post process. (STR 2 byte masked :"00").	

SCREEN$

Purpose Passes messages that control the behavior of the screen.

Used By CONNECT SCREEN from a method.

Remarks in *italics* below indicate how the screen behaves in certain circumstances, and are provided for information.

Contents

SCREEN$[ALL]	Prior to a CONNECT SCREEN directive, SCREEN$ can be dimensioned to 16 and the values shown below can be set. The values will determine certain SCREEN characteristics.
SCREEN$[0]	Execution status.
	If SCREEN$[0]="OO3A" and NUM(SCREEN$[3])=0 on entry, then SCREEN$[4] contains the secondary key value directly from the sort file, instead of the display values of the sort fields.
SCREEN$[1]	Screen name.
SCREEN$[2]	Key of last record edited.
SCREEN$[3]	Sort number.
SCREEN$[4]	Key value of record to edit.
SCREEN$[5]	Format data string upon exit.
SCREEN$[6]	Error status upon exit (when "F" option used in [15]) "X"=failed.
SCREEN$[7]	CONNECT message.
	If "AUTO-EXIT" found in CONNECT message or SCREEN$[15]="F": *1. Set SCREEN$[13]="YY Y".* *2. If SCREEN$[4]="" then put key value from format data area into MSG$[4].*
SCREEN$[8]	View name for **F9**.
SCREEN$[9]	LET SCREEN[9]=$0102030A0B$ This will only allow entry into elements 1,2,3,10, and 11 as they appear on the screen.
SCREEN$[10]	Not used.
SCREEN$[11]	Last record read or changed *(set after each read and after each record is successfully added or changed).*
SCREEN$[12]	After read method.

SCREEN$[13]	On/Off indicators (YN) (space applies defaults):		
	(1,1)	Exit screen after editing first record. The default is **N**.	
	(2,1)	Skip maintenance mode window. The default is **N**.	
	(3,1)	Auto key range for views. The default is **N**.	
	(4,1)	Skip printing mode. The default is **N**.	
	(5,1)	Sort change allowed. The default is **Y**.	
	(6,1)	Print screen description in window title:	
		" "	Do not print title.
		"C"	Center heading.
		"R"	Right justify heading.
		"L"	Left justify heading
		The default is **" "**.	
	(7,1)	Clear key value. The default is **N**.	
	(8,1)	Create window with border. The default is **Y**.	
	(9,1)	Skip **F7** Special Functions Menu: "**N**" **-** The Screen Special Functions menu is displayed "**Y**" **-** The user is taken directly to the Help Type toggle menu without first displaying the Special Functions Menu.	
SCREEN$[14]	Application help code (LLHHHHHHHH). The user can toggle between Application help, Data help, and General help by pressing **F7** on any field to display the Help type toggle menu.		
SCREEN$[15]	Maintenance modes:		
	" "	All modes are available.	
	"A"	Add.	
	"C"	Change.	
	"D"	Delete.	
	"I"	Inquire.	
	"F"	Logical screen entry. In this mode, records cannot be added, changed, or deleted. If SCREEN$[15]="F" Set SCREEN$[13]="YY Y"	
	Modes can be used in any combination, except for the "F" mode. For example, IC starts in inquiry mode and enables only change and inquiry.		
	An attempt is made to open the file and read using the key value supplied in SCREEN$[4]. If the file open and read are successful, the data read from the record is supplied as the default data for screen entry. Otherwise, format defaults are applied. The screen terminates after the last field is entered.		

SCREEN$[16]	Window disposition on exit:	
	"N"	Delete screen window before exit. (Default)
	"Y"	Do not delete screen window.
	"R"	Do not delete screen window on exit, Read and print last record entered before **F9** to View. Do not clear last record on exit.
	"r"	Do not delete screen window, Read and print last record entered before **F9** to view.

V$

Purpose Passes data to a view method from a view.

Used By A view with a view method specified, after a row has been read and before it is displayed.

Contents

V$[0,0]	(1,1)	Length of column attribute entry.
	(2,1)	Number of columns used in view.
	(3,1)	Number of columns deleted from view.
	(4,1)	Number of heading rows in view.
	(5,1)	First new column ID.
	(6,1)	First new column number.
	(7,1)	Second new column ID.
	(8,1)	Second new column number.
	(9,n)	The new column n ID and number.
V$[0,1]	(1,1)	Column 1 window address (column).
	(2,1)	Column 1 window address (row).
	(3,1)	Column 1 width.
	(4,1)	Column 1 data name number. 00 implies new column.
	(5,1)	New column ID.
	(6,1)	New column data, built by view method.
V$[0,n]	As V$[0,1] for column n.	

VIEW$

Purpose Passes messages that control the behavior of the view.

Used By CONNECT VIEW called from a method.

Contents

VIEW$[ALL]	Prior to a CONNECT VIEW directive, VIEW$ can be dimensioned to 18 and the values shown below can be set. The values will determine certain view characteristics.	
VIEW$[0]	Execution status.	
VIEW$[1]	View name [,link-name]	
	If view name not found, create view using link specified. If view name not found and link name not specified, a search for a link using the view name will be done. If link found using view name, a view will be created with the same name as found link.	
VIEW$[2]	Selected key.	
VIEW$[3]	Which sort.	
VIEW$[4]	Starting key value when view first displayed.	
VIEW$[5]	Starting key range allowed while in view.	
VIEW$[6]	Ending key range allowed while in view.	
VIEW$[7]	CONNECT message. (SORT n, USING [RANGE FROM TO], SELECT WHEN|secure SELECT WHEN). See VIEW$[13].	
VIEW$[8]	Screen name for **F9**.	
VIEW$[9]	Cursor mode:	
	C	Cursor mode.
	E	Entire field in reverse video (uses view color bar).
	F	Full row (modes same as E).
	P	Simulate PRINT VIEW functionality.
VIEW$[10]	Carriage return action:	
	F	Field edit.
	S	Single record maintenance. Return on **F4** at maintenance options.
	s	Same as S except the return is performed after one record is edited.
	E	Exits and returns current key in entry [2].
VIEW$[11]	Not used.	
VIEW$[12]	Not used.	

VIEW$[13]	On/Off indicators (**YN**) (space applies defaults):		
	(1,1)	Key change allowed. The default is **Y**.	
	(2,1)	Data change allowed. The default is **Y**.	
	(3,1)	View alterations allowed. The default is **Y**.	
	(4,1)	Leave view window displayed after exit. The default is **N**.	
	(5,1)	Sort change allowed. The default is **Y**.	
	(6,1)	Exit after view display. The default is **N**.	
	(8,1)	Enable secure SELECT WHEN. The default is N.	
VIEW$[14]	Application help code (LLHHHHHH),prompt-msg-help-name. Note: The comma (,) separating the application help name and the prompt message help name is required when no application help name is supplied.		
VIEW$[15]	Active select condition.		
VIEW$[16]	Command processing mode:		
	(1,1)	N	Count
		L	List (same as a view)
		S	Sum
		P	Print (hard copy)
		C	Copy
		M	Move
		D	Delete
		c	Change
	(2,1)	Verify for C,M,D,c (Y/N)	
	(3,8)	Link name for C,M,D	
	or		
	(2,1)	Verify for C,M,D,c (Y/N)	
	(3,n)	Change expression.	
	(3+n),x)	Command description.	
	or		
	(2,1)	SUM number elements.	
	(3,n)	Elements to sum.	
	(3+n,x)	Command description.	
VIEW$[17]	Select and/or change expression CPP'd program.		
VIEW$[18]	View FILE-SUFFIX.		
VIEW$[19]	For internal use.		

VIEW$[20]	For internal use.
VIEW$[21]	user supplied override of the view window size and location, format is: comma delimited number of cols for view - defines view width number of rows for view - defines view height starting col for view - defines left most position of the view starting row for view - defines top most position of the view For example: 60,10,2,3 creates a view window 60 cols (characters) wide, 10 rows high, starting at col 2 row 3.
VIEW$[22]	Optional View Heading
VIEW$[23]	Optional data name list. View will display only these datanames. pass as: dn1,dn3,dn3,…dnx Replace "," with a ";" to signify locked column. Define new/joined columns with the column id A-Z.
VIEW$[24]	Number of data rows to return. Specifying this option will allow the V$ array to be returned to the calling program populated with all of the view column data and the number of records specified in this field. The view will not be displayed on the screen. OO3A will just exit. For example, if VIEW$[24]="10" then the CONNECT VIEW will not display the view, instead it will return the 10 rows of data in the v$ array. The first row would be the first row of data that would be normally be displayed as defined by all the other parameters in the array that determine the first row of a view.
VIEW$[25]	Joined column display flag. N = Do not display joined column data names. A = Display all data names defined by the joined column Link definition.
VIEW$[26]	RESERVED for internal use to pass joined column information to command processing method OO3A3.
VIEW$[27]	RESERVED for internal use to pass information to command processing method OO3A3 for **F8**-Query option.

VIEW$[28]	View heading col/row substitution values: The first byte defines the substitution set delimiter used in the string. This is followed by one or more substitution sets: 1 byte substitution value delimiter + 1 byte row number (1,2,3) + 3 byte column number + substitution text + delimiter + 1 byte row number (1,2,3) + 3 byte column number + substitution text + (next entry) Accepts any number of substitution values in any sequence. All substitution values will be padded to column width with trailing spaces. Unless leading spaces are supplied, substitution headings will be left aligned. If the substitution values exceed the defined column width, the substitution value will be truncated. To clear a column heading within a row, define a substitution value of 1 space. This will be padded to full column width and have the effect of clearing out the column heading for that row.

VIEW$[29]	VIP View color settings Row and column color settings can be supplied in one of two formats: SETCOLOR syntax or a view color method name. Support for the view color method provides flexibility for processing complex color rules beyond the scope of the SETCOLOR syntax. SETCOLOR syntax BY ROW: SETCOLOR foreground+background BY ROW WHEN condition SETCOLOR WHITE+RED BY ROW WHEN #OEFCUST.CUST-SALES > 17000 Multiple SETCOLOR commands are supported. Use a ";" to separate commands. VIEW$[29] = "SETCOLOR WHITE+RED BY ROW WHEN"+ "#OESCUST.CUST-SALES > 15000;"+ "SETCOLOR RED+WHITE BY COL.CUST-CODE WHEN "+ "OESCUST.CUST-SALES >17000" Commands are evaluated in order. In the above example an entire row might display with white text on a red background and with the CUST-CODE column displayed with red text on a white background. The color rules are applied each time a row is to be displayed and each time the user moves off a column after entering data. For more information please see the SetColor Method earlier in this manual.
VIEW$[30]	Set to an "R" will cause the view to be presented in reverse order from the selected sort.