Department of Veterans Affairs
Decentralized Hospital Computer Program

Unwinder (XQOR)
Technical Manual

Version 7.1

August 1994

Information Systems Center
Salt Lake City, Utah

Preface

This document describes the operation of the XQOR routines, which are used in conjunction with the Protocol file to create modular building blocks for applications. The *Unwinder Technical Manual* is intended for DHCP developers and possibly for IRM (Information Resource Management) personnel at VAMCs.

Table of Contents

Introduction

{ XE "Introduction" }

The Unwinder is a utility that is used in conjunction with the Protocol file (#101) to create modular building blocks for applications.

The Unwinder allows hierarchical traversing of menus, as found in Menu Management, and also the structuring of order protocols into independent, reusable modules. Each node becomes a "building block" from which more sophisticated modules may be built. For instance, the node "Order Shirt" may have as sub-items, "Get Size," "Get Color," "Get Style," and "Get Delivery Date." Each of these sub-items may, in turn, be used to build other modules.

Provisions have been made to allow additional building blocks to be placed at the item level of the node. Their purpose is to allow modifying actions to be executed and thus increase the flexibility of each module.

The following sections describe how developers can use the Unwinder for their applications.

Implementation and Maintenance

{ XE "Implementation and Maintenance" **}**

Description of Protocol File Operations

{ XE "Protocol File Operations" **}**
Information in the Protocol file is arranged into hierarchies. The Unwinder works by navigating down the hierarchies, stacking the path taken, so it may return back up the hierarchy by the same path. The Unwinder is also capable of navigating the Option file.

This navigation works as follows: The system begins with an initial node in the Protocol file and executes the entry action. If the node is a menu, the items are displayed, selections are allowed, and the selections are stacked as new nodes. If the node is not a menu, the items are simply stacked as new nodes. (However, they may be screened, just as menu items may be screened.) The same process is then repeated with each new node. When there are no more new nodes, the system returns back up the path it came down, executing exit actions.

What this allows is not only the hierarchical traversing of menus, as found in Menu Management, but the structuring of order protocols into independent, reusable modules. Each node then becomes a "building block" from which more sophisticated modules may be built. For instance, the node "Order Shirt" may have as sub-items, "Get Size" "Get Color," "Get Style," and "Get Delivery Date." Each of these sub-items may in turn be used to build other modules. Provisions have been made to allow additional building blocks to be placed at the item level of the node. Their purpose is to allow modifying actions to be executed and thus increase the flexibility of each module. For further clarification, the following illustration demonstrates the sequence of events. (This is only an example. Because of performance considerations and the way packages are already set up, it is certainly not necessary to go to this level of detail in setting up nodes in the Protocol file.)

Assume the following entries are in the PROTOCOL file:

NAME:	MY CLOTHES MENU
ITEM TEXT:	Order Clothes Menu
TYPE:	Protocol Menu
ENTRY ACTION:	
EXIT ACTION:	
ITEM:	MY SHIRT
ITEM:	MY SOCKS

NAME:	MY SHIRT
ITEM TEXT:	Shirt
TYPE:	Protocol
ENTRY ACTION:	S STYLE="SHIRT"
EXIT ACTION:	K STYLE
ITEM:	MY CLOTHES ORDER
ITEM:	

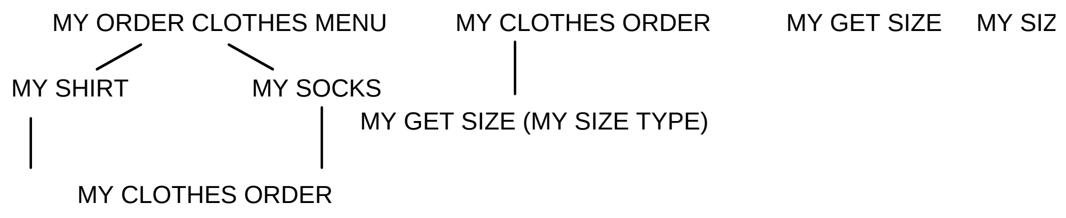
NAME:	MY SOCKS
ITEM TEXT:	Socks
TYPE:	Protocol
ENTRY ACTION:	S STYLE="SOCKS"
EXIT ACTION:	K STYLE
ITEM:	MY CLOSES ORDER
ITEM:	

NAME:	MY CLOTHES ORDER
ITEM TEXT:	Clothing Order
TYPE:	I '$D(STYLE)
ENTRY ACTION:	D EN^GETSTYLE
EXIT ACTION:	K STYLE
ITEM:	MY GET SIZE
MODIFYING ACTION:	MY SIZE TYPE

NAME:	MY GET SIZE
ITEM TEXT:	Get Size
TYPE:	Protocol
ENTRY ACTION:	D EN^GETSIZE
EXIT ACTION:	D SIZE^CLEANUP
ITEM:	

NAME:	MY SIZE TYPE
ITEM TEXT:	Size Type
TYPE:	Protocol
ENTRY ACTION:	D EN^SIZETYPE
EXIT ACTION:	
ITEM:	

These entries would create nodes related in the following ways:



If a user selected "Shirt" from the Order Clothes Menu, the sequence of actions executed would be as follows:

```
S STYLE="SHIRT"                    ;Entry action for "Shirt"
       I '$D(STYLE) D EN^GETSTYLE ;Entry action for "Clothing Order"
            D EN^SIZETYPE                ;Entry action for "Size Type" (Note:
                                                This is the modifying action of
                                                "Get Size" when it is an item of
                                                "Clothing Order")
            D EN^GETSIZE                 ;Entry action for "Get Size"
            D SIZE^CLEANUP               ;Exit action for "Get Size"
       K STYLE                       ; Exit action for "Clothing Order"
K STYLE                            ;Exit action for "Shirt"
```

Note that the protocol, MY CLOTHES ORDER, prompts for style if it is not yet defined. This allows the protocol to be used independently, rather than be dependent on the path which led to it. Also, the protocol, MY SIZE TYPE, is used to show how a modifying action might be used to make the principal protocol, MY GET SIZE, work more generically.

To summarize, the Unwinder works by navigating through a hierarchy of menus and actions. Each node of the hierarchy represents a specific function to be performed. These functions are developed for the various packages and placed in the PROTOCOL file as a point of integration.

Protocol Types

{ XE "Protocol Types" **}**
There are several types of protocols. The type field of the protocol determines the way the Unwinder operates on that particular protocol. The XQOR routines are also executed when options of type Protocol, Protocol Menu, and Extended Action are invoked from the Option file.

General Types

M (menu)	A menu of selections is presented to the user. Fields in the Protocol file and XQORM variables affect the formatting and operation of menus. The menus generally have multiple columns and allow multiple selections.
X (extended action)	An extended action processes all sub-items (entries in the ITEM multiple) of the protocol after the entry action and before the exit action. Sub-items may, in turn, be extended actions (thus, the term "unwinder"). The sub-items are processed in SEQUENCE order, if the SEQUENCE field is defined.
A (action)	An action only processes the entry and exit actions. Sub-items are ignored.

OE/RR Types

The following types are specific to OE/RR and should be used only in the context of placing orders.

Q (protocol menu)	A protocol menu is the same as a menu, except that an OE/RR context is assumed. A provider prompt and an "OE/RR Accept Orders" screen are presented appropriately.
O (protocol)	A protocol is the same as an extended action, except that the Unwinder assumes orders are being placed.
L (limited protocol)	A limited protocol is the same as an action, except that the Unwinder assumes orders are being placed.

T (term)	A term is a protocol that may be defined as a prompt in a dialog. DIR calls are used to process the prompting defined by a term protocol. Currently, this type of protocol is used only for OE/RR generic order definitions.
D (dialog)	A dialog is a list of term protocols (listed in the ITEM multiple). The individual prompts are presented in sequence and up-arrow navigation between prompts is allowed. This allows a dialog to occur with the user without database updates. Currently, this type of protocol is used only for OE/RR generic order definitions.

All links to OE/RR are made through the routine, XQORO**{** XE "XQORO" **}**.

+***NOTE:*** *If OE/RR is not installed and a protocol type specific to OE/RR is executed, an error message is displayed.*

Routine Descriptions

{ XE "Routine Descriptions" }

Routines exported:

XQOR	XQOR1	XQOR2	XQOR3	XQOR4	XQORD	XQORD1
XQORI001	XQORINI1	XQORINI2	XQORINI3	XQORINI4	XQORINI5	XQORINIS
XQORINIT	XQORM	XQORM1	XQORM2	XQORM3	XQORM4	XQORM5
XQORM6	XQORMX	XQORNTEG	XQORO			

Routine Descriptions:

XQOR	Prepare to Unwind Options
XQOR1	Main Unwinding Loop
XQOR2	Process Extended Actions, Protocols
XQOR3	Process Menus, Protocol Menus
XQOR4	Process "^^" jump
XQORD	Dialog Utility
XQORD1	Process Menus, WP during dialog
XQORM	Menu Utility
XQORM1	Display selections & prompt
XQORM2	Lookup for Menu Utility
XQORM3	Lookup (cont.)
XQORM4	Menu Messages
XQORM5	Menu Help
XQORM6	Function Key Reader
XQORMX	Compile formatted menus
XQORNTEG	Package checksum checker
XQORO	Order Entry Calls

25 ROUTINES

Files

{ XE "Files" }

There are no files in the Unwinder utility. However, the Unwinder provides its utility by operating on the Protocol File (exported with OE/RR) and the Option File (exported with Menu Management). These files are accessed by the Unwinder in a read-only manner.

Exported Menus & Options

{ XE "Exported Menus & Options" }

There are no menus and options in the Unwinder utility.

Cross-References

{ XE "Cross-References" }

There are no cross-references in the Unwinder utility.

Archiving & Purging

{ XE "Archiving & Purging" }

There are no archiving and purging functions in the Unwinder utility.

EN^XQOR Entry Point

{ XE "EN^XQOR Entry Point" }
This is the main routine for navigating protocols. The routine processes the initial protocol and the subordinate protocols. This processing of subordinate protocols happens according to the type of protocol and the navigation variables that get set along the way. For example, by defining a set of protocols you could create the following entries:

Name	**Item Text**	**Type**	**Entry Action**	**Exit Action**
`MYTOP`	`My Top`	`X`	`W !,"Top Entry"`	`W !,"Top Exit"`
`MYITM1`	`My Item 1`	`X`	`W !,"Item 1 Entry"`	`W !,"Item 1 Exit"`
`MYITM2`	`My Item 2`	`X`	`W !,"Item 2 Entry"`	`W !,"Item 2 Exit"`
`MYITM3`	`My Item 3`	`X`	`W !,"Item 3 Entry"`	`W !,"Item 3 Exit"`
`MYSUBITM`	`My SubItem`	`A`	`W !,"SubItem Entry"`	`W !,"SubItem Exit"`

If MYITM1, MYITM2, and MYITM3 are placed in the item multiple of MYTOP and MYSUBITM is placed in the item multiple of MYITM2, calling EN^XQOR with MYTOP as the initial protocol should produce the following results:

```
Top Entry
Item 1 Entry
Item 1 Exit
Item 2 Entry
SubItem Entry
SubItem Exit
Item 2 Exit
Item 3 Entry
Item 3 Exit
Top Exit
```

This assumes that MYITM1, MYITM2, and MYITM3 are entered in that sequence or assigned sequence values of 1, 2, and 3, respectively.

Input Variable{ XE "Input Variable" }

X (required)	Identifies the initial protocol that EN^XQOR should process. X should be in variable pointer format. For example, X="1234;ORD(101," would cause the processing to start with the *protocol* that has an internal entry number of 1234. Similarly, X="1234;DIC(19," would cause the processing to start with the *option* that has an internal entry number of 1234. An alternative to using variable pointer format is to set X equal to the name or number of the protocol and DIC equal to the number or global reference of the file you are working in (generally the Protocol file). For example, X="MYTOP" and

DIC=101 will begin processing at the MYTOP protocol. For backwards compatibility, if X is not in variable pointer format and DIC is not defined, operation in the *Option* file is assumed.

Navigation Variables{ XE "Navigation Variables" }

Navigation variables are optional and may be set anywhere inside the code that is being executed by EN^XQOR. These variables affect the way XQOR displays information, passes control to subsequent protocols, etc.

XQORQUIT{ XE "XQORQUIT" **}** Signals the Unwinder to not process any protocols that are subordinate to the current protocol. Control is passed to the next sibling protocol. In the above example, setting XQORQUIT in the entry action of MYITM2 would prevent MYSUBITM from being processed.

XQORPOP{ XE "XQORPOP" **}** Signals the Unwinder to not continue processing sibling protocols. Control is returned directly to the parent protocol. For example, if XQORPOP is set in the entry action of MYITM2, MYSUBITM and MYITM3 are not processed and control is returned to MYTOP.

XQORFLG("SH"){ XE "XQORFLG(\"SH\")" **}** If set to 1, a subheader is displayed just before processing any subordinate options. The subheader contains the menu text of the option. Subheader displays may be turned off by setting XQORFLG("SH") to 0. The default value for XQORFLG("SH") is 0. For example, setting XQORFLG("SH")=1 in the entry action of MYITM2 would cause MYSUBITM to display as follows:

```
    --- My Sub Item ---
SubItem Entry
SubItem Exit
```

XQORNOD{ XE "XQORNOD" **}** Reference variable that identifies the protocol currently being processed. This is in variable pointer format. For example, if the currently executing protocol were MYITMZ, with an internal entry number of 2456, then XQORNOD would be:

```
2456;ORD(101,
```

XQORNOD(0){ XE "XQORNOD(0)" **}** Reference variable that provides information about the current protocol if the parent protocol was a menu type. The information is in four pieces delineated by up-arrows (^):

1) Internal entry number (inside the item multiple) of the selected item
2) .01 field of the entry selected
3) Text that was displayed on the menu
4) What the user typed in to select the item

For example, if the MYTOP protocol were a menu instead of an extended action, the three MYITM* protocols would be presented as possible menu selections. If you select the second, XQORNOD(0) might look something like:

```
1^2456^My Item 2^MY ITEM 2
```

When the Unwinder processes menu types, it is internally calling the EN^XQORM entry point. Therefore, the following subscripted XQORM variables may be set in the entry action of a protocol that is a menu type:

```
XQORM(0)
XQORM("A")
XQORM("B")
XQORM("H")
XQORM("S")
XQORM("?")
XQORM("??")
XQORM("KEY",keyword)
XQORM("XLATE",function key)
```

The subscripted XQORM variables will affect the way the menu is displayed and processed. Setting one of these variables in the entry action will override the equivalent setting defined in the Protocol file, if one exists. For example, setting XQORM("S") will override what is set in the SCREEN field of the protocol. See the description of the entry point, EN^XQORM for a description of the XQORM variables.

Note: While the subscripted XQORM variables may be set, XQORM itself should not be set, as EN^XQOR handles proper setting of XQORM before the menu is displayed.

Output Variables**{** XE "Output Variables" **}**

There are no output variables used with the Unwinder.

EN1^XQOR Entry Point

{ XE "EN1^XQOR Entry Point" **}**
This entry point is identical to EN^XQOR, except that the entry and exit actions of the initial protocol are not executed. This entry point provides backwards compatibility with the way Kernel 6 processed protocols that were defined in the Option file.

EN^XQORM Entry Point

{ XE "EN^XQORM Entry Point" **}**
This entry point handles the display of and selection from a menu. Note that this routine processes a single menu only. This is the call EN^XQOR uses to obtain menu selections. The caller is responsible to handle any selections from the menu that are returned in the Y array. If you want navigation to the selected items handled for you, use the EN^XQOR entry point. The menus handled by this routine are the multiple selection, multiple column menus that are typical in OE/RR.

Input Variables**{** XE "Input Variables" **}**

XQORM{ XE "XQORM" **}** (required)	A variable pointer to the menu that should be displayed. For example, XQORM="1234;ORD(101," will display the menu in the protocol that has an internal entry number of 1234 and process it according to the field entries in the protocol itself and the other XQORM variables that are set up.
XQORM(0){ XE "XQORM\(0\)" **}** (required)	Is a string of flags that control the display and prompting of the menu. If a numeric is included, it must be at the beginning of the string. The following parameters are allowed:

numeric	Maximum number of selections allowed. If a number is not specified, as many selections as items on the menu are allowed.
A	Prompt for a selection from the menu (display a "Select Item(s):" prompt, for example). If the A is not included, selections will not be prompted for.
D	Display the menu. If the D is not included, the menu is not displayed. If prompting is allowed (A), typing "?" will display the items.

\	Suppress the line feed before the "Select Item(s):" prompt. Used to control vertical spacing on the screen.
X	Find only exact matches. You must type in the entire text of the menu item for it to be selected.
F	Disables saving selected items into DISV for spacebar recall. Otherwise, items will be saved and typing spacebar will recall the previous selections.
+	Allows "+" and "-" to be returned as valid selections, even though they are not on the menu. This flag is generally not used, as the keyword mechanism (see XQORM("KEY") array) provides similar functionality.
R	Save keywords that have been entered for spacebar recall. Normally, spacebar recalls only items selected from the menu and not keywords.
r	Save up arrow jumps for spacebar recall. If a user jumps to another protocol (enters "^^Health Summary," for example), the jump is typically not saved for spacebar recall. The "r" flag saves the jump for recall.

The following are some examples of how these flags may be used -

2A	allow a maximum of two selections, prompt the user to select items but do not display the menu initially (wait for the user to type "??"), and save all selections for spacebar recall
AD	allow unlimited selections, display the menu initially, prompt the user for selections, place the "Select" prompt immediately under the menu with no white space, and save selections for spacebar recall
D	display the menu only, do not prompt for selections

XQORM("A"){ XE "XQORM(\"A\")" } Text to use for the "Select" Prompt. For example, XQORM("A")="Choose From Items 1-4:" will present the user with that prompt. "Select Item(s):" is the default prompt.

XQORM("B"){ XE "XQORM(\"B\")" } Text to use as the default menu selection. XQORM("B")="Item 1" would result in a prompt like "Select Item(s): Item 1//."

XQORM("H"){ XE "XQORM(\"H\")" } MUMPS code that, when executed, displays a header for the menu. For example, XQORM("H")="W #,""Menu of Shoe Styles""".

XQORM("S"){ XE "XQORM(\"S\")" } MUMPS code that is executed before displaying each menu item. If $T is true after the code is executed, the menu item is selectable. If $T is false, the menu item is not selectable and parentheses are placed around it. When executed, DA(1) will be the internal entry of the menu in the Protocol file and DA will be the internal entry number of the item on the menu. The naked reference will NOT be set. For example, XQORM("S")="I DA#2" will cause menu items with even internal entry numbers to not be selectable.

NOTE: There are frequent requests to allow separate screening logic for each item on the menu. This may be accomplished by doing the following:

1) For each item that you wish to screen, place screening code in the SCREEN field of the protocol for that item.

2) In the SCREEN field of the menu protocol, place the following code:

```
I 1 X:$D(^ORD(101,+$P(^ORD(101,DA(1),10,DA,0),
"^",1),24)) ^(24)
```

XQORM("?"){ XE "XQORM(\"?\")" }; MUMPS code that replaces the default single question mark help. For example, XQORM("?")="W !,""Type the name of an item.""". The default double question mark help{ XE "Double question mark help" } is still provided. The double question mark help explains the extended syntax for making selections from the menu but is somewhat specific to OE/RR.

XQORM("??"){ XE "XQORM(\"??\")" } MUMPS code that replaces all default help{ XE "Default help" }, both single and double question mark. For example, XQORM("??")="D HELP^MYRTN"

XQORM("NO^"){ XE "XQORM(\"NO^\")" } If defined, disallows exiting the menu with "^." The user is forced to make a selection.

XQORM("NO^^"){ XE "XQORM(\"NO^^\")" } If defined, disallows the use of the "^^" syntax by the user for jumping to another protocol.

XQORM("KEY"){ XE "XQORM(\"KEY\")" }; An array of keywords { XE "Keywords " }that may be typed at the menu prompt that are not shown on the menu. The format of the keyword array is:

```
XQORM("KEY",keyword)=pointer to protocol ^ branch flag
```

The 'pointer to protocol' must be the internal entry number of an entry in the Protocol file. This protocol is executed when the user types the keyword. If the branch flag is set (i.e., equal to 1), control branches to that protocol as if it were a menu selection. Otherwise, EN^XQOR treats the entry as if it were a ^^-jump. For example,

`XQORM("KEY","TIME")=1234`	If "TIME" is typed, a protocol that displays the current time is run and then the user returns to the menu.
`XQORM("KEY","QUIT")=5678^1`	If "QUIT" is typed, the system treats QUIT as if it were part of the menu and the quit logic is executed.

XQORM("XLATE"){ XE "XQORM(\"XLATE\") " } If function key interpretation{ XE "Function key interpretation" } is allowed, ;the XQORM("XLATE") array allows a mapping between the strings returned by the function keys and menu selections or keywords. For example,

```
XQORM("XLATE","UP")="Scroll Up"
XQORM("XLATE","DOWN")="Scroll Down"
XQORM("XLATE","HELP")="??"
```

Function key processing is allowed if the Kernel routine ^XGF is present.

Output Variables{ XE "Output Variables" }

Y	All results are returned in Y. Y itself will be greater than 0 if items have been selected. Y is -1 if enter was pressed without a selection or if '^' was entered.

Y(n)

Each selection is in a numbered subscript. You should traverse the subscripts with $ORDER, as they are not always in strictly sequential order.

For items selected normally from the menu -

```
Y(n)=item IEN ^ protocol IEN ^ displayed name ^ actual
input
```

Item IEN (first piece) is the internal entry number of the item within the item multiple (generally not that useful). The protocol IEN (second piece) is the internal entry number of the protocol for the selected item (this is what you usually want to look at). The displayed item name (third piece) is the item name as it appeared on the menu. The actual user input (fourth piece) is what the user actually typed. This is useful if the user entered additional information using the "=" convention.
For example, if you enter the following

```
Select Item(s): ED=3, DT
```

The Y array might look like:

```
Y=2
Y(1)=3^2345^Edit^ED=3
Y(2)=7^3456^Details^DT
```

When obtaining input from the user, special syntax is allowed. This syntax allows users to select things that are not on the menu. Typing "^^protocol name" allows the user to jump directly to the entry identified by the protocol name. When finished with the process executed by that entry, the user is returned to the menu where the "^^" was typed.

If keywords**{** XE "Keywords " **}** are set up (see the XQORM("KEY") description), the user may type these keywords to select actions that are not directly on the menu. When the user types "^^protocol name" or a keyword, that portion of the Y array looks different. This is because an item has not been directly selected from the menu.

For entries preceded by "^^"(jump syntax) -

```
Y(n)=^^^text of jump
```

In other words, there are three circumflexes, then the text of what was typed after '^^'. In this case, it is up to you to locate the protocol that matches the typed text and code the jump.

For keywords{ XE "Keywords " } *that were entered -*

```
Y(n)=^^keyword^`IEN=text after keyword=
```

The third piece contains the keyword. The fourth piece begins with "`" (accent grave) followed by the internal entry number of the protocol associated with the keyword. The "`" (accent grave) aids in looking up the protocol to be executed by forcing FileMan to look up by internal entry number. If the user typed additional text after the keyword, this is placed after the internal entry number and delimited by equals signs. If the keyword should be branched to, rather than treated as "^^jump," the internal entry number of the protocol is also in the second piece.

So if the user types the following (assuming proper keywords, etc. are set up) -

```
Select Item(s): ED=1,^^HEALTH SUMMARY,SHOW ORDERS,DT
```

The Y array might look like:

```
Y=4
Y(1)=3^2345^Edit^ED=3
Y(2)=^^^HEALTH SUMMARY
Y(3)=^^SHOW^`4394=ORDERS=
Y(4)=7^3456^Details^DT
```

Remember, you don't need to worry about processing the Y array and handling protocol navigation if you use EN^XQOR. This is done for you automatically.

The EN^ZQORM entry point is a very low-level call that is used by EN^XQOR. It is documented here to allow those who desire the flexibility of processing things at this very low level.

Examples

If you are in an account where OE/RR is installed, the following will demonstrate how EN^XQORM is called and what is returned in the Y array. You should be in programmer mode in a partition where you have logged in through ^XUP (to set up the IO variables).

First, display the OE/RR review screen{ XE "Review screen" } and make some selections by entering the following commands:

```
S XQORM=$O(^ORD(101,"B","ORR REVIEW SCREEN",0))_"; ORD(101,"
S XQORM(0)="AD"
D EN^XQORM
ZW Y
```

If you selected items from the menu, these were listed in the Y array when you displayed it using ZWRITE. Now if you want to allow the user to enter a keyword, say "TIME," enter the following commands, and type "TIME" at the select item(s) prompt:

```
S XQORM("KEY","TIME")=$O(^ORD(101,"B","OR GKEY TIME",0))
D EN^XQORM  ; type time as one of the selections
ZW Y
```

When you view the Y array this time, the entry for the "TIME" keyword should be there. To now change the select item(s) prompt to something else and give it a default, enter the following commands:

```
S XQORM("A")="Type in something - "
S XQORM("B")="Quit"
D EN^XQORM
ZW Y
```

When the menu was displayed, the prompt at the bottom should have been different. Again, any selections you made were listed in the Y array. You may wish to continue with other experiments, such as seeing what happens to the Y array when "^^jump" syntax is used, other input variables are changed, etc.

DISP^XQORM1 Entry Point

{ XE "DISP^XQORM1 Entry Point" **}**
If you have replaced the standard help **{** XE "Help " **}**by setting XQORM("??"), the menu selections may be displayed from your help code by calling DISP^XQORM1 with X="?". For example,

```
S XQORM("??")="W !,""These are the selections:"" S X=""?"" D
DISP^XQORM1"
```

DISP^XQORM1 should only be called from within the code used by XQORM("??").

Input Variables**{** XE "Input Variables" **}**

X (required)	Must be "?"

XREF^XQORM Entry Point

{ XE "XREF^XQORM Entry Point" }
Menus are compiled into the XUTL global. This should happen automatically. If you need to force a menu to recompile, XREF^XQORM can be used to do that.

Input Variables{ XE "Input Variables" }

XQORM{ XE "XQORM" } (required) Variable pointer to the protocol that should be recompiled. For example,

```
S XQORM=$O(^ORD(101,"B","MY KEYWORD MENU",0))_";
ORD(101,"
D XREF^XQORM
```

External Relations

{ XE "External Relations" }

Required DHCP packages

Package	Minimum Version
Kernel	7.1
OE/RR	2.5

Database Integration Agreements

```
               NAME: DBIA344-A           ENTRY: 344
  CUSTODIAL PACKAGE: ORDER ENTRY/RESULT         Salt Lake City
SUBSCRIBING PACKAGE: UNWINDER                   Salt Lake City
              USAGE: Private          APPROVED: APPROVED
             STATUS: Active            EXPIRES:
           DURATION: Till Otherwise Agr  VERSION:
               FILE: 101                  ROOT: ORD(101,
        DESCRIPTION:                      TYPE: File
   The following DBIA is granted between the Unwinder and OE/RR.

   Read Access to File 101:  The XQOR routines navigate the Protocol file
   (101).  To provide this navigation, XQOR needs read access to File 101.

     ROUTINE:

                                ********************

               NAME: DBIA351-A           ENTRY: 351
  CUSTODIAL PACKAGE: KERNEL                     San Francisco
SUBSCRIBING PACKAGE: UNWINDER                   Salt Lake City

              USAGE:
                     Private          APPROVED: APPROVED
             STATUS: Active            EXPIRES:
           DURATION: Till Otherwise Agr  VERSION:
               FILE: 19                   ROOT: DIC(19,
        DESCRIPTION:                      TYPE: File
   Read Access to File 19:  When an Option that is a protocol (O) or protocol
   menu (Q) is encountered by menu manager, control is turned over to XQOR.
   XQOR needs to have read access to File 19 to be able to provide the
   navigation of these protocols.  This agreement would replace DBIA #5,
   which was between OE/RR and Menu Driver.

     ROUTINE:
```

```
               NAME: DBIA344-C            ENTRY: 847
  CUSTODIAL PACKAGE: ORDER ENTRY/RESULT          Salt Lake City
SUBSCRIBING PACKAGE: UNWINDER                    Salt Lake City
              USAGE: Private           APPROVED: APPROVED
             STATUS: Active             EXPIRES:
           DURATION: Till Otherwise Agr  VERSION:
              FILE:
                     100.99                ROOT: ORD(100.99,
       DESCRIPTION:                        TYPE: File
   Read Access to File 100.99:  The OE/RR Parameters file (100.99) is
   accessed in setting up some of the OE/RR variables and in determining if
   OE/RR is running.

     ROUTINE:

                                  ********************

               NAME: DBIA351-B            ENTRY: 858
  CUSTODIAL PACKAGE: KERNEL                      San Francisco
SUBSCRIBING PACKAGE: UNWINDER                    Salt Lake City
              USAGE: Private           APPROVED: APPROVED
             STATUS: Active             EXPIRES:
           DURATION: Till Otherwise Agr  VERSION:
               FILE:                       ROOT: XUTL
       DESCRIPTION:                        TYPE: File
   Use of ^XUTL:  The XQOR routines use ^XUTL("XQORM") and ^XUTL("XQORW") to
   store compiled protocol menus.  An agreement to allow use of these global
   nodes would partially replace DBIA #4 (which erroneously identifies the
   node used as ^XUTL("ORUM")).  The portion of DBIA #4 which allows OE/RR to
   use ^XUTL("OR",$J,package namespace) would need to remain as is.

     ROUTINE:

                                  ********************

               NAME: DBIA351-C            ENTRY: 859
  CUSTODIAL PACKAGE: KERNEL                      San Francisco
SUBSCRIBING PACKAGE: UNWINDER                    Salt Lake City
              USAGE: Private           APPROVED: APPROVED
             STATUS: Active             EXPIRES:
           DURATION: Till Otherwise Agr  VERSION:
               FILE:                       ROOT: DISV(
       DESCRIPTION:                        TYPE: File
   Use of ^DISV:  The Unwinder uses ^DISV(DUZ,"XQORM") to store the items
   that were selected for spacebar recall.  I couldn't find a DBIA agreement
   for this.  When we originally discussed this (years ago), the condition
   for using ^DISV was that the first subscript be DUZ so that Kernel could
   maintain it.  I need a new agreement for read/write access to
   ^DISV(DUZ,"XQORM").

     ROUTINE:
```

```
                NAME: DBIA899                ENTRY: 899
   CUSTODIAL PACKAGE: ORDER ENTRY/RESULT            Salt Lake City
 SUBSCRIBING PACKAGE: UNWINDER                      Salt Lake City
               USAGE: Private             APPROVED: APPROVED
              STATUS: Active               EXPIRES:
            DURATION: Next Version         VERSION: 2.5
                FILE: 101.11                  ROOT: XUTL("XQORW"
         DESCRIPTION:                         TYPE: File
   When OE/RR handles the prompting for generic orders, prompts that require
   word processing answers store the entered text temporarily in file 101.11
   (XQOR WORD PROCESSING).  The lifespan of an entry in this file is several
   minutes.  The Unwinder uses this file when handling the order prompting
   for OE/RR.  Since this file is an OE/RR file, the Unwinder needs an
   integration agreement for both read and write access to this file.

   It would be possible to move this file into the Unwinder package.
   However, this portion of the Unwinder is currently used exclusively by
   OE/RR.  Since the portion of OE/RR that handles order dialogs is being
   rewritten (so OE/RR can handle the front door), I would prefer to wait
   until version 3 of OE/RR before shifting around the custody of files too
   much.

     ROUTINE:

                            ********************

                NAME: DBIA344-B              ENTRY: 846
   CUSTODIAL PACKAGE: ORDER ENTRY/RESULT            Salt Lake City
 SUBSCRIBING PACKAGE: UNWINDER                      Salt Lake City
               USAGE: Private             APPROVED: APPROVED
              STATUS: Active               EXPIRES:
            DURATION: Till Otherwise Agr   VERSION:
                FILE:                         ROOT:
         DESCRIPTION:                         TYPE: Other
   The Unwinder was originally written as part of OE/RR in the namespace OR.
   When the Unwinder functionality was separated into the XQOR routines, all
   the links to OE/RR were isolated into the routine, XQORO.  This routine
   uses OE/RR variables, and calls into OE/RR entry points.  The following
   integration agreements are needed to support this routine (XQORO).

   OE/RR Variables:  The XQORO routine makes sure OE/RR variables are set to
   the proper values between each protocol that is executed.  The following
   variables are killed between each protocol to protect the OE/RR
   environment -

        ORIFN,ORCOST,ORIT,ORSTRT,ORSTOP,ORTO,ORPURG,ORTX,ORSTS,ORPK,ORLOG,
        ORPCL,OR,ORZ,ORNS

        The following variables are reset between each protocol -

        ORVP,ORPV,ORL,ORTS,ORDUZ,ORNP,OROLOC,ORGY,ORACTION,OROLD,ORNS,
        ORTX,ORUP

        ORPRFRM is used in conjunction with response time monitoring.

     ROUTINE:
```

```
                NAME: DBIA344-D              ENTRY: 848
   CUSTODIAL PACKAGE: ORDER ENTRY/RESULT            Salt Lake City
 SUBSCRIBING PACKAGE: UNWINDER                      Salt Lake City
               USAGE: Private             APPROVED: APPROVED
              STATUS: Active               EXPIRES:
            DURATION: Till Otherwise Agr   VERSION:
                FILE:                         ROOT:
         DESCRIPTION:                         TYPE: Routine

     ROUTINE: OR1
   COMPONENT:  ADD
   VARIABLES:     Use of ADD^OR1:  This is called to set up a context for
                  adding orders.  OREND and ORPTLK are checked after this
                  call to see if the context was successfully established.
                  This agreement, along with the one concerning the use of
                  AFT^OR1, would replace DBIA #8 and DBIA #46.
   COMPONENT:  AFT
   VARIABLES:     Use of AFT^OR1:  This is called to present and OE/RR review
                  screen and to clear the 'add orders' context.

                             ********************

                NAME: DBIA344-E              ENTRY: 849
   CUSTODIAL PACKAGE: ORDER ENTRY/RESULT            Salt Lake City
 SUBSCRIBING PACKAGE: UNWINDER                      Salt Lake City
               USAGE: Private             APPROVED: APPROVED
              STATUS: Active               EXPIRES:
            DURATION: Till Otherwise Agr   VERSION:
                FILE:                         ROOT:
         DESCRIPTION:                         TYPE: Routine

     ROUTINE: ORX2
   COMPONENT:  PT1
   VARIABLES:     Use of PT1^ORX2:  This unlocks the patient when exiting an
                  'add orders' context.


                             ********************

                NAME: DBIA344-F              ENTRY: 850
   CUSTODIAL PACKAGE: ORDER ENTRY/RESULT            Salt Lake City
 SUBSCRIBING PACKAGE: UNWINDER                      Salt Lake City
               USAGE: Private             APPROVED: APPROVED
              STATUS: Active               EXPIRES:
            DURATION: Till Otherwise Agr   VERSION:
                FILE:                         ROOT:
         DESCRIPTION:                         TYPE: Routine

     ROUTINE: ORUTL
   COMPONENT:  READ
   VARIABLES:     Use of READ^ORUTL:  This awaits user input in a manner
                  consistant with OE/RR.
               NAME: DBIA344-G            ENTRY: 851
  CUSTODIAL PACKAGE:
                     ORDER ENTRY/RESULT          Salt Lake City
 SUBSCRIBING PACKAGE: UNWINDER                    Salt Lake City
               USAGE: Private           APPROVED: APPROVED
              STATUS: Active             EXPIRES:
            DURATION: Till Otherwise Agr  VERSION:
                FILE:                       ROOT:
         DESCRIPTION:                       TYPE: Routine

      ROUTINE: ORGKEY
    COMPONENT:  SET
    VARIABLES:     Use of SET^ORGKEY:  This sets up keywords that are allowed
                   during the 'add orders' context.

                              ********************

               NAME: DBIA344-H            ENTRY: 852
   CUSTODIAL PACKAGE: ORDER ENTRY/RESULT          Salt Lake City
 SUBSCRIBING PACKAGE: UNWINDER                    Salt Lake City
               USAGE: Private           APPROVED: APPROVED
              STATUS: Active             EXPIRES:
            DURATION: Till Otherwise Agr  VERSION:
                FILE:
                                            ROOT:
         DESCRIPTION:                       TYPE: Routine

      ROUTINE: ORUHDR
    COMPONENT:  OE
    VARIABLES:     Use of OE^ORUHDR:  This sets up the menu headers
                   appropriate for OE/RR with the display of a protocol menu
                   during an 'add orders' context.  The variable ORUIEN, which
                   identifies the menu, is set and killed.

                              ********************

               NAME: DBIA351-D            ENTRY: 860
   CUSTODIAL PACKAGE: KERNEL                      San Francisco
 SUBSCRIBING PACKAGE: UNWINDER                    Salt Lake City
               USAGE: Private           APPROVED: APPROVED
              STATUS: Active             EXPIRES:
            DURATION: Till Otherwise Agr  VERSION:
                FILE:                       ROOT:
         DESCRIPTION:                       TYPE: Routine
    Call to ABT^XQ12:  The Unwinder calls ABT^XQ12.  I believe this is part of
    the response time monitoring.  The local variable, XQXFLG, is also checked
    when making this call.  Agreement is made to call ABT^XQ12 and check the
    XQXFLG variable or this needs to be placed on the list of supported
    references.


      ROUTINE: XQ12
    COMPONENT:  ABT
    VARIABLES:
```

Internal Relations

{ XE "Internal Relations" }

There are no internal relations in the Unwinder utility.

Package-Wide Variables

{ XE "Package-Wide Variables" }

There are no package-wide variables in the Unwinder utility.

How to Generate On-Line Documentation

{ XE "How to Generate On-Line Documentation" }{ XE " On-Line Documentation" }

Routines

{ XE "Routines" }
The namespace for the Unwinder utility is XQOR. A listing/printout of any or all of the Unwinder routines can be produced by using the Kernel option XUPRROU (List Routines), which is on the XUPR-ROUTINE-TOOLS menu on the XUPROG (Programmer Options) menu, a sub-menu of the EVE (Systems Manager Menu) option.

```
Systems Manager Menu [EVE]
      Programmer Options [XUPROG]
           Routine Tools [XUPR-ROUTINE-TOOLS]
                List Routines [XUPRROU]
```

When prompted with "routine(s) ? >:" type in XQOR* to get a listing of all Unwinder routines.

The first line of each routine contains a brief description of the general function of the routine. A first-line listing can be produced by using the Kernel option XU FIRST LINE PRINT (First Line Routine Print).

```
Systems Manager Menu [EVE]
      Programmer Options [XUPROG]
           Routine Tools [XUPR-ROUTINE-TOOLS]
                First Line Routine Print [XU FIRST LINE PRINT]
```

%INDEX

{ XE "%INDEX" }
%INDEX is a routine that produces a report called the VA Cross-Referencer—a technical and cross-reference listing of one routine or a group of routines, with a summary of errors and warnings for routines that do not comply with VA programming standards and conventions, a list of local and global variables and what routines they are referenced in, and a listing of internal and external routine calls. In programmer mode: D ^%INDEX. When selecting routines, select XQOR*.

Checksum Routine

{ XE "Checksum Routine" }

XQOR	8446896
XQOR1	12239132
XQOR2	4783934
XQOR3	6956058
XQOR4	5404137
XQORD	5245554
XQORD1	2990726
XQORI001	2683365
XQORINI1	5626907
XQORINI2	5232646
XQORINI3	16095121
XQORINI4	3357818
XQORINI5	366739
XQORINIS	2218558
XQORINIT	10854706
XQORM	2970763
XQORM1	3913139
XQORM2	8125756
XQORM3	6109644
XQORM4	4355781
XQORM5	2832749
XQORM6	8695
XQORMX	5081453
XQORO	11593614

Glossary

{ XE "Glossary" }

Action	M code invoked by a protocol.
Dialog	A type of protocol that contains other protocols which are term protocol. Each term protocol issues a single prompt.
Item	A protocol which has a child relationship to another protocol.
Keyword	A word that is not contained on a menu, but which, when entered by the user, will allow a specified action to be taken. This provides a function similar to secondary menus in Menu Management.
Menu	A selection list from which the user may choose. The selection(s) determine which protocols are executed next.
Navigation	The process of selecting which branches of the Protocol file logic tree should be taken. The protocol file sets up modules of code in a tree-like fashion. User entries determine the pathway to take through the branches of the tree.
Protocol	A file entry which can be viewed as a module of code. Each module of code may optionally contain other modules of code. Other fields in each file entry determine conditions under which the module should be executed, what should be presented to the user, etc.
Stack	A data structure that allows information to be stored in a fashion such that the last information stored is the first information retrieved.
Term	A type of protocol that allows the definition of an individual prompt within a dialog.

Unwind	Another term for navigation. Unwind is sometimes used to convey the idea of a single protocol causing the invocation of other protocols, which in turn invoke additional protocols, etc.
^^jump	A double up-arrow jump. This is the syntax a user may use to begin execution of a protocol which is not on the current menu. When execution of the protocol is completed, the user is returned to the original menu and the context of the original menu is restored.

Index