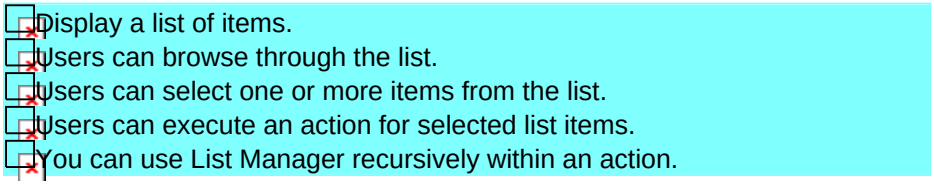
List Manager Developer's Guide

Department of Veterans Affairs
Decentralized Hospital Computer Program
Software Service



Last Updated: April, 1998

Converted to Adobe Acrobat format: November, 1999

Draft

Contents

Getting Started

Introduction

The List Manager Developer's Guide is designed to provide you, the Department of Veterans Affairs (VA) developer, with how to information on creating applications using List Manager. This manual is a full reference for creating List Manager Applications. It is the first revision of the original "draft" List Manager Developer's Guide.

List Manager was originally developed as an interface for the Scheduling module of DHCP's MAS V. 5.2 package. Since then it has been used as an interface for a number of other applications, including Text Integration Utility and NOIS.

List Manager provides a generic method of presenting lists of items to terminal users. Its core functions are:

- Display a list of items.
- Users can browse through the list.
- Users can select one or more items from the list.
- Users can execute an action for selected list items.
- You can use List Manager recursively within an action.

Orientation

Screen Dialog

At a few places in this manual, you are shown a simulation of your interaction with the computer. In order to distinguish computer-supplied prompts from your responses, responses will be in bold type. Like this:

```
    COMPUTER'S PROMPT:  USER'S RESPONSE
```

The return key, the key used to terminate "reads," is shown as **<RET>**.

Entry Points

For entry points that take input variables, the input variable is labeled optional if it is optional; otherwise, it is a required variable.

For entry points that take parameters, parameters are listed in lowercase. This is to convey that the listed parameter name is merely a placeholder; M allows you to pass a variable of any name as the parameter or even a string literal (if the parameter is not being passed by reference).

The following is an example of the documentation format for input parameters:

```
D XGLMSG^XGLMSG(msg_type,[.]var[,timeout])
```

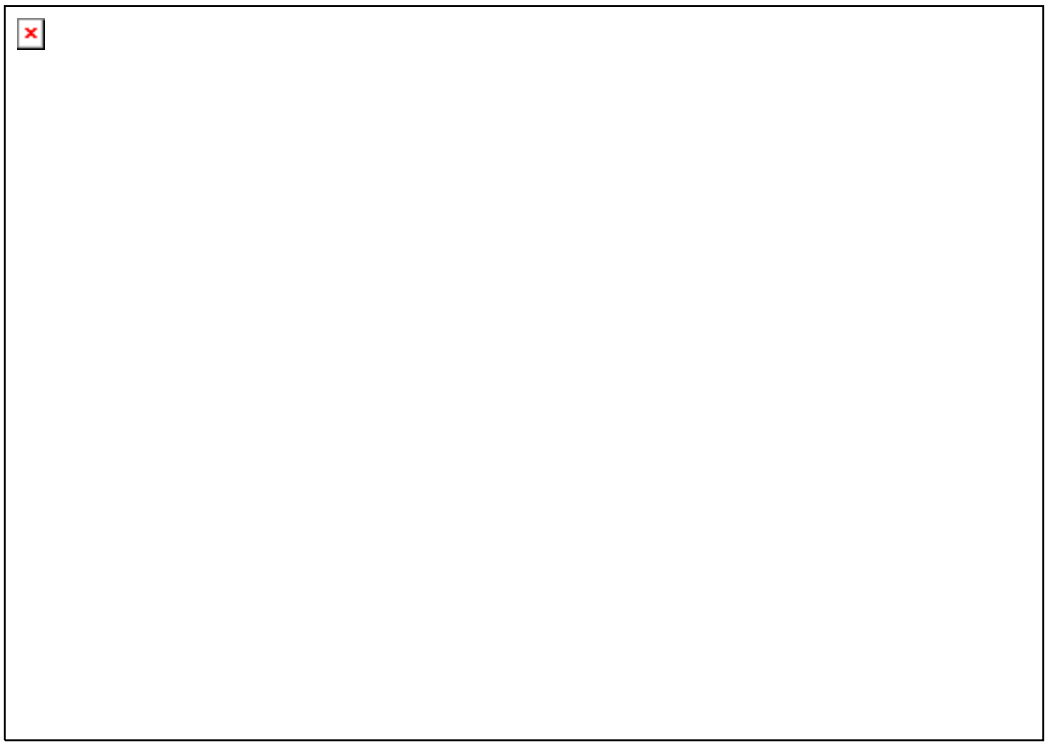
Rectangular brackets [ ] around a parameter are used to indicate that passing the parameter is optional. Rectangular brackets around a leading period in front of a parameter indicate that you can optionally pass that parameter by reference.

List Manager Main Screen

Below is an illustration of the components of a typical List Manager display. The screen is divided into three regions:

- Header area
- List Area
- Action Area

Key	Controlled By
1	Header Code
2	Expand Code
3	Top Margin
4	Bottom Margin, Right Margin
5	Screen Title
6	Caption Line Columns
7	Column
8	Array Name
9	Display Text
10	Date Range Limit



You are only allowed to directly WRITE to the action area. The List Manager controls the other two areas. However, you can modify the contents of header and list areas by using calls in the List Manager API, and by changing the header and list arrays passed to the List Manager.

List Manager Workbench: ^VALMWB

The Workbench allows the development of a List Manager application without having to move from one development tool to another. Load the workbench by calling the routine ^VALMWB.

The Workbench allows you to **edit** all of the data for a list template, action protocols, menu protocols, input templates, and routines; in short, every part of a List Manager application.

You can **run** a List Template from the Workbench. When you run a template, you are prompted for any 'set-up' code to initialize variables. (This is needed if the template is not a top-level template.) After 'running' the template, you are returned to the workbench. (The workbench is List Manager application.)

We recommend that you do all List Template development using the workbench. As new features become available, the workbench will automatically present them to you.

Installation and Setup

Major List Manager Components

1. The List Template file (#409.61)
2. The Protocol file (#101)
3. Routines in the VALM* namespace. (List Manager routines)
4. Routines in the XQOR* namespace. (Protocol Processing routines)

Package Requirements

The following packages and versions must be present to run properly:

Package	Version
OERR	2.5 or greater
Kernel	6.5 or greater

Installation

To install the List Manager, do the following routine in Programmer Mode: D ^VALMINIT.

Terminal Type Attributes for List Manager Users

In order to effectively use the List Manager, the following terminal type attributes must be defined for List Manager users:

TERMINAL TYPE Field	Example Field Values for VT-100 Terminal
Form Feed:	#,$C(27,91,50,74,27,91,72)
XY CRT	W $C(27,91),DY+1,$C(59),DX+1,$C(72)
Erase to End of Page	$C(27,91,74)
Insert Line	$C(27,91),"1L"
Underline On	$C(27,91,52,109)

TERMINAL TYPE Field	Example Field Values for VT-100 Terminal
Underline Off)	$C(27,91,109
High Intensity	$C(27,91,49,109)
Normal Intensity	$C(27,91,109)
Save Cursor Position	$C(27,55)
Restore Cursor Pos	$C(27,56)
Set Top/Bottom Marg	$C(27,91),+IOTM,$C(59),+IOBM,$C(114)
SGR Attributes Off	$C(27,91,109)

How to Make a List Manager Application

1. Define List Template

The first step to create a List Manager application is to create the List Template for your application. A List Template is the core of a List Manager application; all the crucial information that determines how a list works is stored in an application's List Template. The best way set up (and maintain) a List Template is to use the Workbench.

Create a New List Template

When you invoke the Workbench, it asks you for a List Template name. You can either enter an existing one or create a new one.

Create an Outline Routine

List Templates depend on calling several subroutines to perform specific actions, including initializing your application, and creating the array of list items that becomes your list. As such, creating these subroutines is central to your List Template. That is why the next question you're asked after you name your template is "Enter Routine Name:".

The Workbench can create an outline routine that contains subroutines to perform all of the functions List Manager requires. Entering a name is optional. However, if you enter a name for a routine, the Workbench will create an outline routine for your application with stub tags and code for the template. The created List Template is then immediately executable.

Here is what the dialog looks like when you let the Workbench set up an outline routine for your application:

```
Select LIST TEMPLATE NAME: ZZLIST
Are you adding 'ZZLIST' as a new LIST TEMPLATE (the 14TH)? Y (Yes)

>>> The system will create a stub routine...

>>> Enter Routine Name: ZZLIST

I am going to create a series of 'ZZLIST*' routines.
Is that OK? Yes// <RET>(Yes)


>>> Building 'ZZLIST' stub routine.........................
ZZLIST has been filed............
```

A fully functional List Manager application (with a "dummy" list of items) has now been created; and you are placed in the Workbench with the new List Template loaded.

Edit the List Template

The Workbench lets you edit all of the fields in the List Template. It organizes the fields in a list template into six distinct groups:

Demographics	Set up the list name, generic prompt, and screen title.
Protocol Information	Set up the menus for your list.
List Region	Set the screen region for the list.
Other Fields	Set miscellaneous list attributes.
MUMPS Code Related	Specify the routines for Header, Entry, Exit, Expand, and Help; optionally enter array name that list is kept in. When List Manager creates an outline routine, it uses that routine for most of these tasks.
Caption Line Information	Define the contents of the caption line (list headings).

The Workbench also lets you perform a number of actions beyond editing the List Template. One of the actions you can perform is running the list (Run List action). Try running the list now as setup by default by List Manager. This will give you an idea of what a bare bones List Manager application looks like.

Later, as you add enhancements to your application, you will use the Workbench to edit a number of your List Template's fields.

Edit the Outline Routine

Now let's look at the outline routine that was created. It contains six specific subroutines. By going through each subroutine, we will see the beginning of our application.

```
ZZKYLM  ; ; 08-OCT-1996
        ;; ;
EN      ; -- main entry point for ZZLIST
        D EN^VALM("ZZLIST")
        Q
        ;
HDR     ; -- header code
        S VALMHDR(1)="This is a test header for ZZLIST."
        S VALMHDR(2)="This is the second line"
        Q
        ;
INIT    ; -- init variables and list array
        F LINE=1:1:30 D SET^VALM10(LINE,LINE_"     Line number "_LINE)
        S VALMCNT=30
        Q
        ;
HELP    ; -- help code
        S X="?" D DISP^XQORM1 W !!
        Q
        ;
EXIT    ; -- exit code
        Q
        ;
EXPND   ; -- expand code
        Q
        ;
```

Outline Rtn Tag	Description
EN	**Application Entry Point**: This section of the code in the outline routine is the line of code to independently invoke List Manager and load your List Template (and your list). If you were to make an option for your List Manager application, you would set the option's RUN ROUTINE field to this tag and routine.
HDR	**Header Code**: In this very simple section of the outline routine, two nodes of the VALMHDR array are set. These should be set to the text lines to display in the Header area of the List Manager screen. List Manager calls this subroutine when initializing your list.

Outline Rtn Tag	Description
INIT	**List Creation**: In this section of the outline routine, all the work is done to create the list of items that is displayed to the user by List Manager. Setting up your list is discussed in more detail in the next section (Define List).
HELP	**Help**: You can set up custom help in this subroutine. When a user enters a "?" at the menu prompt, your custom help would be called. This is an optional feature.
EXIT	**Exit Code:** Use this subroutine to clean up variables and any other exit processing your application needs to perform before exiting.
EXPND	**Expand Code:** This subroutine is for placing MUMPS code to display a detailed inquiry-type report/screen for a specific entry in the list. This is an advanced, optional feature.

In the next section, Define List, you will edit the outline routine's INIT subroutine, replacing the "dummy" list of items created in the stub subroutine with **your** application's list items. This is the next step in your application - setting up the list of items for List Manager to display to your list user.

What Comes Next?

You've created a list template for your application. You've created an outline routine for your application. So what comes next?

You need to set up the list of items that your application will display to your list user. Setting up the list is the **second** of four steps in creating a List Manager application.

To add functionality to your application, you need to create Action-type protocols. These are akin to menu options, and are the actions available to your list users in the "action area" at the bottom of the List Manager screen. These actions let your list users select items and perform actions with the select items. Creating actions is the **third** of 4 steps in creating a List Manager application.

Finally, once you create some Action-type protocols, you need to create a Menu-type protocol. Then, attach all of your actions to the Menu-type protocol, and designate the menu protocol as your list template's Protocol Menu. Then, run your application and test out all of your actions. Organizing your menu is the **fourth** of 4 steps in creating a List Manager application.

2. Define List Array

Once you have created a List Template to define your List Manager application, the next step is to set up the array (list) of items that will be displayed to your list user. You set up the list array using M code in the routine specified in the List Template's ENTRY field.

Routine to Create List

The routine specified in the ENTRY field in the MUMPS Code Related section of the Workbench is what List Manager calls to set up your list. So you must set your list array up in a routine.

If you let List Manager create an outline routine for your List Template, it sets this field in the List Template to the INIT label of the routine it creates. In the created outline routine, it sets up a "dummy" list using the SET^VALM10 entry point. If you look at the code it puts in this subroutine, you can see one way to create a list. You can set up a list entirely yourself, or you can use some of List Manager's entry points. Both methods are described below.

Array to Store the List In

The ARRAY field in a List Template, in the MUMPS Code Related section of the Workbench, should contain the name of the array will hold your list of items to be displayed. Note that a space character must precede the array name. This is needed to allow global specification. (FM will not allow '^' as the first character.) The array can be either a local or global variable.

The array of list items you create needs to follow the format used in word processing fields:

```
^TMP("SDAM",$J,line #,0)=display_string
```

There is one case in which you don't need to specify the array name in the ARRAY field. By making calls to SET^VALM10, you can have the List Manager decide where to store the list array. This method of creating a list is discussed below.

Build the List Array Yourself

You can easily create a list of items yourself. To do this:

1. In the routine called by the ENTRY field of the List Template, make an array of items in the list. Make sure your array is in the same format as word processing fields, that is, ^TMP("SDAM",$J,line #,0)=display_string). The list array should start with list item 1, and there should be no gaps in the array line sequence.
2. It's a good idea to include the line number as the first part of the text of each display line. This aids list users when selecting items.
3. Set the ARRAY field of the List Template to the name of the array.
4. Set the variable VALMCNT equal to the number of items in your list.
5. You're done!

Somewhere else, you may want to store a corresponding index of the entry number for items in your list, if your items correspond to entries in a file. Then when you get to making actions, you will be able to associate an item in the list with the entry number from which it came.

Build the List Array Using List Manager's API

List Manager provides an API, which includes entry points for creating and maintaining lists.

Creating the Array with SET^VALM10

You can create the array entries in your list using the SET^VALM10 entry point. When you do this, you don't need to set an explicit array name in the List Templates ARRAY field. List Manager will maintain the array itself, without you needing to know where it is stored. If you need to reference lines in the array, you can use the @VALMAR@(<line #>,0) syntax.

To setup and maintain your array using SET^VALM10:

1. All of the code that follows should be in the routine called by the ENTRY field of the List Template.
2. Keep in mind that your list array should start with list item 1, and that there should be no gaps in the array sequence of lines.
3. To add a line to the list, make a call to SET^VALM10:

```
D SET^VALM10(line_num,display_text)
```

4. It's a good idea to include the line number as the first part of the text of each display line. This aids list users when selecting items.
5. If the items in your list correspond to file entries, you may want to keep track of the internal entry number for each list item. Simply use the optional third parameter of the SET^VALM10 call to associate an internal entry number with your list item. You can then retrieve the associated internal entry number for any line with the code:

```
S Y=$O(@VALMAR@("IDX",56,"")).
```

6. When you are done adding lines to the list, set the variable VALMCNT equal to the number of items in your list.
7. You're done!

Setting up the Text Lines with Captions and $$SETFLD^VALM1

To help formatting each line of text for display, you may want to consider using captions and $$SETFLD^VALM1. This lets you format text in a line based on any **caption items** you may have set up in your list template. In the Caption Line Information section of the Workbench, you can enter caption items. Each caption item has a name, length, column position, default video attributes, and display text fields. $$SETFLD^VALM1 lets you **position** pieces of text in your list lines based on how you set up captions for your line in the List Template.

So, supposing you have set up 4 caption items in your List Template, named "LINENO", "NAME", "INITIAL", and "FM ACCESS CODE". When you create your list array, you could loop through entries in the NEW PERSON file, and format a line to display for each NEW PERSON entry as follows:

```
 S LINE=0,EN=.9 F  S EN=$O(^VA(200,EN)) Q:'+EN  D
 .S LINE=LINE+1
 .S ZZNODE0=$G(^VA(200,EN,0)),LINEVAR=""
 .S ZZNA=$P(ZZNODE0,U,1),ZZIN=$P(ZZNODE0,U,2),ZZFM=$P(ZZNODE0,U,4)
 .S LINEVAR=$$SETFLD^VALM1(LINE_".",LINEVAR,"LINENO")
 .S LINEVAR=$$SETFLD^VALM1(ZZNA,LINEVAR,"NAME")
 .S LINEVAR=$$SETFLD^VALM1(ZZIN,LINEVAR,"INIT")
 .S LINEVAR=$$SETFLD^VALM1(ZZFM,LINEVAR,"FM ACCESS CODE")
 .D SET^VALM10(LINE,LINEVAR) ; adds formatted line to list array
```

Now your lines of text are set up according to your captions in your List Template. And if you adjust the positions of your List Template captions, your text lines are automatically adjusted too!

Note: If you have a large NEW PERSON file, and you try this example, make sure you loop only through some subset of it; lists become difficult to use once there are more than a certain number of screens in the list (10 screens in a list is probably a good limit!).

Setting and Displaying Video Attributes for List Lines with FLDCTRL^VALM10

In the Caption Line Information section of the Workbench, you can enter caption items. Each caption item has a name, length, column position, default video attributes, and display text fields. This provides a way to organize your lines of text, based on caption positions.

Using the FLDCTRL^VALM10 entry point, you can set the video attributes for different portions of your line based on the default video attributes entered for every caption in the line. For example, you may have a caption of length 10 starting at column 40, with a default video attribute of REVERSE. If you call FLDCTRL^VALM10 for a line number, all default video attributes for the line will be activated, and the region of that line from column 40 to column 49 will be displayed in reverse.

To activate the default video attributes for all lines in your array:

1. Using the Workbench, set up caption items for each portion of your display line. Set default video attributes as desired for each caption item.
2. **After** you add each line to the list array, make a call FLDCTRL^VALM10(line_num). So you will need to call FLDCTRL^VALM10 once for each line you add to the array.
3. When you run your list, each line you called FLDCTRL^VALM10 for will be displayed with the video attributes set up in the List Template captions.

3. Define List Actions

Once you've created your list template, and your list, the next step is to create actions for your list. Actions are what appear as menu items in the bottom of the List Manager screen. They allow you to launch any routine from a List Manager menu. Actions are stored as protocols, of type **action**, in the PROTOCOL File.

List Manager supplies a set of pre-defined actions that you include with your List Manager application. It's usually a good idea to make use some of these, such as VALM DOWN A LINE, VALM UP A LINE, VALM NEXT SCREEN, etc. to provide the basic list functionality users expect.

In addition, you will probably want to **define your own actions** to add your own custom functionality to your list.

How To Define an Action

1. From the Workbench, choose PE for Protocol Edit.
2. Add a new protocol.
3. Set the new protocol's TYPE to ACTION.
4. Set the ITEM TEXT field to the menu item text for this action.
5. Set the ENTRY ACTION field to call a routine that will perform your action(s).
6. Use the EXIT ACTION field to set List Manager status variables before returning control to List Manager.
7. Add your new action-type protocol to the menu-type protocol that is the main menu for your application - this makes it a menu item in List Manager. See the next section of this manual, Define a Menu, for how to do this.

Here are some more issues to consider for your actions:

- How to select item(s) from the list in your action.
- How to determine what screen mode the user is in.
- Getting control of the screen.
- What List Manager should do when your action completes.
- How to display a custom message after completing an action.

How to Select List Items

In the routine called by an action, you may want to select an item or items from the List Manager list. One easy way to do this is to make a ^DIR call in your action. Set up the DIR(0) input variable to ask for number(s) in the range of the entire list, or only what items are displayed on the current screen, as follows:

1 item from entire list	`S DIR(0)="N^1:"_VALMCNT_":0"`
1 item from current screen	`S DIR(0)="N^"_VALMBG_":"_VALMLST_":0"`
Set of items from entire list	`S DIR(0)="L^1:"_VALMCNT`
Set of items from current screen	`S DIR(0)="L^"_VALMBG_":"_VALMLST`

The interaction with the user takes place in the lower part of the screen. From the output of the ^DIR call, you have the array number(s) of the selected item(s); you can then perform whatever action you would like with the selected item(s). If the user chooses an item or set of items (as reflected in the output variables from the ^DIR call), you can either process the items immediately, or highlight them (current screen only) for further action.

Another way to select entries is to use the List Manager entry point EN^VALM2. This is a generic selector that prompts the user to select list items from the current screen only. Here is a sample of the code you would could call to select a single entry using EN^VALM2:

```
N ZZVALM,ZZEN
S ZZVALM="DUZ^1^ASDF^ASDF" ;?? Need to confirm how to set this up!
D EN^VALM2(ZZVALM,"O")
S ZZEN=$O(VALMY("")) ; get line number of selected entry
I '+ZZEN W !,"No Entry Selected!" H 5 Q
W !,"You selected ",@VALMAR@(ZZEN,0) H 5
Q
```

Using the Entire Screen

If your action needs control over the **entire screen**, make a call to FULL^VALM1 at the beginning of your action's code. This call will change the scrolling region to be the full screen, and turns word wrap on, and all user interaction will be in scrolling mode. When you return control back to the List Manager, set VALMBCK to 'R'. This refreshes the screen and resets the scrolling region as needed by List Manager.

When Your Action Completes

When returning to the List Manager from a protocol action, make sure the variable **VALMBCK** is set. This tells List Manager what to do when returning from your action:

Value	Description
R	Refresh Screen
<null>	Clear bottom portion of screen and prompt for action
Q	Exit (quit) List Manager

If not defined after an action, the List Manager will act like it was set to 'Q'.

If you want to display a **custom message** in the message window after completing an action, set the variable **VALMSG** with the text desired. The message area allows up to 50 characters. For more information, see the description of MSG^VALM10.

4. Define List Menu

The final step in building a List Manager application is to create the menu for your list. This provides the set of choices at the bottom of the List Manager screen. You can create new actions to add to your menu, and/or use generic List Manager actions as well.

Steps to Set Up Your Application's Menu

1. From the Workbench, choose PE for Protocol Edit.
2. Add a new protocol.
3. Set the new protocol's TYPE to MENU.
4. Set the new protocol's COLUMN WIDTH as follows:

# of Columns Desired	Column Width Setting
1	1
2	40
3	26

5. Set the new protocol's MNEMONIC WIDTH to a width that provides for the length of your longest menu item mnemonic, plus white space to separate the mnemonics from the menu text. If your longest mnemonic is 2 characters, setting this field to 4 provides 2 characters of white space.
6. Add any actions (either custom actions created by you, or generic actions) as ITEMS to the new protocol. You can set a mnemonic and a sequence number for each item.
7. You **must** include the following code in the HEADER field of the menu protocol:

```
D SHOW^VALM
```

   This routine properly displays the list of actions to the user in the action area.
8. In the MENU PROMPT field, set the text for users to be prompted by. "Select Action: " is a good choice, for example.
9. Once you finish editing the menu protocol, return to the Workbench. Set the TYPE OF LIST to PROTOCOL (not DISPLAY!). This enables the list to use your new protocol, instead of the standard VALM DISPLAY protocol.
10. Set the PROTOCOL MENU to the name of the menu-type protocol you just created.
11. Test your new menu by choosing "Run List" from the Workbench.

You should consider the following additional issues when setting up protocols for use by the List Manager:

The Hidden Menu

In the workbench, you can set your list's Hidden Menu to the name of any menu protocol. This is typically used to provide some of the more basic actions like line up and line down, especially when the main menu has a lot of custom items. By default, the workbench sets up lists to use the generic VALM HIDDEN ACTIONS protocol as the hidden menu. This provides access to all of the generic List Manager actions for negotiating the list. You can set the hidden menu to your own hidden menu, if you wish.

Columnar Arrangement of Menu Items

If the number of columns desired for your menu items is more than one and if you want to place each action in a particular column, you should use a SEQUENCE numbering scheme for the items in the menu.

List Manager displays your menu items in the minimum number of rows possible, given the number of items and the number of columns you've specified. It will place items in sequence as follows:

1	4	7
2	5	8
3	6	9

So knowing how List Manager places items, you can use sequencing to control which column an item is placed in.

If the number of items to appear in each column is not equal then you must add 'blank' items and place the blank protocol in the appropriate column as described above.

A 'blank' protocol is an action protocol with the ITEM TEXT and ENTRY ACTION fields left blank.

Sub-Menus

If you use a sub-menu, then the HEADER field of the (?? top menu?) should contain a W "". (?? need more on submenus.)

Overriding the Default Action

The List Manager will automatically provide a default action of 'next screen' or 'quit'. However, you can override this default action by setting XQORM("B") as part of the ENTRY ACTION code for a PROTOCOL menu.

5. Fine Tune Your Application

A number of ways that you can fine-tune a basic List Manager application are discussed in this section:

Entry Selection and Light Bar Scrolling

List Manager does not support a scrolling "light bar" for entry selection. When the user presses the up and down arrow keys, there is not a way to hook those key presses to a scrolling light bar in the list of entries.

For entry selection, the best method is to make sure that in the text of each line, the line number is shown (preferably on the left hand side of the line). Then, you can make your own call using ^DIR, or use the EN^VALM2 generic selector, to let your users choose entries. If you want to select an entry and perform an action all at once, you can do this. Another style is to have one action that "selects" entries. You can then use SELECT^VALM10 to highlight that line of the array. This is useful if there are multiple actions a user can perform on a selected entry or entries; you can let the user select the entries, highlight them, and then have the user perform actions on the set of highlighted entries.

Setting Video Attributes in Your List Line

One enhancement you can make to your list application is setting and changing the video attributes in your list lines.

Before you load your list, for example, you can set what the video attributes (highlight, reverse video, underline, or blinking) should be for any given caption field in a line. Do this in the List Template, by editing the Default Video Attributes for your captions. Then when you build your array list initially, you can activate these list template attributes for each line by making calls to FLDCTRL^VALM10.

Once your list is already up and displayed, you can still change the video attributes of your lines. To change video attribute based on screen position, use CNTRL^VALM10. You can save (SAVE^VALM10) and restore (RESTORE^VALM10) a line's video attributes.

You can also "select" a line using SELECT^VALM10.

Updating Items in the List

Another enhancement you can make to your list application is actively updating the lines in your list. While you cannot **add** lines to the list, you can change the contents of existing lines. This is useful, particularly if in your actions you are editing file entries, whose contents correspond to what is displayed in your list.

When a user updates an entry, you can update the corresponding list array line with a call to FLDTEXT^VALM10, and then re-paint the line on the display with a call to WRITE^VALM10. You can also insert text into an existing line based on caption position, using FLDTEXT^VALM10.

When the User Is In Scrolling Mode (not Screen Mode)

The variable **VALMCC** will always be available to indicate the user's **screen mode** in List Manager: (1 means 'screen mode' and 0 means 'scrolling mode'.)

If the user is signed on to the system using a terminal type that does not support the cursor control fields needed by the List Manager, List Manager automatically defaults to scrolling mode. This means that the list array and headers will always be totally re-painted to the screen after each action.

There may be times that the application code will need to know if the job is in scrolling mode. For example, if only one field in one entry is to be changed as a result of an action and the user was working totally in the 'action area' of the screen, then the code could simply use the appropriate call to update just that field and set VALMBCK to null. However, if the user is in scrolling mode, then you would not update the screen and would set VALMBCK to 'R'.

Scroll-Locking Columns

If your list display is going to be more than will fit on a user's screen (greater than 80 or 132 columns), you can set a scroll lock, so that to the left of the scroll lock, no scrolling will occur. This feature is based on caption fields (another good reason to set up your lines using caption fields). You can only set one caption field as the point at which no scrolling will occur. That

field, and everything to the left of it, will be stationary when the user scrolls the rest of the list to the right.

Browsing Word Processing Fields

It is easy to "browse" word processing fields using List Manager. Set the Type of your template to Display. This will provide a menu of standard actions (line up, line down, etc.). Then, for the array, simply set the ARRAY NAME to the global location of your word processing field. List Manager expects the array to be in the format of a word processing field, so at that point you are done.

You can also launch the VA FileMan Browser from within List Manager to browse a word processing field or global array. As different mix of features is offered when browsing word processing fields with the VA FileMan browser.

Long Lists

You should not use List Manager to display very long lists of entries. Although there is no limit other than that of system resources on the size of a list, you may find that users have difficulty if there are more than, for example, 10 screens in the list. The exact limit on the number of screens may depend on the type of information in the list, and how willing your user is to go through such a list. At some point, performance also becomes a consideration, especially if you are building your list array.

Calling List Manager and Other Programs from Actions

From an action in your List Manager application, you can call List Manager again. It is re-entrant. You can also call other applications, for example ScreenMan, the VA FileMan Browser. You do not need to NEW any variables when calling these applications.

6. Export Your List Manager Application

Kernel V. 8.0's Kernel Installation and Distribution System (KIDS) made List Manager Templates and protocols standard package components. This enables List Manager applications to be distributed just like any other package, using KIDS.

To export your List Manager application, you need to export your application's **protocols** and your application's **list template**, as well as routines, options, and any other supporting components.

Protocols

With Kernel V. 8.0's Kernel Installation and Distribution System (KIDS), you can include protocols as package components in a KIDS build. You can then export your List Manager application in a KIDS build.

Prior to Kernel V. 8.0, in order to export protocols, you would have needed to use the ORVOM tool (for more information of the ORVOM process, see the 'Order Entry/Results Reporting Developer's Guide'.)

List Templates

With Kernel V. 8.0's Kernel Installation and Distribution System (KIDS), and with Kernel patch XU*8*2 is installed, you can include list templates as package components in a KIDS build. You can then export your List Manager application in a KIDS build.

Before Kernel 8.0

Prior to Kernel V. 8.0, in order to export list templates, you would have needed to use the ^VALMW3 List Manager utility.

Example Code

LIST TEMPLATE PROTOCOL MENU

This is an example of a protocol menu that would be attached to a list template that has a type of PROTOCOL.

```
NAME: SDAM MENU
ITEM TEXT: Appointment Management
TYPE: menu
PACKAGE: SCHEDULING
DESCRIPTION:   This menu contains all the activities for the appointment
management option.
COLUMN WIDTH: 26
MNEMONIC WIDTH: 4

ITEM: SDAM APPT CHECK IN            MNEMONIC: CI            SEQUENCE: 11
ITEM: SDAM APPT UNSCHEDULED         MNEMONIC: UN            SEQUENCE: 12
ITEM: SDAM APPT MAKE                MNEMONIC: MA            SEQUENCE: 13
ITEM: SDAM APPT CANCEL              MNEMONIC: CA            SEQUENCE: 21
ITEM: SDAM APPT NO-SHOW             MNEMONIC: NS            SEQUENCE: 22
ITEM: SDAM LIST MENU                MNEMONIC: AL            SEQUENCE: 23
ITEM: SDAM PATIENT CHANGE           MNEMONIC: PT            SEQUENCE: 31
ITEM: SDAM CLINIC CHANGE            MNEMONIC: CL            SEQUENCE: 32
ITEM: SDAM DATE CHANGE              MNEMONIC: CD            SEQUENCE: 33

HEADER: D SHOW^VALM
MENU PROMPT: Select Action:
```

PROTOCOL MENU

This menu is a sub-menu of the SDAM APPOINTMENT MENU. Please note the header.

```
NAME: SDAM LIST MENU
ITEM TEXT: Appointment Lists
TYPE: menu
PACKAGE: SCHEDULING
COLUMN WIDTH: 40
ITEM: SDAM LIST CHECKED IN            MNEMONIC: CI
ITEM: SDAM LIST NO SHOWS              MNEMONIC: NS
ITEM: SDAM LIST ALL                   MNEMONIC: TA
ITEM: SDAM LIST NO ACTION             MNEMONIC: NA
ITEM: SDAM LIST CANCELLED             MNEMONIC: CA
ITEM: SDAM LIST FUTURE                MNEMONIC: FU
ITEM: SDAM LIST INPATIENT             MNEMONIC: IP
ITEM: SDAM LIST NON-COUNT             MNEMONIC: NC

EXIT ACTION: S:'$D(VALMBCK) VALMBCK="" D EXIT^SDAM
ENTRY ACTION: S XQORM(0)="1A"
HEADER: W ""
MENU PROMPT: Select List:
MENU DEFAULT: No Action Taken
```

PROTOCOL ACTION

```
NAME: SDAM LIST CANCELLED
ITEM TEXT: Cancelled
TYPE: action
PACKAGE: SCHEDULING
DESCRIPTION:  This list will display all the cancelled appointments for the
date range specified.
ENTRY ACTION: S X="CANCELLED" D LIST^SDAM
 Appendix B  - Sample List Template File Entries
PROTOCOL TYPE

NAME: SDAM APPT MGT
TYPE OF LIST: PROTOCOL
HIDDEN PROTOCOL MENU: VALM HIDDEN ACTIONS
LEFT MARGIN: 1
RIGHT MARGIN: 80
TOP MARGIN: 5
BOTTOM MARGIN: 14
RIGHT MARGIN: 80
OK TO TRANSPORT?: OK
USE CURSOR CONTROL: YES
ENTITY NAME: Appointment
PROTOCOL MENU: SDAM MENU
SCREEN TITLE: Appt Mgt Module
ALLOWABLE NUMBER OF ACTIONS: 1
DATE RANGE LIMIT: 999
ARRAY NAME:  ^TMP("SDAM",$J)
ITEM NAME: NAME     COLUMN: 9     WIDTH: 22    DISPLAY TEXT: Patient or Clinic
ITEM NAME: DATE     COLUMN: 32    WIDTH: 20    DISPLAY TEXT: Appt Date/Time
ITEM NAME: STAT     COLUMN: 53    WIDTH: 22    DISPLAY TEXT: Status
ITEM NAME: APPT#    COLUMN: 5     WIDTH: 3
ITEM NAME: TIME     COLUMN: 75    WIDTH: 5

EXPAND CODE: D EN^SDAMEP
EXIT CODE: D FNL^SDAM
HEADER CODE: D HDR^SDAM
HELP CODE: D HLP^SDAM5
ENTRY CODE: D INIT^SDAM
```

DISPLAY TYPE

```
NAME: SDAM APPT PROFILE
TYPE OF LIST: DISPLAY
HIDDEN PROTOCOL MENU: VALM HIDDEN ACTIONS
TOP MARGIN: 5
BOTTOM MARGIN: 17
RIGHT MARGIN: 80
OK TO TRANSPORT?: OK
USE CURSOR CONTROL: YES
SCREEN TITLE: Expanded Profile
ALLOWABLE NUMBER OF ACTIONS: 2
ARRAY NAME:  ^TMP("SDAMEP",$J)
EXIT CODE: D FNL^SDAMEP
HEADER CODE: D HDR^SDAMEP
HELP CODE: D HLP^SDAM5
ENTRY CODE: D INIT^SDAMEP
```

Application Code Examples

Examples of List Manager application code:

```
SDAM    ;; - main code

EN      ; -- main entry point
        K XQORS,VALMEVL D EN^VALM("SDAM APPT MGT")
        Q
        ;
INIT    ; -- set up appt man vars and list man array and other vars
        K I,X,SDBEG,SDEND,SDB,XQORNOD,SDFN,SDCLN,DA,DR,DIE,DNM,DQ
        S DIR(0)="43,213",DIR("A")="Select Patient name or Clinic name"
        D ^DIR K DIR I $D(DIRUT) S VALMQUIT="" G INITQ
        S SDY=Y
        I SDY["DPT(" S SDAMTYP="P",SDFN=+SDY D INIT^SDAM1
        I SDY["SC(" S SDAMTYP="C",SDCLN=+SDY D INIT^SDAM3
INITQ   Q
        ;
HDR     ; -- screen header set up
        N X
        I SDAMTYP="P" D HDR^SDAM10
        I SDAMTYP="C" D HDR^SDAM3
        S X=$P(SDAMLIST,"^",2)
        S VALMHDR(2)=$$SETSTR^VALM1($$FDATE^VALM1(SDBEG)_" thru
"_$$FDATE^SSDEND),X,59,22)
        Q
        ;
FNL     ; -- what to do upon exiting list man
        K ^TMP("SDAM",$J),^TMP("SDAMIDX",$J),^TMP("VALMIDX",$J)
```

```
        K SDAMCNT,SDFLDD,SDACNT,VALMHCNT,SDPRD,SDFN,SDCLN,SDAMLIST,SDT,SDAT
EG,SDEND,DFN,Y,SDAMTYP,SDY,X,SDCL,Y,SDDA,VALMY
        Q


HLP     ; -- help for list
        I $D(X),X'["??" D HLPS,PAUSE^VALM1 G HLPQ
        D CLEAR^VALM1
        F I=1:1 S SDX=$P($T(HELPTXT+I),";",3,99)
        Q:SDX="$END"
        D PAUSE^VALM1:SDX="$PAUSE" Q:'Y W !,$S(SDX["$PAUSE":"",1:SDX)
        W !,"Possible actions are the following:"
        D HLPS,PAUSE^VALM1 S VALMBCK="R"
HLPQ    K SDX,Y Q
        ;
HLPS    ; -- short help
        S X="?" D DISP^XQORM1 W ! Q
        ;
HELPTXT ; -- help text
        ;;Enter actions(s) by typing the name(s), or abbreviation(s).
        ;;
        ;;ACTION PRE-SELECTION:
        ;; Actions may be pre-selected by separating them with ";".
        ;; .
        ;; .
        ;; .
```

SDAMEP ;; - expand code

```
EN      ; Selection of appointment
        K ^TMP("SDAMEP",$J)
        S VALMBCK=""
        D SEL G ENQ:'$D(SDW)!(SDERR)
        W ! D WAIT^DICD,EN^VALM("SDAM APPT PROFILE")
        S VALMBCK="R"
ENQ     Q

VALMD   ;List Manager Sample Routine; APR 2, 1992
        ;
EN      ; -- option entry point
        K XQORS,VALMEVL
        D EN^VALM("VALM DEMO APPLICATION")
ENQ     Q
        ;
        ;
INIT    ; -- build array
        W ! S DIC("A")="Select Package:",DIC="^DIC(9.4,",DIC(0)="AEMQ" D ^DIC
K DIC
        I Y<0 S VALMQUIT="" G INITQ
PKG     ; -- entry pt if package known
        N VALMX,VALMCNTI,VALMPRO,VALMIFN,X,VALMPRE,Z
        S VALMPKG=+Y
        D CLEAN^VALM10
```

```
       S
(VALMCNTI,VALMCNT)=0,(VALMPRE,VALMPRO)=$P($G(^DIC(9.4,VALMPKG,0)),U,2)
        F S VALMPRO=$O(^ORD(101,"B",VALMPRO))
        Q:$E(VALMPRO,1,$L(VALMPRE))'=VALMPRE
        S VALMIFN=0 F S VALMIFN=$O(^ORD(101,"B",VALMPRO,VALMIFN)) Q:'VALMIFN
I $D(^ORD(101,VALMIFN,0)) S VALMX=^(0) D
        .S VALMCNTI=VALMCNTI+1 W:(VALMCNTI#10)=0 "."
        .S X=$$SETFLD^VALM1(VALMCNTI,"","NUMBER")
        .S X=$$SETFLD^VALM1($P(VALMX,U),X,"NAME")
        .S X=$$SETFLD^VALM1($P(VALMX,U,2),X,"TEXT") K Z S
$P(Z,$E(VALMCNTI),240)=""
        .S VALMCNT=VALMCNT+1
        .D SET^VALM10(VALMCNT,$E(X_Z,1,240),VALMCNTI) ; set text
        .S ^TMP("VALMZIDX",$J,VALMCNTI)=VALMCNT_U_VALMIFN
        .D:'(VALMCNT#9) FLDCTRL^VALM10(VALMCNT) ; defaults for all fields
        .D FLDCTRL^VALM10(VALMCNT,"NUMBER") ; default for 1 field
        .D:'(VALMCNT#5) FLDCTRL^VALM10(VALMCNT,"NAME",IOUON,IOUOFF) ; adhoc
        D NUL:'VALMCNT
INITQ   Q
        ;
HDR     ; -- demo header
        N VALMX
        S VALMX=$G(^DIC(9.4,VALMPKG,0)),X=" Package:"_$P(VALMX,U)
        S VALMHDR(1)=$$SETSTR^VALM1("Prefix:"_$P(VALMX,U,2),X,63,15)
        S VALMHDR(2)="Description: "_$E($P(VALMX,U,3),1,65)
        Q
        ;
NUL     ; -- set null message
        I 'VALMCNT D
        .F X=" "," No protocols to list." S VALMCNT=VALMCNT+1 D
SET^VALM10(VALMCNT,X)
        .S ^TMP("VALMZIDX",$J,1)=1,^(2)=2
        Q
        ;
FNL     ; -- clean up
        K DIE,DIC,DR,DA,DE,DQ,VALMY,VALMPKG,^TMP("VALMZIDX",$J)
        D CLEAN^VALM10
        Q
        ;
EXP     ; -- expand action
        D FULL^VALM1
        N VALMI,VALMAT,VALMY
        D EN^VALM2(XQORNOD(0),"O") S VALMI=0
        F S VALMI=$O(VALMY(VALMI)) Q:'VALMI D
        .S VALMAT=$G(^TMP("VALMZIDX",$J,VALMI))
        .W !!,@VALMAR@(+VALMAT,0),!
        .S DA=+$P(VALMAT,U,2),DIC="^ORD(101,",DR="0"
        D EN^DIQ,PAUSE^VALM1
        S VALMBCK="R"
        Q
        ;
EDIT    ; -- edit action
        N VALMA,VALMP,VALMI,VALMAT,VALMY
        D EN^VALM2(XQORNOD(0),"O") S VALMI=0 ; allow the user to "O"ptionally
answer
        F S VALMI=$O(VALMY(VALMI)) Q:'VALMI D
```

```
        .D SELECT^VALM10(VALMI,1) ; -- 'select' line
        .S VALMAT=$G(^TMP("VALMZIDX",$J,VALMI))
        .W !!,@VALMAR@(+VALMAT,0)
        .S DA=+$P(VALMAT,U,2),VALMP=$G(^ORD(101,DA,0)),DIE=19,DR="1" D ^DIE K
DIE,DR
        .S VALMA=$G(^ORD(101,DA,0))
        .I $P(VALMP,U,2)'=$P(VALMA,U,2) D UPD($P(VALMA,U,2),"TEXT",.VALMAT)
        .D SELECT^VALM10(VALMI,0) ; -- 'de-select' line
        S VALMBCK=$S(VALMCC:"",1:"R")
        Q
        ;
DESC    ; -- display description action
        N VALMI,VALMY,VALMAT
        D EN^VALM2(XQORNOD(0),"OS") S VALMI=0 ; select only a "S"ingle
protocols
        F S VALMI=$O(VALMY(VALMI)) Q:'VALMI D
        .S VALMAT=+$P($G(^TMP("VALMZIDX",$J,VALMI)),U,2)
        .I '$D(^ORD(101,VALMAT,1)) W !!,"No Description entered." D
AUSE^VALM1 Q
        .D WP^VALM("^ORD(101,"_VALMAT_",1)",$P($G(^ORD(101,VALMAT,0)),U))
        S VALMBCK="R"
        Q
        ;
UPD(TEXT,FLD,VALMAT) ; -- update data for screen
        D:VALMCC FLDCTRL^VALM10(+VALMAT,.FLD,.IOINHI,.IOINORM,1)
        D FLDTEXT^VALM10(+VALMAT,.FLD,.TEXT)
        Q
        ;
CHG     ; -- change package action
        K X I $D(XQORNOD(0)) S X=$P($P(XQORNOD(0),U,4),"=",2)
        I X="" R !!,"Select Package: ",X:DTIME
        S DIC="^DIC(9.4,",DIC(0)="EMQ" D ^DIC K DIC G CHG:X["?"
        I Y<0 D G CHGQ
        .W !!,*7,"Package has not been changed."
        .W ! S DIR(0)="E" D ^DIR K DIR
        .S VALMBCK=""
        D PKG,HDR S VALMBCK="R" S VALMBG=1
CHGQ    Q
```

* Example of stub routine created when adding a new List Template using the Workbench.

```
ZZDEMO  ;; 24-JAN-1993
        ;; ;
EN      ; -- main entry point for DOCUMENTATION DEMO
        D EN^VALM("DOCUMENTATION DEMO")
        Q
        ;
HDR     ; -- header code
        S VALMHDR(1)="This is a test header for DOCUMENTATION DEMO."
        S VALMHDR(2)="This is the second line"
        Q
        ;
INIT    ; -- init variables and list array
        F LINE=1:1:30 D SET^VALM10(LINE,LINE_" Line number"_LINE)
        S VALMCNT=30
```

```
        Q
        ;
HELP    ; -- help code
        S X="?" D DISP^XQORM1 W !!
        Q
        ;
EXIT    ; -- exit code
        Q
        ;
EXPND   ; -- expand code
        Q
        ;
```

List Template Reference

Demographics Fields

NAME (.01)

Name of the List Template. The template should be namespaced.

ENTITY NAME (.09) [optional]

This field contains the term that will be displayed to the user that best describes the items in the list. This field is used by the select entry point (EN^VALM2).

SCREEN TITLE (.11) [optional but recommended] **Screen Title field**

This field contains the text that will be displayed/printed in the upper left hand corner of the screen display.

The screen title can be changed at run time by setting the variable VALM("TITLE") during ENTRY CODE or action processing. If you have one basic List Template definition that could be used for more then one application, then setting VALM("TITLE") allows you to re-use the template but change the title as it appears to the user.

Protocol Information Fields

TYPE OF LIST (.02)

Indicates the type of list template. The type determines what actions are presented to the user.

PROTOCOL type will use the menu protocol specified in the PROTOCOL MENU field.

DISPLAY type will use the standard VALM DISPLAY PROTOCOL supplied by the List Manager

PROTOCOL MENU (.1)

This field contains the name of the protocol menu that will be used by the List Manager if the TYPE OF LIST is 'protocol'. This field is not used for 'display' types.

PRINT PROTOCOL (1.01) [optional]

This field contains the name of the protocol that will be called when the user selects the generic 'Print List' action. Normally, this field is blank and the generic printing action is sufficient.

HIDDEN MENU (1.02) [optional but recommended]

This field contains the name of the protocol menu that will be used by the List Manager for the 'hidden' actions available to the user. Normally, the application enters the 'VALM HIDDEN ACTIONS' menu in this field. However, there maybe applications that would require a different set of 'hidden' actions.

If the List Template has a 'hidden' menu defined the List Manager will automatically display help for the hidden menu when the user enters '??'.

List Region Fields

TOP MARGIN (.05)

This field contains the number of the top row of the scrolling region where the list will be displayed.

BOTTOM MARGIN (.06)

This field contains the number of the bottom row of the scrolling region where the list will be displayed.

RIGHT MARGIN (.04) [optional]

This field indicates the maximum number of characters a row can contain. If this parameter is not set, 80 is used.

(Range: 80 to 240 characters.)

Other Fields

OK TO TRANSPORT ? (.07)

This field indicates to the transport utility if this list template should be distributed.

Note: this field is obsolete now that KIDS is used to transport List Manager applications.

USE CURSOR CONTROL (.08)

This field indicates whether the cursor positioning and character enhancement capabilities of the device should be used. If set to 'NO', then lists will be presented in scrolling mode.

(See X. Site Preparation and Installation section.)

ALLOWABLE NUMBER OF ACTIONS (.12)

This field indicates the number of actions a user can select at one time.

For example, if this parameter is set to 1 then the user can only enter one action.

Allowed: Select Action: NX
Not allowed: Select Action: NX,EP

If this parameter is not entered then the system defaults to 1.

DATE RANGE LIMIT (.13) [optional]

This field contains the maximum number of days that can be specified by the user while entering a date range. This parameter is only used if the applications calls the List Manager's date range entry point (RANGE^VALM1). Date Range Limit field

AUTOMATIC DEFAULTS (.14) [optional]

This field indicates whether List Manager should always supply a default action at the 'Select' prompt for 'Protocol' type List Templates.

If set to 'NO', a default is not provided automatically. It is your responsibility to indicate a default, if desired. This default can be indicated by setting XQORM("B") as part of the protocol menu's HEADER code. For example:

```
D SHOW^VALM S XQORM("B")="Your action")
```

This parameter only is valid for 'Protocol' type List Templates.

If the parameter is set to 'YES' or is blank, a default will be provided by List Manager. If the current screen contains the last line in the list, then the default will be 'Quit'. Otherwise, it will be "Next Screen". However, as discussed above, you can override this default by setting XQORM("B").

MUMPS Code Related Fields

HEADER CODE (100)

This MUMPS field contains the code that the List Manager will execute to create the application specific screen header array. This header must be stored in VALMHDR( ).

The subscripting for VALMHDR() is a simple integer number. For example:

S VALMHDR(1) = "This is the 1st line of the header"

S VALMHDR(2) = "This is the 2nd line of the header"

During action processing, if the header needs to be changed, you can KILL VALMHDR and then SET VALMBCK="R". This will cause List Manager to automatically invoke this HEADER CODE, as part of the re-display of the screen.

ENTRY CODE (106)

This field contains MUMPS code that is executed when the List Manager is called. This code is usually used by the application to initialize variables. Any application specific variables should also be set up here.

List Manager variables to be initialized are:

VALMCNT [required] The number of lines in the list.

VALMBG [optional] The number of the line you want the List Manager to start displaying from a line other than 1. If not defined, it will be set to 1 by List Manager.

VALMQUIT [optional] If during the building of the array, the software determines that the List Manager application cannot continue, this variable should be set. Setting this variable will cause the List Manager to quit the current List Manager application.

The array specified in the ARRAY NAME field is also set up at this time. This array contains the list of items to display. The subscripting of the array should conform to FileMan word processing format.

For example: If ARRAY NAME equals ^TMP("SDTEST",$J) then the list would be stored as follows:

```
^TMP("SDTEST",$J,1,0) = " 1 Smith,John "
```

```
^TMP("SDTEST",$J,2,0) = " 2/2/93@0800am"
```

If you plan to use the entry selection call, EN^VALM2, then the following must also be set:

```
^TMP("SDTEST",$J,"IDX",<line #>,<entry #>) = ""
```

The 'line #' corresponds to the 1 and 2 shown in the above example. The 'entry #' corresponds to an entry in your application. In the example, the two lines each correspond to appointment entry number . So the "IDX" nodes would be set up in the following manner:

```
^TMP("SDTEST",$J,"IDX",1,1)=""
^TMP("SDTEST",$J,"IDX",2,1)=""
```

Also, see ARRAY NAME field for more information on that list template field.

EXIT CODE (105) [optional but recommended]

This field contains MUMPS logic that will be executed by the List Manager when the user exits the list. This should be used to clean up variables and any other exit processing the application needs to perform.

EXPAND CODE (102) [optional]

This field contains the MUMPS code that displays a detailed inquiry-type report/screen for a specific entry in the list. If this field is filled in, then the standard 'display' protocol will have an 'Expanded' action.

The standard VALM EXPAND protocol uses this field to expand an entry. If the type of list is Protocol then add the VALM EXPAND protocol to your custom protocol menu and enter the code in this EXPAND CODE field.

A possible method for expand is to create another List Template that is a DISPLAY type. You need only build display array and set this EXPAND CODE field to be another call to the List Manager, passing in the display template name.

HELP CODE (103) [optional]

This field contains the MUMPS code for custom application help. This code will be executed when the user types a '?' at the 'Select Action: ' prompt.

This field is optional. If this field is left blank, the normal help given by the XQOR* driver will take effect.

If the List Template has a 'hidden' menu defined the List Manager will automatically display help for the hidden menu when the user enters '??'.

ARRAY NAME (107) [optional]

This field contains the name of the array that holds the list of items to be displayed. The code specified in the ENTRY CODE field must create this array initially.

Note: The array name must be preceded by a space character. This is needed to allow global specification. (FM will not allow '^' as the first character.) The array can be either a local or global variable.

The array needs to follow the format used in word processing fields. e.g. ^TMP("SDAM",$J,line #,0)=string

Finally, you do not have to indicate the array in which the list will be located. By making calls to SET^VALM10, you can have the List Manager decide where to store the list array. If you need to reference lines in the array, the use of the @VALMAR@(<line #>,0) syntax is supported. This feature is ideal for a short list of items(e.g. <10 items).

Caption Line Information Fields

CAPTION LINE COLUMNS (200) [optional]

This multiple field contains column definitions for the data displayed in the list. Adding entries to this multiple is optional. The column parameters are used when the List Manager writes the line indicating the top of the list's scrolling region.

ITEM NAME (.01)

This field contains the reference name of the column. The DISPLAY TEXT field contains the text that will be used when the caption line is written. The text in this field is used when the application refers to this column during programming.

COLUMN (.02)

This field contains the column number where the data/caption starts.

WIDTH (.03)

This field contains the number of characters this field will use.

DISPLAY TEXT (.04) [optional]

This field contains the text that will appear on the caption line for this column/field. If the text is longer than the WIDTH parameter, it will be truncated to the WIDTH specification when written as part of the caption line. This field is optional and can be left blank.

DEFAULT VIDEO ATTRIBUTES (.05) [optional]

This parameter allows you to indicate the default video attributes that should be applied when program calls are made to the FLDCTRL^VALM10 entry point.

The following is the list of attributes and abbreviations used for this parameter:

H - for highlight
R - for reverse video

U - for underline
B - for blinking

SCROLL LOCK (.06) [optional]

If you want to lock one for more columns into place as the user scrolls horizontally through the list, you can place a 'scroll lock' on the right most column field that should be locked in place on the screen. Only one column can have this 'scroll lock' parameter set to 'yes'. If you attempt to set more than one, the system will not allow it and will issue a warning.

If this parameter is set to 'YES', this caption field and any other caption field, with a COLUMN parameter set to less than this current caption fields, will always be displayed by the List Manager.

This parameter does not need to be filled in for List Templates with a RIGHT MARGIN of 80. For those templates with a RIGHT MARGIN of over 80, this field also does not need to be entered. However, the use of this field allows you to indicate the list's identification fields for user readability.

Only 1 caption field can have this parameter set to 'YES'.

The local array VALMDDF () is available to you at run time. This array is subscripted by the column field's name and contains information described above:

VALMDDF(<column name>)=<column name> ^ <column> ^ <width> ^ <caption> ^ <video> ^ <scroll lock>

APIs

List Manager Variables

This section lists all of the variables within List Manager that you can either set or refer to in your List Manager application code.

Variable	Description
VALM(TITLE)	The screen title can be changed at run time by setting this variable, during ENTRY CODE or action processing. If you are one basic List Template definition that could be used for more then one application, then setting VALM("TITLE") allows you to re-use the template but change the title as it appears to the user.
VALMBCK	When returning to the List Manager from a protocol action, you should set the variable VALMBCK. This tells List Manager what to do when returning from an action. If not defined after an action, List Manager acts as if it was set to "Q". **R** refresh screen **null** Clear bottom portion of screen and prompt for action **Q** Exit (quit) List Manager
VALMBG	An optional variable you can set in the INIT code that sets up your list. This tells List Manager what line in your list to start displaying the list in (default is line 1). In action protocols, you can also refer to the value of this variable to find the number of the first list line currently displayed on the user's screen.
VALMCC	Always available to indicate the user's screen mode. 1 means screen mode and 0 means scrolling mode.
VALMCNT	The number of the lines in the list. In the INIT code that sets up the list, you must set VALMCNT equal to the number of lines in your list.

Variable	Description

VALMDDF()	This array is available at runtime. It is subscripted by caption field name, so there is one node per caption field in your List Template. Each node contains the following ^-pieces:
	
	1. caption field name
	2. column
	3. width
	4. caption
	5. video (if defined)
	6. scroll lock (if defined)
	
	For example:
	
	`VALMDDF("INIT")=INIT^37^5^Init.`
	`VALMDDF("NAME")=NAME^1^35^ Name^`

VALMHDR()	The header is stored in VALMHDR(). The subscripting for VALMHDR() is a simple integer number. For example:
	
	`S VALMHDR(1) = "1st line of header"`
	`S VALMHDR(2) = "2nd line of header"`
	
	During action processing, if the header needs to be changed, you can kill VALMHDR and then set VALMBCK="R". This will cause List Manager to automatically invoke what is called by the HEADER CODE field as part of the re-display of the screen.

VALMLST	In action protocols, you can refer to the value of this variable to find the number of the last list line currently displayed on the user's screen.
VALMQUIT	If in the INIT code, while building a list, you decide that List Manager should not continue, set this variable to tell List Manager to quit.
VALMSG	To display a custom message in the message window after completing an action, set this variable with the desired text (up to 50 characters).
@VALMAR@(#,0)	If you built your array using SET^VALM10, you can use the@VALMAR@(line#,0) syntax to reference text lines in the array.

Variable	Description

@VALMAR@("IDX")	Location of entry index when you set up an array using SET^VALM10, and pass index entries with each line. The relationship of the list line to the indexed value stored in the global referenced by @VALMAR@("IDX") is:
	
	`^..."IDX",line_num,index_num)=""`
	
	So to retrieve the entry number indexed for line 54 in the array, you could use:
	
	`S Y=$O(@VALMAR@("IDX",56,""))`

XQORM("B")	List Manager automatically provides a default action of 'next screen' or 'quit'. However, you can override this default action by setting XQORM("B") as part of the ENTRY ACTION code for a PROTOCOL menu. Set it to the text of the menu item you would like to be the new default.

Kernel Video Variables

You can use the following standard video control variables in List Manager:

Attribute	**Variable**
Normal Video	IOINORM
High Intensity	IOINHI
Reverse Video On	IORVON
Reverse Video Off	IORVOFF
Underline On	IOUON
Underline Off	IOUOFF
Blink On	IOBON
Blink Off	IOBOFF

These variables can be used in ON and OFF parameters outlined in a number of List Manager calls. If other video attributes are needed, you will need to make the appropriate call to Kernel's ENDR^%ZISS entry point to set up variables for those attributes.

The variables listed in the above table should always remain defined and should not be killed by application code.

Finally, you can specify more than one video attribute in a single call by concatenating the variables. For example, ' D CNTRL^VALM10(1,20,30,IOINHI_IOUON,IOINORM)' would highlight **and** underline 30 characters starting at column 20.

List Manager Generic Action Protocols

The following table lists generic actions in the PROTOCOL file that you can use in your List Manager application.

Note: These generic actions are all attached to the VALM HIDDEN ACTIONS protocol. This is so that you can set your list's HIDDEN MENU protocol to VALM HIDDEN ACTIONS and have your list automatically make all of these actions available to your list users.

Protocol Name	**Protocol Description**
VALM DOWN A LINE	Move down a line.
VALM UP ONE LINE	Move up a line
VALM FIRST SCREEN	This action will display the first screen.
VALM LAST SCREEN	The action will display the last items.
VALM NEXT SCREEN	This action will allow the user to view the next screen of entries, if any exist.
VALM PREVIOUS SCREEN	This action will allow the user to view the previous screen of entries, if any exist.
VALM PRINT LIST	This action allows the user to print the entire list of entries currently being displayed.
VALM PRINT SCREEN	This action allows the user to print the current List Manager display screen. The header and the current portion of the list are printed.
VALM REFRESH	This actions allows the user to re-display the current screen.
VALM SEARCH LIST	Finds text in list of entries.
VALM TURN ON/OFF MENUS	This toggles the menu of actions to be displayed/not displayed automatically.
VALM GOTO PAGE	This protocol will allow the user to move to any page in the list.
VALM RIGHT	This protocol will allow the user to move the screen to the right if the List Template is set up for a width of more then 80 characters.

Protocol Name	Protocol Description
VALM LEFT	This protocol will allow the user to move the screen to the left if the List Template is set up for a width of more then 80 characters.
VALM QUIT	This protocol can be used as a generic 'quit' action.
VALM HIDDEN ACTIONS	This menu protocol contains all the above action protocols. You usually would specify this protocol as the 'Hidden Menu' protocol in the List Template set up. The Workbench automatically designates this protocol as the 'Hidden Menu' protocol when a List Template is initially created.

General

EN^VALM

Invoke ListMan to load a List Manager template/application.

Format

```
D EN^VALM(template_name)
```

Input

template_name	Name of a List Manager template to load.

Output

(none)

SHOW^VALM

Use a call to SHOW^VALM in the HEADER field of all of your menu protocols. This displays the menu to the user.

Format

```
D SHOW^VALM
```

Input

(none)

Output

(none)

PAUSE^VALM1

This will pause the screen. The call uses a ^DIR call with DIR(0) set to "E" for end of page. The prompt will look like:

```
Press RETURN to continue or '^' to exit:
```

Format

```
D PAUSE^VALM1
```

Input

(none)

Output

(none)

RANGE^VALM1

This sub-routine lets the user change a date range.

Format

```
D RANGE^VALM1
```

Input

DATE RANGE LIMIT field	Value as stored in the List Template file.
VALMB	(optional) Default beginning date.

Output

VALMBEG	Beginning date in FM date format.
VALMEND	Ending date in FM date format.

EN^VALM2

This sub-routine is a generic selector that can be used within an action call.

In order to use this call, the List Manager ENTRY CODE **must** to set up the @VALMAR@("IDX") index array. This is done by setting up the list array line-by-line with the SET^VALM10 entry point, and associating an ien with each line created.

Format

```
D EN^VALM2(valmnod, options)
```

Input

valmnod	String in XQORNOD(0) four-piece format: 1. ien of selected item (?? what does this mean) 2. ien of menu (??what does this mean) 3. menu text (??what does this mean) 4. text user entered to select item (?? what does this mean) Example: `S VALMNOD="3^1312^Misc. Consult^3"`	
Options	Selection option flags	
	O	Selection is optional. Otherwise, the user must make a selection or enter an up-arrow.
	S	User can only select one entry. Otherwise, the user can select more than one item.

Output

VALMY()	Array with selected entries as subscripts.

List Line Text

FLDUPD^VALM1

Updates a specific caption field of a specified list line on the display screen. The field name must match a field defined in the CAPTION LINE COLUMNS multiple of the LIST TEMPLATE file.

Format

```
D FLDUPD^VALM1(text, field, entry)
```

Input

text	Text to insert.
field	Caption field name.
entry	Line number of line in the list.

Output

(none)

$$SETFLD^VALM1

This function inserts text in a string based on the column position of Caption fields stored in the current List Template. Typically this is used when you are building the lines to place in your list's array. It helps you easily place text strings in your list lines based on the position of caption headers in the active List Template. If your List Template has 3 captions, you would typically make 3 calls to this function to construct your line - one call each to insert the text corresponding to each caption header.

Format

```
S X=$$SETFLD^VALM1(text, string, field)
```

Input

text	Text to insert.
string	String for text to be inserted into.
field	Caption field name in list template whose column position determines the position in string to insert text at.

Output

return value	String with text inserted.

$$SETSTR^VALM1

This extrinsic function will setup a string for display. Once the string has been set up for display, you would typically set it in the ARRAY specified in the list template; e.g., S ^TMP("SDAM",$J,SDLN)=X.

Format

```
S X=$$SETSTR^VALM1(text, string, column, length)
```

Input

text	Text to insert
string	String to insert text into.
column	Column position to insert text at.
length	Number of characters to clear.

Output

return value	String with text inserted.

Example

```
>S X=$$SETSTR^VALM1("This","",10,4) W !,X
This

>S X=$$SETSTR^VALM1("is",X,20,2) W !,X
```

```
This is

>S X=$$SETSTR^VALM1("an",X,30,2) W !,X
This is an

>S X=$$SETSTR^VALM1("example.",X,40,8) W !,X
This is an example.
```

FLDTEXT^VALM10

Inserts text at the column where the specific field starts in a LINE in the list array.

The FIELD name must match a field defined in the CAPTION LINE COLUMNS multiple of the LIST TEMPLATE file.

Format

```
D FLDTEXT^VALM10(line, field, text)
```

Input

line	Line number in list array to insert text into.
field	Name of a caption field in the List Template. Text will be inserted at the column position corresponding to the specified caption field.
text	Text to insert.

Output

(none)

SET^VALM10

Used to construct the initial list array before displaying the list to the user. Adds one line at a time to the list array.

Note: If the List Template does not define an ARRAY NAME, then you must use this call to build lines in the list array.

Format

```
D SET^VALM10(line, string[, ien])
```

Input

line	Line number in the array to set line at. The list array, when completed, must start at line number 1, and there cannot be any gaps in the line numbering sequence.
string	Text of the line.
ien	(optional) Entry number to associate with the line. If passed, then the line will also be indexed for use by the EN^VALM2 generic list selection call.

Output

(none)

List Line Video

CNTRL^VALM10

Sets the video attributes for a line in the current list.

Format

```
D CNTRL^VALM10(line, column, width, on, off[, save])
```

Input

line	Line number of line to set video attributes for.
column	Screen column position where code should be invoked.
width	How many screen columns the code should be in effect for.
on	Beginning control sequence. See Kernel Video Variables for a set of variables you can use here.
off	Ending control sequence. See Kernel Video Variables for a set of variables you can use here.
save	(optional) 1 to save control sequence for later use (to be restored with RESTORE^VALM10). Otherwise, 0.

Output

(none)

FLDCTRL^VALM10

Activates the appropriate video control sequences for a LINE in the list array based on the DEFAULT VIDEO ATTRIBUTES in the CAPTION LINE definition for the template.

Format

```
D FLDCTRL^VALM10(line, [field], [on], [off][, save])
```

Input

line	Line number in the list array to activate video attributes for.
field	(optional) If passed, only the video attributes defined for text that falls within the specified caption field will be activated. Must be the name of a caption field in the List Template.
on	(optional) If defined, then the code in this variable is used at the starting column position to turn on video attributes instead of the default. See Kernel Video Variables for a set of variables you can use here.
off	(optional) If defined, then the code in this variable is used at the ending column position to turn off video attributes instead of the default. See Kernel Video Variables for a set of variables you can use here.
save	(optional) 1 to save control sequence for later use (to be restored with RESTORE^VALM10). Otherwise, 0.

RESTORE^VALM10

Restores the video attributes that have been saved for the indicated line. This subroutine does **not** re-write the line to the screen; use WRITE^VALM10 after restoring video attributes to actually write the line.

Format

```
D RESTORE^VALM10(line)
```

Input

line	Line number to restore video attributes for.

Output

(none)

SAVE^VALM10

Saves the current video attributes for the indicated line.

Format

```
D SAVE^VALM10(line)
```

Input

line	Line number to save the current video attributes for.

Output

(none)

SELECT^VALM10

Highlight/unhighlight a line in the list. The call will set up or delete the proper video controls and then 'writes' the line to the screen.

Format

```
D SELECT^VALM10(line, mode)
```

Input

line	Line number of line to highlight/unhighlight. The line must be one that is currently displayed on the screen.
mode	1 to highlight; 0 to unhighlight and restore to original state.

Output

(none)

WRITE^VALM10

Re-write a line to the screen.

Format

```
D WRITE^VALM10(line)
```

Input

line	Number of the line in the list to re-write to the screen.

Output

(none)

Screen Control

CHGCAP^VALM

Change a label on a caption header for a field defined in CAPTION LINE COLUMNS multiple in the List Template file.

Format

```
D CHGCAP^VALM(field,label)
```

Input

field	Caption Field Name.
label	Text for caption header.

Output

(none)

CLEAR^VALM1

Use this call in programmer mode during development to clean up the screen after an error occurs. It changes the screen from screen mode to the full scrolling region and clear the screen. Also, it turns off the following:

- underline
- high intensity
- reverse video
- blinking

Format

```
D CLEAR^VALM1
```

Input

(none)

Output

(none)

FULL^VALM1

Sets the screen to the full scrolling region.

Format

```
D FULL^VALM1
```

Input

(none)

Output

(none)

INSTR^VALM1

Insert text on the display screen at the row and column specified.

Format

```
D INSTR^VALM1(string, column, row, [length][, erase])
```

Input

string	String to insert.
column	X coordinate.
row	Y coordinate.
length	(optional) Number of characters to clear.
erase	(optional) If a value (any value) is passed for this parameter, the screen cells from (row,col) to (row,col+length) are erased before the string is displayed.

Output

(none)

RE^VALM4

This call re-displays the list header and list areas for the active list application. It is often used to display the results of a change an action has caused **before** passing control back to the List Manager. (Normally, you set VALMBCK="R" and then returns control to the List Manager.)

Format

```
D RE^VALM4
```

Input

(none)

Output

(none)

CLEAN^VALM10

Kills the data and video control arrays associated with the active list. This call is commonly used to kill the array related data before re-building the array.

Format

```
D CLEAN^VALM10
```

Input

(none)

Output

(none)

KILL^VALM10

This subroutine deletes video attributes. If LINE is defined then only the attributes for that line are deleted.

Format

```
D KILL^VALM10([line])
```

Input

line	(optional) Line number to delete video attributes for. If this parameter is not passed, then all video attributes for the current list are deleted.

Output

(none)

MSG^VALM10

This call allows you to immediately post a message to the 'message window' located in the lower frame bar of the List Manager display screen.

Note: To display a custom message when List Manager re-displays the screen after an action is performed, set the variable VALMSG to the desired message text.

Format

```
D MSG^VALM10([message])
```

Input

message	(optional) Text up to 50 characters. If you don't pass this string, any custom message currently displayed is turned off, and List Manager's standard message is re-displayed.

Output

(none)

Conversion

$$FDATE^VALM1

This extrinsic function returns a date in 'mm/dd/yy' format (e.g., 12/12/92).

Format

```
S X=$$FDATE^VALM1(fmdate)
```

Input

fmdate	VA FileMan formatted date/time.

Output

return value	Date in 'mm/dd/yy' format.

$$FDTTM^VALM1

This extrinsic function returns a date/time in 'mm/dd/yy@hh:mm' format (e.g., 12/12/92@09:00).

Format

```
S X=$$FDTTM^VALM1(fmdate)
```

Input

fmdate	VA FileMan formatted date/time.

Output

return value	Date in 'mm/dd/yy@hh:mm' format.

$$FTIME^VALM1

This extrinsic function returns a date/time in the 'mmm dd, yyyy@hh:mm' format (e.g., DEC 12, 1992@09:00).

Format

```
S X=$$FTIME^VALM1(fmdate)
```

Input

fmdate	VA FileMan formatted date.

Output

return value	Date in 'mmm dd, yyyy@hh:mm' format.

$$LOWER^VALM1

This extrinsic function will convert a string from upper case to lower case. It parses the string, using a space, comma and a '/', It starts with the second character after each delimiter.

If your line of text contains many consecutive spaces, it is often faster to execute this function as you build each portion the line, instead of after the line has been completely built.

Format

```
S X=$$LOWER^VALM1(string)
```

Input

string	String to convert.

Output

return value	Converted string.

Example

```
> S X="PATIENT,ONE AND/OR PATIENT,TWO"
> S X=$$LOWER^VALM1(X)
> W X
Patient,One And/Or Patient,Two
```

$$NOW^VALM1

This extrinsic date/time function returns the value of 'NOW' in external format.

Format

```
S X=$$NOW^VALM1
```

Input

none

Output

return value	Value of 'now' in $$FTIME^VALM1 format (e.g., "Mar 06, 1993 11:15:29").

$$UPPER^VALM1

This converts a string from lower case to upper case.

Format

```
S X=$$UPPER^VALM1(string)
```

Input

string	String to convert.

Output

return value	Converted string.

Index