Introduction

The Kernel is a vendor-independent applications development environment, as well as a run-time environment providing standard vendor-independent services to applications software. It is not an operating system, but a set of utilities and associated files that are executed in an M environment. The Kernel is central to VA DHCP software strategy, in that it permits any DHCP software application to run without modification on any hardware/software platform that supports American National Standards Institute (ANSI) Standard M. All operating system-specific, M implementation-specific, or hardware-specific code is isolated to Kernel. Therefore, porting DHCP to a new environment requires modification only to a handful of Kernel routines.

As a whole, the Kernel provides a computing environment that permits controlled user access, presents menus for choosing from various computing activities, allows device selection for output, enables the tasking of background processes, and offers numerous tools for system management and application programming. Kernel also provides tools for software distribution and installation.

DHCP users see the same user interface, regardless of the underlying system architecture, because DHCP applications are built using Kernel facilities for sign-on, database access, option selection, and device selection. As a result, user interaction with the system is constant across DHCP applications.

If the reader is not already familiar with VA FileMan or MailMan, the respective user, programmer, and technical manuals for each should be obtained and reviewed. Other source documents describing overall DHCP policy are:

- **VA Programming Standards and Conventions (SAC).**
- ***MIRMO/ISC Operations Document.***
- ***National Verification Document.***

User Introduction

Kernel provides the doorway into the DHCP (Decentralized Hospital Computer Program) computer system, the menus that tie together the options and utilities to enhance those options.

For the doorway, Kernel provides the access and verify code system that you use to establish your identity to the DHCP computer system.

Once you have signed on, Kernel provides your menus. Each user on the computer system, as identified by their access code, has their own individual set of menus and options.

The person or department managing the computer system organizes each user's menus. From your menu, you can run any application the computer system managers have made available to you. Kernel's menu system is what is used to make DHCP applications such as Scheduling, Nursing, and Personnel available to users.

To produce output from DHCP applications (e.g., to printers or to the terminal screen), Kernel provides a common device interface called the Device Handler. To queue a job rather than run it directly, the device handler links to a common queuing system called Task Manager.

This manual contains information about these and other parts of Kernel. The intent of this manual is to help you learn to use Kernel and take fullest advantage of the facilities it provides. This manual also includes information for system managers and programmers; to find the information of interest to you, the general user, look for chapters and sub-chapters containing the phrase "User Interface" in their titles.

Another overview of the Kernel environment is presented in the *User's Guide to Computing*, an instructional manual for new users.

ADP Application Coordinators (ADPACs) may want to skim through the Systems Manual and concentrate on the user interface chapters and sub-chapters, particularly issues concerning every Kernel user, such as sign-on process and menu navigation.

System Manager Introduction

Kernel provides the backbone of an M computing platform, providing a mechanism to organize M programs as options, and a way to organize those options into a menu system for users. Kernel provides the following major system management components:

- **Alerts provide an integrated notification system.**
- **Device Handler provides a common device interface.**
- **Electronic Signature Codes provide a secure electronic approval system.**
- **File Access Security system manages access to VA FileMan files.**
- **Kernel Installation and Distribution System (KIDS) provides a package distribution and installation system.**
- **Menu Manager provides a common menu management system.**
- **Sign-On/Security organizes users and allows secure logons.**
- **Task Manager provides a common job queuing system.**

Kernel provides the system manager the means to manage a secure, multi-user M-based computer system. Some typical daily tasks performed by system managers using Kernel system management tools include:

- **Setting up accounts for new users and terminating accounts for expired users.**
- **Adding and subtracting options from users' menus.**
- **Controlling file access for users.**
- **Monitoring Task Manager task queues.**
- **Terminating unwanted tasks.**
- **Monitoring devices.**
- **Creating and modifying links to output devices in the DEVICE file.**
- **Installing software packages.**

You can find system manager information in the chapters and sub-chapters of this manual that contain "System Management" in their titles. In addition, the User Interface and Programmer Tools sections of this manual provide useful background information on other parts of the Kernel.

Instructions for installing Kernel are provided in the *Kernel Installation Guide*. Information on recommended system configuration and setting Kernel's site parameters, as well as lists of files, routines, options, and other components are documented in the *Kernel Technical Manual*. A detailed description of techniques that may be used to monitor and audit computing activity is presented in the *Kernel Security Tools Manual*.

Programmer Introduction

Kernel provides programmers with a number of tools. These tools include application programmer interfaces (APIs), and direct-mode utilities. These tools let you create applications that are fully integrated with Kernel and that take advantage of Kernel's features.

The major Kernel APIs are:

- **Alerts**
- **Device Handler**
- **Kernel Installation and Distribution System (KIDS)**
- **Menu Manager**
- **Sign-On**
- **Task Manager**
- **XGF Function Library**
- **XLF Function Library**

You can find programmer information in the chapters and sub-chapters of this manual that contain "Programmer Tools" in their titles. You may want to concentrate on the particular chapter of this manual that impacts a project you are working on. For example, if you are working on a project requiring tasking a job, you should familiarize yourself with the information in the Task Manager: Programmer Tools chapter.

Orientation

Organization

The discussion of each part of Kernel in this manual is organized in the following order:

1. **User Interface**
2. **System Management**
3. **Programmer Tools**

That is to say, on a given topic within Kernel, information of relevance to the general end-user is presented first. Next, information of relevance to system managers is presented. Finally, information of relevance to programmers is presented.

When a subject is large enough, such as Sign-On/Security, separate chapters are devoted to the User Interface, System Management, and Programmer Tools discussions. In other cases where the subject matter is smaller, such as the discussion of the Browser device, the three divisions of audience are contained entirely within a chapter or sub-chapter.

Beyond this general layout for each subject area of User Interface, System Management, and Programmer Tools, this manual is divided into six major parts, based on the following functional divisions within Kernel:

- **Sign-On/Security**
- **Menu Manager**
- **Device Handler**
- **Task Manager**
- **Kernel Installation and Distribution System (KIDS)**
- **Other Tools**

Conventions

This manual uses several methods to highlight different aspects of the material. Descriptive text is presented in a proportional font. "Snapshots" of computer dialogue or other on-line displays are shown in a non-proportional font and enclosed within a box. User's responses to on-line prompts are highlighted in Boldface. The pressing of the return key (used to terminate "reads" on a line), is illustrated as <RET>.

The following example is a screen capture of computer dialogue, and indicates that the user should enter two question marks followed by <RET> when prompted to select a menu option:

```
Select Primary Menu option: ?? <RET>
```

Other special keys are represented within < > angle brackets. For example, pressing the PF1 key can be represented as pressing <PF1>.

File names and field names for VA FileMan files are presented in the same case as in the data dictionary (upper and/or lower).

Headings for programmer entry point descriptions (e.g., supported for use in application packages and on the Database Integration Committee (DBIC) list) are prefaced with a leading bullet and include the routine tag (if any), the up-arrow used when calling the routine, and the routine name. The following is an example:

- **EN1^XQH**

The entry points listed in the XLF Function Library chapter are an exception to this format. Entry points in this chapter are listed in an abbreviated format, without a leading bullet. They are clearly labeled as entry points, however.

For entry points that take input variables, the input variable will be labeled optional if it is optional; if not labeled optional, 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 formatting for input parameters:

Usage `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.

Headings for descriptions of direct mode utilities are prefaced with the standard M ">" prompt to emphasize that the call is to be used only in direct mode. They also include the M command used to invoke the utility. The following is an example:

>D ^XUP

This manual refers in many places to the M programming language. Under the 1990 ANSI (American National Standards Institute) standard, MUMPS is the primary name of the MUMPS programming language, and M is considered an alternate name. Under the proposed 1995 ANSI standard, M will become the primary name of the MUMPS programming language, and MUMPS will be considered an alternate name. In anticipation of the 1995 standard, this manual uses the name M.

Package Management

Throughout this manual, advice and instruction are offered about the numerous tools the Kernel provides for overall DHCP management. Site parameters, for example, are discussed in various sections; techniques for granting user access and monitoring computing activity are presented in the Sign-On/Security section; techniques for managing menus are presented in the Menu Manager section; and so forth.

Information about managing computer security is provided in the *Kernel Security Tools Manual*. The *Kernel Installation Guide* also includes information about package management, such as recommended settings for site parameters and scheduling time frames for tasked options.

To protect the security of DHCP systems, distribution of this software for use on any other computer system by DHCP sites is prohibited. All requests for copies of the Kernel for non-DHCP use should be referred to the DHCP site's local ISC.

Otherwise, there are no special legal requirements involved in the use of the Kernel.

Part 1: Sign-On/Security

Chapter 1 Sign-On/Security: User Interface

The first step you take each time you use the computer system is called signing on. When you sign on to the DHCP computer system, you are required to enter an access code and a verify code. These codes identify you to the computer system, and, as these codes are private to you, serve to prevent unauthorized access to your account.

You are shielded from most steps in the sign-on process. In the background, the Kernel's Sign-On/Security establishes the proper environment, records and monitors the sign-on event, and takes you to Menu Manager. Menu Manager presents a list of menu options which let you interact with other parts of the Kernel and application packages. When you complete a session on the computer system, you sign out to exit.

Signing On

To sign on, you need to enter your access and verify codes. You can recognize the Kernel's front door by its access code prompt. Entering an access code and pressing <RET> brings up the verify code prompt. Entering a matching verify code and pressing <RET> completes your sign-on and takes you beyond Sign-On/Security into Menu Manager.

Your access code establishes your unique identity to Kernel. Your matching verify code corroborates your identity. "Echo" is turned off when you enter access and verify codes so that the characters are not displayed (echoed back) on the screen. Codes are encrypted after they are entered and compared with the encrypted stored code for a match. Codes must be from 6 to 20 alphanumeric characters, with a mix of alphabetic and numeric.

```
ACCESS CODE: <RET>
VERIFY CODE: <RET>
Device: _LTA8628:
Not a valid ACCESS CODE/VERIFY CODE pair.

ACCESS CODE: <RET>
VERIFY CODE: <RET>
Good evening FRIEND     You last signed on Apr 21,1992 at 07:57

There was 1 unsuccessful attempt since you last signed on:

You were last executing the 'MailMan Menu' menu option.
Do you wish to resume? YES//
```

If you have not been assigned a primary menu (see the Menu Management section), Kernel displays a message indicating that access is not allowed, and signs you out from the computer system. Similarly, if your primary menu has been marked as "out-of-order" (an option attribute), Kernel also denies you access.

```
ACCESS CODE:
VERIFY CODE:
Device: _LTA8628:
No access allowed for this user.
```

Defining a New Verify Code

While access codes are a unique identifier for your user record in Kernel's NEW PERSON file, verify codes are like secret passwords assuring that the person signing on is the one for whom the user record was established. You rarely need to be issued a new access code, but should change your verify code if you suspect that someone else has used it to gain access to the system. You can change your verify code with the option called Edit User Characteristics (which is available from the common menu User's Toolbox).

You must change your verify code at periodic intervals as specified by IRM. You will be prompted during sign-on to pick a new code. Access and verify codes must contain at least one alpha and one numeric character. Codes must be between 6 and 20 characters in length. In addition, the access and verify codes for any user and cannot be identical. These restrictions are enforced whenever access or verify codes are created or changed.

If you have forgotten your verify code, the site's security officer should delete the existing code, and then instruct you to sign on again but just press return at the verify code prompt. You will be prompted for a new verify code and also re-prompted to enter the same verify code again as confirmation. If you don't want to bother inventing a verify code, entering a question mark at the verify code prompt displays a possible although cryptic choice, like DKM493. Entering a question mark a second time displays another choice. When you log off, you're reminded to remember the new verify code for use at next sign-on.

LOGIN Menu Template

You can execute a script of options on your first sign-on of the day by having a menu template called LOGIN. For more information, see the Menu Manager: User Interface chapter.

Sign-On Shortcuts

To reach the primary menu in one step at the access code prompt, you can enter the access and verify code as one string separated by a semicolon:

```
ACCESS CODE: accesscode;verifycode <RET>
Good afternoon.     You last signed on today at 12:00
```

To "jump start" directly to a particular option, you can specify the name of an option after another semicolon:

```
ACCESS CODE: accesscode;verifycode;^Intro <RET>
Good afternoon.     You last signed on today at 12:00
^INTROductory text edit
```

To force the Kernel query of the terminal type identity, you can include a colon anywhere in the string. If you want to avoid the terminal type query, see "The Almost Extinct Terminal Type Prompt" below.

Normal Sign-off

When you complete a session on the computer system, you should sign off the system so that no one can come along and use the computer system under your identity. There are several ways you can sign off of the system.

```
SYSTEM COMMAND OPTIONS                                        [XUCOMMAND]
  Halt                                                           [XUHALT]
  Continue                                                    [XUCONTINUE]
  Restart Session                                               [XURELOG]
```

One way to sign off is to enter "halt" at any menu prompt. When you sign off using "halt", at next sign-on, after entering access and verify codes, your normal primary menu will be your first menu.

Or, to sign off, you can enter "continue". At your next sign-on, after entering access and verify codes, your last-used menu when you signed off will be your first menu for that session.

If remotely connected via modem or other network device, you can enter "restart" to sign out of Kernel without dropping the communication line.

Finally, you may sign off without using any of these shortcuts simply by pressing <RET> at each menu prompt to step back up the menu pathway and finally exit (see the Menu Management section for more information.)

Abnormal Sign-off and Error Handling

If you encounter an error while using the computer system, the Kernel will trap it, issue the message "Sorry 'bout that", and attempt to return you to your primary menu. The Kernel can recover from most error conditions and, given a suitable environment, will permit you to continue. Some error conditions, however, cause an abnormal exit such that you are immediately logged off the computer system. When this happens, you can sign on again if you still need to use the computer system.

The Almost Extinct Terminal Type Prompt

When signing on, you may be prompted to enter a terminal type. You should not see this prompt very often, however, since Kernel usually can identify your terminal type without needing to prompt you to enter one. If you are prompted, you should enter the name of the actual terminal type to use (for example, C-VT220). The entered terminal type tells Kernel how to support screen-oriented and other enhanced displays. If unusual circumstances arise and the wrong terminal type is in effect, you can redefine it by using the Edit User Characteristics option (available through the User's Toolbox, discussed below).

The Edit User Characteristics option lets you edit a setting (ASK DEVICE TYPE AT SIGN-ON) that allows you to decide whether to bypass the usual terminal type query. If you always work at the same terminal and want to save a small amount of time during the sign-on process, you can set ASK DEVICE TYPE AT SIGN-ON to DON'T ASK. Kernel then assumes that your last terminal type should be used as the default.

If you have ASK DEVICE TYPE AT SIGN-ON set to DON'T ASK, and sign on using a terminal whose terminal type is different from the one normally used, you should sign-on by including a colon (:) after your access code. This forces Kernel to query the terminal for its identity. Alternatively, once signed on, you could invoke the Edit User Characteristics option to change your terminal type to the one currently in use. Or, you could use this option to reset the ASK DEVICE TYPE AT SIGN-ON question to ASK, log off and sign back on (whereby Sign-On/Security will obtain the correct terminal type identification).

Escaping from a Jumbled Screen

One consequence of your sign-on terminal type not matching the actual one being used is that full-screen display could appear jumbled. To escape from a ScreenMan form, such as Edit User Characteristics, all you need to do is enter two up-arrows, each followed by <RET>. To escape from VA FileMan's Screen Editor, you should enter <PF1>E to exit.

Alerts

After signing on, you could be presented with an alert notice just before the menu prompt. If so, you need to pick the Common option for viewing alerts to take care of urgent, pending matters. For more information about alerts, see the Alerts chapter, in the Menu Manager section of this manual.

```
SYSTEM COMMAND OPTIONS ...                                    [XUCOMMAND]
  View Alerts "VA"                                             [XQALERT]
```

User's Toolbox

The User's Toolbox is available from any menu prompt, by entering "TBOX or USER'S TOOLBOX. It makes available, from one menu, some of the most frequently used Kernel options. The following lists the options contained in the User's Toolbox, and the chapters in this manual where each option is described (refer to the listed chapter for more information on each option):

Option	Chapter Described
Display User Characteristics	Sign-On/Security: User Interface
Edit Electronic Signature code	Electronic Signature Codes
Edit User Characteristics	Sign-On/Security: User Interface
Menu Templates ...	Menu Manager: User Interface
Spooler Menu ...	Spooling
TaskMan User	TaskMan User Interface
User Help	(accesses online help)

The Edit User Characteristics and Display User Characteristics options of the User's Toolbox are described on the following pages.

Edit User Characteristics

Edit User Characteristics, one of the options available from the User's Toolbox, allows you define some characteristics of your online environment. There are a number of values you can edit with the Edit User Characteristics option:

- **Initial, Nick Name: You can enter both your initials, and a nick name. Both of these can serve as an alternate way for users to specify your account, e.g., when sending mail to you.**

- **Phone, Office Phone, Voice Pager, Digital Pager: You can enter phone numbers in these fields.**

- **Ask Device Type at Sign-On: This field controls whether Kernel should determine what kind of terminal you are using when you sign on. If this is set to DON'T ASK, Kernel assumes you are using the same kind of terminal you used the last time you signed on. This can cause problems if you are using a different kind of terminal (screen displays may not work properly), so this should normally be set to ASK.**

- **Auto Menu: The setting of Auto Menu determines whether, in the menu system, a list of items on the current menu is displayed with the menu prompt. Beginning users should usually set Auto Menu to YES so that they can see menu items for each menu. Experienced users who are familiar with their menus may prefer to set this field to NO, which makes menu displays speedier since individual items on each menu aren't displayed.**

- **Type-Ahead: This setting controls whether characters you type faster than the system can process end up being processed or not. Normally you should set type-ahead to YES, so that keystrokes you enter are not lost due to system slowness.**

- **Text Terminator: The text terminator is a setting used by VA FileMan's Line editor. When you are using the Line editor and are importing text from an external source, you may not want a blank line to indicate the end-of-file, which could prematurely terminate the text transfer. By default, the text terminator in the VA FileMan line editor is the carriage return character (<RET>). Setting this to another character string, like ZZ (something that will not be encountered in the target text) may permit downloading without interruption. If you change the setting of the text terminator from the default of the carriage return character, you will need to remember your text terminator when using the Line editor; otherwise, you will be unable to exit the Line editor. For more information on the text terminator, please see the *VA FileMan User Manual*.**

■ Edit User Characteristics Form

```
                          EDIT USER CHARACTERISTICS
NAME: FLEW,ANTHONY                                              PAGE 1 OF 1
_____________________________________________________________________________

                      INITIAL:  AF                PHONE:    484-7520
                    NICK NAME:TONY          OFFICE PHONE:
                                              VOICE PAGER:
                                             DIGITAL PAGER:


   ASK DEVICE TYPE AT SIGN-ON:
                     AUTO MENU:  NO MENUS GENERATED
                   TYPE-AHEAD:  ALLOWED
              TEXT TERMINATOR:
            PREFERRED EDITOR:

Want to edit VERIFY CODE (Y/N):
_____________________________________________________________________________
ExitSaveRefresh

Enter a command or '^CAPTION' to jump to a field in the current window.

COMMAND:                                                          INSERT
```

- **Preferred Editor: You can choose which editor the Kernel uses when you edit word-processing fields on the system. You can choose any editor defined on your system.**

- **Verify Code: You can change your verify code by answering YES to this field. First enter your current verify code; then, enter a new verify code. You will be asked to confirm the new verify code by entering it a second time; if you confirm it, the new verify code will take effect immediately.**

Display User Characteristics

Display User Characteristics, like Edit User Characteristics, is an option in the User's Toolbox. It prints out a description of many of the characteristics of your current computing environment, including some of the characteristics that can be set through the Edit User Characteristics option.

■ Sample of Display User Characteristics Output

```
SMITH,JANE (#1163)  DEVICE: LAT DEVICE  ($I: _LTA769:)JOB: 547366821

ENVIRONMENT                              ATTRIBUTES
-----------                              -----------
   Site ........ SAN FRANCISCO              Type-ahead ....... Y
   UCI ......... KRN,KDE                    Time-out ......... 300
   Signed on ... 08:48                      Fileman code(s) .. @
   Terminal type C-VT320

KEYS HELD
---------
   XUPROG          XUMGR          XUPROGMODE     XQSMDFM
   XUAUDITING      XUARCHIVE      XUSCREENMAN    XUFILEGRAM

MENU PATH
---------
Systems Manager Menu (EVE)
  User's Toolbox (XUSERTOOLS)
    Display User Characteristics (XUUSERDISP)

'^' to escape, <CR> to view Mailman user info:




Last used MailMan: 19 Aug 94 11:01

Mail Groups:
```

Summary

Kernel's Sign-On/System Security module provides the means for signing into Kernel with a unique identity. Once you complete the sign-on process, you are sent to Kernel's menu system, where you can run any option your system manager has placed in your menus. When you finish a computer session, always be sure to sign off; this protects your account from misuse by someone else.

Chapter 2 Sign-On/Security: System Management

This chapter describes the system management tools for Kernel's Sign-On/Security module.

The Sign-On Process

If sign-ons are enabled, as shown in the Sign-On Flow Chart that follows, the sign-on process begins with a gathering of information from the KERNEL SYSTEM PARAMETERS file (#8989.3) and then from the DEVICE file (#3.5) to determine whether to allow sign-on for this session and, if so, how to create an appropriate environment. If, for example, the MAX SIGNON ALLOWED limit has been reached, the sign-on attempt will fail. If the current device is tied to a routine (as specified in the TIED ROUTINE field of the DEVICE file), that routine is executed and the session is halted. If not, the user is prompted for access and verify codes. After a successful sign-on, attributes for that user are then retrieved from the NEW PERSON file (#200). Sign-On/Security then sends the user to Menu Manager. If a primary menu is associated with the device (PRIMARY MENU OPTION field in the DEVICE file), that menu is presented. Otherwise, the user's primary menu is presented. If the user does not have a primary menu (the PRIMARY MENU OPTION field in the NEW PERSON file is null), the session is halted.

The sign-on flow chart in this section illustrates the procedural steps taken by the Kernel's Sign-On/Security system to determine whether to permit sign-ons and, if so, how to create an appropriate computing environment. Typically, after site parameters and device characteristics are checked, the user is prompted for access and verify codes, user attributes are collected, and a primary menu prompt is presented.

Introductory Text

Before gathering system parameters or prompting for access and verify codes, Sign-On/Security displays contents of the INTRO TEXT field (File #8989.3). The text may be edited with the Enter/Edit Kernel Site Parameters option, or with Introductory text edit (an option specially designed for this purpose).

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Operations Management ...                                   [XUSITEMGR]
   Introductory text edit                                    [XUSERINT]
```

Parameters Checked during Sign-On

Various parameters are checked as an initial step in the sign-on process. The KERNEL SYSTEM PARAMETERS file (#8989.3) stores the default values for most of the parameters. Values for critical fields should be defined by IRM when the Kernel is installed. The values in the KERNEL SYSTEM PARAMETERS file may be edited any time, though, with the Enter/Edit Kernel Site Parameters option.

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Operations Management ...                                   [XUSITEMGR]
   Kernel Management Menu ...                                [XUKERNEL]
      Enter/Edit Kernel Site Parameters                     [XUSITEPARM]
```

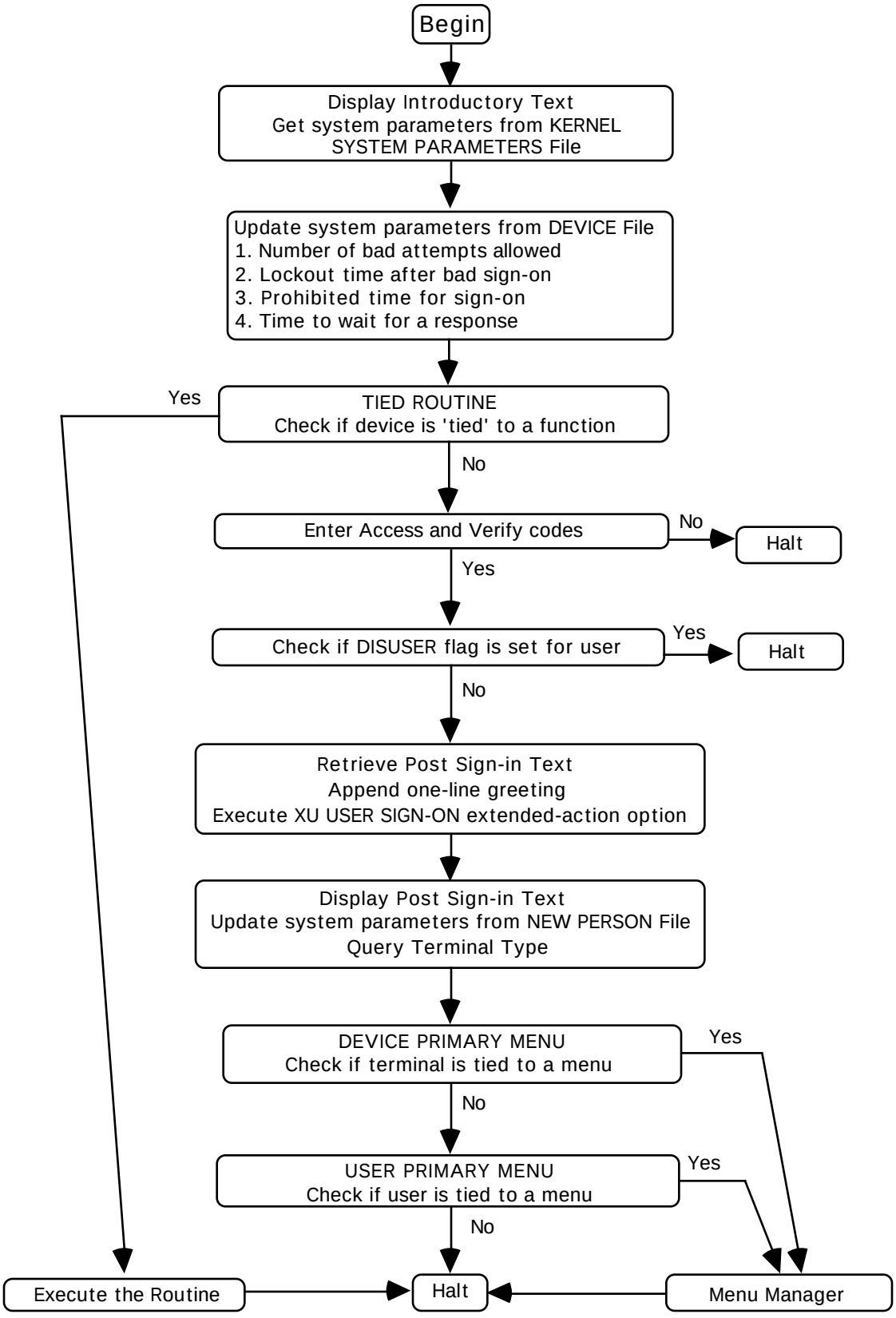
Sign-On Attempts and Device Lock-out Times: The DEFAULT # OF ATTEMPTS field (File #8989.3) holds the default limit of the number of times a user can try to enter a valid access/verify code pair. When the limit is reached, Sign-On/Security is unresponsive for the duration specified by the Default Lock-out Time. The values for number of attempts and lock-out time are overridden by any values for the current device specified by comparable fields in the DEVICE file (#3.5). Device values are ignored, however, if the BYPASS DEVICE LOCK-OUT site parameter (File #8989.3) is set to YES. In particular, the fields that are bypassed are OUT-OF-SERVICE DATE, SECURITY, and PROHIBITED TIMES FOR SIGN-ON. Device values are put back into effect for the current device if the DEVICE file's PERFORM DEVICE CHECKING field is set to YES.

Max Signon Allowed: One Kernel site parameter used in the initial sign-on screening is MAX SIGNON ALLOWED. It is a field within the VOLUME SET multiple (File #8989.3). Its value sets an upper limit for number of M processes (interactive, background, and system) that can run concurrently on the specified volume set or CPU. The TASKMAN JOB LIMIT, a field in the TASKMAN SITE PARAMETERS file (#14.7), should be set to a number slightly lower than MAX SIGNON ALLOWED to leave room for a few interactive logons when TaskMan is busiest.

For OpenVMS sites: the OpenVMS interactive logins parameter (set by the DCL command SET LOGINS/INTERACTIVE) should be set to a number less than the Kernel Max Signon to conserve system resources. If the OpenVMS limit is set too high in relation to the Kernel limit, users will try to access the Kernel only to be rejected when reaching Sign-On/Security. That means that they would waste system resources by creating a new OpenVMS process and activating a DSM image, all to no avail.

Prohibited Times for Sign-On: Time periods can be specified, during which interval sign-ons can be barred by device or by user. This is controlled by the PROHIBITED TIMES FOR SIGN-ON field in the DEVICE file (#3.5) and a comparable field in the NEW PERSON file (#200).

■ Sign-On Flow Chart



Multiple Sign-On Restriction: The DEFAULT MULTIPLE SIGN-ON field (File #8989.3) controls whether users may create two or more simultaneous sessions by signing on to more than one device. The setting is overridden by comparable fields in the DEVICE (#3.5) and NEW PERSON (#200) files, respectively. The value is checked at sign-on to prevent unauthorized multiple sessions.

If multiple sign-ons are prohibited, problems can occur if users experience an abnormal exit such that the sign-on record cannot be cleared. To clear an individual user, the Release User option can be used (described later in this chapter). To make sure all users are clear when the system is brought up after a crash, IRM can use the option Clear all users at startup (also described later in this chapter).

Interactive User's Priority: This parameter (in File #8989.3) should usually be left null. A setting here affects the job priority of interactive users and could result in poor response time.

Ask Device Type at Sign-On: The ASK DEVICE TYPE AT SIGN-ON parameter controls whether the user's current device at sign-on is queried for its display attributes (DA). The correct terminal type can thus be identified without prompting the user.

It is recommended that ASK DEVICE TYPE AT SIGN-ON be set to ASK so that Sign-On/Security performs the DA query and allows the Device Handler to set up the correct terminal type attributes. This has become more important with the advent of screen control. VA FileMan's Screen Editor and Screen Manager, for example, will not function properly if the terminal type recorded by the Kernel fails to match the actual terminal type being used.

As with other parameters, the site default (ASK DEVICE TYPE AT SIGN-ON field in File #8989.3) is overridden by a DON'T ASK setting for the device (like-named field in File #3.5), which would similarly be overridden by a DON'T ASK setting for the user (like-named field in File #200). A null value functions as ASK. The user override can be set by any user via the Edit User Characteristics option.

If the parameter is set to DON'T ASK, Sign-On/Security does not perform the DA query and assumes the user's last terminal type is still appropriate. Although the difference in resource consumption is negligible, a split second's savings in time may be appreciated by the user. Bypassing the DA query may thus be acceptable if the same terminal type is always being used. But if the user should sign onto another terminal type, problems may occur with the presentation of screen-oriented displays unless the user knows how to change the terminal type to match the actual current one.

If the device is non-ANSI-standard, Sign-On/Security may not find a DA but will continue to determine the terminal's identity by querying its answerback message. All known non-ANSI devices, such as the Qume 102 terminal,

should thus have their answerback messages programmed. This is accomplished by using the terminal type setup mechanism and entering C-QUME as the Qume 102's answerback message. The name must match an entry in the Kernel's TERMINAL TYPE file (#3.2) to take effect. If the answerback message contains additional characters, such as a serial number, the message will not match an entry in the TERMINAL TYPE file and will be useless for sign-on purposes.

If the terminal's DA return code doesn't match an entry in the DA RETURN CODES File, or if the terminal is non-ANSI and cannot be programmed with an appropriate answerback message, Sign-On/Security prompts the user to identify the terminal type if the user's ASK DEVICE TYPE AT SIGN-ON setting is set to ASK. This is the only case in which the terminal type prompt is asked during sign-on. The last terminal type used will be presented as the default (it is stored in the NEW PERSON file). If ASK DEVICE TYPE AT SIGN-ON is set to DON'T ASK, Sign-On/Security assumes that the last terminal type is appropriate and does not prompt the user for validation.

Display Attributes (DA) Return Codes: The DA RETURN CODES file (#3.22) is used to equate DA return codes to entries in the TERMINAL TYPE file. You can use the DA Return Code Edit option to automate the population of the DA RETURN CODES file. For more information, please see the "Display Attributes (DA) Return Codes " section of the Device Handler: System Management chapter.

Devices Selectable at Sign-On: IRM can also control which devices may be selected at sign-on with a field in the TERMINAL TYPE file. The SELECTABLE AT SIGN-ON flag should be set to YES for all devices commonly used for sign on. Ordinarily, it should not be set for printers (e.g., P- terminal types such as P-DEC or P-OTHER). To allow the loading of ScreenMan forms and proper functioning of other screen-oriented displays, the flag should also not be set for PK- types, that is, printers with keyboards. This is not an actual restriction, however, but a recommendation.

Lifetime of Verify Code: To insure that users change their verify codes at periodic intervals, IRM should set the LIFETIME OF VERIFY CODE parameter (File #8989.3) to a certain number of days (90 is the recommended number of days). Users must then change their verify codes at least as often as the specified lifetime. Sign-On/Security checks whether the verify code needs to be changed and, if so, prompts the user at sign-on to enter a new code.

Auto-generate Access Codes: When assigning access codes, the security officer or IRM staff may invent an alphanumeric string or may ask the Kernel to generate one. If the AUTO-GENERATE ACCESS CODES site parameter (File #8989.3) is set to YES, only generated, cryptic codes can be assigned. It is not necessary to pick the first one presented; others can be generated for selection.

Default Institution and Agency: The institution running the Kernel software is defined during the Kernel installation when prompted for the DEFAULT INSTITUTION (File #8989.3). This field is a pointer to the INSTITUTION file (#4). One or more institutional affiliations may also be associated with a user, such as a VA Outpatient Clinic and an Army Medical Center. This data is stored in the Division multiple in the NEW PERSON file (#200). If a user is associated with more than one Institution, the user will be prompted at sign-on to select a division. In this way, the local variable DUZ(2) can be set to the appropriate value. If the user's Division multiple is blank, the DEFAULT INSTITUTION field (File #8989.3) is used to define DUZ(2). Since the INSTITUTION file contains a pointer to the AGENCY file, the signed-on user's agency affiliation may also be determined.

The KERNEL SYSTEM PARAMETERS file (#8989.3) also contains a field named AGENCY. This field is not a pointer but is instead a set-of-codes such as N for Navy. This field is presented for editing during Kernel installation. Its value is used at sign-on to set the DUZ("AG") local variable. The agency associated with the overall Kernel system may thus be determined.

Auto Menu Display: The AUTO MENU flag, stored in the local variable DUZ("AUTO"), is used by Menu Manager to control whether all items on a menu are presented automatically after each cycle through the menu system. If the items are not displayed, the user can always invoke the display by entering a question mark. New users often like to see all the menu choices. Experienced users probably do not need to see the choices and the display can be suppressed to save system resources. The user setting for AUTO MENU (in File #200) will override any comparable device setting (File #3.5), which will, in turn, override the site parameter default (File #8989.3). Users may edit the setting with the Edit User Characteristics option.

Type-Ahead: If type-ahead is disabled, any keystrokes that the user enters while computer system processes previously issued instructions will not register. If type-ahead is enabled, keystrokes entered in advance of processing will be stored in the type-ahead buffer and will be interpreted when the earlier process is finished. New users may experience unwanted results if type-ahead is enabled and they had not anticipated the effect. Experienced users may prefer type-ahead for efficiency. The user setting overrides the device setting, which, in turn, overrides the site parameter setting. Users may edit the setting with the Edit User Characteristics option.

Timed-Read: The value for the timed-read parameter is stored in the local variable DTIME and is used to calculate how long the Kernel should wait before terminating a read. If, for example, a user does not respond to a menu prompt in the number of seconds defined by the timed-read, the Kernel will take steps towards sign-off and, without subsequent user response, will halt the user session. The user setting overrides the device setting, which, as usual, will override the site default.

Post Sign-In Text: The post sign-in text is similar to introductory text, except that Kernel displays it only after a successful sign-on. Like the introductory text, you can edit the message text using the Enter/Edit Kernel Site Parameters option; alternately, you can use the Post sign-in Text Edit option, which is specially designed for this purpose:

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Operations Management ...                                  [XUSITEMGR]
   Post sign-in Text Edit                                  [XUSERPOST]
```

Applications can append information to the post-sign in text (on a per-user, per sign-on basis only) by attaching to the new XU USER SIGN-ON option (described in the Sign-On/Security:Programmer Tools chapter).

XU USER SIGN-ON Option

The XU USER SIGN-ON option is a new option in Kernel V. 8.0. Packages can attach action-type options to this extended-action-type option, so that package-specific actions can be performed at sign-on. For more information, see the Sign-On/Security: Programmer Tools chapter.

Clear All Users at Startup

```
Parent of Queuable Options ...                    [ZTMQUEUABLE OPTIONS]
  Clear all users at startup                          [XUSER-CLEAR-ALL]
```

If multiple sign-ons are prohibited, users may be prevented from signing on to the system when it is brought up after a crash (which can cause numerous abnormal exits). To prevent this problem from occurring, IRM can use an option called Clear all users at startup. Kernel recommends this option be scheduled to run at system startup. Although this option can be invoked interactively without ill effects, it was designed as a background process and, so, is placed along with other tasked options on the Parent of Queuable Options menu. To release a single user, see the Release User option, described in the User Management Options on the Operations Menu section of this chapter.

Enabling and Disabling Logons

IRM has full control over whether logons are enabled. Access to a particular volume set may be disabled by setting the INHIBIT LOGONS? flag in the VOLUME SET file (#14.5). Setting the flag to YES sets the ^%ZIS("14.5","LOGON","*volume set*") node, whose presence disallows user logons. That is, logons through Sign-On/Security, invoking the ^ZU routine, will fail (terminals for user access are usually linked to ZU within the operating system setup. Some special terminals, like the console, are untied.) The ^%ZIS("14.5","LOGON","*volume set*") node is also checked after each cycle through the menu system; signed-on users will be logged off as soon as they return to a menu prompt.

Adding New Users

Creating a new user account involves adding a record to the NEW PERSON file, assigning an access code, and assigning a primary menu. You need the XUMGR key to assign primary menu options. Even the programmer's at-sign is insufficient, as checked by the PRIMARY MENU OPTION field's input transform.

```
SYSTEMS MANAGER MENU ...                                        [EVE]
User Management ...                                           [XUSER]
   Add a New User to the System                             [XUSERNEW]
   Grant Access by Profile <locked: XUMGR>                  [XUSERBLK]
   User Inquiry                                             [XUSERINQ]
```

Add a New User to the System [XUSERNEW]

Add a New User to the System is an option that you can use to set up user accounts one-by-one. The option presents a standard scrolling-mode editing sequence for user attributes.

When using this option, entry of a social security number in the SSN field is usually required. While SSN is not required in File #200's data dictionary, it is a required field when using this option. If the option is used by someone who holds the XUSPF200 key, however, entry of an SSN is *not* required.

You can also print security forms for the new user with this option.

When signing on for the first time, the new user should simply press <RET> at the verify code prompt, which then lets them enter their own secret verify code.

New Person Required Fields

When adding new users, a default set of fields is required, at a minimum. This set is defined by a field in the KERNEL SYSTEM PARAMETERS file (#8989.3) called NEW PERSON IDENTIFIERS. If it is null, the default set of required fields for NEW PERSON file entries is Initials (field 1), Sex (field 4), and SSN (field 9). If, given local site policy, a different set should be used, IRM may use this field to specify other identifiers.

Note that SSN is not required if the person entering accounts holds the XUSPF200 key.

Grant Access by Profile [XUSERBLK]

The Grant Access by Profile option includes features unavailable in the Add a New User option. With Grant Access by Profile, you can grant access to one or more people based on a typical user profile. All characteristics of the typical user, including menus, keys, and service/section, are copied to the new user or replace the characteristics of an existing user. For new users, access security forms are generated as part of the process. These forms can be delivered to the service/section coordinator by inter-office mail and can be distributed to the new users.

Grant Access by Profile is locked with the XUMGR key and is strictly limited for use by IRM. It must be restricted because any user profile, even that of a programmer, can be copied to another user. As with the Add a New User option, the SSN is required when adding new records except by holders of the XUSPF200 key or if another default set of New Person Identifiers has been defined.

Access is assigned according to an existing user profile. Characteristics of the new user are cloned from the existing one. Rather than copying the characteristics from an actual user, creating several dummy users with profiles of typical positions may be worthwhile. A user such as PHARMACY,TECH or RESIDENT,SURGERY could be created with the appropriate user attributes, including menu options, keys, and service/section codes.

Several steps are involved in copying access to new or existing users. First you enter the name of the user account to clone from. Then, optionally, you can specify a termination date. Next, you enter the names of the new users to create. The system will pause for each new user as it verifies identifiers, checks for duplicates, and updates the NEW PERSON file. You must enter a device to print the computer account notification letters on. You can either run the access assignment immediately or queue it for a later time.

Security Forms

```
SYSTEMS MANAGER MENU ...                                        [EVE]
User Management ...                                           [XUSER]
   Reprint Access agreement letter                      [XUSERREPRINT]
```

Two security forms are printed for each new user. The first, the Computer Account Notification, includes the user's auto-generated access code and the name of the service/section coordinator who can answer questions. The second is the Computer Access Policy, a contract to which users must adhere. It states the terms of granting access to sensitive information; the user must accept these terms as a condition of being given system access.

The forms should be edited for local use. They are stored in the help frame XUSER COMPUTER ACCOUNT. Place-holder text should be replaced with the actual name and address of the facility.

VA FileMan word processing "windows" are used to retrieve the user's name, service/section, and service/section coordinator's name. To be effective, the SERVICE/SECTION field in the NEW PERSON file must be filled in for the new user. The COORDINATOR (IRM) field, a new field in the SERVICE/SECTION file, must also be filled in and updated when necessary. Word processing "windows" are also used for formatting, like |TOP|, to separate the two forms. When using the File Access Security system, READ access to the SERVICE/SECTION file is needed to retrieve the Coordinator's name within the window command. See the *VA FileMan User Manual* for more information on using word processing "windows," the File Access Security system, and navigation.

The Reprint Access Agreement Letter option allows you to reprint the computer access agreement letter in case there was a problem printing the first form (e.g., the first form is jammed in the printer). It does not reprint the access code on the letter, however.

Editing Existing Users

```
SYSTEMS MANAGER MENU ...                                        [EVE]
User Management ...                                            [XUSER]
   Edit an Existing User                                    [XUSEREDIT]
```

The attributes of an existing user may be edited with the Edit an Existing User option. This option invokes a screen-oriented display, using ScreenMan. It is impossible to exit the form and save changes unless all required fields, like the SERVICE/SECTION field in File #200, are filled in.

Listed below are descriptions of each of the user attributes you can edit with the Edit an Existing User option.

Name (required): The user's name should be entered in capital letters. The syntax should be "LAST,FIRST MI." with only a comma (no spaces) between the last and first name. A middle initial may follow, separated with a space and followed with a period. It is not appropriate to add credentials, such as M.D., since there are other ways to specify such additional information (by the Title and the Signature Block Printed Name). Furthermore, the parsing algorithms commonly used in application packages only recognize two pieces, before and after the comma, rearranging them and using upper/lower case to generate "First MI. Last".

Initial: The user's initials may be entered, usually two or three capital letters with no spaces. The NEW PERSON file (#200) contains a look-up type cross-reference by Initial (C), so if the INITIAL field is filled in, the user can be found in File #200 by entering the initials. For example, just the initials can be used at the "Select NEW PERSON Name:" prompt, or when addressing mail messages, or for other look-up purposes. Users may edit their initials at any time since this field is included in the common option Edit User Characteristics.

Title: The TITLE field points to the TITLE file (#3.1), a file exported with the Kernel but without data (records). The User Management options to add or edit a user's record allow LAYGO into the TITLE file, so titles can be added via File #200. Although not required, it may be wise to assign appropriate titles to users so this field can be referenced by other packages. MailMan, for example, displays titles in message headers if the user who is reading mail has so indicated with a flag in MailMan's Edit User Options called Show Titles.

Nick Name: Like INITIAL, NICK NAME has a look-up type cross-reference (D) in File #200 so that look-ups will succeed simply by using the nick name. This field is also included in Edit User Characteristics.

■ Screen 1 of Edit an Existing User

```
                          EDIT AN EXISTING USER
NAME:FLEW,ANTONY                                              PAGE 1 OF 3
_____________________________________________________________________________

     NAME:FLEW,ANTHONY                              INITIAL:AF
   TITLE:HISTORIAN                                NICK NAME:TONY
     SSN: 999999999                               MAIL CODE:

          PRIMARY MENU OPTION:XMUSER
 Select SECONDARY MENU OPTIONS:ZZ MAIN

Want to edit ACCESS CODE (Y/N):      FILE MANAGER ACCESS CODE:
Want to edit VERIFY CODE (Y/N):

             PREFERRED EDITOR:
              Select DIVISION:GREAT BRITAIN
               SERVICE/SECTION:INFORMATION RESOURCES MGMT
_____________________________________________________________________________
Exit     Save    Next Page    Refresh

Enter a command or '^' followed by a caption to jump to a specific field.

COMMAND: _                          Press <PF1>H for help      INSERT
```

SSN: The SSN field is not a required field in the data dictionary for File #200. SSN is required when using the User Management options to add a new user unless the XUSPF200 key is held by the person using the option. It is highly recommended that each new user have the SSN field filled in to minimize the problem of subsequent duplicate entries. Since many existing users do not have an SSN entered, however, the Edit an Existing User option does not require that one be entered.

Mail Code: The user's mail code may be entered for purposes of interoffice routing of manually delivered mail.

Primary Menu (required for functional access): Users must be assigned a primary menu option in order to reach Menu Manager after successfully entering access and verify codes. The Primary Menu should provide a route to all the computing functions the user can be expected to need. The XUMGR key must be held by the person assigning the menu (unless delegated options are available for use with the Secure Menu Delegation system). Building and rearranging menus is discussed in the Menu Management chapter.

Secondary Menu Options: The secondary menu may be used to assign particular options to individual users to customize their menu choices. While a user may have a standard primary menu to carry out the usual functions of a department or service, additional special functions just for this user can be assigned as secondary options. This is a multiple field, unlike the Primary Menu Option, so additional items can easily be added.

Access/Verify Code Edit: These fields may be used to edit a user's access or verify code as needed. If a user has forgotten the verify code, or needs a new one, IRM should delete the existing code so that when the user logs on and presses return at the verify code prompt, a new (secret) verify code can be entered. To accomplish this, "Y" should be entered at the "Want to edit VERIFY CODE (Y/N) :" prompt. An at-sign (@) should then be entered to delete the existing code. The change will be filed immediately, unlike other changes that will be processed as part of the overall transaction when leaving the ScreenMan form.

Users may edit their verify code at any time via the Edit User Characteristics option on the Common menu. If this option uses a local template, the ability to edit the verify code should probably remain, as a security measure. IRM may even choose to add the ability to edit the access code as well.

File Manager Access Code: The File Manager access code is stored in the local variable DUZ(0). If DUZ(0)=@, the user is a programmer with the highest level of access authority. Other symbols may be assigned, depending on the user's needs. The pound sign (#) is often used for IRM support staff. Application packages will indicate which symbols are needed for package-specific access.

If the File Access Security conversion has been run, the VA FileMan access code is not used to control file-level access as it was before the conversion. The File Access Security system (formerly known as Part 3 of the Kernel installation) permits the association of a user with a file whereby explicit access can be granted. While the conversion process is somewhat involved, the benefits resulting from implementing the File Access Security system are worthwhile.

The VA FileMan access code continues to serve several functions, though, even after running the file access conversion. If a user has been granted full file access privileges for a particular file, a further restriction may be placed at the file or field level to prohibit modification of the definition or entry of data. Files have top-level restrictions of READ, WRITE, or DELETE as do fields and templates. If the file, field, or template is protected with the programmer's at-sign (@), the user must also have the at-sign in the VA FileMan access code field of the NEW PERSON file.

The Device Handler also checks the VA FileMan access code of the user if the security field in the DEVICE file has been defined with a character string. The user would not be able to select the device unless at least one of the characters in the user's code matched at least one character in the device code.

The most important of the File Manager access code characters is the programmer's at-sign, the @ symbol. It has special meaning and overrides other file access restrictions or other VA FileMan access code characters. It is

not recommended that the at-sign be allocated unless absolutely needed. Allocation is, in part, restricted by the fact that only those few users who have programmer access to the system can give other users the at-sign. Note that a set statement from programmer mode can be used to temporarily assign DUZ(0)="@" without storing the code in File #200 (which would give permanent programmer access).

Use of the at-sign is less common now than in the past since alternative security measures have been developed. It is still required for several critically sensitive checks, however, such as entering M code into VA FileMan files such as the OPTION and FUNCTION files.

For more information on the File Access Security System, please see the File Access Security chapter.

Preferred Editor: If a user's PREFERRED EDITOR field is null, Kernel uses VA FileMan's Line Editor to edit word-processing fields. If the Preferred Editor is set to another entry in the ALTERNATE EDITOR file (#1.2), like VA FileMan's Screen Editor, Kernel uses that editor when the user edits word-processing fields. As described in VA FileMan's documentation, users can switch from the Line Editor to another editor by using the Utility sub-option on the Edit option menu.

```
1>_ <RET>(enter one space character)
2><RET> (press return)
EDIT Option: U <RET> tilities in Word-Processing
UTILITY Option: E <RET> ditor
Select ALTERNATE EDITOR: S <RET> CREEN EDITOR - VA FILEMAN
```

If the Preferred Editor is the Screen Editor, it is also possible to switch to another editor, like the Line Editor, to take advantage of line editor features such as File Transfer from Foreign CPU. Note that other editors, such as WordMan or VA LetterMan, do not support switching to the Line Editor, which may be a limitation in some circumstances.

This field is also included in Edit User Characteristics and MailMan's Edit User Options so that all users may define a Preferred Editor if they so choose.

Division: The Division multiple has a corresponding site parameter, the Default Institution, that sets users' DUZ(2) if this field is not filled in. A user setting, however, takes precedence over the site parameter. This is a multiple field and if the user is associated with more than one institution, the user is prompted at sign-on to pick the one corresponding to the computing activities to be carried out in that session.

Service/Section (required): This field points to the SERVICE/SECTION file (#49) distributed with the Kernel virgin installation. No data is included. It is a required field since applications have begun to use it in various utilities. The Kernel's CPU/Service/User/Device Stats [XUSTAT] option, for

example, can summarize sign-on information for all users in the same Service/Section. The Grant Access by Profile option also makes use of this field to specify the Service/Section Coordinator to whom the access forms of the new users should be delivered.

Timed Read: As discussed with other site parameters earlier in this chapter, timed read defines the length of time the Kernel should wait for a user response to a "read". A setting for the user attribute overrides the site default. It is used to define the local variable DTIME.

Multiple Sign-On: As discussed with other site parameters, multiple sign-on controls whether the user will be permitted to have two or more concurrent sign-on sessions. The user setting takes precedence.

Auto Menu: As discussed with other site parameters, auto-menu controls whether the entire list of menu options is automatically presented or whether the user needs to enter a question mark to invoke the display. The user setting takes precedence.

Ask Device Type at Sign-On: As discussed with other site parameters, the ASK DEVICE TYPE AT SIGN-ON field controls whether the device being used at sign-on will be queried for its terminal type. The user setting takes precedence.

Type-Ahead: This field controls whether the user can enter text faster than the computer can read it. If set to YES, the computer buffers input from the user. If set to NO, keystrokes from the user are lost if they are typed faster than the computer can process them.

Allowed to Use Spooler: This field controls whether a user may pick the spool device at the device prompt to send output to the spooler.

Programmer Access Code (PAC): For users who have been granted the Programmer Mode option along with the XUPROG key and XUPROGMODE key, a programmer access code may be assigned as additional security. If a PAC is defined, Kernel prompts for the PAC just before allowing a user to enter programmer mode. If this field is null, a PAC is not asked.

Can Make into a Mail Message: This field controls whether a spooled document can be transformed into a regular mail message for use within MailMan.

DISUSER: If set to YES, disables access to the system for this user (without terminating the user's account).

File Range: Users who have VA FileMan privileges to create files may be given a numeric range of numbers to use as file numbers. Assigning number ranges acts as a safeguard to keep users from picking a number within a range that is nationally reserved for an application package. It can also serve

local database administration needs of segmenting local development by number ranges.

Termination Date: As mentioned in the section about terminating users, the termination date indicates when a user's access privileges should be revoked.

Always Show Secondaries: If set to YES, contents of a user's secondary menu are shown when the user enters one question mark at a menu prompt. Otherwise, the user must enter two question marks to see their secondary menu.

Prohibited Times for Sign-On: As discussed with other sign-on parameters, prohibited times for sign-on can be used to regulate when the user may sign-on. The user setting takes precedence over any corresponding device setting.

Phone, Office Phone, Commercial Phone, Fax Number: Set up phone numbers for the user in these fields.

Voice Pager, Digital Pager: Set up pager numbers for the user in these fields.

Language: Overrides the setting of the DEFAULT LANGUAGE field in the KERNEL SYSTEM PARAMETERS file. Both of these are used to set the DUZ("LANG") flag for each user. VA FileMan uses this setting to enable the display of language-specific dates and times, numeric formats, and dialogs.

Additional Attributes Editable by Users

Some but not all of the user attribute fields can be edited by users using the Edit User Characteristics option. See the Sign-On/Security: User Interface chapter for a description of the fields users can edit (using the default Edit User Characteristics form and template). The only field the user can edit that is not part of the system manager's Edit an Existing User form is the TEXT TERMINATOR field.

Edit User Characteristics Form and Template

The Kernel exports a ScreenMan form and a template to be used in the Edit User Characteristics option. Both are called XUEDIT CHARACTERISTICS. The input template by the same name is invoked if the ScreenMan form cannot be loaded on the current terminal type.

IRM can substitute a locally-developed template by entering its name in the User Characteristics Template field in the KERNEL PARAMETERS file (#8989.2). IRM may also design a customized form with the same name as the local input template that will be displayed instead, terminal type setup permitting. In other words, to invoke a locally modified display, an input template must exist. If a ScreenMan form by the same name also exists, an attempt will be made to display the form before defaulting to the input template.

See the *Kernel Installation Guide* for more information on creating a local Edit User Characteristics form and template.

Deactivating and Reactivating Users

Kernel provides options to deactivate and reactivate users. When users no longer need access privileges, IRM can partially or entirely close access to their account.

```
SYSTEMS MANAGER MENU ...                                          [EVE]
User Management ...                                              [XUSER]
   Deactivate a User                                          [XUSERDEACT]
   Purge Inactive Users' Attributes                        [XUSERPURGEATT]
   Reactivate a User                                          [XUSERREACT]
```

Deactivating a User

The Deactivate a User option lets you temporarily or permanently disable access for users. You can schedule termination of a user for a future date. The Deactivate a User option loads a ScreenMan form with four fields, described below.

1. Disable User

Setting the Disable User field to YES prevents a user from signing on, but leaves all of their menus, keys, and other attributes (essentially the user's entire account) still enabled. It sets a field in the user's NEW PERSON file entry, called DISUSER.

You might want to use the Disable User feature to prevent access to your system by an external support person, except during pre-approved times (where you may want to monitor their actions). Setting DISUSER to YES prevents them from logging on to the system until you clear the field.

If you set Disable User to YES, do not set any other fields in the Deactivate a User form (they only apply to terminating a user). Then, to re-enable access, use the Reactivate a User option (described below).

2. Termination Date

Terminating a user is the way to formally deactivate a user (as opposed to temporarily disabling their account). Setting a termination date effectively terminates that user's account, effective from the termination date forward.

The Deactivate a User option automatically performs the following steps when you deactivate a user:

- **Revokes the user's status as an authorized sender of any mail groups.**
- **Revokes the user's status as a surrogate.**
- **Revokes the user's status as a Secure Menu Delegation delegate.**
- **Deletes the user's access code, verify code, electronic signature code, FileMan access code, and programmer access code.**
- **Deletes the user's menu templates.**
- **Deletes the user's delegated options.**
- **Purges the ^DISV global on that CPU for that user.**

You can also decide whether all mail messages and all keys for the account will be deleted on the termination date with the final two fields in the Deactivate a User option (DELETE ALL MAIL ACCESS and DELETE KEYS AT TERMINATION). If the user is expected to return to the facility and will need to have the user account reopened, keys and mail could be retained.

Note: Please see the XU USER TERMINATE section in the Sign-On/Security: Programmer Tools chapter for more information on cleaning up user access and privileges at termination.

3. Delete All Mail Access

Setting the DELETE ALL MAIL ACCESS field causes all mail messages for the user to be deleted when their account is terminated on the termination date.

4. Delete Keys at Termination

Setting the DELETE KEYS AT TERMINATION field causes all keys for the user to be deleted at termination (except keys marked "keep at terminate").

As discussed in the Security Keys chapter, the application developer may export a key with the KEEP AT TERMINATE field set to YES in such a situation. The Provider key, included with the Kernel, has the flag set to YES for this purpose. Although a user may have been deactivated, it could be important to continue a processing activity that the user had authorized, based on privileges associated with a security key. A medical order could continue to hold an approved status, for example, even though the authorizing provider had been deactivated.

Automatic Deactivation of Users

Kernel's XUAUTODEACTIVATE option finds all users in the NEW PERSON file with a termination date in the past, but who still have an access code. Any such users are users that had been scheduled for termination but who were not terminated (usually because the task that should have terminated them did not run). XUAUTODEACTIVATE terminates any users it finds in this category. It acts as a safety net to ensure that all users who were scheduled for termination are, in fact, terminated. It should be regularly scheduled (see the *Kernel Installation Guide* for recommended frequency of scheduling). Because this option is not intended for interactive use it is placed on the ZTMQUEUABLE OPTIONS menu.

Purging Mail and Keys for Inactive Users

You can use the Purge Inactive Users' Attributes utility to clean up files. It removes all mailboxes, messages, mail groups, and keys for users who have been terminated. If any of these users still retain access codes, they are deleted.

This is particularly significant with mail. A mail message cannot be completely removed from a system until all recipients have deleted it from their mail baskets. If a user is no longer active, then it becomes unlikely that the message will ever get purged.

There are two modes of running this option. You can VERIFY the process for each user that the computer selects as eligible. If you choose not to verify the process for each user, then for every user with a non-future termination date, their set of keys, mail groups, messages, and mail baskets will be deleted.

Reactivating Users

You can use the Reactivate a User option to re-enable access for a user who has either been terminated, or whose access has been temporarily disabled. To re-enable access for someone whose account is merely disabled (with the DISUSER field set to YES), use this option to simply clear the DISUSER field. Otherwise, using this option, you can fill in all the fields needed for an active account (FILE MANAGER ACCESS CODE, PRIMARY MENU OPTION, etc.).

When you reactivate a user, you are asked whether to deny access to old mail messages. If the reactivated user account is a less privileged account than previously, it may be appropriate to deny the user access to messages that were received in the user's prior capacity. Even if that user's Mail Box was deleted at termination, once the user is reactivated, an old message would be delivered if responded to by another recipient.

User Management Options on the Operations Menu

Kernel provides a set of options for IRM to monitor and support users logged on to the system. These options are on the User Management Menu, under the Operations menu.

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Operations Management ...                                    [XUSITEMGR]
   User Management Menu ...                                  [XUOPTUSER]
      Find a user                                          [XU FINDUSER]
      List users                                             [XUSERLIST]
      Print Sign-on Log                                      [XUSC LIST]
      Release user                                            [XUSERREL]
      User Inquiry                                            [XUSERINQ]
      User Status Report                                   [XUUSERSTATUS]
```

Find a User: You can use this option to find a user who is currently signed on to the system in this UCI group. If you are on the same CPU as the user, this option will also show the menu path of the user. The option finds users based on the "CUR" cross-reference of the SIGN-ON LOG file.

List Users: This option lists all users known to the system.

Print Sign-on Log: This option prints out the SIGN-ON LOG.

Release User: If multiple sign-ons are prohibited, problems can occur if users experience an abnormal exit such that the sign-on record cannot be cleared. IRM can use the Release user option to remedy the problem for individual users. To clear all users on startup, schedule the Clear All Users at Startup option.

User Inquiry: This option displays various attributes of a specified user. If the user is currently signed on, it displays the job and device numbers, the sign-on time, and what option is being executed. Otherwise, it displays the last sign-on time. It also displays which keys are held by the user.

User Status Report: This option produces a report of the users currently signed on to this CPU and UCI. It shows the option each user is running and when they signed on, as well as their device and job numbers.

Sign-On Audits

Sign-on events are recorded in the SIGN-ON LOG file (#3.081). Statistics such as the time of access and the user's identity are stored for audit purposes. If the user exits normally (is not "bumped" off the system), the sign-on record will include the time of exit. If the user exits abnormally with an error or enters programmer mode, the sign-on record cannot include a time of exit.

Information about sign-on activity may be reviewed with options on the Operations and System Security menus.

The Sign-On Log is purged with an option (XUSCZONK) that should be tasked to run on a regular schedule, such as every night. This option cannot be reached from Menu Manager; like other options that should only be queued, it is on the Parent of Queuable Options menu.

Sign-On Statistics

Statistics about active sessions may be obtained with the CPU/Service/User/Device Stats option. This option permits sorting by CPU, by the user's Service/Section such as MAS, by individual users, or by particular devices.

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Operations Management ...                                  [XUSITEMGR]
   CPU/Service/User/Device Stats                              [XUSTAT]
```

Failed Access Attempts Audit

When a user enters invalid access/verify code pairs, the number of attempts is recorded and the device appears to lock after the site parameter limit of failed access attempts is reached. After this point, Sign-On/Security continues to record what the user types (but only to create a record in the FAILED ACCESS ATTEMPTS LOG file). If a valid access code is entered, Sign-On/Security can link the attempt with a known user and will record that user's name in the log. Since it is a valid code, its text is not recorded in the log. The text of subsequently entered invalid verify codes can, however, be recorded as clues to the source of the access attempt. If the access code is not valid, a user's name cannot be associated but the text of the attempt can be recorded. The log also records the time of day, device used, and CPU/UCI location.

■ Kernel Sign-On Auditing Files

SIGN-ON LOG (#3.081): ^XUSEC(0,

Set Parameters:	**predefined**
Display Parameters:	**N/A**
Initiate/Terminate:	**always done**
Print Reports:	**Print Sign-on Log [XUSC LIST]**
Purge Logs:	**Purge Sign-on Log [XUSCZONK]**

FAILED ACCESS ATTEMPTS LOG (#3.05): ^%ZUA(3.05,

Set Parameters:	**Establish System Audit Parameters [XUAUDIT]**
Display Parameters:	**Display the Kernel Audit Parameters [XU-SPY-SHOW]**
Initiate/Terminate:	**on/off switch**
Print Reports:	**devices: Device Failed Access Attempts [XUFDEV]** **users: User Failed Access Attempts [XUFDISP]**
Purge Logs:	**Failed Access Attempts Log Purge [XUFPURGE]**

OLD ACCESS AND VERIFY CODES (#200 XREF): ^VA(200,

Set Parameters:	**predefined**
Display Parameters:	**N/A**
Initiate/Terminate:	**always done**
Print Reports:	**N/A**
Purge Logs:	**Purge Log of Old Access and Verify Codes [XUSERAOLD]**

Purge Old Access and Verify Codes

```
SYSTEMS MANAGER MENU ...                                        [EVE]
User Management ...                                            [XUSER]
   Purge Log of Old Access and Verify Codes                [XUSERAOLD]
```

This option purges all inactive access and verify codes, which allows for the recycling of codes. Old access and verify codes are stored so that users may not pick a previously used code when required to choose a new code. If old codes are stored indefinitely, though, it may become difficult for users to invent new codes. When you use this option interactively, you can purge codes older than a retention period you specify, from 7 to 90 days. When scheduled, the retention period defaults to 90 days, but can be changed to anything from 30 to 90 days by putting the number of days in the TASK PARAMETERS field.

The log of access codes is stored in the whole-file AOLD cross reference of the NEW PERSON file (#200). The log of verify codes is stored per user in the VOLD cross reference of File #200 (not a whole-file cross-reference). So, verify codes are not necessarily unique between users, while access codes are.

Chapter 3 Sign-On/Security: Programmer Tools

The Kernel's Sign-On/Security module sets up a standard DHCP programming environment as a foundation for application packages. Once a sign-on session has been created, application packages may assume that system-wide variables exist for common reference. For example, key variables defined via Sign-On/Security include the user's institution and agency (DUZ(2) and DUZ("AG"), respectively).

Direct Mode Utilities

Several direct mode utilities are available for programmers to use at the MUMPS prompt. They are not callable entry points and cannot be used in application package routines. These utilities do allow programmers to simulate ordinary user sign-on and yet work from programmer mode to test code and diagnose errors.

>D ^XUP: Programmer Sign-On

The ^XUP routine can be called as a quick way to enter the Kernel and set up a standard environment. It sets up DT, calls ^%ZIS, prompts for access code if DUZ is zero or undefined, kills and rebuilds ^XUTL("XQ",$J), and kills ^UTILITY($J). Finally, it calls ^XQ1 to prompt for an option if one should be run. If a non-menu-type option is specified, returning from the option will display the "Select:" prompt as though the option was a menu-type. Although this construction may at first appear misleading, restricting option selection to menu-type only would be a functional limitation to the call.

>D ^XUS: User Sign-On; No Error Trapping

^XUS determines whether access to the computer is allowed, and then sets up the user with the proper environment. This routine can be called to establish the sign-on environment. A recommended alternative for programmers is to call ^XUP, which establishes sign-on conditions as well as calling ^XQ1 for an option name. Neither ^XUP nor ^XUS sets the error trap. Entering through ^ZU sets the trap and then calls the ^XUS routine.

>D H^XUS: Programmer Halt

This is an obsolete utility. It simply transfers control to ^XUSCLEAN.

>D ^XUSCLEAN: Programmer Halt

Programmers are advised to call the ^XUSCLEAN routine when signing off. It is the same code that the Kernel uses when a user signs off or restarts. It notes the sign-off time in the SIGN-ON LOG file and kills the $J nodes in ^XUTL and ^UTILITY. It then performs a normal halt.

>D ^ZU: User Sign-On

This routine sets the error trap and then calls ^XUS. User sign-ons should be tied to ^ZU.

Callable Entry Points

Entry points related to Sign-On/Security are listed below. You can reference these callable routines within application code.

• $$KSP^XUPARAM: Return Kernel Site Parameter

Usage `S X=$$KSP^XUPARAM(param)`

Input

param: **Site parameter to retrieve. Currently, the following values for param are supported:**

- **INST**
- **SPOOL DOC**
- **SPOOL LIFE**
- **SPOOL LINE**
- **WHERE**

Output

return value: **Requested site parameter value.**

Description

You can use the $$KSP^XUPARAM to retrieve a Kernel site parameter. The following parameters are currently supported:

INST:	**Internal entry number of site's institution, in the site's INSTITUTION file.**
SPOOL DOC:	**MAX SPOOL DOCUMENTS PER USER (internal value) from the site's KERNEL SYSTEM PARAMETERS file.**
SPOOL LIFE:	**MAX SPOOL DOCUMENT LIFE-SPAN (internal value) from the site's KERNEL SYSTEM PARAMETERS file.**
SPOOL LINE:	**MAX SPOOL LINES PER USER (internal value) from site's KERNEL SYSTEM PARAMETERS file.**
WHERE:	**Site's domain name (free text value), from the site's DOMAIN file.**

Examples

```
S A6ASITE=$$KSP^XUPARAM("WHERE")
S A6ASPLLF=$$KSP^XUPARAM("SPOOL LIFE")
```

• SET^XUS1A: Output Message During Sign-On

Usage `D SET^XUS1A(string)`

Input

string: **String to output. First character is stripped from string; if the first character is an exclamation point, a line feed is issued before the string is displayed; otherwise, no line feed is issued.**

Output **none**

Description

Use this function during a package-specific action executed at sign-on (see the XU USER SIGN-ON section). New with Kernel V. 8.0, packages can attach an action-type option to a Kernel extended action-type option called XU USER SIGN-ON. This option, and all attached action-types, are executed during every sign-on.

This function should only be used by action-type options attached to and executed by Kernel's XU USER SIGN-ON extended action.

You should use the SET^XUS1A entry point to perform any output during package-specific actions during sign-on.

Display of the string is not immediate; instead, every call to SET^XUS1A appends a node to an array containing the post sign-on text. When all package-specific sign-on actions have completed, the sign-on process then displays the post sign-on text array, which will also contain any strings registered with the SET^XUS1A function, appended at the end.

• KILL^XUSCLEAN: Clear all but Kernel Variables

Usage `D KILL^XUSCLEAN`

Input **none**

Output **none**

Description

A call to this entry point clears the partition of all but key variables essential to the Kernel. Application programmers are allowed to use this call to clean up application variables and leave the local symbol table unchanged when returning from an option or as otherwise required by SAC Standards.

In the past, options that have called KILL^XUSCLEAN have occasionally created problems for other options that had defined package-wide variables. For example, a user might enter the top-level menu for a package, which could have an entry action that retrieved site parameters into a local variable that is supposed to remain defined while in any menu of that package, between options. But if the user could then reach a secondary menu option that happened to call KILL^XUSCLEAN, a side effect would be the killing off the previously defined package-wide variable.

KILL^XUSCLEAN now provides a way for sites and developers to work around this problem. For any menu-type option, a new field in the OPTION file, PROTECTED VARIABLES, allows you to enter a comma-delimited list of variables to protect from being killed by KILL^XUSCLEAN. Once a user enters a menu subtree descendant from the protected menu, the variables are protected until the menu subtree is exited.

So, for example, to protect a package-wide variable for an entire package, you can enter that variable in the PROTECTED VARIABLES field for the top-level menu in the package. As long as a user doesn't exit the top-level menu of the package's menu tree, the package-wide variable will be protected from all calls to KILL^XUSCLEAN. Up-arrow jumps into a menu tree also work fine, as long as the menu that has been protected is in the menu path made by the jump.

• $$ADD^XUSERNEW: Add New Users

Usage `S X=$$ADD^XUSERNEW([dr_string][,keys])`

Input

dr_string: **[optional] Additional fields to ask when adding the new user, in the format for a DR string as used in a standard DIC call (see the VA FileMan documentation for information about DIC).**

keys: **[optional] A comma-delimited string of keys to assign to the newly created user.**

Output

return value: **Return value is in a similar format to the value of Y returned from a standard DIC call (see the VA FileMan documentation for information about DIC):**

return value=-1	**User neither existed nor could be added.**
return value=N^S	**User already exists in the file; N is the internal number of the entry in the file, and S is the value of the .01 field for that entry.**
return value=N^S^1	**N and S are defined as above, and the 1 indicates the user has just been added to the file.**

Description

Use this extrinsic function to add new entries to the NEW PERSON file. It prompts for the user's name, and for the default identifiers for NEW PERSON entry, which are Initials (field 1), Sex (field 4), and SSN (field 9). If the user of this function has the XUSPF200 key, entry of SSN is not required. The default identifiers can be locally modified by modifying the New Person Identifiers field in the KERNEL SYSTEM PARAMETERS file.

To prompt for additional fields during this call, you pass a DR string containing the field to prompt for as a parameter to this function. If the person adding the entry up-arrows out before filling in all the identifiers and requested fields, the entry will be removed from the NEW PERSON file, and -1 will be returned.

Examples

To add a new user, asking default fields for new entry:

```
>S X=$$ADD^XUSERNEW
```

To add a new user, specifying a key to add:

```
>S X=$$ADD^XUSERNEW("","PROVIDER")
```

To add a new user, specifying additional fields to ask, plus two keys to add:

```
>S X=$$ADD^XUSERNEW("5;13;53","PSMGR,PSNARC")
```

• ^XUVERIFY: Verify Access and Verify Codes

The ^XUVERIFY entry point validates access and verify codes. You can use it anytime within an application program to verify that the person using the system is the same person who signed on.

Usage D ^XUVERIFY

Input

%: **If %="A", ^XUVERIFY checks the access code;**
If %="V", ^XUVERIFY checks the verify code;
If %="AV", ^XUVERIFY checks both the access and verify code.

%DUZ: **The user's number (DUZ value).**

Output

%: **% can be returned with the following values:**

- **-1 An up-arrow was entered.**
- **0 A question mark was entered.**
- **1 Success (the correct code was entered).**
- **2 Failure (the incorrect code was entered).**

Description

Use ^XUVERIFY anytime in an application to check if the current user can enter the access or verify code that matches the currently signed-in user.

XU USER SIGN-ON Option

New Kernel Call to Application Routines

Some packages have asked for the means to execute an action at user sign-on, but not through the alert system. Kernel is providing a new option, called XU USER SIGN-ON, that packages can attach to, and perform package-specific tasks on user sign-on.

XU USER SIGN-ON: Package-Specific Sign-On Actions

Kernel V. 8.0 introduces a new method to support package-specific sign-on actions. Kernel exports an extended-action option called XU USER SIGN-ON. Packages that want Kernel to execute a package-specific user sign-on routine can accomplish this by attaching their own option, of type action, to Kernel's XU USER SIGN-ON option. Your action-type option should call your package-specific user sign-on routine.

To attach your option to the XU USER SIGN-ON option, make your option an item of the XU USER SIGN-ON protocol; then, export your option with a KIDS action of SEND, and export the XU USER SIGN-ON option with a KIDS action of USE AS LINK FOR MENU ITEMS.

During sign-on, Kernel executes the XU USER SIGN-ON option, which in turn executes any options that packages have attached to XU USER SIGN-ON. No database integration agreements are required to attach to the XU USER SIGN-ON option.

If you need to perform any output during your action, you should use the SET^XUS1A function to perform the output. Output is not immediate, but occurs once all package-specific sign-on actions have completed. Also, you should not perform any tasks requiring interaction in an action attached to the XU USER SIGN-ON option.

The variable DUZ will be defined at the time the sign-on actions are executed; DUZ will be set as it normally is to the person's internal entry number in the NEW PERSON file.

Take care to make code efficient, since executed by every sign-on. A few examples of tasks you might want to accomplish during sign-on are:

- **Alert the user to a package status.**
- **Issue a reminder.**
- **Notify the package of the sign-on of a package user.**

Example

The following option, when attached to the XU USER SIGN-ON protocol, outputs one line during sign-on:

```
NAME: ZZTALK PROTOCOL                    MENU TEXT: TALKING PROTOCOL
  TYPE: action                           E ACTION PRESENT: YES
 DESCRIPTION:   USE TO TEST EXTENDED ACTION PROTOCOLS
  ENTRY ACTION: D SET^XUS1A("!This line is from the ZZTALK option.")
  UPPERCASE MENU TEXT: TALKING PROTOCOL
```

XU USER TERMINATE Option

New Kernel Call to Application Routines

Kernel V. 8.0 introduces a new method to support package-specific user termination actions. Kernel V. 8.0 exports an extended-action option called XU USER TERMINATE. Packages that want Kernel to execute a package-specific user termination action can accomplish this by attaching their own option, of type action, to Kernel's XU USER TERMINATE extended action.

Discontinuation of USER TERMINATE ROUTINE

Kernel V. 7.1 introduced a method for packages to have Kernel execute a package-specific routine when Kernel terminated a user. The method was for the package to have a routine tag and name in fields 200.1 (USER TERMINATE TAG) and 200.2 (USER TERMINATE ROUTINE) of the package's PACKAGE file entry. When Kernel V. 7.1 terminated a user, it executed the TAG^ROUTINE entry point stored in these fields, if any.

Kernel V. 8.0 will continue to execute the entry point, if any, stored in package's PACKAGE file entry. However, Kernel V. 8.0 will be the last version to support that method of package-specific user termination routines.

Creating a Package-Specific User Termination Action

Beginning with Kernel V. 8.0, you should create an action-type option that calls your package-specific user termination routine. To attach it to the XU USER TERMINATE option, export your option with a KIDS action of SEND, and export the XU USER TERMINATE option with a KIDS action of USE AS LINK FOR MENU ITEMS.

Kernel defines the variable XUIFN at the time your action executes; it is defined as the internal entry number in the NEW PERSON file of the user being terminated.

When terminating a user, Kernel executes the XU USER TERMINATE option, which in turn executes any options attached to XU USER TERMINATE. No database integration agreements are required to attach to the XU USER TERMINATE option.

A few examples of user clean up you might want to accomplish when Kernel terminates users are:

- **Removal of HINQ access.**
- **Removal of Control Point access.**
- **Removal from health care teams.**

Chapter 4 Electronic Signature Codes

User Interface

An electronic signature is a security tool that application packages can use as an additional identification check. A package may, for example, require that an electronic signature be applied to a particular form or document before subsequent processing can continue.

If you need create an electronic signature for yourself, you can choose the Edit Electronic Signature Code option, available from the User's Toolbox menu.

You can enter a new electronic signature code, or change an existing code. The length of the code must be between 6 and 20 uppercase characters. Requiring all uppercase allows the code to be verified with either uppercase or lowercase input (since lowercase will be converted to uppercase in the matching process). You should choose a code that other users are not likely to guess, as this code verifies that it is actually you who are signing off on some important action.

```
SYSTEM COMMAND OPTIONS ...                                  [XUCOMMAND]
  User's Toolbox ...  "TBOX"                               [XUSERTOOLS]
    Edit Electronic Signature code                            [XUSESIG]
```

The Edit Electronic Signature Code option also allows you to edit the following fields:

- **Initial**
- **Signature Block Printed Name**
- **Signature Block Title**
- **Office Phone**
- **Voice Pager**
- **Digital Pager**

Applications may print some or all of these fields when printing an electronically signed document. You should therefore ensure that the values entered in these fields are accurate.

System Management

```
SYSTEMS MANAGER MENU ...                                              [EVE]
User Edit ...                                                        [XUSER]
   Electronic Signature Block Edit                                [XUSESIG BLOCK]
   Clear Electronic signature code  <locked:  XUMGR>               [XUSESIG CLEAR]
```

Electronic signature codes are stored in the NEW PERSON (#200) file.

The Electronic Signature Block Edit option [XUSESIG BLOCK] lets you edit the electronic signature code for any user on the system. When you create an electronic signature code for a user, the SIGNATURE BLOCK PRINTED NAME field is initially filled in by a cross-reference on the Name field (and will be overwritten if the Name field is changed). Credentials such as M.D. may be added to customize the printed name. As a security feature, an input transform requires that the user's last name (first comma piece of the Name field) be included in the printed name. (This field cannot be edited through FileMan since it is write-protected with an up-arrow.)

Another option available to IRM allows the clearing (deleting) of an electronic signature code. This option is locked with the XUMGR key. This option can be used to clear a user's electronic signature code if the user has forgotten the code. The user can then enter a new code with the Edit Electronic Signature Code option in the User's Toolbox.

Programmer Tools

Several entry points are available for programmers to work with electronic signature code. These entry points are described below.

• SIG^XUSESIG: Verify Electronic Signature Code

Usage `D SIG^XUSESIG`

Input	**DUZ:**	**User number.**
Output	**X1:**	**If the user entered the correct electronic signature code, the encrypted electronic signature code as stored in File #200 is returned in X1. Otherwise, X1 is returned as null.**

Description

Use SIG^XUSESIG to request and verify the electronic signature code of the current user.

• DE^XUSHSHP: Decrypt Data String

Usage `D DE^XUSHSHP`

Input	**X:**	**Encrypted string generated by a call to EN^XUSHSHP.**
	X1:	**Identification number used as input variable X1 in EN^XUSHSHP call.**
	X2:	**Number used as input variable X2 in EN^XUSHSHP call.**
Output	**X:**	**The decrypted string (can be printed).**

Description

Use this entry point to decrypt a string encrypted by a call to EN^XUSHSHP. Typically this entry point would be used to decrypt strings when printing a document containing encrypted strings.

• EN^XUSHSHP: Encrypt Data String

Usage D EN^XUSHSHP

Input

X:	**The string to be encrypted (such as the contents of the SIGNATURE BLOCK PRINTED NAME field in the NEW PERSON file).**
X1:	**An identification number such as DUZ.**
X2:	**A document number (or the number one).**

Output

X:	**Encrypted string.**

Description

This entry point encrypts a string, and associates the encrypted string with an identification number and a document number. To decrypt the string, a call must be made to the DE^XUSHSHP (decrypt) entry point, with the encrypted string, identification number, and document number as input variables. Typically this entry point would be used to encrypt strings within a document.

• HASH^XUSHSHP: Hash Electronic Signature Code

Usage D HASH^XUSHSHP

Input

X:	**Electronic Signature code as entered by the user.**

Output

X:	**Hashed form of the electronic signature code submitted as input to function.**

Description

This entry point uses as input the text string (signature) entered by the user. The routine then hashes the string. The hashed result can then be used to verify the user's identity by comparison with the stored electronic signature code (in the NEW PERSON file).

Chapter 5 File Access Security

The File Access Security system is an optional Kernel module. It provides an enhanced security mechanism for controlling user access to VA FileMan files.

User Interface

As a user, you typically access DHCP data by use of application options. You enter data into files and retrieve information from files through the menu options within an application package. Except under a few unusual circumstances, your use of the system will not be affected by the File Access Security system.

If you need to work directly with files by using VA FileMan options, however, you will be affected. VA FileMan options provide direct access to data files; an example of some VA FileMan options is as follows

```
Select VA FileMan Option: ? <RET>

          Enter or Edit File Entries
          Print File Entries
          Search File Entries
          Inquire to File Entries
```

If the File Access Security system is implemented, the only files you can access directly through VA FileMan options are those listed in your ACCESSIBLE FILE multiple in the NEW PERSON file. IRM grants file access by using a sub-menu on the User Management menu. Six levels of access are possible:

1. **DATA DICTIONARY**
2. **READ**
3. **WRITE**
4. **DELETE**
5. **LAYGO**
6. **AUDIT**

Each level of access is granted as yes or no. If the File Access Security system is implemented, file access is controlled by these yes/no flags, not by the matching of your FileMan access code string with security placed on the file.

If you have not been granted any accessible files, entering two question marks when prompted for a file shows no files to access:

```
Select VA FileMan Option: Enter or Edit File Entries <RET>
INPUT TO WHAT FILE: ?? <RET>

INPUT TO WHAT FILE:
```

In this case, you need to contact IRM and be granted access to the files you need.

File Access Security is also invoked when an option uses the VA FileMan line editor. In particular, the Transfer Lines from Another Document option on the line editor's Edit menu does not permit access to other word processing documents in the current file or other files unless read access to that file has been explicitly granted. If you need to transfer text from other files using the line editor, contact IRM to request access to those files.

System Management

Prior to introduction of the File Access Security system, user access to VA FileMan files through VA FileMan options was controlled by matching a character in a user's File Manager access code (the DUZ(0) string) with a character in the file's top level file security fields.

Kernel's optional File Access Security system uses a different method. It allows you to control access to files for any user using VA FileMan options directly. Access is granted (or denied) by adding (or removing) a file from a user's ACCESSIBLE FILE multiple in their NEW PERSON file entry.

As before, the File Access Security system does not affect access to files through non-VA FileMan options; security in this case is managed by controlling the availability of the option. The exceptions to this are discussed below, in the "When is File Access Security Checked?" section.

As before, if a user's DUZ(0) is set to the programmer's at-sign (@), VA FileMan options allow complete file access. If it is set to anything else, VA FileMan options use the ACCESSIBLE FILE specifications in the NEW PERSON file to grant varying levels of file access.

This higher degree of control over a user's file access comes at a price: more management on IRM's part is required to provide each user access to the files they need access to. The payoff in using the File Access Security system is in enhanced control and security for VA FileMan files.

When is File Access Security Checked?

When using VA FileMan options, access to files through the File Access Security system is checked.

When accessing data through package options (e.g., options using VA FileMan programmer calls), File Access Security is not checked when initially accessing a file. It is checked when in ^DIC calls when adding an entry to the top level of a file (LAYGO access), and is checked in ^DIE calls when deleting an entry at the top level of a file (DELETE access). Programmers can bypass these LAYGO and DELETE access checks using the variables DLAYGO and DIDEL, respectively.

When accessing data through package options, File Access Security is also checked when a file is navigated to (READ, WRITE, DELETE, and LAYGO access) from another file. There is not currently a way for programmers to override access checks when navigating to a file from another file, so explicit access to files navigated to from an application option must be granted by the IRM.

What in FileMan is Still Protected by the File Manager Access Code?

When the File Access Security system is enabled, access to templates (input, print, sort, etc.) is denied (as before) when using VA FileMan options, if the user's DUZ(0) string does not contain a matching character. Similarly, when editing fields via VA FileMan's Enter or Edit File Entries option, the DUZ(0) matching process is invoked to permit or deny editing for protected fields. The DUZ(0) value is also checked by some non-VA FileMan applications. Finally, if a user's DUZ(0) is "@", they are allowed complete access to all files.

The Purpose for Granting File Access

IRM is responsible for granting file access. The needs of each user must be determined and an appropriate degree of access authority assigned. Too much access may risk the security of your system, while too little may inhibit productive activity.

What is the purpose of file access? Why bother specifying who has access to which files? The answer is threefold:

- **To monitor the use of VA FileMan.**
- **To regulate the extent of VA FileMan access from among six forms which allow READ, WRITE, DELETE, LAYGO, DD, or AUDIT access.**
- **To reserve DUZ(0), the VA FileMan access code, as a security measure to protect just templates and fields, not files, from VA FileMan options.**

With file access security, it is possible to know who has access to which files and what kind of access they have. This information can also be retrieved by user or by file. In addition, privileges can also be entirely restricted for an individual user or for a single file that may contain sensitive information.

Who Needs File Access

You need to grant file access in the following cases:

- **A user needs to access files directly through VA FileMan options.**
- **Within an application option, VA FileMan is used to navigate from one file to another.**
- **Within an application option that calls ^DIE to edit a file entry, a user is unable to add or delete entries in a pointed-to file.**
- **Within an application option that calls ^DIE or ^DIC to edit a file entry, a user is unable to add or delete entries in the primary file (because the application didn't set DLAYGO or DIDEL).**

- **A user needs to use the VA FileMan line editor's "Transfer Lines from Another Document" function.**

Application programmers can document which files need to be granted to whom, or can modify their code or data dictionary specifications to allow access.

Levels of File Access Authority

READ ACCESS	**Permits display of data on file. This allows use of the VA FileMan Print, Search, Inquire, Check/Fix DD Structure, and List File Attributes options. Read access is also required to use some of the Filegram and Audit options. To transfer text (responding to the Line Editor's "Edit option:" prompt), the user needs read access to the file from which text is being transferred. Similarly, read access is needed for the file from which entries are being transferred with the Transfer File Entries option.**
WRITE ACCESS	**Permits changing data values that are currently on file via the Enter or Edit File Entries option. It will not permit the adding of new entries to the file. Write access is needed for the file being "transferred to" when using the Transfer File Entries option. It is also needed for the Compare/Merge File Entries option.**
DELETE ACCESS	**Permits deletion of an entire file entry with Enter or Edit File Entries. It does not permit deletion of the file or any of its attribute fields. Delete access is needed to the "transfer from" file when using the Transfer File Entries option.**
LAYGO ACCESS	**Permits adding a new entry to a file (note that write access is also required if using VA FileMan's Enter or Edit File Entries option.)**
DD ACCESS	**Permits data dictionary access for changing or deleting the structure of the file itself. The Modify File Entries and Utility options require DD access. To use the Map Pointer Relations option, DD access is needed to the PACKAGE file and to the files one selects for mapping. It is also needed when using the Check/Fix DD Structure option to fix (change) the file's structure.**
AUDIT ACCESS	**Allows the audit of data changes provided that DD access has also been granted.**

These are the six levels of access. Any or all may be enabled for each of the user's accessible files. This is done by changing the field value from null to YES. This flag is overridden for programmer-users whose DUZ(0)=@. Granting all of the first four levels of access (READ, WRITE, DELETE, and LAYGO) permits adding and deleting file entries as well as editing their attribute field data values. This is true unless the attribute field has been protected. If so, that is, if there is read, write, or delete protection within the data dictionary for a given field, the user's File Manager Access Code, DUZ(0), is checked. Access is denied if the user's DUZ(0) doesn't contain a character matching the field protection. Again, DUZ(0)=@ overrides this restriction.

The last two kinds of access (DD and audit) pertain to the structure of the file itself. While this provides a generous scope for VA FileMan data dictionary modification, it falls short of, for example, deleting a field protected with the at-sign (@).

The same applies to templates. If the template is protected, the user who has access to the file will not have access to the template from VA FileMan options unless there is a match in the DUZ(0) character string.

Audit Access to Files

Audit privileges might be granted to advanced VA FileMan users who are interested in developing new audit capabilities. With audit access, which must be accompanied by DD access, VA FileMan's Modify option may be used to set an audit flag for a particular field within a file. This access does not include setting audit conditions with M code, which is reserved for users with a File Manager access code containing "@".

The data values for attribute fields may be recorded in the Audit File by setting an audit flag in the data dictionary for that field. The SSN field in the PATIENT file, for example, could be audited. There are two choices for the audit. An entry may be made in the AUDIT file when a value is entered or changed, or an entry may be made only when the value is changed, that is, edited or deleted. The second method may be all that's needed. In the SSN example, you would monitor just the circumstances of the change, not of the initial SSN assignment.

To display the results of the audit, your DUZ(0) must equal @. Then, you can query the AUDIT file in the usual way with FileMan's Inquire option.

How to Grant File Access

IRM specifies the particular files and levels of access for users. The File Access Security menu, on the User Management menu, provides options to grant file access security. These options edit a multiple field in the NEW PERSON file called ACCESSIBLE FILE.

The options for granting file access privileges fall into three functional categories:

EDITING — **To assign file access to an individual user or a group of users. One user's profile may also be duplicated or copied to another user or group of users. To simplify adding files, number ranges may be specified.**

LISTING — **To display one user's profile, a name-sorted list of all user's profiles, or a file or range of files with associated users and the access levels of each.**

RESTRICTING — **To entirely limit access by user or by file, or to delete a range of files for a user or group of users.**

The options are designed to facilitate queries by user or by file. You may add or delete file access for one user or for many users. Or you may begin with the file and list users with access or restrict access.

Using the File Access Options

```
SYSTEMS MANAGER MENU ...                                          [EVE]
User Management ...                                              [XUSER]
   File Access Security ...                                 [XUFILEACCESS]
      Grant Users' Access to a Set of Files                  [XUFILEGRANT]
      Copy One User's File Access to Others                   [XUFILECOPY]
      Single file add/delete for a user                   [XUFILESINGLEADD]
      Inquiry to a User's File Access                       [XUFILEINQUIRY]
      List Access to Files by File number                      [XUFILELIST]
      Print Users Files                                       [XUFILEPRINT]
      Delete Users' Access to a Set of Files               [XUFILESETDELETE]
      Remove All Access from a Single User                 [XUFILEREMOVEALL]
      Take away All access to a File                          [XUFILEDELETE]
      Assign/Delete a File Range                        [XUFILERANGEASSIGN]
```

When using the file access options, you may have the following questions:

- **What is the DUZ# that appears next to the user's name?**
- **How is a range of file numbers specified?**
- **What are the queuing questions all about?**

Understanding DUZ (the User Number)

When listing the file accesses by user or by file, the user's name is followed by a number in parentheses. The heading indicates that this is the User #. It is the same as the DUZ#.

DUZ is a local variable that identifies the user who has signed-on. Once the user enters an access and verify code, the Kernel's Sign-On/Security uses this variable to identify an entry in the NEW PERSON file. It must be unique, so the user's name will not do. Instead, the internal entry number is used. That is what becomes the value of DUZ.

Some users have low numbers while others have high ones. This simply indicates the order their names were entered into the NEW PERSON file. Users with low numbers are often people who began using your system some years ago, while users with high numbers tend to be recent entries in the file.

Using Ranges of File Numbers

Can files be specified by number ranges? Yes; it is useful to do this when granting several files at once. First find out the number of the files. Typing a question mark at the "to files:" prompt will display the number and name of the files. Note the numbers and then put them together on one line. You can use dashes to indicate a consecutive range and commas to separate the single numbers and dashed groups as follows:

2,3,4,6,7,8,125,236,799 or 2-4,6-8,125,236,799

File numbers are also used when printing a group of consecutive files. The prompt asks for a place to start with a default file name presented. To print just this one file, respond to the next prompt by pressing <RET>, thereby accepting the default of ending after printing that one file.

To print a consecutive range of files, the lowest number is entered as the starting point and the highest number as the ending point. All files that fall in this range will be printed.

Queuing File Access Specifications

Most of the options provide the opportunity to queue, after specifying who is to be granted which files. Queuing sends the specifications to the Task Manager to assign to users at a later time. TaskMan can work at an off-peak time, like midnight, to avoid consuming system resources during the daytime. If the system is not busy, queuing is still a good idea since your terminal will otherwise be tied up while the report is being printed.

Running The File Access Security Conversion

Advantages

To implement the file Access Security, you need to run a conversion. Some advantages of implementing the File Access Security system are:

- **File Access Security: Running the conversion makes it possible to identify the levels of access each individual user has to each file.**
- **System Performance: Checking file access by user is slightly faster in terms of global accesses and CPU time.**

Advance Preparation for the Conversion

The conversion is designed to allocate access privileges to all of your users according to their current FileMan access code (DUZ(0), combined with information about their file access through options stored in the ^DISV global. After the conversion you should get only a few user requests for file access. The File Access Security menu, an option on the User Management menu, should then be used to add a file to a user's Accessible File multiple.

The conversion uses the FileMan access code (DUZ(0)) string) to assign file access according to the characters in the string. If a file is protected with a particular character that matches one in the user's code, that file is entered into the user's ACCESSIBLE FILE multiple. Levels of access are granted according to the file's original security (field-level security continues to function the same, by checking the FileMan access code). Note that users with programmer-level access (VA FileMan access code=@) will not need to have any files in their ACCESSIBLE FILE multiple since they will be able to access all files without restriction.

^DISV Global

The conversion process makes use of the ^DISV global to identify which files had recently been accessed by which users. The conversion adds all files that the user had been able to access (select from) to the user's ACCESSIBLE FILE list. It grants read access to these files.

Using the ^DISV global to grant file access has the benefit of permitting option usage "as usual" the day after the conversion is run. Killing the ^DISV global just before the conversion is not advised since many users will suffer inappropriate access restrictions and will need special attention by IRM Service just after the conversion. Killing the ^DISV global a week or two before the conversion, however, may be worthwhile as a way of purging obsolete user data.

In multi-CPU environments, where each CPU has its own copy of the ^DISV global, you should choose the busiest user node to run the conversion on (so as to pick up the most comprehensive information from that node's ^DISV). MSM-DOS sites should run the conversion from their busiest compute server; DSM for OpenVMS sites should run the conversion from their busiest user node.

It is assumed that ^DISV is not translated, so K ^DISV on the CPU where the conversion will be run. Do this about two weeks before you perform the conversion, as advance preparation. ^DISV will be reset as soon as a user responds to a "Select:" prompt.

```
>K ^DISV <RET>     (only on the CPU where the conversion will run,
                   about two weeks beforehand, as advance
                   preparation.)
```

Adding Explicit File Access for IRM

If there are any files that are neither protected nor accessed by users (e.g., the DOMAIN file) the conversion will not list them in any user's ACCESSIBLE FILE multiple. Before the conversion, such files are accessible to everyone, while after the conversion such files will only be accessible to users with programmer-level access. So, before the conversion, assign a character such as the pound sign (#) to otherwise unprotected files. This will ensure that at least those users with the pound sign (usually IRM staff) will be granted access. VA FileMan's Edit File option may be used to edit the codes.

■ Updating File Access Settings (Before Conversion)

```
Select OPTION: UTILITY FUNCTIONS <RET>
Select UTILITY OPTION: EDIT FILE <RET>

MODIFY WHAT FILE: USER// DOMAIN <RET>          (227 entries)
NAME: DOMAIN// <RET>
DESCRIPTION:. . .
Select APPLICATION GROUP: <RET>
DEVELOPER: <RET>
VERSION: <RET>
DATA DICTIONARY ACCESS: (Enter a code such as '#" for each level of
READ ACCESS: <RET>             access so that such unprotected files
WRITE ACCESS: <RET>                  will be assigned to IRM staff.)
DELETE ACCESS:  <RET>
LAYGO ACCESS:  <RET>
AUDIT ACCESS:  <RET>
```

Summary of How the File Access Security Conversion Works

The File Access Security system conversion prepares the NEW PERSON file for VA FileMan's new method of file access (lookup into a user's record for file access). VA FileMan's ability to protect data within files on fields and templates, remains the same. The steps that occur when the conversion is run are outlined below.

1. **The structure for implementing the new file access method is set up by the following:**

   a. **Placing the data dictionary for the ACCESSIBLE FILE multiple in the NEW PERSON file. (This multiple will be permanently put in place by running the File Access Security conversion.)**

   b. **Installing menu options, help frames and templates used for maintaining the new user file access method (entries with the XUFI namespace).**

2. **Each user's VA FileMan access code is used to add entries in the ACCESSIBLE FILE multiple by:**

   a. **Creating a list of files to be processed by examining each file's protection codes. Files that meet *both* of the following requirements are temporarily stored in the ^UTILITY($J global:**

      - **Files that have protection defined.**
      - **Files with protection *not* equal to "@".**

      Note: Files that lack any protection will thus be bypassed. Such unprotected files will not later be listed in anyone's ACCESSIBLE FILE multiple. Protection (such as #) should therefore be applied before the running conversion so that at least some users (IRM Staff) will be granted access.

   b. **Examining each user in the NEW PERSON file. Each user meeting *all* of the following requirements is selected for further processing:**

      - **Users *not* terminated.**
      - **Users with an Access Code.**
      - **Users with a VA FileMan access code.**
      - **Users with a VA FileMan access code *not* equal to "@".**

      The user's VA FileMan access code is parsed. Each character is compared with the list of files in the ^UTILITY($J global. All files that have a protection code matching this character are added to the

user's ACCESSIBLE FILE multiple. If the character is used as the file's DD security, the user is granted DD access; if it is used as LAYGO, the user is granted LAYGO access, and so on.

3. **Files accessed by the user through options since the last time the ^DISV global was killed are added to the user's ACCESSIBLE FILE multiple by the processing of the ^DISV global. Entries in ^DISV that meet *both* of the following requirements will be added to the ACCESSIBLE FILE multiple, with READ access:**

   - **The file must not be in VA FileMan's file number range (file number must be equal to or greater than 2).**
   - **The user does not already have access to this file.**

File Access Security Conversion Instructions

1. **Identify unprotected files and assign protection codes as desired (as described in Advance Preparation section earlier). The DOMAIN file, for example, may need to be protected with the pound sign (#) so that it will be granted to users having a FileMan access code containing this character.**

2. **Review VA FileMan access codes of VA FileMan users. The codes should contain characters matching those used to protect the files that these individuals use. Since the conversion will automatically grant files to users according to previous privileges as indicated by the FileMan access code, add any additional characters to their FileMan access codes to take advantage of the conversion's automated file assignment according to levels of access.**

3. **Be ready to use the File Access Security Menu to review and grant file access privileges after the conversion.**

4. **In the production account, enable File Access Security system features and options with ENABLE^XUFILE3, as illustrated below:**

```
In VAH:

>D ENABLE^XUFILE3 <RET>

>
```

5. **In the production account, begin the conversion with ^XUINCON:**

```
In VAH:

>D ^XUINCON <RET>

Version 7 of the Kernel defined a new multiple-valued field
in the New Person File called Accessible File.  This conversion
will store file access in this multiple in the following manner:

Those Users who have a FileMan Access Code (DUZ(0)) which
is not null, i.e., contains some character string,
will have their access string matched to the protection
currently on your files.  For each match between the file
and the user, the file will be listed in the user's
Accessible File multiple as will the type of access
(dictionary, delete, laygo, read, write, audit).

NOTE: Files with no protection will NOT be assigned to any user.

Would you like to run the conversion now? NO//
```

6. **If you are ready to run the conversion, answer YES:**

```
Would you like to run the conversion now? NO// YES <RET>
56237,36565
Build Table.
Convert Users.
Give access from DISV file.
X-ref.
Done56237,36565.
>
```

7. **Review the newly assigned access settings. Use the File Access Security menu (from the User Management Menu) to display file access by user and by file.**

8. **Any XUFILE* templates that are associated with File #3 may be deleted. They are obsolete and replaced by the comparable ones associated with File #200.**

After the Conversion

After the conversion, users may complain about not being able to add entries to files as they previously could. This typically results from use of an option that navigates from one file to another. To be able to add entries to the navigated-to file, the user needs LAYGO access to that file. IRM can solve the problem by granting LAYGO, using the File Access Security options.

As this form of security is implemented, IRM will find that it provides a more accurate and precise knowledge of who has what level of access to which files. When the conversion is run, privileges are granted to existing users by making use of information stored in the FileMan record of file manipulation activity, the ^DISV global. The file access conversion grants each user read access to files that user had recently accessed as indicated in ^DISV. IRM may grant file access privileges to new users by copying the profile of an existing user with similar duties, such as a laboratory application coordinator or admissions clerk.

To be sure that appropriate levels of access have been allocated, IRM staff should determine who has what level of access to which files. Access to sensitive files, such as the NEW PERSON file, should be reviewed and readjusted for individual users as appropriate. All files on your system should be reviewed before and after running the conversion. The following capture shows how to create a print template for this purpose:

```
Select OPTION: PRINT FILE ENTRIES <RET>

OUTPUT FROM WHAT FILE: FILE <RET>
SORT BY: NAME// @NUMBER <RET>
START WITH NUMBER: FIRST// 3 <RET>     (Enter starting file number.)
GO TO NUMBER: LAST// 4 <RET>         (Enter the ending file number.)
  WITHIN NUMBER, SORT BY: <RET>
FIRST PRINT ATTRIBUTE: NUMBER;L8;S;"" <RET>
FIRST PRINT ATTRIBUTE: NAME;L25;"" <RET>
THEN PRINT ATTRIBUTE: DD ACCESS;R6 <RET>
THEN PRINT ATTRIBUTE: RD ACCESS;R6 <RET>
THEN PRINT ATTRIBUTE: WR ACCESS;R6 <RET>
THEN PRINT ATTRIBUTE: DEL ACCESS;R6 <RET>
THEN PRINT ATTRIBUTE: LAYGO ACCESS;R6 <RET>
THEN PRINT ATTRIBUTE: AUDIT ACCESS;R6 <RET>
THEN PRINT ATTRIBUTE:<RET>
HEADING: FILE LIST// FILE SECURITY <RET>
STORE PRINT LOGIC IN TEMPLATE:    (Store in a local template for later use,
                                        using a name like ZZFILE SECURITY.)
```

Once the conversion has been run, you can use the File Access Security menu to print the accessible files for individual users. You may thus establish profiles that would be typical of groups of users in Nursing, Pharmacy, or other services. Then, when establishing an account for a new user or reactivating the access of a previously terminated user, the profile will be available for copying to the new user.

Programmer Tools

For an overview of the functionality provided by the File Access Security system, please see the System Management section of this chapter.

Field Level Protection

As before, the DUZ(0) check is not performed when a user traverses fields in a DR string or in a template (field-level protection is checked during the template-building process, but not subsequently when the template is invoked by a user.) If you want to make the presentation of fields conditional, based on a user's DUZ(0), branching logic may be used as described in the *VA FileMan Programmer's Manual*.

File Navigation

Edit-type options that navigate to a second file do so by calling VA FileMan and, hence, depending on the type of navigation and the existing file protection, will require that the user have write access to change data in the pointed-to file, delete access to delete an entry, and perhaps LAYGO access to add a new entry.

Adding new entries when navigating to a file is controlled by LAYGO access. If a pointing field allows LAYGO, as specified in the data dictionary, and the pointed-to file also allows LAYGO, the user will not need explicit file access to add entries. If the pointed-to file is protected, however, the user will need explicit LAYGO access to the file. Delete access is checked at the moment the user tries to delete a file entry.

When coding calls, if DIC(0) contains 'L', DIC allows the user to add a new entry if one of three conditions is met:

- **The user has been granted LAYGO access to the file.**
- **The user's DUZ(0) is equal to '@'.**
- **The variable DLAYGO is defined equal to the file number.**

Use of DLAYGO When Navigating to Files

Use of input templates or ^DIE calls as part of edit-type options permits user access to the first file. But if navigation to a second file is involved, LAYGO is not automatically be granted. One of the three conditions mentioned above must be met to allow navigation to the second file: LAYGO access is granted, DUZ(0)=@, or the variable DLAYGO is set.

Providing LAYGO access by using the DLAYGO variable obviates the need for IRM to grant LAYGO file access to the pointed-to file via the File Access system. An example of setting DLAYGO in a template is shown below:

```
INPUT TO WHAT FILE: RENTAL <RET> (A file pointed-to by the Line Item file.)
EDIT WHICH FIELD: TRANSACTION NUMBER <RET>
THEN EDIT FIELD: DATE RENTED <RET>
THEN EDIT FIELD: S DLAYGO=800265 <RET>      (Set DLAYGO to the number of the file to be
                                             navigated-to via backward pointing.)
THEN EDIT FIELD: LINE ITEM: <RET>
   By 'LINE ITEM', do you mean the LINE ITEM File,
       pointing via its 'RENTAL TRANSACTION' Field? YES// Y <RET> (YES)
WILL TERMINAL USER BE ALLOWED TO SELECT PROPER ENTRY IN 'LINE ITEM' FILE?
YES// <RET> (YES)
DO YOU WANT TO PERMIT ADDING A NEW 'LINE ITEM' ENTRY? NO// Y <RET> (YES)
WELL THEN, DO YOU WANT TO **FORCE** ADDING A NEW ENTRY EVERY TIME? NO// <RET>
(NO)
DO YOU WANT AN 'ADDING A NEW LINE ITEM' MESSAGE? NO// N <RET> (NO)
   EDIT WHICH LINE ITEM FIELD: LINE ITEM <RET>
   THEN EDIT LINE ITEM FIELD: RENTAL TRANSACTION <RET>
   THEN EDIT LINE ITEM FIELD: K DLAYGO <RET>     (Kill DLAYGO upon exit.)
   THEN EDIT LINE ITEM FIELD:
```

Use of DLAYGO in ^DIC Calls

When a user attempts to add an entry at the top level of a file in a ^DIC call, their file access security is checked for LAYGO access to the file. Programmers can override this check (and save the site from having to grant explicit LAYGO access) by setting DLAYGO to the file number in question. For more information on DLAYGO as used in ^DIC calls, see the *VA FileMan Programmer Manual*.

Use of DIDEL in ^DIE Calls

When a user attempts to delete an entry at the top level of a file in a ^DIE call, their file access security is checked for DELETE access to the file. Programmers can override this check (and save the site from having to grant explicit DELETE access) by setting DIDEL to the file number in question. Use of DIDEL does not override a file's "DEL" nodes, however. For more information on DIDEL as used in ^DIE calls, see the *VA FileMan Programmer Manual*.

Part 2: Menu Manager

Chapter 6 Menu Manager: User Interface

The Kernel's menu system presents menu options within DHCP application packages in a standard fashion. Once you become familiar with using the menu system in one application, using other applications will be easier since the same rules apply.

Navigating Kernel's Menus

When you successfully sign into the computer system, Menu Manager presents your primary menu options. Your primary menu is the top-level menu assigned to you by IRM. Most options that are available to you are available from your primary menu, or from a sub-menu attached to your primary menu.

The menu system prompts you with a "Select (menu name) Option:" prompt. For example, in a menu named IFCAP, Menu Manager would prompt you with "Select IFCAP Option:". You can navigate through the menu system by responding to this prompt in different ways, which are described in this chapter.

You can enter question marks to see option choices and obtain on-line help. You can enter an option's synonym or the first few letters of its menu text, using upper or lowercase, to select the option. Or you can enter an up-arrow along with the option specification (option menu text or synonym) to jump to the destination option rather then traversing the menu pathways step-by-step.

Choosing Options

You can choose an option from your current menu at the select prompt. Choosing the option launches the computer application associated with the option. To choose an option, type in the first few letters of the option as it is displayed and press <RET>. If the option is another menu, indicated by trailing ellipses (...), it will become the current menu, and so on down the menu pathway.

To come back up the menu pathway, press <RET> at the select prompt. Each time you press <RET>, Menu Manager will return you to the next higher menu level, until you reach your highest menu, the primary menu. If you press <RET> at the primary menu, Menu Manager asks if you want to halt. If you answer yes, your Kernel session will be ended.

Entering One Question Mark to List Options

When you enter a menu, the items may or may not be displayed automatically, based on whether you have "auto-menu" turned on. The auto-menu feature, as described in the Sign-On/Security: User Interface chapter, is a flag that controls the menu display. If you don't have a setting specified for auto-menu, the site parameter default will be used. Often, to save system resources, the site parameter may be set to disable automatic display. In this case, to display your menu's items, simply enter a question mark.

```
Select Any Level Menu Option: ? <RET>

          First Item
          Second Item
          Third Item of Menu Choices ...
          Fourth Item

Enter ?? for more options, ??? for brief descriptions, ?OPTION for
help text.

Select Any Level Menu Option:
```

Entering ?Option to Display Option Help

To obtain a lengthier description of an individual option, enter a single question mark, and the first few letters of the option name. If there is an extended description of the option, or a help frame describing the option, they are displayed.

```
Select User's Toolbox Option: ? <RET>

          Display User Characteristics
          Edit User Characteristics
          Electronic Signature Code Edit
          Menu Templates...
          Spooler Menu...
          TaskMan User
          User Help

Select User's Toolbox Option: ?DISPLAY <RET>



'Display User Characteristics'     Option name: XUUSERDISP
    Display the user's name, location, and characteristics

 **> Press 'RETURN' to continue, '^' to stop: <RET>

Select User's Toolbox Option:
```

Entering Two Question Marks to List Secondary and Common Options

At any select prompt you can enter two question marks to see options on the Secondary and Common menus, as well as options available on the current branch of your menu tree.

The Secondary menu and the Common menu contain options that you can select at any location in the menu system. Options on the Secondary menu are typically created by your system manager. Options on the Common menu are standard Kernel options available from anywhere in the menu system. Options on the current menu, on the other hand, can only be directly selected while that menu is the current menu.

The two-question-mark display shows the option's synonym (a short abbreviation), if one exists . You can select an option by its synonym as well as by its full name. On the same line, it lists the option's full name followed by the formal option name in capital letters enclosed in square brackets. (The name is the .01 field of the OPTION file.) It also shows any option restrictions such as out-of-order, locked, or prohibited times.

■ Listing Current, Secondary, and Common Menu Options

```
Select Systems Manager Menu Option: ?? <RET>

          Core Applications ... [XUCORE]
          Device Handler ... [XUTIO]
   FM     VA FileMan ... [DIUSER]
          Menu Management ... [XUMAINT]
          Programmer Options ... [XUPROG]
             **> Locked with XUPROG
          Operations Management ... [XUSITEMGR]
          Spool Management ... [XU-SPL-MGR]
          Task Manager ... [ZTMMGR]
          User Edit ... [XUSER]

You can also select a secondary option:

          Kermit menu ... [XT-KERMIT MENU]

Or a Common Option:

   TBOX   User's Toolbox ... [XUSERTOOLS]
          Halt [XUHALT]
          Continue [XUCONTINUE]
          Restart Session [XURELOG]
   XMU    MailMan Menu ... [XMUSER]
   VA     View Alerts [XQALERT]
          Time [XUTIME]
          Where am I? [XUSERWHERE]
```

Entering Three Question Marks to Display Option Descriptions

Entering three question marks at any select prompt displays option descriptions (from a word processing field in the OPTION file). If entered at the select prompt for a menu within the primary tree, the top-level options are described; then you are prompted whether you want to see descriptions for secondary or Common options.

```
Select Spooler Menu Option: ???<RET>

'Allow other users access to spool documents'     Option name: XU-SPL-ALLOW
     This option edits the 'OTHER AUTHORIZED USERS' field of the SPOOL
     DOCUMENT file to allow other users access to a spool document.

'Delete A Spool Document'     Option name: XU-SPL-DELETE
  **> Extended help available.  Type "?Delete" to see it.
     Delete a spool document from the spool document file and delete the
     associated message if they are still linked.

'List Spool Documents'     Option name: XU-SPL-LIST
  **> Extended help available.  Type "?List" to see it.
     This option lists entries in the spool document file.

'Make spool document into a mail message'     Option name: XU-SPL-MAIL
  **> Extended help available.  Type "?Make" to see it.
     This option will take a spool document and post it as a mailman
     message to the user's IN basket.  This doesn't move the data at all
     but does decrease the number of lines charged to the user.

  **> Press 'RETURN' to continue, '^' to stop, or '?[option text]' for more
                         help: <RET>

'Print A Spool Document'     Option name: XU-SPL-PRINT
  **> Extended help available.  Type "?Print" to see it.
     This allows the printing of a document that has been spooled.


  Shall I show you your secondary menus too? No// <RET>
  Would you like to see the Common Options? No// <RET>


Select Spooler Menu Option:
```

You should be ready to use three question marks to learn more about unfamiliar options, such as ones distributed in a new package release.

Jumping to Options (the Up-arrow Jump)

The pathways of the primary, secondary, and Common menus have tree-like structures. You can step up or down the pathways to reach your destination, or invoke the menu system's up-arrow jump feature as a shortcut. To jump to an option, enter an up-arrow before the option specification (the option's menu text or synonym in upper or lowercase letters). You only need to use the first few characters needed to uniquely identify the option. You can use the option's synonym to limit ambiguity, especially if the synonym is distinct from other synonyms or menu texts.

```
Select Systems Manager Menu Option: ^Intro <RET> ductory text edit
```

The menu system carries out the necessary footwork to reach the desired option. If, along the way, there are pathway restrictions, such as locks or prohibited times, access to the option will be denied, just as when stepping to an option. If a match is found within the primary or secondary menus, that option is executed (the menu system will not search the Common menu if it can find a match in the primary or secondary menus).

If the menu system finds *more than one* matching option on *either* the primary, secondary, or Common menu tree, the menu system presents a list of matching choices. Entering an up-arrow followed by a question mark will display all of the options available to you.

```
Select Systems Manager Menu Option: ^List Names<RET>

    1    List Namespaces  [XUZ NAMESPACES]
    2    List Namespaces  [ZZ NAMESPACE LIST]

Type '^' to stop, or choose a number from 1 to 2 :
```

IRM should assign "shallow" secondary menus to facilitate menu jumping. When a jump is requested, the menu system searches all the way through the primary as well as the secondary, looking for a match. Users will be inconvenienced and system resources will be consumed if secondary menus are "deep" in terms of their hierarchical tree-like structure.

You may occasionally find jumping disabled; when you try to jump, you may get a message that quick access is temporarily disabled. Jumping will stay disabled until the needed menu trees are rebuilt.

The Rubber Band Jump

The menu system's jump feature includes the ability to jump out to a destination option and then back again, something like the motion of a rubber band. The syntax for the rubber band jump request is the use of a double up-arrow followed by the usual option specification. For example:

```
Select Systems Manager Menu Option: ^^TASKMAN USER <RET>
```

As with the single up-arrow jump, restrictions along the menu pathways are checked.

If you enter two up-arrows without a following option, you are returned to the primary menu. This technique is a quick way for you to "go home" to the menu that is displayed at sign-on, and is called the go home jump.

It is important to note that when you invoke the rubber band jump, there is no attempt to protect variables that may be set or killed, via Entry or Exit Actions, as you jump through the menu tree. The rubber band jump may thus be inappropriate under certain circumstances since it could cause significant alteration of your environment.

The Common Menu

The Common menu is designed as a collection of options that are available to all users. The standard Common menu items are:

- **User's Toolbox: As described in the Sign-On/Security: User Interface chapter, the User's Toolbox is a menu containing options that allow you to control some aspects of your computing environment.**
- **Halt, Continue, Restart Session: As described in the Sign-on: User Interface chapter, these options are three different ways to log out of the system.**
- **View Alerts: As described in the Alerts chapter and the Sign-on: User Interface chapter, View Alerts is an option that lets you process Alerts.**
- **Time: The Time option simply displays the date and time.**
- **Where am I?: This option lists information identifying what computer system you are signed into (e.g., UCI, Volume Set, Node, and Device).**

Selecting Common Options with the Double Quote

Since Common options are intended to be readily accessible, there is a shortcut method to reach them. While you could use an up-arrow jump, it is quicker to enter a quotation mark followed by the option specification. The following example selects the User's Toolbox option from the Common menu via its synonym, TBOX:

```
Select Sample Menu Option: "TBOX <RET>

          Display User Characteristics
          Edit User Characteristics
          Electronic Signature code Edit
          Menu Templates ...
          Spooler Menu ...
          TaskMan User
          User Help

Select User's Toolbox Option:
```

Menu Templates

Menu templates are like scripts. You can use them to execute a fixed series of options, in sequence. Tools for creating, deleting, listing, and renaming templates are options on the Menu Templates menu, part of the User's Toolbox (TBOX) menu:

```
Select Menu Templates Option: ? <RET>

          Create a new menu template
          Delete a Menu Template
          List all Menu Templates
          Rename a menu template
          Show all options in a Menu Template

Select Menu Templates Option:
```

When you create a menu template, you are prompted for a series of options that lead to a final non-menu (i.e., executable) destination option. Once you choose one non-menu option to be executed, you can navigate to other options and choose them to be executed as well, if you wish. When you have selected each executable option to be part of the template, enter a plus sign (+) to store the sequence of options. You will be asked to confirm the sequence of options in the template, and then to give the template a name.

To invoke the template, simply enter a left square bracket followed by the template name:

```
Select Option: [mytemplate <RET>

Loading MYTEMPLATE...
```

The template will then execute each option that is part of the template, in the same order as the options were selected for the template.

Menu templates are stored in the MENU TEMPLATE multiple of the NEW PERSON file, so you can use any name for menu templates. If your menu template points to options that are subsequently removed from the OPTION file, you receive a message that the menu template will no longer function properly and needs to be deleted and/or rebuilt.

Use menu jumping (the up-arrow jump) when you want to jump immediately to an option. Use menu templates when you have a series of options that you will need run in the same order repeatedly, over a period of time.

LOGIN Menu Template

Beginning with Kernel V. 8.0, you can have a menu template execute automatically, on your first sign-on of the day. If you have a menu template named LOGIN (all uppercase), the menu template will be executed on your first sign-on of the day. So if you have a series of options you execute on your first sign-on every day, an easy way to execute them is to create a menu template; store the series of options in the template; and name the template LOGIN.

Summary

Once you learn how to navigate Kernel's menu tree, you can use some of Menu Manager's additional features to help increase your productivity in the DHCP computer system. These features include the up-arrow jump, the rubber-band jump, using three question marks to obtain online option help, and using menu templates as scripts.

Chapter 7 Menu Manager: System Management

Menu Manager is built around options, which are entries in the OPTION file (#19). There are several types of options. One type of option, Menus, has subentries in the MENU (item) multiple, a multiple that points back to the OPTION file itself. Other types of options are designed as items that plug into the MENU (item) multiple of a menu-type option.

Kernel provides a number of tools to create and manage menus and options.

Kernel's Menus

Most of the options exported with a package are tied to a parent option, or master menu, as a collection point. The Kernel exports three menu tree "roots." The Systems Manager menu [EVE] is the master menu for IRM. The Common menu [XUCOMMAND] is linked through the display function of the menu system rather than the OPTION file. The Kernel also exports a menu that is a miscellaneous collection of options that should not normally be invoked by the interactive user; it is the Parent of Queuable Options [ZTMQUEUABLE OPTIONS]. Most of the options on this menu should be scheduled to run as TaskMan jobs.

■ Kernel Menu Tree Roots

```
SYSTEMS MANAGER MENU ...                                    [EVE]
SYSTEM COMMAND OPTIONS ...                              [XUCOMMAND]
PARENT OF QUEUABLE OPTIONS ...                [ZTMQUEUABLE OPTIONS]
```

Creating Menus and Options

```
SYSTEMS MANAGER MENU ...                                          [EVE]
  Menu Management ...                                           [XUMAINT]
    Edit options                                               [XUEDITOPT]
```

One task IRM performs frequently is defining local primary menus that are appropriate for their users. This task of menu creation is accomplished by grouping exported menus from various packages together on a new master menu. You can use Edit options, on the Menu Management menu, to define a new menu if READ, WRITE, and LAYGO access to the OPTION file has been granted (either through File Manager access code, or through the File Access Security system if that is enabled). Only a few fields need to be defined, as shown below. The new menu may then be assigned to a user, as described in the Sign-on chapter, with one of several options on the User Edit menu.

```
Select OPTION to edit: ZZSTAFF MENU <RET>
  Located in the Z (Local) namespace.
  ARE YOU ADDING 'ZZSTAFF MENU' AS A NEW OPTION (THE 721ST)? Y <RET> (YES)
   OPTION MENU TEXT: Staff Menu <RET>
NAME: ZZSTAFF MENU// <RET>
MENU TEXT: Staff Menu// <RET>
PACKAGE: <RET>
OUT OF ORDER MESSAGE: <RET>
LOCK: <RET>
REVERSE/NEGATIVE LOCK: <RET>
DESCRIPTION:
  1>This is the primary menu for staff members. <RET>
  2><RET>
EDIT Option: <RET>
TYPE: menu <RET>
Select ITEM: XUCORE <RET>    Core Applications
  ARE YOU ADDING 'XUCORE' AS A NEW MENU (THE 1ST FOR THIS OPTION)? Y <RET>
(YES)
   MENU SYNONYM: <RET>
  SYNONYM: <RET>
  DISPLAY ORDER: 10 <RET>
Select ITEM: XUSPY <RET>   System Security
  ARE YOU ADDING 'XUSPY' AS A NEW MENU (THE 2ND FOR THIS OPTION)? Y <RET>
(YES)
   MENU SYNONYM: <RET>
  SYNONYM: <RET>
  DISPLAY ORDER: 20 <RET>
Select ITEM: XT-KERMIT MENU <RET>    Kermit menu
  ARE YOU ADDING 'XT-KERMIT MENU' AS A NEW MENU (THE 3RD FOR THIS OPTION)?
YES <RET> (YES)
   MENU SYNONYM: <RET>
  SYNONYM: <RET>
  DISPLAY ORDER: 30 <RET>
Select ITEM: <RET>
CREATOR: SITE,MANAGER// <RET>
HELP FRAME: <RET>
PRIORITY: <RET>
Select TIMES PROHIBITED: <RET>
Select TIME PERIOD: <RET>
RESTRICT DEVICES?: <RET>
Select PERMITTED DEVICE: <RET>
```

Option Name and Menu Text: By convention, the formal option name is usually entered in all capital letters. According to namespacing conventions, it must begin with a namespace that identifies the associated package. It is the .01 field of the OPTION file. The menu text is what is displayed to the user at the select prompt. Like the words of a heading or title, initial capitalization is used for all words except prepositions and articles, all of which are presented in lowercase. To minimize the number of keystrokes needed to select an option, different first letters should be used for the text of each menu item. Menus should be limited to about seven items so they will all appear together on one screen. The most frequently used items should be presented first.

Synonyms and Display Order: By default, the items on the menu are displayed in alphabetical order by menu text. If any of the items is assigned a synonym, those items will be displayed before others lacking synonyms. To facilitate menu jumping, synonyms should ideally be unique; numbers are not good choices for synonyms.

To customize the order of the display, each item on the menu can be assigned a Display Order. This field is an option attribute that will be presented when using Edit options. When first assigning a number for the display order, you may want to use 10, 20, and 30 rather than 1, 2, and 3 to permit easier modification in the future if another item needs to be inserted.

PRIORITY: You can set an option's PRIORITY field to set a run priority for an option. Experimentation will be needed to determine the effect of priority settings.

HELP FRAME: You can specify a help frame for an option. The help frame is displayed if, at the "Select..." menu prompt, the user enters ?OPTION (where OPTION is the name of an option).

DISPLAY OPTION Field: If auto-menu is in effect for a user, the items on that user's current menu are always displayed. A problem can arise when, if an option displays output and then quits, auto-menu's automatic display of menu options scrolls the output off the screen. Since the auto-menu display usually scrolls the option's output off the screen faster than the user can read the output, it can effectively render the option unusable. You can avoid this problem by setting the option's DISPLAY OPTION field to YES. If set to YES and the user has auto menu turned on, Menu Manager will ask "Press RETURN to continue..." after the option completes, but before displaying the list of menu options. The user will then have a chance to review the output before returning to their menu.

Other OPTION File Fields: For more information on fields in the OPTION file, including how to create options of a type other than Menu, please see the Menu Manager: Programmer Tools chapter.

If the Option Invokes Non-DHCP Applications

If you create an option that invokes non-DHCP applications, such as WordMan or CalcMan, include a call to the Device Handler with the code D HOME^%ZIS in the Exit Action field of the OPTION file so that the required IO variables will be present when leaving these options. Do the same for any other utility that is known to kill IO variables upon exit.

If the Option Should Be Regularly Scheduled

If an option should be regularly scheduled to run through TaskMan, you should set its SCHEDULING RECOMMENDED field (field #209 in the OPTION file) to YES. You will not be able to use Schedule/Unschedule Options to schedule an option unless this field is set to YES for the option.

Auditing Option Use

```
SYSTEM MANAGER MENU...                                          [EVE]
  System Security...                                          [XUSPY]
    Audit Features ...                                  [XUAUDIT MENU]
      Maintain System Audit Options ...                [XUAUDIT MAINT]
        Establish System Audit Parameters                     [XUAUDIT]
        Audited Options Purge                               [XUOPTPURGE]
      Audit Display ...                                        [XUADISP]
        Option Audit Display                                [XUOPTDISP]
```

You can establish an audit on options to record every time an option is used. You can do this with the Establish System Audit Parameters option, which is in the Audit Features [XUAUDIT MENU] menu tree. Simply enter a time to initiate audit and a time to terminate audit. Then enter the specific options you want to audit (you can also choose all options).

Each time a user uses an audited option, an entry is made in the AUDIT LOG FOR OPTIONS file (#19.081). You can display these entries using the Option Audit Display option [XUOPTDISP]. You can purge the AUDIT LOG FOR OPTIONS file with the Audited Options Purge option [XUOPTPURGE].

If Kernel Toolkit is installed at your site, you can also use its Alpha/Beta Test Option Usage menu to count the number of times an option is invoked. For more information, see the Kernel Toolkit documentation.

For more information on auditing, please see the *Kernel Security Tools Manual*.

Displaying Menus and Options

```
SYSTEMS MANAGER MENU ...                                          [EVE]
  Menu Management ...                                          [XUMAINT]
    List Options by Parents and Use                             [XUXREF]
    Display Menus and Options                        [XQDISPLAY OPTIONS]
      Abbreviated Menu Diagrams                             [XUUSERACC2]
      Diagram Menus                                          [XUUSERACC]
      Inquire                                                [XUINQUIRE]
      Menu Diagrams (with Entry/Exit Actions)               [XUUSERACC1]
      Print Option File                                        [XUPRINT]
```

Kernel provides a number of options to display and diagram menus and options.

Diagramming Options

To discover the menu tree roots of other packages and how options and sub-options are related, you can use the following menu diagramming options:

Abbreviated Menu Diagrams	**Outlines the menu tree.**
Diagram Menus	**Outlines the menu tree, and shows option attributes such as locks and prohibited times.**
Menu Diagrams (with Entry/Exit Actions)	**Outlines the menu tree, shows option attributes, and shows entry/exit and header actions as well.**

Also, the List Options by Parents and Use option identifies which options have "no parents" and are thus stand-alone roots. It also indicates whether options are used as primary menus, secondary menus, and/or as regularly scheduled tasks.

Option Descriptions

To learn more about the options included in a package, you can use Print Option File (from the Display Menus and Options menu) to print the option description, type, and other information. This listing can be sorted by namespace. To print all the VA FileMan options, you can sort from DD to DI.

Displaying Options

To display an option, use the Inquire option:

```
Select Display Menus and Options Option:  Inquire <RET>
|
Which OPTIONS item to display: XT-KERMIT MENU <RET>     Kermit menu
|
NAME: XT-KERMIT MENU                    MENU TEXT: Kermit menu
  TYPE: menu                            CREATOR: POSTMASTER
  PACKAGE: KERNEL                       E ACTION PRESENT: YES
  X ACTION PRESENT: YES
 DESCRIPTION:   This is the top level menu for kermit functions.  It
gives access to the send, receive, and edit options.
ITEM: XT-KERMIT RECEIVE                 SYNONYM: R
ITEM: XT-KERMIT SEND                    SYNONYM: S
ITEM: XT-KERMIT EDIT                    SYNONYM: E
  EXIT ACTION: D CLEAN^XTKERM4          ENTRY ACTION: D INIT^XTKERM4
  UPPERCASE MENU TEXT: KERMIT MENU
```

Option Access by User

```
Menu Management ...                                          [XUMAINT]
  Show Users with Selected Primary Menu                     [XUXREF-2]
  Option Access By User                                     [XUOPTWHO]
```

To show which users have been assigned a particular option as a primary or secondary menu, the Show Users with Selected Primary Menu option can be invoked. Option Access by User is another cross referencing tool.

Managing Menus and Options

Managing Primary Menus

When IRM receives new packages, existing primary menus should be modified to include the new menus. It is not wise to create a new primary menu for every new or unusual circumstance. This would lead to a tremendous variety of menus that would be difficult to sort out and use in the future. Primary menus can be customized with security keys (see the Security Key section). If there are a few menu options that require special privilege, they can be locked and the keys assigned to the appropriate users. In this way, a smaller number of primary menus can serve the needs of a larger number of users.

Also, while putting new master menus onto users' secondary menus can be a quick fix, it isn't a good idea to do this. Too many options on a user's secondary menu can be cumbersome for the user. In addition, in the long run, it is easier for IRM to manage access to a menu reached from a few well-defined primary menus than to manage access to a menu reached from a large number of users' secondary menus.

Assigning Secondary Menus

An easy way to allocate menu options is to assign them to users individually as secondary menu options. Secondary options are unique for each user and are stored in a multiple in the user's NEW PERSON file entry. Assignment of secondary options should be limited to the essential few, and should not involve deep structures with multiple levels. Instead, new primary menus should be built or existing ones modified. During menu jumping, all branches of both the primary and secondary menu trees are searched each time a jump request is received by the menu system. Greater efficiency and user convenience will result if the depth of the secondary menu trees is confined.

The ALWAYS SHOW SECONDARIES Field

You can set the ALWAYS SHOW SECONDARIES field in a user's NEW PERSON file entry. If set to YES for a user, that user will always have their secondary and common options listed when options on their primary menu are listed (which occurs either by the user entering two question marks at the "Select..." menu prompt, or when auto-menu is turned on).

Redefining the Common Menu

All users automatically have access to the Common options (the [XUCOMMAND] menu) by virtue of the menu system's design. As described earlier, entering two question marks at any select prompt will display the Common menu. The only way to deny access to a particular user is to lock the Common option with a reverse key (see the Security Key section) and then allocate the key to the same user.

The items on the Common menu may be left as they are distributed by the Kernel, or modified locally as desired. For example, an item may be added to display on-line help about local computer access policies. This is accomplished by using Edit options to edit the XUCOMMAND option. The Item multiple lists the existing menu choices; other locally namespaced options may be added.

If options are locally added to the standard XUCOMMAND option set, new installations of the Kernel will not overwrite the changes. During installation, items on the local XUCOMMAND option are compared with the exported items. Any previously exported items that were removed by the site will not be added back. Brand new items, however, will be added and any matching items will be updated. Other items that the site may have added will be left in place.

Altering Exported Menus

Generally speaking, exported menu structures should stay intact. If local modifications to exported menus are made, great care must be taken to preserve any logic that may exist in the exported structure. For example, the entry action of one option may set up key variables that are then assumed to exist when another option, one further down on the menu tree, is invoked. Although each one of a package's options should be able to be invoked independently once the steps described in the Technical Manual for creating and killing package-wide variables have been taken (according to the Programming Standards and Conventions (SAC)), this is not always the case and cannot be assumed.

If an option cannot be invoked independently, the developer can set that option's INDEPENDENTLY INVOCABLE field to NO, as an alert that some other option or action must be done before the option can be called.

To give users the options associated with new packages, IRM should try to allocate the menus as whole entities. If dissection appears necessary, the Internal Relations section of the package documentation should be consulted before rearranging any of the items.

Deleting Unreferenced Options

```
Programmer Options ...  <locked:  XUPROG>                    [XUPROG]
  Delete Unreferenced Options                      [XQ UNREF'D OPTIONS]
```

All options for interactive use (not designed exclusively as queuable tasks) should normally be tied to a menu that is used as a primary menu or at least as a secondary menu. Stand-alone options that have no parents and are not menu-type options should be reviewed. They may be obsolete package options or local test options and could be candidates for deletion. The option to delete unreferenced options can be used to cycle through the entire OPTION file and delete non-menu options that are not referenced by other options. Deletion should obviously be done with care. Use of the option is limited to those who hold the XUPROG key.

Fix Option File Pointers

```
Menu Management ...                                          [XUMAINT]
  Fix Option File Pointers                                  [XQOPTFIX]
```

After performing maintenance work on the OPTION file, such as deleting obsolete options that may have been items on a menu, you can use the Fix Option File Pointers (shown below) to remove any dangling pointers that may have been left in the Item multiple. Running this option is an alternative to having VA FileMan update the pointers each time an individual option is deleted.

```
Select OPTION NAME: ZZTEST3 <RET>          Test Option
NAME: ZZTEST3// @ <RET>
   SURE YOU WANT TO DELETE THE ENTIRE 'ZZTEST3' OPTION? Y<RET> (YES)
SINCE THE DELETED ENTRY MAY HAVE BEEN 'POINTED TO'
BY ENTRIES IN THE 'USER' FILE, ETC.,
DO YOU WANT THOSE POINTERS UPDATED (WHICH COULD TAKE QUITE A WHILE)?
NO// <RET>
```

Testing a User's Menus

```
User Management...                                             [XUSER]
  Switch Identities                                       [XUTESTUSER]
```

You can test a user's menus using the Switch Identities option. It lets you test the user's menus and keys. It doesn't allow you to execute any bottom-level menu options, however; it only lets you navigate menu trees. You are reminded at each prompt whose menu it is that you are testing. To exit this mode and return to your own menus, simply enter an asterisk ("*").

Managing Out-Of-Order Option Sets

```
Menu Management ...                                          [XUMAINT]
  Out-Of-Order Set Management...                            [XQOOMAIN]
    Create a Set of Options To Mark Out-Of-Order            [XQOOMAKE]
    List Defined Option Sets                                [XQOOSHOW]
    Mark Option Set Out-Of-Order                              [XQOOFF]
    Options in the Option File that are Out-of-Order      [XQOOSHOFIL]
    Protocols Marked Out-of-Order in Protocol File        [XQOOSHOPRO]
    Recover Deleted Option Set                              [XQOOREDO]
    Remove Out-Of-Order Messages from a Set of Options         [XQOON]
    Toggle options/protocols on and off                     [XQOOTOG]
```

Menu Manager, starting with Kernel V. 8.0, provides a mechanism for defining sets of options and protocols, and a way to disable and enable access for these pre-defined option and protocol sets. This can be handy when you need to repeatedly disable and enable sets of options and protocols.

Use the Create a Set of Options to Mark Out-Of-Order option to define a set of options. You are prompted first to select options, and then to select protocols. For both options and protocols, you can use the wildcard asterisk with or without a namespace to add a group of options to the set; the - sign followed by a namespace to remove options from the set; and NAM1-NAM2 to add a range of options from NAM1 to NAM2 to the set.

Use Mark Option Set Out-Of-Order to disable access to a set of options. You are asked to enter the message used to place all options in the set out-of-order. The option then places the message in each option's OUT OF ORDER MESSAGE field. Use Remove Out-Of-Order Messages from a Set of Options to enable access to an option set. To toggle the status of an individual option only, use Toggle Options/Protocols On and Off.

Out-of-Order Option sets are stored in ^XTMP, with a purge date set for seven days in the future. If you place a set of options out of order, but the option set is purged from ^XTMP before you enable access to it, you can rebuild the out-of-order option set using Recover Deleted Option Set. It asks you to specify the exact text of the message used to place the set of options out of order; it then recreates an out-of-order option set containing all options currently placed out of order with the specified message (note: make sure the message you specify is unique to the set of options you are re-enabling). You can then enable access to the rebuilt option set with Remove Out-Of-Order Messages from a Set of Options.

To see what sets of options have been grouped in sets on the system, use List the Defined Options Sets. To show all options and protocols currently marked out of order, use Options in the Option File that are Out-of-Order and Protocols Marked Out-of-Order in Protocol File.

Restricting Option Usage

```
Menu Management ...                                          [XUMAINT]
  Restrict Availability of Options                         [XQRESTRICT]
```

Options can be restricted in terms of when users may select them and when devices may be used to invoke them. Many of the option restrictions are included in Restrict Availability of Options.

OUT OF ORDER MESSAGE: To completely restrict access, you can mark an option to be out-of-order. Do this by entering text in an option's OUT OF ORDER MESSAGE field. If a user attempts to invoke the option, the Out of Order Message will be displayed.

Locks: Both the normal lock, and also the Reverse/Negative lock can be associated with options (as described in the Security Key chapter). Also, M code can be entered in the header, entry action, or exit action fields to restrict the use of an option given certain conditions.

Prohibited Times: You can prohibit the use of an option at certain times during the day by assigning a set of prohibited time periods at the Select TIMES PROHIBITED prompt.

Permitted Devices: If the RESTRICT DEVICES flag is set to YES, the option can only be invoked on one of the devices listed in the PERMITTED DEVICES multiple. The running of an option may thus be restricted. This flag does not affect the choice of devices used for the output from options. It instead controls the processing involved in the use of the option itself.

Queuing Required Flag: Using Edit Options, you can allow users to invoke an option, but force any output to be queued outside of certain times of day, by editing the option's QUEUING REQUIRED multiple field. In the .01 and .02 fields of the multiple, TIME PERIOD and DAY(S) FOR TIME PERIOD, enter the time periods and days in which you do not want the option's output to be produced. During these time periods, the output of the options can only be queued. When a user requests a time for queuing, the menu system will determine the next permissible day and time for output. Users may thus invoke the option and use it to define the parameters for the subsequent processing, but the actual work will be done during a later time period, presumably when the system is less busy.

Menu Manager Options that Should Be Scheduled

This section describes the two Menu Manager options that should be regularly scheduled.

The Kernel exports a number of other options that should be scheduled to run at regular intervals. Most of these are located on the Parent of Queuable Options menu. The *Kernel Installation Guide* contains a complete list, along with suggested scheduling frequencies.

Clean Old Job Nodes in XUTL

The Clean old Job Nodes in XUTL option (XQ XUTL $J NODES) is Kernel's purge option for Kernel globals. As well as ^XUTL, this option also purges the ^UTILITY, ^TMP, ^XTMP, and ^XUSEC globals.

```
Operations Management ...                                  [XUSITEMGR]
  Clean old Job Nodes in XUTL                        [XQ XUTL $J NODES]
```

User stacks for each user's job are stored in the ^XUTL global (see "The ^XUTL Global: Structure and Form" later in this chapter.) This is also called the compiled menu system. If a job ends abnormally, such as upon error, UCI switching, or programmer exits that bypass ^XUS, the entries remain in the global (this explains why programmers are advised to halt out of programmer mode with D ^XUSCLEAN rather than simply halting.)

The purge routine sets a purge date of seven days in the past. Any user stack in ^XUTL older than seven days is purged. Any entries with a matching $J at the top level of ^UTILITY and ^TMP are also killed.

Next, after cleaning out the user stacks in ^XUTL, the purge routine checks ^UTILITY and ^TMP. Any entry at subscript ($J) or (namespace, $J) that doesn't have a matching entry in the user stacks in ^XUTL is killed.

Next, the purge routine checks ^XTMP. Any entry in ^XTMP at subscript (namespace) lacking a header node at (namespace,0), or with a purge date in the header node less than the purge date determined by the purge routine is killed.

Finally, the purge routine goes through the sign-on nodes stored at ^XUSEC(0,"CUR",DUZ,DATE). Any nodes older than the purge date are killed.

The XQ XUTL $J NODES option should be queued to run on a regular basis. If separate copies of ^XUTL are maintained on different CPUs, separate entries should be made in the OPTION SCHEDULING file for each CPU so that a separate job will purge each CPU's XUTL global. Because this option

deletes any user stacks that are time-stamped with a date earlier than the purge date determined by this option (seven days) you need to take care how frequently you schedule it (in the unusual event of a seven-day long job, this option should obviously not be run).

Also, at MSM-DOS sites, the purge option only works correctly if ^XUTL, ^UTILITY, and ^TMP are either common to all CPUs, or have separate copies on all CPUs. If any mixing of common copies versus separate copies of for these globals at MSM-DOS sites, this option may not work correctly.

Rebuilding Primary Menu Trees

```
PARENT OF QUEUABLE OPTIONS                          [ZTMQUEUABLE OPTIONS]
  Non-interactive Build Primary Menu Trees               [XQBUILDTREEQUE]
  Copy the compiled menus from the print server     [XU-486 MENU COPY]
Menu Management ...                                            [XUMAINT]
  Build Primary Menu Trees                                  [XQBUILDTREE]
```

The menu system uses local menu trees to process requests. When changes are made to the menu structure, the local menu trees are rebuilt (a process also known as microsurgery). If a user attempts an up-arrow jump when the local trees need to be rebuilt or are being rebuilt, a message is issued about quick access being temporarily disabled; the user will not be able to jump to reach the option. Microsurgery is triggered in the following situations:

- **The Edit Options option is used.**
- **An Out-of-Order option set is enabled or disabled.**
- **A sufficiently large number of changes have been made to a menu tree.**

It is also recommended to rebuild all primary menu trees every other day during non-peak hours, using the XQBUILDTREEQUE option. If separate copies of ^XUTL are maintained on different CPUs, separate entries should be made in the OPTION SCHEDULING file for each CPU so that a separate job will rebuild each CPU's ^XUTL global.

For MSM-DOS sites (which usually run TaskMan only on the Print Server), an alternative to running the full menu rebuild on every CPU is to run the XU-486 MENU COPY option. It should be scheduled to run after the menu tree rebuild has finished on the Print Server. As distributed, this option only copies the compiled menus to a UCI named VAH on Compute Servers named CSA, CSB, etc. It uses VA FileMan's %RCR routine to copy the data.

Primary menu trees may also be built/repaired immediately using the Build Primary Menu Trees option. In particular, if menu jumping has stopped working and microsurgery is not fixing the menus, use Build Primary Menu Trees to force a menu rebuild to fix the problem.

Error Messages During Menu Jumping

There are some conditions under which a menu jump may not be completed. In these cases the user will see one of the following error messages:

```
I NEED TO REBUILD MENUS .... QUICK ACCESS IS TEMPORARILY DISABLED
Please proceed to {target option's menu text}
```

This means that the time stamps on the OPTION file and the ^XUTL global indicate that the OPTION file has been modified since the menus were compiled in ^XUTL and the global is therefore locked until XQ8 can recompile the modified menus. This error message can be generated by both user-generated jumps and phantom jumps.

```
*** WARNING ***
Illegal jump requested to option '{option's menu text}'  Jump pathway
locked at option '{locked option's menu text}'
```

This indicates that a locked option for which the user does not possess the key has been encountered in the tree between the option where the jump was requested and the target option to which the jump was requested. This error message can be generated by both user-generated jumps and phantom jumps.

```
*** WARNING ***
Illegal jump was requested to option '{option menu text}'  Jump path
out of order from '{option's menu text}'  with message '{out of order
message}'
```

This means that an option on the tree between the option where the phantom jump was requested and the target option has been marked as out of order (Field #2 of the OPTION file). This error message can be generated by both user-generated jumps and phantom jumps.

```
*** WARNING ***
Illegal jump was requested to option '{option menu text}'  Variable
XQUIT encountered at option '{option name}'
```

This means that the jump logic has encountered the variable XQUIT (detected with a $DATA statement). This variable is usually set by an Entry Action (Field # 20 of the OPTION file) and causes the menu system to refuse to run or jump past that option. This error message can be generated by both user-generated jumps and phantom jumps.

```
*** WARNING ***
Background jump requested to option '{value in XQMM("J")}'  but this
option does not exist on your system.
```

A VA FileMan lookup was attempted for the option set in the variable XQMM("J") but no such option was found in the OPTION file. This error message can only be generated from a phantom jump.

```
*** WARNING ***
Background jump requested to option '{option's menu text}' but  you
do not have access to this option.  See your computer
representative.
```

This means that the target option requested by XQMM("J") is not in the tree of options to which this user has access (that is, the target option was neither in the user's primary menu tree nor specifically listed as a secondary menu for that user). This error message can only be generated from a phantom jump.

For more information on phantom jumps, please see the Menu Manager: Programmer Tools chapter.

The ^XUTL Global: Structure and Function

The ^XUTL global is an account-specific global. It should exist in each production account on your system. This global is created primarily from information in the OPTION file [^DIC(19)] and is therefore sometimes referred to as "*the compiled menu system*."

^XUTL is divided into 3 main sections: the user stacks, the display nodes, and the jump nodes:

`^XUTL("XQ",$J)`	**User stacks**
`^XUTL("XQT",$J)`	**User stacks (menu templates only)**
`^XUTL("XQO",ien)`	**Display nodes**
`^XUTL("XQO","P"_ien)`	**Jump nodes**

User Stacks

User stacks are stored in nodes in ^XUTL("XQ",$J) and ^XUTL("XQT",$J).

The example on the next page shows a typical user stack. In this case the $J is 541065826.

The "XQ" nodes may be divided into meaningful sets according to what is contained in the third subscript. The numeric third subscripts begin with the zero node which is set to the date and time in VA FileMan format by the program ^XUS1 when the user logs on or ^%XUCI when the user is changing UCIs.

The other numeric, third subscripts (in this case the numbers 1 to 3) reflect the user's progression through the menu system. Each time a new option is invoked, a new node is created which contains the option number, concatenated with a "P", the number of the option whose compiled menu tree contains the current option, an up arrow, and the zero-node of the OPTION file for that option. A different format is used for options in a user's secondary menu tree.

A pointer in the node ^XUTL("XQ", $J, "T") indicates which option in this list of numbered nodes the menu driver is currently using. This pointer is set and reset by the menu driver as the user moves up and down the menu tree. In the example, XUPROGMODE is the option that the menu driver is currently using.

Other "XQ" nodes of the global which have a non-numeric third subscript are used to store various pieces of Kernel information which are set up at sign-on. ^XUTL("XQ",$J,"XQM") points to the user's primary menu. In the example, the user's primary menu is OPTION file entry #29.

■ User Stack Example

```
^XUTL("XQ",541065826,0) = 2920113.081624
^XUTL("XQ",541065826,1) = 29P29^EVE^Systems Manager
                          Menu^^M^.5^^192^^^^^^n^1^^^
^XUTL("XQ",541065826,2) = 31P29^XUPROG^Programmer Options^^M^^
                          XUPROG^^^^^^^n^^
^XUTL("XQ",541065826,3) = 49P29^XUPROGMODE^Programmer mode^^R
                          ^^XUPROGMODE^^^^^^^ n^^
^XUTL("XQ",541065826,"DUZ") = 63
^XUTL("XQ",541065826,"DUZ(0)") = LlPp
^XUTL("XQ",541065826,"DUZ(2)") = 16000
^XUTL("XQ",541065826,"IO") = _LTA5103:
^XUTL("XQ",541065826,"IOBS") = $C(8)
^XUTL("XQ",541065826,"IOF") = #,$C(27,91,50,74,27,91,72)
^XUTL("XQ",541065826,"IOM") = 79
^XUTL("XQ",541065826,"ION") = LAT DEVICE
^XUTL("XQ",541065826,"IOS") = 158
^XUTL("XQ",541065826,"IOSL") = 24
^XUTL("XQ",541065826,"IOST") = C-VT100HIGH
^XUTL("XQ",541065826,"IOST(0)") = 149
^XUTL("XQ",541065826,"IOT") = VTRM
^XUTL("XQ",541065826,"IOXY") = W $C(27,91)_((DY+1))_$C(59)_((DX+1))_$C(72)
^XUTL("XQ",541065826,"T") = 3
^XUTL("XQ",541065826,"XQM") = 29
```

XQT Nodes (Menu Templates)

The "XQT" nodes are used to create a stack of options similar to the "XQ" stack when a menu template is invoked. These nodes are translated from the subfile ^VA(200,DUZ,19.8) when a user precedes an option selection with a left square bracket character, "[", much like a print template is invoked in FileMan. For example, if the user has defined a menu template named "DOIT" using the Menu Template options of the User's Tool Box, typing "[DOIT" will load that sequence of options into the "XQT" nodes and begin executing them. When a menu template is requested by the user, the option tree of that template is loaded into the "XQT" nodes and remains loaded as long as the user is logged on. Further requests for "[DOIT" will use that same stack.

Display Nodes

Display nodes are stored in ^XUTL("XQO", internal number).

The first example on the following page shows the display nodes for EVE, the System Manager's Menu. The internal number of EVE in this particular OPTION file is 29. In the first part of the example the option names and menu texts, along with a limited number of fields for that option compiled from the OPTION file, are concatenated together. It is from this part that XQ2 (the menu display program) gets the information it needs.

In the second part, all the menu texts and synonyms are listed in order in upper case. It is here that XQ tries to match what the user entered at the terminal with the correct option. The third part of the example, the 0th node of the options, is listed by number and provides the remaining information that the Menu System may need to make the option work. To understand what the various "^" pieces mean, look at a VA FileMan global format data dictionary listing of the OPTION file.

Illustrated in the second example on the following page is the display node for the secondary menus of a user whose DUZ is equal to 66. Here, the user has only a single secondary menu called "Secondary Menu" (with an internal number of 580 in the OPTION file). The various parts of this example are identical to those of the Display Nodes for the EVE example above. Note that the second subscript, instead of pointing to a menu in the OPTION file, is a "U" concatenated with the user's DUZ which points to the NEW PERSON file entry. This is because secondary menu options are stored in the NEW PERSON file entry for each user.

Display Nodes for EVE

```
^XUTL("XQO",29,0) = 2^55048,38923
^XUTL("XQO",29,0,1) = ^XUCORE^Core Applications ...^NOT
                          AVAILABLE^^^^^^XUTIO^Device Handler
                          ...^^^^n^^FM^DIUSER^VA FileMan ...^^^^n^^^XMMGR^
                          Manage Mailman ...^^^^^^^XUMAINT^Menu Management
                          ...^^^^n^^^XUPROG^Programmer Options ...^^XUPROG^^^
                          ...^
^XUTL("XQO",29,0,2) = ^XUSITEMGR^Operations Management ...^^^^^^^XU-SPL-MGR
                          ^Spool Management ...^^^^^^^XUSPY^System Security
                          ...^^^^^^^ZTMMGR^Task Manager ...^^^^n^^^XUSER^User
                          Edit ...^^^^^^
^XUTL("XQO",29,"CORE APPLICATIONS") = 40^1
^XUTL("XQO",29,"DEVICE HANDLER") = 32^1
^XUTL("XQO",29,"FM") = 19^0
^XUTL("XQO",29,"MANAGE MAILMAN") = 30^1
^XUTL("XQO",29,"MENU MANAGEMENT") = 9^1
^XUTL("XQO",29,"OPERATIONS MANAGEMENT") = 174^1
^XUTL("XQO",29,"PROGRAMMER OPTIONS") = 31^1
^XUTL("XQO",29,"SPOOL MANAGEMENT") = 415^1
^XUTL("XQO",29,"SYSTEM SECURITY") = 226^1
^XUTL("XQO",29,"TASK MANAGER") = 83^1
^XUTL("XQO",29,"USER EDIT") = 39^1
^XUTL("XQO",29,"VA FILEMAN") = 19^1
^XUTL("XQO",29,"^",9) = ^XUMAINT^Menu Management^^M^^^105^^^n^n^^n^^^^
^XUTL("XQO",29,"^",19) = FM^DIUSER^VA FileMan^^M^^^^^^n^^^n^1^^
^XUTL("XQO",29,"^",30) = ^XMMGR^Manage Mailman^^M^^^299^^^^^54^^1^1^^^
^XUTL("XQO",29,"^",31) = ^XUPROG^Programmer Options^^M^^XUPROG^^^^^^^n^^
^XUTL("XQO",29,"^",32) = ^XUTIO^Device Handler^^M^^^413^^^n^^20^n^^
^XUTL("XQO",29,"^",39) = ^XUSER^User Edit^^M^^^153^^^^^^n^^
^XUTL("XQO",29,"^",40) = ^XUCORE^Core Applications^1^M^^^^^^^^^n^^
^XUTL("XQO",29,"^",83) = ^ZTMMGR^Task Manager^^M^^^^^^n^^50^^1^^
^XUTL("XQO",29,"^",174) = ^XUSITEMGR^Operations Management^^M^^^^^^^y^^n^^
^XUTL("XQO",29,"^",226) = ^XUSPY^System Security^^M^^^^^^^^119^n^^
^XUTL("XQO",29,"^",415) = ^XU-SPL-MGR^Spool Management^^M^^^419^^^^^20^^
```

Display Nodes for a Secondary Menu

```
^XUTL("XQO","U66",0) = 1^54927,30758
^XUTL("XQO","U66",0,1) = ^ZZTSTSM^Secondary Menu ...^^^^n^^
^XUTL("XQO","U66","SECONDARY  MENU") = 580^1
^XUTL("XQO","U66","^",580) = ^ZZTSTSM^Secondary Menu^^M^^^^^^n^^^^1^1^^1
```

Jump Nodes

Jump nodes are stored in ^XUTL("XQO","P"_internal number), where there is one "P_..." entry in ^XUTL("XQO") for each primary menu that exists. The jump nodes, for each primary menu, store the pathways to all options that can be jumped to.

The jump nodes are created in the XQ8* series of programs. They are very similar to display nodes, except that (1) they have a "P" concatenated on the front of the primary option's number in the second subscript, and (2) these nodes describe the entire primary menu tree rather than just the single level tree.

Examples of the jump nodes for a single primary menu are shown on the following page. Since these nodes can be very extensive in number, some nodes have been removed from the example to save space.

In the first example are the "lookup" nodes, where the jump software tries to match a menu text or synonym with what the user has entered at the terminal. Each node is set to its internal number in the OPTION file and, in the second "^" piece, a 0 if it is a synonym or a 1 if it is menu text.

In the second example, the "menu pathway" entries below the "P580" node show all of the options that can be jumped to from the primary menu whose internal entry number is 580. Each entry contains list(s) of the series of options that must be navigated through in a jump from the primary menu. In the case of the option DILIST (# 17), the list of options that will have to be processed is 520,519,518,411,17. If, as in the case of ZZTEST4 (# 318), there is more than one possible pathway, then each is listed along with various other necessary pieces of information, such as locks, time restraint, etc.

Jump Nodes: Lookup Nodes

```
^XUTL("XQO","P580",0) = 55165,28536
^XUTL("XQO","P580","19^") = 394^0
^XUTL("XQO","P580","2ND SECOND LEVEL MENU TEST^") = 575^1
^XUTL("XQO","P580","3^") = 518^0
^XUTL("XQO","P580","ACTN^") = 391^0
^XUTL("XQO","P580","ALL^") = 420^0
```

Jump Nodes: Menu Pathways

```
^XUTL("XQO","P580","LIST FILE ATTRIBUTES^") = 17^1
^XUTL("XQO","P580","TEST 4^") = 318^1
...
^XUTL("XQO","P580","TOOL^") = 581^0
^XUTL("XQO","P580","X-TYPE OPTION TEST^") = 576^1
^XUTL("XQO","P580","X^") = 576^0
^XUTL("XQO","P580","ZDAVE^") = 411^1
^XUTL("XQO","P580","^",5) = ^XUEDITOPT^Edit
                            options^^E^581,5,^^106^^^^^20^n^^^^
^XUTL("XQO","P580","^",17) = ^DILIST^List File Attributes^^A^
                            520,519,518,411,17,^^^^^n,^y^^n^1^^^
...
^XUTL("XQO","P580","^",318) = ^ZZTEST4^Test
                            4^^O^520,575,397,318,^^^^^n,^^^^^^
^XUTL("XQO","P580","^",318,0) = 2
^XUTL("XQO","P580","^",318,0,1) = 520,575,578,397,318,^^^n,^
^XUTL("XQO","P580","^",318,0,2) = 520,575,578,318,^^^n,^
...
^XUTL("XQO","P580","^",579) = ^ZZLEVEL3B^Phantom
                            Mother^^M^520,575,579,^^^^^n,^^^^1^1^^1
^XUTL("XQO","P580","^",580) = ^ZZTSTPM^Primary
                            Menu^^M^^^^^^n^^^^1^1^^1
^XUTL("XQO","P580","^",581) = ^ZZLUKTOOLS^Luke's
                            Tools^^M^581,^^^^^^^^^1^1^^1
```

Menu Manager Variables (Troubleshooting)

There is a group of Menu Manager variables that is always defined. It may be useful for IRM staff to know what these variables signify when investigating errors. If an error is reported in VA FileMan's DIP routine, for example, knowing the value of XQY at the time of the error indicates which option was invoking the DIP routine. The option can then be reviewed to discover the name of the routine that was calling DIP.

XQABTST Flag that signals whether alpha-beta testing is in effect.

XQDIC Internal entry number (ien) of the option's parent (which must be a menu) in the OPTION file, if an option is executing. If the user is in a menu, XQDIC is set to the ien of the current menu's parent (unless they are in their primary menu, in which case XQDIC is set to the ien of the primary menu).

The value of XQDIC also corresponds to the second subscript in the display nodes portion of the ^XUTL global, ^XUTL("XQO",) for the menu in question.

XQPSM Like XQDIC, a look-up value into the second subscript of ^XUTL, the compiled menu global. XQPSM points to the tree of the target option in the jump. It resulted from the ability to jump to any option, not just ones on the primary menu tree. It can help identify jumps from a primary, secondary, or Common option.

XQT Current option's type (e.g., M for menu, A for action).

XQUR User's response to the menu prompt (replaces A).

XQUSER User's name in the form JAMES L. AUSTIN.

XQY Internal entry number of the current option or menu (replaces Y).

XQY0 First node (subscript of zero) of the current option (replaces Y(0)).

XQXFLG Contains several flags, including whether capacity management testing is active.

Chapter 8 Menu Manager: Programmer Tools

Creating Options

You can develop application packages quickly and easily using Menu Manager. Once you have defined a set of files using VA FileMan, you can use Menu Manager to provide a menu of options including entering, editing, displaying, and printing information. You can use M code to tailor the functioning of an option, in the option's header, entry, or exit action. You can create specialized routine-type options. And you can associate help frames with options (as described in the Help Processor chapter) to further enhance option creation and custom tailoring.

Option Types

Several different option types exist:

- **Edit, Inquire, and Print are mainly used to access VA FileMan files.**
- **Action and Run Routine types are available for invoking M code.**
- **Menu types, as discussed earlier in this section, are used to group other options for presentation to the user at the select prompt.**
- **Servers are options that can be addressed through MailMan (sending to S.SERVER NAME). The server activity, such as the running of a routine, is then carried out. For a complete description, see the Server chapter later in this Menu Manager section.**
- **Protocol, Protocol Menu, Extended Action, and Limited option types are specific to the XQOR (Unwinder) package. Control is passed to the XQOR (Unwinder) software for processing. The Extended Action type, for example, "unwinds" the items on a menu in a specific order. Protocol Menus are formatted in multiple columns allowing several items to be selected at once. The Protocol-type option prompts the user for a selection. Limited protocols involve patient-oriented processing, rather than application-specific tasks. Any of these option types are included, like other options, when a package is exported. See the OE/RR or Unwinder (XQOR) documentation for more information.**

Creating Options (Edit Options)

```
MENU MANAGEMENT...                                          [XUMAINT]
   Edit options                                           [XUEDITOPT]
```

You can define options with the Edit Options template, available from the Menu Management menu. Depending on what type of option you are editing, the Edit Options template branches to the fields in the OPTION file appropriate for that option type.

Some option types (Edit, Inquire, and Print) have fields whose names correspond to VA FileMan DI variables. The Edit Options template branches to the DI fields that have relevance to the type of VA FileMan call being made by the option.

For Edit type options, the DI fields presented correspond to the input variables for an ^DIE call. Likewise, inquire-type options correspond to ^DIQ calls, and print options to ^DIP calls. See the VA FileMan Programmer Manual for a complete description of the meaning of the variables represented by each of the DI fields.

Options that Should Be Regularly Scheduled

If an option should be regularly scheduled to run through TaskMan, you should set its SCHEDULING RECOMMENDED field (field #209 in the OPTION file) to YES. Sites will not be able to use Schedule/Unschedule Options to schedule an option unless this field is set to YES for the option.

Variables for Programmer Use

The appearance and functioning of the menu system can be modified by programmers by using several variables. The variables may be defined within application packages, such as in an option's Entry Action, Exit Action, or Header. These variables are listed below.

The XQMM variables may be used individually or together. It is strongly recommended that you test the effects of XQMM variables with the auto-menu display, DUZ("AUTO"), turned on and off.

XQUIT: Quit the Option

This variable may be set in an option's Entry Action to cause Menu Manager to quit and not invoke the option. The menu system will not run the option, either as a foreground job or background task, and will not jump past the option. If an option's use depends on the existence of certain application-specific key variables, for example, the Entry Action logic can set XQUIT if those variables are not defined. Menu Manager simply checks for the existence of the XQUIT variable, so it can be set to null (S XQUIT="") or to a value as the programmer chooses.

XQMM("A"): Menu Prompt

If XQMM("A") exists, it is used as the prompt by the menu system instead of the normal "Select...option" menu prompt. This variable is killed immediately after it is used. It does not inhibit the auto-menu display. If the user has chosen to have options displayed at each cycle of the menu system, then the options will be displayed *before* the XQMM("A") prompt is presented. Unlike the phantom jump, prompts must be set singularly, and may not be concatenated with a semicolon.

XQMM("B"): Default Response

If XQMM("B") is defined, it is used by the menu system as the default response and is presented along with the usual two slashes (//). If the user accepts the default by pressing <RET>, the default will become the user's response.

XQMM("B") identifies an option if set to a unique synonym or a unique string of text from the beginning of the option's menu text. This option must exist on the user's current menu. If the option cannot be found, Menu Manager will respond with two question marks (??), kill both XQMM("A") and XQMM("B"), and display the standard menu prompt.

XQMM("J"): The Phantom Jump

This variable may be used to force a menu jump to an option within the user's menu tree. Set it equal to the exact option name (.01 field of the OPTION file) to which Menu Manager should jump. For example:

```
S XQMM("J")="XUMAINT"
```

would jump to the Menu Management option if that option is within the user's menu tree.

The phantom jump automatically turns off the user's menu display for one cycle through the menu system so that the user does not see a list of choices before jumping to an option that is not on that list.

The phantom jump may also be used to designate a set of options for a series of jumps, called a script. The exact option names should be separated with semicolons. For example:

```
S XQMM("J")="XUMAINT;DIUSER"
```

After jumping to Menu Management, the menu system would jump to VA FileMan (provided that all of the access and security requirements are met).

After all the options in a script have been completed, the phantom jump logic returns the user to the option that was last run before the script was invoked. If for some reason this cannot be accomplished, the user is returned to their primary menu.

XQMM("N"): No Menu Display

This variable may be used to suppress the "auto-menu" display of menu options for one menu cycle. XQMM("N") is then killed and the display resumes as usual. XQMM("N") may be used in conjunction with XQMM("A") and ("B") to present only the custom tailored menu prompts.

Setting XQMM("N") does not change the display for users who already suppress the auto-menu display. For users who have auto-menu turned on, XQMM("N") takes precedence over DUZ("AUTO").

It is not necessary to define XQMM("N") when using the phantom jump, XQMM("J"), since the display will already be suppressed. If XQMM("J") is present, then XQMM("N") will not be killed after the first cycle since the phantom jump is already inhibiting the display. In this case, XQMM("N") will be killed after the second cycle (the display of menus after the jump is completed). If several phantom jumps are chained together, XQMM("N") will not be killed until one cycle after the final jump unless code is added to explicitly kill it between jumps.

Direct Mode Utilities

>D ^XQ1: Test an Option

^XQ1 asks you to select an option; it then uses the selected option as the primary menu option for entry into the menu system (at the top of ^XQ). This provides a way for an individual in programmer mode to enter into the menu system at a desired option.

This entry point is also called by ^XUP. Programmers are advised to use ^XUP instead of ^XQ1 to enter the Kernel from programmer mode since the ^XUP routine sets up a standard environment and takes care of cleanup activities. The ^XUP direct mode utility is described in the Sign-On/Security: Programmer Tools chapter.

Note that while D ^XQ1 is a direct mode utility, it is not a callable entry point.

Callable Entry Points

• $$ADD^XPDMENU: Add Option to Menu

Usage `S X=$$ADD^XPDMENU(menu,option,[syn],[order])`

Input

menu:	**Name of the menu to add an option to.**
option:	**Name of the option being added to the menu.**
syn:	**[optional] Synonym to add to the SYNONYM field in the new menu item.**
order:	**[optional] Order to place in the DISPLAY ORDER field in the new menu item.**

Output

return value:	**1 if operation succeeded, 0 if operation failed.**

Description

Use this entry point to add an option as a new item to an existing menu.

• OUT^XPDMENU: Edit Option's Out of Order Message

Usage `D OUT^XPDMENU(option,text)`

Input

option:	**Name of option in which to place an OUT OF ORDER MESSAGE value.**
text:	**Text of message to place in option's OUT OF ORDER MESSAGE field.** **If this is not null, the text is stored in the option's OUT OF ORDER MESSAGE field and the option is placed out of order.** **If this parameter is passed as a null string, the current OUT OF ORDER MESSAGE value is deleted, and the option is put back in order.**

Output **none**

Description

Use this entry point to create or delete an out of order message for an option; this action effectively puts the option out of order or back in order.

• RENAME^XPDMENU: Rename Option

Usage `D RENAME^XPDMENU(old,new)`

Input

	old:	**Current option name (.01 field of OPTION file entry). Must be an exact match.**
	new:	**New name for option.**

Output **none**

Description

Use this entry point to rename an existing option.

• NEXT^XQ92: Restricted Times Check

Usage `D NEXT^XQ92`

Input

	Y:	**Internal entry number of the option in the OPTION file.**

Output

	X:	**The date/time in VA FileMan format of the next unrestricted runtime when the option can run. If the option is able to run at the current time, X is returned as the current time. If the option is prohibited for the entire next week, X is returned as null and a message is issued regarding the time restriction.**

Description

NEXT^XQ92 returns the next time an option can run, checking any time or date restrictions placed on the option. If there are no times in the next week when the option can be run, X is returned as null and a message is issued regarding the time restriction.

• OP^XQCHK: Current Option Check

Usage D OP^XQCHK

Input **none**

Output **XQOPT:** **String, in the following format:**

Option/Protocol Name^Menu Text

If neither an option nor a protocol can be identified, XQOPT is returned as:

"-1^Unknown"

Description

The OP^XQCHK routine returns the current option or protocol name and menu text in the first and second pieces of the output variable XQOPT. It looks for the local variable XQY, the internal number of the option, or XQORNOD, the internal number of the protocol.

If the search is unsuccessful, because the job is not running out of the menu system or is not a tasked option, XQOPT is returned with -1 in the first piece and "Unknown" in the second. Note that XQCHK cannot return option/protocol information if the job is a task that did not originate from an option.

• ^XQDATE: Current Date/Time

Usage D ^XQDATE

Input **none**

Output **%:** **Current date and time, in VA FileMan format.**

%Y: **Current date and time, in human readable format.**

Description

The ^XQDATE entry point returns the current date/time in VA FileMan format in %, and in human readable format (e.g., Jan. 9, 1990 1:37 PM) in %Y.

Chapter 9 Security Keys

User Interface

Security keys are primarily used to allow access to specially protected options. If a package exports a menu that has one or two options that require a secured level of access, they may use keys to lock those special options. When an option is locked, you can only use the locked option if you hold the key matching the key the option was locked with.

Entering Double question marks at the menu system's select prompt displays the current options. If any of the options are locked, that fact is listed also, along with the names of any associated security keys. In the following example, the option Programmer Options is locked with a security key named XUPROG:

```
Select Systems Manager Menu Option: ?? <RET>
          Device Handler ... [XUTIO]
          Menu Management ... [XUMAINT]
          Programmer Options ... [XUPROG]
             **> Locked with XUPROG
```

You can list which keys you currently hold by using the Display User Characteristics option on the Common menu. It displays a list of all keys you hold, similar to the following:

```
KEYS HELD
---------
   XUPROG          XUMGR          XUPROGMODE      XUAUTHOR        ZTMQ
```

The keys you need to carry out computing activities should be assigned by IRM when your computer account is first added to the system. Other keys may be allocated at a later time by IRM or by an IRM designee, such as an application coordinator, with use of the Secure Menu Delegation utilities.

System Management

Identifying Locked Options

IRM can list which keys lock what options by using Menu Management's Diagram Menus option. The following example shows that the Programmer Options menu is locked with the XUPROG key. It also shows that one of its options, Programmer Mode, is locked with the XUPROGMODE key:

```
Select Menu Management Option:  Diagram Menus <RET>
Select USER (U.xxxxx) or OPTION (O.xxxxx) name: O.XUPROG <RET>
Programmer Options (XUPROG)
**LOCKED: XUPROG**
-------------------------PG Programmer mode
                  [XUPROGMODE]
                    **LOCKED: XUPROGMODE**
```

Security keys are stored in the SECURITY KEY file (#19.1). Keys given to users are stored in the users' NEW PERSON file (#200) entries, in the KEYS multiple.

Options are locked by a given key when the name of that key is entered into the LOCK field of the OPTION file. If an option is locked, users need to be given the key in order to invoke the option.

Key Management

Keys are defined and allocated to users with options on the Key Management menu.

```
SYSTEMS MANAGER MENU ...                                          [EVE]
  Menu Management ...                                           [XUMAINT]
     Key Management ...                                        [XUKEYMGMT]
       Allocation of Security Keys                              [XUKEYALL]
       De-allocation of Security Keys                         [XUKEYDEALL]
       Enter/Edit of Security Keys                              [XUKEYEDIT]
       All keys a user needs                                      [XQLOCK1]
       Change user's allocated keys to delegated keys  [XQKEYALTODEL]
       Keys for a given menu tree                                 [XQLOCK2]
       Delegate keys                                             [XQKEYDEL]
       List users holding a certain key                          [XQSHOKEY]
       Remove delegated keys                                    [XQKEYRDEL]
       Show the keys of a particular user                       [XQLISTKEY]
```

Allocating and De-Allocating Keys

The main option to assign security keys to a user or users is the Allocation of Security Keys option. Allocating a key to a user lets them invoke options that are locked with the key. For options with reverse locks, allocating the key locks the user out from the option. In either case, allocating the key to a user does not allow the user to give the key to anyone else.

To remove a key from a user, use the De-Allocation of Security Keys option.

Unless you have been delegated a key (see below), the only way you can allocate or de-allocate keys is if you hold the XUMGR key or have a VA FileMan access code of "@".

All of the keys that a new user needs to use their assigned options can be determined by using the option All Keys a User Needs on the Key Management menu. This produces a list of the primary and secondary menus for that user, and compiles a list of the keys for that menu tree. This list can then be assigned or delegated. It may also be edited before the keys are given to the user. Similarly, the option Keys For a Given Menu Tree will examine a menu and list all of the keys associated with all sibling options.

Delegating Keys

Delegating keys allows you to give a user the ability to assign specific keys to other users (as opposed to the XUMGR key and "@" VA FileMan access code, which allow all keys to be assigned).

One way to delegate keys is to use the Change user's allocated keys to delegated keys option. This option delegates to a user all of the keys that are currently allocated to that user. Any entries in their KEYS multiple are entered in the DELEGATED KEYS multiple as well. They may now use the Allocation option to give the keys to others.

Alternatively, IRM can use the Delegate keys option to populate the DELEGATED KEYS multiple one-by-one.

A user who has been delegated a key may allocate that key to others in two ways:

- **Through the Allocation of Security Keys option, if it is on their menu**
- **By delegating an option locked by the key in question; the key will be allocated along with the option.**

The key recipients (excepting holders of the XUMGR key or a VA FileMan access code of "@") cannot assign the key to others, however, even if they have

access to the Allocation option, because the key does not exist in their DELEGATED KEY multiple.

One example of key delegation is an IRM designee, delegated the Provider key, who allocates that key to incoming medical residents.

For security reasons, users who have a key in their delegated key multiple may not allocate that key to themselves. That key must be awarded by another user who has been delegated the key or by an IRM staff member who holds the XUMGR system key.

Creating and Editing Security Keys

Keys may be created using the Enter/Edit option on the Key Management menu. If a key has already been defined, its name cannot be edited. It also cannot be deleted, as discussed below. Other key attributes stored in the SECURITY KEY file (#19.1) may be used for special purposes. Attributes of the Provider key are shown in the following example:

```
Select SECURITY KEY NAME: PROVIDER <RET>
NAME: PROVIDER// <RET>  (No Editing)
DESCRIPTIVE NAME: Provider// <RET>
PERSON LOOKUP: LOOKUP// <RET>
KEEP AT TERMINATE: YES// <RET>
DESCRIPTION:
  1>This KEY is given to all entries in the New Person file that need
  2>to be looked up as a Provider. Those entries that hold this key
  3>are considered to be providers.  It was given to all active
  4>Providers in file 6 at the time of the Kernel 7 install.
EDIT Option: <RET>
Select SUBORDINATE KEY: <RET>
GRANTING CONDITION: <RET>
```

PERSON LOOKUP: As described in the Programmer API section that follows, a special AK cross-reference on File #200 is maintained automatically for anyone who is granted a key that is flagged for Person Lookup. This cross-reference has been introduced to facilitate identification of user groups, like providers.

KEEP AT TERMINATE: As described in the Sign-On/Security section concerning user deactivation, keys that are marked as "Keep at Terminate" will not be removed as a user attribute of terminated users. This allows the continued processing of activities that had been previously authorized because the user held the key.

SUBORDINATE KEY (Exploding Keys): If a key has any associated subordinate keys (entries in this multiple), the subordinate keys are automatically assigned along with the overall key. A key with this feature is

called an exploding key since it and its subordinates are assigned all at once. Note that if entries in the subordinate key multiple are edited, dynamic updating of the keys already assigned to users does not occur.
Exploding keys may not be exported with an application package (although there may be support for this functionality in the future). They are intended to be created by IRM as a time-saving method in the key allocation process.

Deleting Security Keys

Keys should not be deleted from the SECURITY KEY file. Kernel has made the .01 field of the SECURITY KEY uneditable to prevent deletion of keys through VA FileMan. IRM should not attempt to edit the key global directly to remove a key since associated pointing relationships will be left to cause errors. The one mechanism Kernel does provide for deletion of security keys is through KIDS (see the KIDS Programmer Tools: Creating Builds chapter for more information).

Reindexing All Users' Security Keys

```
SYSTEMS MANAGER MENU ...                                        [EVE]
User Management ...                                           [XUSER]
   Manage User File ...                                 [XUSER FILE MGR]
      Reindex the users key's                        [XUSER KEY RE-INDEX]
```

You can use this option to reindex all users' keys in the NEW PERSON file. If a user has a key, but is lacking the corresponding ^XUSEC cross-reference for the key, you can use this option to regenerate the ^XUSEC cross-reference. While the ^XUSEC cross-reference is being rebuilt, there can be an impact on all users with key lookups failing in ^XUSEC until the index is entirely rebuilt; therefore, this option should be used with caution and is best delayed until users are not signed on.

Using Keys with Reverse Locks

If a key is associated with an option via the REVERSE/NEGATIVE LOCK field, rather than the LOCK field, it functions to lock out users who hold the key. The key used for a reverse lock is just like any other key, differing only in the way it is associated with an option. Menu Management's Diagram Menus option indicates the existence of any reverse locks, such as the use of the XMNOPRIV key to prevent access to MailMan's shared mail facility.

The typical use of a key with the REVERSE/NEGATIVE LOCK field is to restrict access to options otherwise available to all users, like MailMan User and other options on the Common menu.

Key Delegation Levels

Starting with Kernel V. 8.0, keys are subject to delegation levels just as options are subject to delegation levels. A field in the NEW PERSON file, DELEGATION LEVEL, stores a user's delegation level (for keys and options). When a key is delegated, the person to whom it is delegated is assigned a level one number lower than the delegation level of the person doing the delegating. This is to prevent the delegated-to person from removing delegated keys from someone with a lower delegation level. For more information about delegation levels, see the Secure Menu Delegation chapter.

Programmer Tools

As well as locking options, programmers can use security keys within options if some part of an option requires special security. One example of this is Kernel's use of the ZTMQ key; it restricts functionality within the Dequeue Task, Requeue Tasks, and Delete Tasks options.

Key Lookup

When writing code that checks whether the current user holds a certain key, do not reference the SECURITY KEY file (#19.1) for this information. Instead, check the ^XUSEC global. The most efficient check is:

```
I $D(^XUSEC(keyname,DUZ))
```

which is (and will continue to be) a supported reference. The ^XUSEC global is built by a cross reference on File #19.1.

Person Lookup

If a key is flagged for Person Lookup, a cross-reference on File #200 will be built and maintained to facilitate programmer calls. It is constructed with the letters "AK" before the key name. The Provider key is exported with the Person Lookup flag set; as a result, providers may be easily identified in this AK.keyname cross reference, at ^VA(200,"AK.PROVIDER",DUZ). Specifically, the lookup would be:

```
S DIC="^VA(200,",DIC(Ø)="AEQ",D="AK.PROVIDER" D IX^DIC
```

Callable Entry Points

• $$RENAME^XPDKEY: Rename Security Key

Usage `S X=$$RENAME^XPDKEY(oldname,newname)`

Input

oldname:	**Name of security key to be renamed.**
newname:	**New name for security key.**

Output

return value:	**1 if operation successful, 0 if operation failed.**

Description

Use this entry point to rename a security key. All necessary indexing is performed to maintain the ^XUSEC global.

Chapter 10 Secure Menu Delegation

The job of allocating menu options to users can be a time-consuming activity, so site managers may want to consider delegating this responsibility to application coordinators. Application coordinators are familiar with the menus for their package and can learn how to assign these to new users in their service area.

Secure Menu Delegation allows the Site Manager to delegate the management of certain menu options to another user, such as an application coordinator. This user, now a delegate, may then assign these as primary or secondary options (along with their keys) to users who fall under their administrative jurisdiction.

For example, the Site Manager might delegate the management of the laboratory options to the Lab Application Coordinator (LAC), and the LAC could then allocate or remove options from everybody in lab. The system is set up in such a way that the LAC could also delegate, with the Site Manager's permission and manager's menu, the management of all the chemistry menus to the head of the Chemistry Section, and so on, creating another level of delegation.

There are two divisions in Secure Menu Delegation:

- **The menu to create and manage delegates.**
- **The menu for the delegates themselves to assign options to end users.**

User Interface: Acting as a Delegate

As a delegate, you have been delegated options (usually by IRM). If you have been delegated options, you can assign these options to computer users on the computer system.

As a delegate, you can assign the following options to your users:

- **Options that have been delegated to you.**
- **Menus that you have created from options delegated to you.**
- **Options you have created from VA FileMan templates.**

As a delegate, you need to understand the basic structure of the OPTION file, which is a file that points back to itself. That is, a menu is an entry in the OPTION file; but items on menus are themselves pointers to other entries in the OPTION file. You should also understand the difference between types of options, be familiar with menu trees, and be sufficiently reluctant to assign great numbers of secondary menus.

Delegate's Menu

To delegate options to users, you need to be assigned a menu called Delegate's Menu Management. The options on this menu are as follows:

```
Delegate's Menu Management                              [XQSMD USER MENU]
  Build a New Menu                                    [XQSMD BUILD MENU]
  Edit a User's Options                             [XQSMD EDIT OPTIONS]
  Copy Everything About an Option to a New Option             [XQCOPYOP]
  Copy One Users Menus and Keys to others             [XQSMD COPY USER]
  Limited File Manager Options (Build)      [XQSMD LIMITED FM OPTIONS]
```

Each of these options on the delegate's menu is discussed below.

Edit a User's Options

Using this option allows you to edit a user's primary and secondary menus. This is the chief method you can use to add (and subtract) options on your users' menus.

Most of your work will be in adding and deleting options on your users' secondary menus. You are only able to add or delete options from a user's secondary menu if the option in question has been delegated to you. That means that you don't have access to a user's entire secondary menu; instead, only those options on the secondary menu that are also delegated to you.

If, when you edit a user's secondary menu, you choose an option that is already on a user's secondary menu, you are asked if you want to delete it from their secondary menu. Otherwise, you are asked if you want to add the option to their secondary menu.

If you are assigning an option that is locked with a key, the delegation process checks whether you have been delegated the key as well. If you have, the key is automatically assigned to the user along with the option. If you have not been delegated the key, you get an error message saying that you haven't been delegated the needed key (the option is assigned to the user, but they won't have the key to unlock the option).

If you delete an option that is locked with a key and that key is delegated to you (and you are at a higher key delegation level than the option holder), the key is deleted along with the option (unless the user holds another option locked by the same key).

Example

In this example, the menu option LRZ MAIN is added to the user's secondary menu. Note that LRZ MAIN is locked with a key and that the key is automatically assigned when the option is assigned:

```
Select Delegate's Menu Management Option: EDit a User's Options <RET>

Select NEW PERSON NAME: SMITH,MARY <RET>
     PRIMARY MENU OPTION: XMUSER// <RET>      MailMan Menu  .
     No keys needed to delete!.
     No keys needed to give!

     SECONDARY MENU OPTION: LRZ MAIN <RET>    Lab User Menu  ...
     ZZLRMAIN key also given!

     SECONDARY MENU OPTION: <RET>

Select NEW PERSON NAME:
```

Unlike secondary menus, you are only able to edit a user's primary menu if their current primary menu is an option that has been delegated to you. Otherwise, you are not allowed to change that user's primary menu. Note that you can't add or subtract options on a user's primary menu; you can only replace the user's entire primary menu with another one.

Build a New Menu

Using this option on the Delegate's Menu Management menu, you can create new menus, with menu items chosen from your delegated options.

First, you need to provide an option name for the new menu you are creating. The menu option name must begin with a namespace assigned to you by IRM. Once you provide a name for the menu, you are asked:

- **Text for the menu.**
- **Description for the menu.**
- **Items for the menu (choose from your delegated options).**

Once you have created a new menu, you can assign it to your users just as if it were an option delegated to you.

Copy Everything About an Option to a New Option

Using this option, you can copy any option on the computer system into a new option. First you are asked which existing option you would like to copy; then, you are asked for a name for the copied option. The option name must begin with a namespace assigned to you by IRM.

Copy One Users Menus and Keys to Others

Using this option, you can copy the menus and keys of one user to another user. Each menu or key you copy, however, must have been delegated to you; otherwise, they are skipped in the copy process. What gets copied from one user into the other user are:

- **Primary menu (and all descendant menus).**
- **Secondary menu items.**
- **Keys.**

The primary menu of the user you're copying from replaces the primary menu of the user you are copying to. The secondary menu items and the keys

of the user you're copying from are merged into the secondary menu items and the keys of the user you're copying to.

Limited File Manager Options (Build)

The Secure Menu Delegation system provides a way for delegates to create options out of VA FileMan templates. Delegates who have enough access to VA FileMan to create input, sort, or print templates can create menu options for their users that directly call these templates.

Characteristics of Intended Users

The Limited File Manager Options (Build) tool is designed for delegates, such as some application coordinators, who have VA FileMan access to a set of files and can create input, sort, or print templates. These delegates may have the VA FileMan options for editing or printing without the ability to modify data dictionaries. They may also have explicit file access to a specified set of files via the File Access Management system. Typically they would be working without the special FileMan Access Code, DUZ(0).

IRM Setup to Enable Building Options from Templates

To allow a user to create menu options from VA FileMan templates, IRM must first assign to the user:

- Delegate's Menu Management menu [XQSMD USER MENU].
- XQSMDFM key.
- A namespace in which to create options. Do this with the secure menu delegation option Specify Allowable New Menu Prefix [XQSMD SET PREFIX]. This forces any options created by the user to be namespaced with the assigned namespace, appended with a Z, appended with an option name.

Building Options

The tool for building options with VA FileMan templates is called Limited File Manager Options (Build). It is part of the Delegate's Menu Management under the Secure Menu Management menu and is locked with the XQSMDFM key.

First, you must have created a sort, print, or input template for a VA FileMan file. Once you have created a template, you can make this template available as an option to your users by turning it into an option.

You can create three types of options:

- **Edit-type option (from an edit template).**
- **Print-type option (from print and sort templates).**
- **Inquire-type option (from either a print template or a file name).**

Once you have turned the template into an option, you can assign that option to your users as you deem necessary. Then, when a user uses the option, they execute the print, sort, or input template that the option was created from.

Example

Suppose you have created a print template called LRZ REFERRAL PRINT for the Lab's REFERRAL file. To turn this print template into an INQUIRE option, use the Limited File Manager Options (Build) option:

```
Select Delegate's Menu Management Option:  Limited File Manager Options
(Build) <RET>
The menu options you build or edit must begin with the namespace:
          LRZ


The option types that may be built are P(rint), E(dit), and I(nquire), and
you must have a template or templates ready to be included in the option.

Or enter D(elete) to DELETE an option


Select Option Type (P/E/I/D): I <RET>
     Enter Print Template Name (Optional): LRZ REFERRAL PRINT <RET>

     Option Name: LRZ REFERRAL INQUIRE <RET>
  Located in the LR (LAB SERVICE) namespace.
  ARE YOU ADDING 'LRZ REFERRAL INQUIRE' AS A NEW OPTION (THE 996TH)? Y <RET>
 (YES)
   OPTION MENU TEXT: Display a Referral <RET>
MENU TEXT: Display a Referral  Replace <RET>
DESCRIPTION:
  1> Display Lab Referral entries (option created by LAB ADPAC). <RET>
  2> <RET>
EDIT Option: <RET>

Select Delegate's Menu Management Option:
```

System Management: Managing Delegates

The options for creating and managing delegates are on the Secure Menu Delegation [XQSMD MGR] menu, which is on the Menu Management menu. Typically, IRM would be the sole holder of this menu. The options on this menu are:

Option Text	Function
Select Options to be Delegated	Delegate options
List Delegated Options and their Users	Print Report
Print All Delegates and their Options	Print Report
Remove Options Previously Delegated	Undo Delegation
Replicate or Replace a Delegate	Copy a Delegate
Show a Delegate's Options	Print Report
Delegate's Menu Management ...	Delegate's menu
Specify Allowable New Menu Prefix	Assign namespaces

The main options to create and manage delegates are:

- **Select Options to be Delegated**
- **Replicate or Replace a Delegate**

Delegating Options: Select Options to be Delegated

To delegate options, use the Select Options to be Delegated option from the Secure Menu Delegation menu. Using this option is a two-step process:

1. **Choose the users to delegate options to.**
2. **Choose which options to delegate to that group of users.**

You can choose to set up one user or many users as delegates. You can choose one option or a group of options to delegate to them.

You also need to assign (not delegate!) the Delegate's Menu Management [XQSMD USER MENU] option to the delegate; this menu gives delegates the means to assign delegated options to users.

■ Delegating Options

```
Select Secure Menu Delegation Option: SELE <RET> ct Options to be
Delegated

Enter the name(s) of your delegate(s), one at a time

 Name: SMITH,JOHN <RET>

 Name: JONES,SUSAN <RET>

 Name: <RET>

Enter options you wish to DELEGATE TO these users

Add option(s): XUINQUIRE <RET>

Add option(s): XUUSERACC <RET>

Add option(s): <RET>

For the following user(s):

1. SMITH,JOHN
2. JONES,SUSAN

You will delegate the following options:

XUINQUIRE   Inquire
XUUSERACC   Diagram Menus

 Delegated by SMITH,MARY on Jul. 21, 1994  3:55 PM.


Ready to delegate these options to these people? Y// <RET>

Request to add delegated options has been queued, task # 465,
     named: SMITH,MARY adding delegated options.
```

Delegating Keys

If options that you intend to delegate are locked with keys, you need to delegate the matching keys to the delegate. Otherwise, the delegate will not be able to assign keys to unlock options they've assigned to their users.

If the option is locked with a key that you possess, the Select Options to be Delegated option branches you to the Key Management program, and lets you allocate (if you so wish) the appropriate keys to the delegates you are creating.

However, to assign keys to users, the delegate must be delegated the key. To do that, you need to use the Key Management option, DELEGATE KEYS [XQKEYDEL]. This option allows you to delegate keys to delegates by

populating the DELEGATED KEYS multiple in their NEW PERSON file entry. Keys entered in a delegate's DELEGATED KEYS subfile allow them to allocate the entered keys to other users (but not themselves).

When a delegate assigns options to a user, they can assign the matching keys as part of that process. However, as an enhancement to a delegate's ability to work with keys, IRM can assign the delegate the following options from the Key Management menu:

- **Allocation of Security Keys**
- **De-allocation of Security Keys**
- **Show the Security Keys of a Particular User**

As long as the delegate does not hold the XUMGR key (which allows any key to be allocated), the Key Management options only allow delegates to allocate and de-allocate keys they've been delegated. Kernel also follows key delegation levels (new with Kernel V. 8.0) with the Allocation of Security Keys and De-allocation of Security Keys options.

Note: Key management options must be separately assigned; they are not a part of the Delegate's Menu Management [XQSMD USER MENU] option.

Delegation Level (Options and Keys)

DELEGATION LEVEL is a field in the NEW PERSON file specifying the number of steps that a person is from the original delegation of options by the Site Manager (whose Delegation Level is 0). Starting with Kernel V. 8.0, the delegation level is also maintained for delegated keys. For instance, if the Site Manager delegates all laboratory options to the Lab ADP Application Coordinator (ADPAC), then the Lab ADPAC would have a Delegation Level of 1. Should the Lab ADPAC further delegate a set of those options to the Chief of Chemistry, the Chief would have a level of 2, and so on.

The use of levels insures that supervision is not compromised such that the lower level user could alter menus or remove keys of the higher level person. No attempt is made to determine who actually works for whom since that information is not available to the software. Delegation chains should therefore be constructed with some care.

To modify the set of options (and accompanying keys) delegated to a particular person, you must have a Delegation Level equal to, or less than, the person you are trying to modify. If you create a new delegate by delegating some (or all) of the options delegated to you, that person will have a Delegation Level equal to your level +1.

It may be necessary to modify Delegation Levels using the VA FileMan as the organization's structure changes over time.

Further Delegation

The only way a delegate can delegate, rather than simply assign, options to someone else is if the delegate has access to the Select Options to be Delegated [XQSMD ADD] option, or the Replicate or Replace a Delegate [XQSMD REPLICATE] option. These options should only be on the Secure Menu Delegation [XQSMD MGR] menu. You should carefully evaluate whether to give this menu to delegates (which gives them the right to further delegate).

Options Too Sensitive to Delegate

Certain options, such as Programmer's Options, are considered too sensitive or powerful to be delegated. They are marked as not delegable in the OPTION file, and the Secure MenuMan Delegation software will not delegate these options. The traditional methods of assigning these menu options must be employed by the Site Manager.

It is highly recommended that the Site Manager, Security Officer, or IRM Chief review the options so marked and, using the VA FileMan, add any locally sensitive options to this list. It is the responsibility of each site to insure that the security of the system is not violated.

It should be noted that a higher level option, such as EVE would still give the delegate access to lower level options, such as XUMAINT even though XUMAINT is itself marked in the OPTION file as non-delegable. The Delegation software does not follow the option trees down to insure that options of options are not delegable.

Replicate or Replace a Delegate

You can copy the Delegated Options of a delegate to another user. Use the Replicate or Replace a Delegate option [XQSMD REPLICATE] to do this. The options that you transfer to another user do not replace any options the user has been previously delegated. They will be added to those options, if any. Like the Select Options to be Delegated option, this option also can branch you to the key allocation program for the new delegate.

You are also asked if the delegated options should be removed from the original delegate. If you say no ('N'), the original delegate remains a delegate. If you say yes ('Y'), all Delegated Options are removed from the original delegate, who will no longer be an active delegate. In order to remove the options from a delegate, however, you must have a Delegation Level lower than they do.

Remove Options Previously Delegated

To simply remove an option from a delegate's list of delegable options, use the Remove Options Previously Delegated option. First, enter the name or names of the delegate(s) you want to remove options from. Second, enter the option or options you want to remove from the specified set of delegates. You're given a chance to review the choices you made; if you say to proceed, a task is queued which removes the options you selected from the delegates you specified.

Specify Allowable New Menu Prefix

Use this option to assign allowable menu prefixes to your delegates. Your delegates need to be given allowable new menu prefixes if they:

- **Build new menus.**
- **Copy options.**
- **Create options from VA FileMan templates.**

Typically, if your delegate works with one particular package, you would assign them that package's namespace as an allowable prefix. Options that the delegate creates must then be prefixed with that namespace, appended with a Z.

If you don't specify an allowable prefix for a delegate, they will not be able to use the following options:

- **Build a New Menu**
- **Copy Everything About an Option to a New Option**
- **Limited File Manager Options (Build)**

You can specify multiple new menu prefixes for a given delegate.

Reports

You can use the following options to generate reports about delegates on your system:

- **List Delegated Options and their Users**
  (Sort by delegated option.)
- **Print All Delegates and their Options**
  (Sort by delegate name.)
- **Show a Delegate's Options**
  (Display all delegated options for one delegate.)

Chapter 11 Alerts

User Interface

When you receive an alert, something on the computer system is requesting your immediate attention. An application package might issue an alert to one or more users when certain conditions are met, such as depleted stock levels or abnormal lab test results, for example.

The first time you reach a menu prompt after receiving a particular alert, the alert's message is displayed to you by the menu system. The alert message is displayed along with a standard notice to select View Alerts on the Common menu to process the alert.

When you receive an alert, you should find out what the alert is asking of you, and attend to it. This is called processing the alert.

Until you process all unprocessed alerts you receive, you'll be reminded that you have pending alerts each time you're at a menu prompt. You will not, however, see the alert message (you only see that the first time you receive an alert and reach the menu prompt).

■ Example of Alert

```
Dr. You need to enter a progress note on 'JONES,WILLIAM'.
          Enter  "VA   VIEW ALERTS     to review alerts


Select Systems Manager Menu Option:
```

Processing Alerts

To process alerts, choose View Alerts from the Common menu. The View Alerts option presents a list of all pending alerts, numbered consecutively with the most recent alerts listed first.

Information-only alerts are displayed with the letter "I" in front of the alert message. When you process information-only alerts, all that happens is that they are removed from the pending alerts list. Their only purpose was to send you the one-line alert message.

When you process alerts that are not information-only, on the other hand, processing the alert may send you to a particular option or program. Afterwards, you are returned to the View Alerts screen if more alerts need processing, or back to the menu prompt if no pending alerts remain.

There are various methods for processing alerts from the View Alerts screen. You can enter:

- **I to process all information-only alerts.**
- **A to process all alerts.**
- **F to forward one or more specific alerts.**
- **P to print the pending alerts.**
- **M to list pending alerts in a mail message and deliver the message to your IN basket.**
- **^ to exit the alert processing screen.**
- **a single number to process a single alert.**
- **a range of numbers to process a range of alerts (e.g., 1,3,5-7).**

The Alert Handler ordinarily deletes alerts once you have processed the alert. If you have processed all pending alerts, and try to select the View Alerts option, nothing is displayed. View Alerts only offers a listing when there are pending alerts; if no alerts are pending, View Alerts just returns you to the menu prompt.

■ View Alerts Screen

```
ACCESS CODE:
VERIFY CODE:
Good evening  Jim  You last signed on Jan 9,1992 at 14:39

Dr. You need to enter a progress note on 'JONES,WILLIAM'.
          Enter  "VA   VIEW ALERTS     to review alerts

Select Clinic Manager Menu Option: "VA <RET>
 1.   Dr. You need to enter a progress note on 'JONES,WILLIAM'.
 2.   Alk Phos elevated, schedule fu bone scan
 3.I  For your information, meeting at 12 noon, room 223
          Select from 1 to 3
          or enter ?, A, I, F, P, M, R, or ^ to exit: ? <RET>

YOU MAY ENTER:
   A number in the range 1 to 3 to select specific alert(s)
      for processing.
   A to process all of the pending alerts in the order shown.
   I to process all of the INFORMATION ONLY alerts, if any, without further ado.
   F to forward one or more specific alerts.  Forwarding may be as an ALERT
to specific user(s) and/or mail group(s), or as a MAIL MESSAGE, or to a
specific PRINTER.
   P to print a copy of the pending alerts on a printer
   M to receive a MailMan message containing a copy of these pending alerts
   R to Redisplay the available alerts
   ^ to exit
```

Forwarding Alerts

Beginning with Kernel V. 8.0, you can forward alerts. You can choose one or more alerts and forward them in the following ways:

- **Forward as alert(s) to a specific user on the computer system.**
- **Forward as alert(s) to a mailgroup on the system.**
- **Copy into mail message(s) and send to users and mailgroups on the system.**
- **Print to an output device on the system.**

System Management

An alert notifies one or more users of a matter requiring immediate attention. Alerts thus function as brief notices that are distinct from mail messages or triggered bulletins.

Starting with version 8.0 of Kernel, alerts are stored in the ALERT file (#8992, stored in ^XTV(8992,). Another new file with Kernel V. 8.0, ALERT TRACKING (#8992.1, stored in ^XTV(8992.1,), provides a means to track alerts and users' responses to alerts. For each user to whom an alert is sent, the ALERT TRACKING file stores:

- **Alert name.**
- **Date created.**
- **Package identifier of alert.**
- **User who generated the alert.**
- **Message text of the alert.**
- **Action associated with the alert.**
- **Data associated with the alert.**

For each recipient of the alert, the ALERT TRACKING file stores:

- **First date and time observed (shown in menu cycle).**
- **First date and time selected for processing.**
- **Date and time processing completed (if any).**
- **Date and time alert was deleted.**
- **If alert was forwarded, user who forwarded, and date and time of forwarding.**

Two new functions, PATIENT^XQALERT and USER^XQALERT, provide access to information in the ALERT TRACKING file. These functions are described in the Programmer Tools section of this chapter.

Alert Management Menu

The Alert Management menu contains the following options, described below:

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Operations Management ...                                  [XUSITEMGR]
  Alert Management...                                      [XQALERT MGR]
    Delete Old (>14 d) Alerts                        [XQALERT DELETE OLD]
    Make an Alert on the fly                               [XQALERT MAKE]
    Purge Alerts for a User                    [XQALERT BY USER DELETE]
```

Alert Purging: Delete Old (>14 d) Alerts

The Delete Old (>14 d) Alerts option performs the following functions:

- **Purges unprocessed alerts from the ALERT file.**
- **Purges alert tracking information from the ALERT TRACKING file.**
- **Forwards unprocessed alerts to supervisors and/or surrogates.**

You can use Delete Old (>14 d) Alerts to purge all alerts that have been unprocessed for longer than a specified retention period (the default is 14 days.) It is assumed that an alert will become obsolete within this period and can be purged by IRM staff. This option also performs additional functions (described below).

This option can be run either directly or as a queued job. You can specify a retention period other than the 14-day default when you queue the option only, by using the new TASK PARAMETERS field of the OPTION SCHEDULING file. If you put a numeric value in the TASK PARAMETERS field, this value replaces the default alert retention value of 14 days.

The Delete Old (>14 d) Alerts option also purges the new ALERT TRACKING file. It purges all entries in the ALERT TRACKING file that are more than 30 days old. The only exception is if, when an alert is created, the call to create the alert specified a retention period different than 30 days; in this case, the different period will be used.

Finally, this option is what forwards unprocessed alerts to supervisors and surrogates (if this was requested when the alert was created). However, if the period to wait before forwarding exceeds the purging retention period used by this option, the alerts will be purged rather than forwarded.

Due to the number of tasks performed by this option, it should be queued through TaskMan on a regular basis. The suggested scheduling frequency is once every day.

Purge Alerts for a User

A new option in Kernel V. 8.0, Purge Alerts for a User, allows you to delete alerts for a user. The main purpose of this option is to provide a way to delete alerts for a user who has been inactive for a period of time (e.g., on leave), and who has accumulated a number of alerts that should not need processing. It is locked with the XQAL-DELETE key, and should only be used by IRM personnel and/or ADPACs.

Make an Alert on the Fly

This option allows you to generate an alert on the fly. It interactively asks you for the alert message, recipients, and alert action, if any (you can specify an alert action type of routine or option). It then generates the alert "on the fly." This option is recommended primarily for IRM personnel and ADPACs; it may or may not be appropriate for other selected users.

Programmer Tools

An application package might want to issue an alert to one or more users when certain conditions are met, such as depleted stock levels or abnormal lab test results.

Alerts are usually generated through programmer calls. The entry point for creating an alert is SETUP^XQALERT.

You may want to send alerts from within an application program or as part of a trigger in a VA FileMan file. Developers and IRM staff are invited to discover imaginative ways to integrate alerts within local and national programming. Remember, however, not to overwhelm the user with alerts.

Once you've sent an alert, one way you can confirm that the alert was sent is to use the VA FileMan Inquire option, and examine the entry in the ALERT file (# 8992) for the user(s) you sent the alert to.

■ Creating an Alert for User #14

```
; send alert
S XQA(14)="",XQAMSG="Enter progress note",XQAOPT="ZZNOTES"
D SETUP^XQALERT
```

■ Checking that the Alert Was Sent

```
>D Q^DI <RET>

Select OPTION: I <RET> NQUIRE TO FILE ENTRIES

OUTPUT FROM WHAT FILE: ALERT <RET>
Select ALERT RECIPIENT: `14 <RET>  EXAMPLE,PERSON
ANOTHER ONE: <RET>
STANDARD CAPTIONED OUTPUT? YES// <RET>
Include COMPUTED fields:  (N/Y/R/B): NO// <RET> - No record number
(IEN), no Computed
 Fields

RECIPIENT: EXAMPLE,USER
ALERT DATE/TIME: DEC 01, 1994@08:02:21
ALERT ID: NO-ID;161;2941201.080221
  MESSAGE TEXT: Enter Progress Note     NEW ALERT FLAG: NEW
  ACTION FLAG: RUN ROUTINE              ENTRY POINT: ZZOPT
```

Package Identifier versus Alert Identifier

Package Identifier: The package identifier for an alert is defined as the original value of the XQAID variable when the alert is created via the SETUP^XQALERT call. Typically the package identifier should begin with the package namespace.

Alert Identifier: The alert identifier consists of three semicolon pieces:

pkgid_";"_duz_";"_time

where pkgid is the original package identifier, duz is the DUZ of the user who created the alert, and time is the time the alert was created (in VA FileMan format). The alert identifier uniquely identifies a particular alert (it is used as the value of the .01 field in the ALERT TRACKING file).

The distinction between package identifier and alert identifier is important. More than one alert can share the same package identifier, but the alert identifier is unique. Some Alert Handler entry points ask for a package identifier (and act on multiple alerts), while other entry points ask for an alert identifier (and act on a single alert).

Package Identifier Conventions

The Order Entry/Results Reporting (OE/RR) package uses a convention for the format of the package identifier consisting of three comma-delimited pieces:

namespace_","_dfn_","_notificationcode

where namespace is the package namespace, dfn is the internal entry number of the patient whom the alert concerns in the PATIENT file, and notification code is a code maintained by the OE/RR package describing the type of alert. Note that this three-comma-piece package identifier is still only the first semicolon piece of an alert identifier.

Several Alert Handler entry points make use of these package identifier conventions:

- **PATIENT^XQALERT returns an array of alerts for a particular patient, based on the second comma-piece of alerts' package identifiers.**
- **PTPURG^XQALBUTL purges alerts for a particular patient, based on the second comma-piece of alerts' package identifiers.**
- **NOTIPURG^XQALBUTL purges alerts with a particular notification code, based on the third comma-piece of alerts' package identifiers.**

Callable Entry Points

• SETUP^XQALERT: Create Alerts

Use this entry point to send alerts to users.

Usage D SETUP^XQALERT

Input

XQA: **Array defining at least one user to receive the alert. Subscript the array with users' DUZ numbers to send to individual users; subscript the array with mailgroup names to send to users in mailgroups:**

```
S XQA(USERDUZ)=""
S XQA("G.MAILGROUP")=""
```

XQAMSG: **Contains the text of the alert. 80 characters can be displayed in the original alert. 70 characters can be displayed in the View Alert listing. The string may not contain a caret (^).**

XQAOPT: **(optional) Name of a non-menu type option on the user's primary, secondary or common menu. The phantom jump is used to navigate to the destination option, checking pathway restrictions in so doing. An error results if the specified option is not in the user's menu pathway.**

XQAROU: **(optional) Indicates a routine or tag^routine to run when the alert is processed. If both XQAOPT and XQAROU are defined, XQAOPT is used and XQAROU is ignored.**

XQAID: **(optional) Package identifier for the alert, typically a package namespace followed by a short character string. Must not contain carets (^) or semicolons (;). If you don't set XQAID, you will not be able to identify the alert in the future, either during alert processing, to delete the alert, or to perform other actions with the alert.**

See the Package Identifier versus Alert Identifier section earlier in this chapter for information on OE/RR conventions for the format of the Package Identifier.

XQADATA: **(optional) Use this to store a package-specific data string, in any format. It will be restored in the variable XQADATA when the user processes the alert and is therefore available to the routine or option that processes the alert.**

You can use any delimiter in the variable, including the caret. You can use it to make data such as patient number, lab accession, or cost center available to your package-specific routine or option without needing to query the user when they process the alert. It is up to your routine or option to know what format is used for data in this string.

XQAFLG: **(optional) Alert flag to regulate processing (currently not supported). The values are:**

- **D to delete an information-only alert after it has been processed (the default for information-only alerts).**
- **R to run the alert action immediately upon invocation (the default for alerts that have associated alert actions).**

This input variable currently has no effect, however.

XQAARCH: **(optional) Number of days that alert tracking information for this alert should be retained in the ALERT TRACKING file. Default time period is 30 days. You can specify a different number of days using this input variable. To retain information forever, a value of 100000 is recommended (good for about 220 years).**

XQASURO: **(optional) Number of days to wait before Delete Old (>14d) Alerts option forwards alert to recipient's MailMan surrogate(s) (if any), if alert is unprocessed by recipient. Can be from a number from 1 to 30.**

XQASUPV: **(optional) Supervisor forwarding. Number of days to wait before Delete Old (>14d) Alerts option forwards alert to recipient's supervisor, if unprocessed by recipient. Can be a number from 1 to 30.**

Supervisor is determined from the recipient's NEW PERSON entry pointer to the SERVICE/ SECTION file, and then the entry (if any) in the pointed-to service/section's CHIEF field.

Output **none**

Description

Use SETUP^XQALERT to send alerts to users.

To send an information-only alert, make sure that XQAOPT and XQAROU are not defined. To send an alert that takes an action, specify either XQAOPT (to run an option), or XQAROU (to run a routine).

When the Alert is Processed

Once the alert is created, the user is then able to receive and process the alert from their View Alerts listing. When this occurs, Alert Handler executes the following four steps for the alert:

1. **Alert Handler sets up the variables:**

   - **XQADATA, if originally set when alert was created.**
   - **XQAID, if originally set when alert was created.**
   - **XQAKILL, the purge indicator; always set to 1 by Alert Handler.**

   If you associated a package identifier, XQAID, with the alert, it is restored along with two additional semicolon pieces: the current user number and current date/time. With the two additional semicolon pieces, the package identifier becomes the alert identifier. If you did not define XQAID when creating the alert, Alert Handler sets XQAID to "NO-ID" followed by the two additional semicolon pieces.

2. **Alert Handler runs the routine or option specified, if any, in the XQAOPT or XQAROU input variables.**

   You can refer to the three variables listed above (XQADATA, XQAID, and XQAKILL) in the option or routine that processes the alert.

3. **Once the routine or option finishes, Alert Handler deletes the alert, under the following conditions:**

   - **If XQAKILL remains at the value of 1 as it was set in step 1 above, the alert is deleted for the current user only.**
   - **To prevent the alert from being deleted, kill XQAKILL during step 2 above. You may not want the alert to be deleted if processing, such as entering an electronic signature, was not completed.**
   - **To delete the alert for all recipients of the alert, not just the current user, set XQAKILL to 0 during step 2 above. When XQAKILL is set to 0, Alert Handler searches for any alerts with a matching Alert Identifier (all three semicolon pieces: the original Package Identifier, the alert sender, and the date/time the alert was sent) and purges them so that other users need not be notified of an obsolete alert. Note that to delete an alert for all recipients, you must define XQAID with appropriate specificity when creating the alert.**

4. **Finally, the Alert Handler cleans up by killing XQADATA, XQAID, and XQAKILL. Alert Handler returns the user to the View Alerts listing if pending alerts remain. Otherwise, Alert Handler returns the user to their last menu prompt.**

■ Sample Call to Send an Alert

```
;send an alert
;assume DFN is for patient DOE,JOHN
K XQA,XQAMSG,XQAOPT,XQAROU,XQAID,XQADATA,XQAFLAG
S XQA(161)=""
S XQAMSG="Elevated CEA for "_$P(^DPT(DFN,Ø),U)_"
("_$E($P(^(Ø),U,9),6,9)_").  Schedule follow-up exam in Surgical
Clinic."
D SETUP^XQALERT Q
```

■ Resulting Alert, from View Alerts Option

```
Select Systems Manager Menu Option: "VA <RET>


 1.I  Elevated CEA for DOE,JOHN (5345).  Schedule follow-up exam in
Surgical
          Select from 1 to 1
          or enter ?, A, I, P, M, R, or ^ to exit:
```

• ACTION^XQALERT: Process an Alert

Usage `D ACTION^XQALERT(alertid)`

Input

alertid: **Alert Identifier of the alert to process (same as ALERT ID field in ALERT file). This contains three semicolon-delimited pieces, the first being the original package identifier, the second being the DUZ of the alert creator, and the third being the VA FileMan date and time the alert was created.**

Output **none**

Description

Use this entry point to process an alert for a user, if that user is the current user. Processing of the alert happens exactly as if the user had chosen to process the alert from the View Alerts menu.

• DELETE^XQALERT: Clear Obsolete Alerts

Usage `D DELETE^XQALERT`

Input

XQAID: **Alert Identifier of the alert to delete. It must be a complete Alert Identifier, containing all three semicolon pieces. The first semicolon piece (Package Identifier) must be in the same form as the alert creator defined it, the second piece being the DUZ of the user who created the alert, and the third piece being the time the alert was created (the second and third pieces are defined by the Alert Handler).**

XQAKILL: **(optional) If XQAKILL is undefined or 0, the Alert Handler deletes the alert for all recipients. If XQAKILL is set to 1, Alert Handler only purges the alert for the current user as identified by DUZ (using a value of 1 only makes sense if the current user is a recipient of the alert, however).**

If the package identifier portion of the alert identifier is "NO-ID", however, the alert is treated as if XQAKILL were set to 1, regardless of how it is actually set.

Output **none**

Description

This entry point deletes (clears) a single alert, for the current user (XQAKILL=1) or all recipients (XQAKILL=0 or XQAKILL undefined). The current user (as identified by the value of DUZ) does not need to be a recipient of an alert; however, in that case, only a value of 0 (or undefined) for XQAKILL makes sense.

DELETE^XQALERT, unlike DELETEA^XQALERT, deletes only a single alert whose alert identifier matches the complete Alert Identifier. For more information on alert identifiers, see the Package Identifier versus Alert Identifier section earlier in this chapter.

• DELETEA^XQALERT: Clear Obsolete Alerts

Usage D DELETEA^XQALERT

Input

XQAID: **All alerts whose package identifier matches the value of this input variable will be deleted, for the alert recipients designated by the XQAKILL input variable.**

The form of XQAID may be exactly as initially set when creating the alert. Or it may contain the two additional semicolon pieces added by the Alert Handler (the full alert identifier). The two additional semicolon pieces are ignored, however; this entry point only requires the original package identifier.

If the alert identifier you specify is "NO-ID", however, (the generic package id assigned to alerts with no original package identifier), this entry point will not delete matching alerts.

XQAKILL: **(optional) If XQAKILL is undefined or 0, the Alert Handler deletes matching alerts for all recipients. If set to 1, Alert Handler deletes matching alerts for the current user only as identified by DUZ (using a value of 1 only makes sense if the current user is also a recipient, however).**

Output **none**

Description

Use DELETEA^XQALERT to delete (clear) all alerts with the same package identifier, for the current user (XQAKILL=1) or all recipients (XQAKILL=0 or XQAKILL undefined). The current user (as identified by the value of DUZ) does not need to be a recipient of an alert; however, in that case, only a value of 0 (or undefined) for XQAKILL makes sense.

One example of the use of DELETEA^XQALERT is when a troublesome condition has been resolved. You can use this entry point to delete any unprocessed alerts associated with the condition. It deletes all alerts whose package identifiers match the package identifier you pass in the XQAID input variable (multiple alerts can potentially share the same package identifier). For more information on package identifiers, see the Package Identifier versus Alert Identifier section earlier in this chapter.

• NOTIPURG^XQALBUTL: Purge Alerts Based on Code

Usage `D NOTIPURG^XQALBUTL(notifnum)`

Input

`notifnum:`	**The notification number for which all alerts should be deleted. Alerts are deleted if the value of this parameter matches the third comma-piece in the alert's Package Identifier.**

Output **none**

Description

NOTIPURG^XQALBUTL deletes all alerts that have the specified NOTIFNUM notification number as the third comma-piece of the alert's Package Identifier (the original value of XQAID when the alert was created).

• PATIENT^XQALERT: Return Alerts for a Patient

Usage `D PATIENT^XQALERT(root,dfn,[startdate[,enddate]])`

Input

root: **Fully resolved global or local reference to return list of matching alerts in.**

dfn: **Internal entry number (in PATIENT file) of patient to return alerts for.**

startdate: **[optional] Starting date to check for alerts. If you pass this parameter, all alerts are returned, open or closed, from the startdate until the enddate (if no enddate is specified, all alerts beyond the startdate are returned). If you omit this parameter (and enddate), only currently open alerts are returned.**

enddate: **[optional] Ending date to check for alerts. If you omit this parameter, but pass a startdate, all alerts are returned beyond the startdate.**

Output

root: **All alerts matching the request are returned in the variable you specified in root, in the format:**

```
root=number of matching alerts
root(1)= "I   "_messagetext_"^"_alertid
root(2)=...
```

where the first three characters are either:

`"I  "`	**if the alert is informational**
`"   "`	**if the alert runs a routine**

and where alertid (Alert Identifier) contains three semicolon-delimited pieces, the first being the original package identifier (value of XQAID), the second being the DUZ of the alert creator, and the third being the VA FileMan date and time the alert was created.

Description

With PATIENT^XQALERT, you can return an array of all alerts for a particular patient that are either a) open, or b) within a given time range (both open and closed).

The association of an alert with a patient is based on the conventions used by the OE/RR package for the Package Identifier (original value of XQAID input variable when creating the alert), where the second comma-piece is a pointer

to the PATIENT file. See the Package Identifier versus Alert Identifier section earlier in this chapter for information on OE/RR conventions for the format of the Package Identifier.

• PTPURG^XQALBUTL: Purge Alerts Based on Patient

Usage `D PTPURG^XQALBUTL(dfn)`

Input **dfn:** **Internal entry number (in the PATIENT file) to delete alerts for.**

Output **none**

Description

PTPURG^XQALBUTL deletes all alerts that have the specified patient internal entry number (dfn) as the second comma-piece of the alert's Package Identifier (the original value of XQAID when the alert was created).

• RECIPURG^XQALBUTL: Purge Alerts Based on User

Usage `D RECIPURG^XQALBUTL(duz)`

Input **duz:** **Internal entry number (in the NEW PERSON file) of the user to delete received alerts for.**

Output **none**

Description

RECIPURG^XQALBUTL deletes all alerts that have been sent to the user in file 200, as indicated by the duz parameter.

• FORWARD^XQALFWD: Forward Alerts

Usage `D FORWARD^XQALFWD([.]alerts,[.]users,type,comment)`

Input

[.]alerts: **Array of alerts to be forwarded, each identified by its full alert identifier (the value of the ALERT ID field in the ALERT DATE/TIME multiple of the current user's entry in the ALERT file. Current user is identified by the value of the XQADUZ variable). The alert identifiers for a user's current open alerts can be obtained using the USER^XQALERT entry point.**

If only a single alert is to be forwarded, only the top node must be set (set it to the alert identifier of the alert to forward, and pass by value). If there are multiple alerts to forward, the value of each entry in the array should be one of the desired alert identifier. For example,

```
A6AALRT(1)="NO-ID;92;2941215.100432"
A6AALRT(2)="NO-ID;161;2941220.111907"
A6AALRT(3)="NO-ID;161;2941220.132401"
```

If using an array, the array must be passed by reference in the parameter list.

[.]users: **Users to forward alert to. For forwarding as an alert or as a mail message (when the type parameter is A or M), the variable may specify one or more users, and/or mailgroups. For users, specify by ien (in the NEW PERSON file). You do not need to precede the user's ien with an accent grave. For mail groups, specify in format G.MAILGROUP.**

If there is only a single user or mailgroup, just set the top node of the array to that value, and pass it by value. If there are multiple values to be passed, pass them as the values of numerically subscripted array nodes (and pass the array by reference). For example,

```
A6AUSER(1)="G.MAS CLERKS"
A6AUSER(2)="G.MAS OVERNIGHT"
```

For forwarding to a printer (when the type parameter is P), there should be only a single value specifying the desired entry in the DEVICE file. You can specify the device either by name or by internal entry number (ien). If specifying by ien, precede the ien with an accent grave (e.g., `202).

type: **Indicates the method of forwarding desired. The options are the single characters 'A' (to forward as an Alert), 'M' (to forward as a Mail Message), and 'P' (to print a copy of the alert). If the value passed is not either A, M, or P, then no action will be taken.**

comment: **A character string to use as a comment to accompany the alert when it is forwarded.**

Output **none**

Description

This entry point can be used to forward alerts (in most cases, for the current user only). It is a silent (no screen input or output) entry point, and so can be used for windowed applications.

Example

```
; get open alerts for current user
K A6AALRT D USER^XQALERT("A6AALRT")
;
I +A6AALRT D  ; if any current alerts...
.; loop through A6AALRT array, parse alert id for each open alert
.K A6AALRT1 S A6ASUB="",A6AI=0
.F  S A6ASUB=$O(A6AALRT(A6ASUB)) Q:'$L(A6ASUB)  D
..S A6AI=A6AI+1,A6AALRT1(A6AI)=$P(A6ASUB,"^",2)
.;
.;forward open alerts of current user to MAS CLERKS mailgroup
.K A6AUSER S A6AUSER="G.MAS CLERKS"
.D FORWARD^XQALFWD(.A6AALRT1,A6AUSER,"A","Forwarded Alert")
Q
```

• USER^XQALERT: Return Alerts for a User

Usage `D USER^XQALERT(root,[duz][,startdate[,enddate]])`

Input

root: **Fully resolved global or local reference to return list of matching alerts in.**

duz: **[optional] DUZ number of user to return alert list for. If you don't pass a number, the current user's DUZ is used.**

startdate: **[optional] Starting date to check for alerts. If you pass this parameter, all alerts are returned, open or closed, from the startdate until the enddate (if no enddate is specified, all alerts beyond the startdate are returned). If you omit the startdate parameter (and enddate), only currently open alerts are returned.**

enddate: **[optional] Ending date to check for alerts. If you omit this parameter, but pass a startdate, all alerts are returned beyond the startdate.**

Output

root: **All alerts matching the request are returned in the variable you specified in root, in the format:**

```
root=number of matching alerts
root(1)= "I   "_messagetext_"^"_alertid
root(2)=...
```

where the first three characters are either :

"I ": if the alert is informational
"   ": if the alert runs a routine

and where alertid (Alert Identifier) contains three semicolon-delimited pieces, the first being the original package identifier (value of XQAID), the second being the DUZ of the alert creator, and the third being the VA FileMan date and time the alert was created.

Description

Use the USER^XQALERT entry point to return a list of alerts for a given user. You can return a list of all alerts for a particular user that are either a) open, or b) within a given time range (open and closed).

Example

```
> D USER^XQALERT("ZZALRT",ZZDUZ,2900101) <RET>

> ZW ZZALRT <RET>
ZZALRT=1
ZZLART(1)="I  Test Message^NO-ID;92;2940729.10312"

>
```

Glossary of Terms for Alerts

Alert
An alert notifies one or more users of a matter requiring immediate attention. Alerts thus function as brief notices that are distinct from mail messages or triggered bulletins.

An alert includes any specifications made by the programmer when designing the alert. This minimally includes the alert message and the list of recipients (an information-only alert). It may also include an alert action, package identifier, alert flag, and alert data. Alerts are stored in the ALERT file.

Alert Action
The computing activity that may be associated with an alert (i.e., an option (XQAOPT variable) or routine (XQAROU variable)).

Alert Data
An optional string that the programmer may define when creating the alert. This string is restored in the XQADATA variable when the alert action is taken.

Alert Flag
An optional tool currently controlled by the Alert Handler to indicate how the alert should be processed (XQAFLG variable).

Alert Handler
The name of the mechanism by which alerts are stored, presented to the user, processed, and deleted. The Alert Handler is a part of Kernel, in the XQAL namespace.

Alert Identifier
A three-semicolon piece identifier, composed of the original Package Identifier (see below) as the first piece; the DUZ of the alert creator as the second piece; and the date and time (in VA FileMan format) when the alert was created as the third piece. The Alert Identifier is created by the Alert Handler, and uniquely identifies an alert.

Alert Message
One line of text that is displayed to the user (the XQAMSG variable).

Package Identifier
An optional identifier that the programmer may use to identify the alert for such purposes as subsequent lookup and deletion (XQAID variable).

Purge Indicator
Checked by the Alert Handler (in the XQAKILL variable) to determine whether an alert should be deleted, and whether deletion should be for the current user or for all users who might receive the alert.

Chapter 12 Servers

System Management

What is a Server?

A server is a special type of option (stored in the OPTION file) which can be triggered by mail messages. Addressing a mail message to a server is termed a "server request." A server request awakens the option and causes it to execute the following:

- **Any M code in the Server's ENTRY ACTION field (#20).**
- **Any M code in the HEADER field (#26).**
- **The routine indicated in the ROUTINE field (#25).**
- **Any M code in the EXIT ACTION field (#15).**

A server-type option (henceforth called simply a server) is similar to a run routine-type option. The difference is that a server is activated by a mail message while a run routine option is activated by a user choosing that option from a menu on a screen. Servers should only be invoked by mail messages (never directly by a user).

The form of the mail message that activates the server is identical to any other mail message except that it is addressed to S.<option name>. The "S." (like the "G." form for sending to mail groups) routes the message to the server request software.

What Can Servers Do?

A server request might trigger a bulletin, send a MailMan reply, and/or initiate an audit of itself. Programmers and local IRM staff may also customize the bulletins or MailMan replies.

Can Server Requests Be Denied?

Only options of type server can be activated by mail messages. The following must be true for a server request to be processed:

- **The server must be set to type "s" in the TYPE field (#4) of the OPTION file. If the type is not "s" and a request is received, it results in an error that, by default, is recorded in the AUDIT LOG FOR OPTIONS file.**
- **The server name must be complete and exact when a server request is made or the request will be denied.**
- **The server must not be disabled (it can be disabled for all requests by setting its LOCK and/or OUT OF ORDER MESSAGE fields).**

As long as the conditions listed above are satisfied, the only mechanism a site has for security for server requests is the setting of the server's SERVER ACTION field. This field has four settings, R, Q, N, and I:

- **If set to R, the server is run immediately provided it is not prevented by a setting in the TIMES/DAYS PROHIBITED field.**
- **If set to Q, the server request is honored by queuing the task to the next time permitted by the server's TIMES/DAYS PROHIBITED field.**
- **If set to N, the server request creates a TaskMan entry but doesn't schedule it; a local mail group is notified; the task can be scheduled at the discretion of the mail group.**
- **If set to I, the server ignores any server requests.**

When a server request is received, the server itself is executed similarly to the way a normal option is executed. That is, if a server request causes a server to be run or queued, the server, (along with its associated entry action code, header code, routine, and exit action code), does not run until the option as a whole runs as scheduled by Task Manager.

How Can the Number of Instances of a Server Be Controlled?

You can use the SERVER DEVICE field in the option file to tie a server to a device of type RESOURCES. This allows you to control how many instances of the server can run at any one time. Only as many server processes can run at any one time as are set up in the associated device's RESOURCE SLOTS field. So if thirty mail messages come in at the same time and attempt to fire off thirty server processes, you can control the maximum number of simultaneous processes that actually run. Additional servers will be able to run when resource slots are freed up from the resource device.

Setting Up a Server

A server has many fields in common with other option types and is set up using the Menu Management option Edit options. This option calls the VA FileMan edit template XUEDITOPT, which prompts for data to be entered in the following fields:

NAME (Field # .01): This should be a namespaced set of 3 to 30 uppercase letters.

MENU TEXT (#1): Since there is never a menu prompt for a server, this field should instead contain an accurate description of what this server does, as it is used by the server software in error messages, bulletins, and MailMan replies. It should be 3 to 50 characters in length.

OUT OF ORDER MESSAGE (#2): If this field contains between 1 and 80 characters of text, the server is placed 'out of order' and will not be activated by a server request. The message itself is included in bulletins or MailMan replies which report the failure.

LOCK (#3): Since servers have no on-line user associated with them, the existence of a lock in this field prevents the execution of a server, much like an out of order message. The user for all servers is the Postmaster. The originator of a server request is recorded, however, in the return address variable.

DESCRIPTION (#3.5): This word processing field should contain an extensive description of the server intended for the local Site Manager and IRM staff. The description should include an exact description of what the server does and the resources it requires.

PRIORITY (#3.8): This field determines the priority at which the server runs.

TIMES/DAYS PROHIBITED (#3.91): This multiple allows the local IRM staff to control the days and times during which the server request is honored. If data is entered which prevents the server from being honored immediately, the software determines the next available time slice that is not prohibited and queues the request for that time. Servers that are marked "R" for Run Immediately in the SERVER ACTION field are instead queued to run at the next non-prohibited time period.

TYPE (#4): This field must always contain the code "s" for Server or the request will be denied and an error will result.

ENTRY ACTION (#20): The M code in this field is executed if the server request is honored. If, as with other options, the variable XQUIT exists after the Entry Action is executed, the request is terminated at that point and an error is generated.

HEADER (#26): This field of M code is executed, if it exists.

ROUTINE (#25): If there is a routine name in this field in the forms ROUTINE, ^ROUTINE, or TAG^ROUTINE, the routine is run.

EXIT ACTION (#15): The M code stored in this field is executed just before the server exits.

SERVER BULLETIN (#220): This field is a pointer to the BULLETIN file; it indicates the bulletin to use to notify the local mail group of a server request on their system. If there is no bulletin entered in this field, the default bulletin XQSERVER is used.

Unless there are pressing reasons to do otherwise, it is recommended that the default bulletin XQSERVER be used by leaving the SERVER BULLETIN field blank.

If the mail group(s) pointed to by XQSERVER (or the bulletin pointed to in this field) does not contain an active user (i.e., a user possessing a verify code and no effective termination date) the software turns on auditing (see below) and sends a MailMan message to the local Postmaster. The most common reason for servers not functioning is that there is no active user associated with the bulletin specified. For security reasons, servers will not run without a locally defined active user associated with the chosen bulletin.

SERVER ACTION (#221): This set of codes field allows the local IRM staff to decide how a server request is to be treated. The action codes are:

- **R** **Run immediately. This causes the server request to be honored in real time as soon as it is received from MailMan provided that it is not prevented by a setting in the TIMES/DAYS PROHIBITED field.**
- **Q** **Queue server. This causes the server request to be honored as soon as permitted by the TIMES/DAYS PROHIBITED field.**
- **N** **Notify local authorities. This causes the server software to create a TaskMan entry that is not scheduled to run, and notify the local mail group with the task number so that it may be approved locally and then scheduled to run using TaskMan's Requeue Tasks option.**
- **I** **Ignore any requests. This code causes the software to ignore all requests for this server. A bulletin or MailMan message may still be sent, however.**

SERVER MAIL GROUP (#222): This field is a pointer to another mail group (the first is pointed to by XQSERVER and/or the bulletin in field #220) to which server notifications are to be sent. The software will notify all

legitimate users in all mail groups pointed to. It is recommended that this field be left blank and a mail group be assigned the chosen bulletin instead. Servers will not work unless there is a local, active user associated with the specified mail group.

SERVER AUDIT (#223): This field causes the server request to be audited in File 19.081 (AUDIT LOG FOR OPTIONS). The default is YES. The information stored for an audited server includes: Option, User (always Postmaster), Device, Job #, Date/Time, CPU, message number, return address of sender, subject of the message, and error message. A server can also be audited using the normal option auditing software. Auditing the Postmaster or the namespace "XQSRV" will capture all server requests.

SUPPRESS BULLETIN (#224): If set to "Y" (YES), it prevents a bulletin from being sent under normal conditions. If there is an error or a possible security breach, a bulletin will still be fired. If the field is not filled in, it takes the default of "N", which means that the sending of bulletins is not suppressed.

SERVER REPLY (#225): This set of codes controls the MailMan reply to a server request. The reply is a message returned to the user who has sent the server request and should not be confused with the local user to whom the bulletin is addressed. If a reply is requested, the software uses the return address of the sender as supplied by MailMan to send a local or network reply (see the standard form below). The possible codes are:

N **No reply is sent (the default).**

E **A reply is sent to the return address of the sender only in the event of an error.**

R **A reply is always sent.**

SERVER DEVICE (#227): Optionally use this field to control the number of server requests for this server that can be processed at any one time. Enter the name of a device of type resource (in the DEVICE file). The number of instances of this server that can run at any one time is limited to the number of resource slots in the selected resource device.

Testing if a Site is Reachable: XQSPING Server

You can use the XQSPING server to invoke the Kernel utility XTSPING at a site. This utility tests to see if the domain to which a message is addressed is reachable. For example, if you want to see if the network link to the San Francisco ISC is working properly, you could address a message to:

```
S.XQSPING@ISC-SF.VA.GOV
```

If the text of the message and the subject are simply the line "Testing", you should get the following message in return:

```
MailMan message for DOE,JOHN  COMPUTER SPECIALIST
Subj: PING reply to: TESTING [#999] 28 Nov 92 12:17  1 line
From: PING SERVER in 'IN' basket.
---------------------------------------------------------------
Testing.
```

The XTSPING utility copies the message addressed to it and returns it to the person who sent it.

Testing a Server: XQSCHK Server

You can use the XQSCHK server to return information about a server on a remote system. You should list the server you want to test in the text of the message addressed to XQSCHK. The subject of the message sent to the XQSCHK server is not important. However, the body of the text must contain the name of the server-option to be checked. When you specify the server to be checked, don't precede the server name with an "S.", instead, list the server option's name exactly as it appears in the OPTION file's .01 field.

The server XQSCHK returns fields 220 to 225 from the OPTION file to show how the option has been set up. In addition, several other things about the option are investigated and error or warning messages may be also returned.

For example, if you want diagnostic information about a server named ZZSERVER, and the option resides on the system at the San Francisco ISC, you should create a message containing the text ZZSERVER and send it to:

```
S.XQSCHK@ISC-SF.VA.GOV
```

The server XQSCHK at San Francisco will unload the name of the server (in this example ZZSERVER). Assuming such a server exists, you would expect to receive a reply in a MailMan message as below.

XQSCHK Server Return Message

```
MailMan message for DOE,JOHN  COMPUTER SPECIALIST
Subj: Server Request Reply from SF-ISC.VA.GOV
From: Postmaster  in 'IN' basket
------------------------------------------------------------------

               Nov. 28, 1992  12:18 PM

   Sender: DOE,JOHN
   Option name: ZZSERVER
   Subject: TESTING XQSCHK
   Message #: 999


   This is a reply from ISC-SF.VA.GOV
   Checking Server Option ZZSERVER.

   Fields 220 to 225 in the Option File:
         220 - No bulletin selected, will use default XQSERVER.
         221 - The server action code is Run Immediately.
         222 - The mail group ZZGROUP is pointed to.
         223 - Auditing is turned off.
         224 - The server's bulletin is not suppressed.
         225 - Reply mail is sent when an error is trapped.
```

Errors and Warnings from the XQSCHK Server

The following is a list of the errors or warnings that might be included in the return message from the XQSCHK server, along with an explanation of each:

Can't unload name of server from message: [message subject].

> **The name of the server to be tested couldn't be unloaded from the text of the message sent to waken the server XQSCHK. The message should contain just the name of the server to be tested and nothing more. XQSCHK ignores blank lines (up to 4) and any lines of text that follow the line where it finds the options' name.**

The option [option name] is not in the Option File.

> **There is no option in the remote site's OPTION file that matches the name of the server that was unloaded from the text of the message. The string it is using to search the OPTION file is returned in [option name].**

Option [option name] is not shown as a server-type option but a [type].

> **The option is not marked in the remote OPTION file as a server, but some other kind of option returned in [type], such as a print-type option.**

[Option name] is marked as Out Of Order with the message: [message].

> **The OUT OF ORDER field for that option has been filled in with the text that is returned in [message].**

The expected data in ^DIC(19,[option number], 220) is missing.

> **There is no information for this option in fields 220 through 225. The 220 node of the OPTION file is missing or blank.**

No bulletin associated with this option default XQSERVER is missing from system.

> **There is no bulletin pointed to by field 220 of this option in the OPTION file, and the default XQSERVER bulletin has been removed from the system. Servers will not run without an associated bulletin, even if it is suppressed.**

Option [option name] points to a bulletin not in the bulletin file.

> **WARNING: there is an invalid pointer in field 220 of the OPTION file that points to a nonexistent bulletin. The default bulletin XQSERVER will be used.**

Option [option name] points to a mail group not in the Mail Group File.

WARNING: there is an invalid pointer in field 222 of the OPTION file indicating a mail group that should receive the bulletin in addition to the mail group pointed to by the BULLETIN file.

There are no mail groups associated with the bulletin [bulletin name].

The bulletin returned in [bulletin name] does not have a mail group associated with it in the BULLETIN file.

There is no active user associated with the bulletin [bulletin name].

When following the pointers from the bulletin to the mail group to the NEW PERSON file, an active user was not found. Each server must be linked to a user who has an access and verify code and is not terminated.

There is no routine in field 25 of the Option File for this option.

This server has no routine associated with it in ROUTINE field of the remote site's OPTION file.

The routine [routine name] is not on the system.

The routine that is named in the ROUTINE field of the OPTION file is not found on the system. It has been removed or is in another UCI.

There is no server action code for this option.

The required server action code in field 221 of the OPTION file is blank.

Programmer Tools

Tools for Processing Server Requests

When a server runs, it can call custom programs to perform server-related tasks such as responding to the sender of the server request, or retrieving the actual text of the server request message. In this way, server requests can act not only as triggers, but also as message carriers. The server can call custom programs via the following fields:

- **ENTRY ACTION**
- **HEADER**
- **ROUTINE**
- **EXIT ACTION**

For more information on the programmer API for processing server requests, see the MailMan (V. 7.1 or higher) Programmer Manual.

Key Variables When a Server is Running

There are key variables that are set up when a server is running. You can reference these key variables during any routine run by the server's ENTRY ACTION, HEADER, ROUTINE, and EXIT ACTION fields. The key variables for servers are set up as follows:

XQSOP	**Server option name.**
XQMSG	**Server request message number.**
XQSND	**DUZ of the sender if the request is local; network address of the sender if the request is not local**
XQSUB	**Subject heading of the server request message.**

Appending Text to a Server Bulletin or Mailman Reply

Servers use bulletins and MailMan messages to communicate with the local IRM staff when a server request is received, or with the sender of a server request, usually in the event of an error. These two kinds of documents look very similar and must contain certain key pieces of data. It is also possible, however, for the sender or the local IRM staff to append other information to the bulletin or MailMan message by setting that information into the array XQSTXT (one line per node). For example, if the following array exists:

```
XQSTXT(0)="Please append these two lines of text"
XQSTXT(1)="to the end of the bulletin XQSERVER."
```

The default bulletin, XQSERVER, would then look like:

```
Subj: Server request notice
From: <Postmaster>
------------------------------------------------------------------

Dec. 21, 1989  3:08 PM

A request for execution of a server option was received.

Sender: <Child,Your@HOME.VA.GOV>
Option name: ZZUPDATECL
Subject: UPDATE CHRISTMAS LIST DATA BASE
Message #: 136771

Menu system Action: No error(s) detected by the menu system.

Please append these two lines of text
to the end of the bulletin XQSERVER.
```

You can use the same method to append text to MailMan messages.

Customizing a Server Bulletin

Please note that the first 6 data elements in a server bulletin are always:

1. **The date and time the request was received.**
2. **The sender.**
3. **The requested option's name.**
4. **The subject of the message requesting a server.**
5. **The requesting message's number.**
6. **A brief statement of the menu system's action or an error message.**

If a customized bulletin is used instead of XQSERVER, these data elements should always be printed first, followed by the contents of XQSTXT.

The easiest way to create a customized local bulletin is to use the VA FileMan copy function to copy the default bulletin XQSERVER to a bulletin of another name. Note that XQSERVER has a line of text in it that says:

```
This is the server bulletin XQSERVER
```

To avoid confusion, you should edit this line using the Bulletin Edit option to reflect the name of the new bulletin.

Chapter 13 Help Processor

User Interface

The Kernel's Help Processor is a utility for displaying help frames. A help frame is a screen of text that explains some part of a package. Each individual help frame can have keyword links to other help frames. Using these keywords, you can navigate through a series of related help frames to learn more about each help frame topic.

Some places you may encounter help frames are:

- **When requesting help on options in the menu system.**
- **When requesting help on a menu in the menu system.**
- **As a standalone option describing some part of a package.**

■ Example of a Help Frame

```
               USING THE 'Help Processor' OPTION
The Help processor is a frame-oriented display system which allows
users and programmers to access and manage help text.

The system is driven off of the  HELP FRAME FIL

There are several LINKSwhich will cause the help text to be
displayed to the user.  The system is interactive, and the user may
select which topic he/she wishes further information on.
The Help Frame Processor Menu contains the following options:

 DISPLAY/EDI      -Displays the text of a help frame and allows for the
                   edit of the name, header, text, or related frames.

 CROSS REFERENC   -Lists all the help frames for a specified package,
                   showing parent help frames, linked to menu option,
                   and invoking routine.

 LIST             -Lists the help frames in several different formats.

 MORE OPTIONS.

Select HELP SYSTEM action or <return>:
```

At the bottom of every displayed help frame is a "Select HELP SYSTEM action..." prompt. You have several choices at this prompt. To back your way out of the help frame system, you can simply press Return. This backs you up one level, or exits you if you are at the top level of a help frame tree. If you want to exit quickly from help frames, you can enter ^Q (and Return) to quit immediately without having to back all of the way out.

You can list other choices at the "Select HELP SYSTEM action..." prompt by entering a question mark. The full list of choices is:

Response	Action
Keyword	Jump to help frame associated with Keyword.
<RET>	Quit to previous help frame (exit if no previous).
^Q	Quit the help system.
^R	Refresh the current frame.
^T	Table of related frames.
^O	On/off switch for bracketing/reverse video of keywords.
^H	How you got to this frame.
^E	Edit this frame (only if authorized as editor of frame).

Keywords in a help frame are displayed by the help processor in reverse video. If you enter the first few letters of a keyword and press return, the help processor will jump to the help frame linked to the entered keyword.

Help Frames in the Menu System

If a menu option has associated help frames, you can display them by entering a question mark followed by an option's menu text or synonym at a menu prompt (e.g., ?option):

```
Select Office Menu Option: ?MAILMAN <RET>
```

Entering three question marks at the menu prompt indicates which options have associated extended help (help frames).

```
Select Office Menu Option: ??? <RET>
```

If a menu itself has an associated help frame, entering four question marks at the menus "Select ... action: " prompt displays the help frame associated with that menu if one exists:

```
Select Help Processor Option: ????<RET>
```

System Management

Help frames are entries in the HELP FRAME file (#9.2). The Header and Text of help frames can be displayed to users to provide instruction about an application package or other topics. Help frames can be distributed with a package or can be created locally to provide information about local policies and procedures.

The options used to create, edit, and link help frames are on the Help Processor menu, shown below:

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Menu Management ...                                          [XUMAINT]
  Help Processor ...                                      [XQHELP-MENU]
    Display/Edit Help Frames                            [XQHELP-DISPLAY]
    List Help Frames                                       [XQHELP-LIST]
    New/Revised Help Frames                             [XQHELP-UPDATE]
    Cross Reference Help Frames                            [XQHELP-XREF]
    Assign Editors                                      [XQHELP-ASSIGN]
    Unassign Editors                                  [XQHELP-DEASSIGN]
    Fix Help Frame File Pointers                             [XQHELPFIX]
```

Use of the Help Processor options is explained by help frames associated with the options. The frames may be displayed with the ?option syntax at the select prompt.

```
Select Help Processor Option: ?DISPLAY <RET>/Edit Help Frames
```

List Help Frames Option

The List Help Frames option can be used to print a series of frames with a table of contents and page numbering to resemble a hard copy manual.

```
Select Help Processor Option: List Help Frames <RET>
Select primary HELP FRAME from which to list: XUDOC NEW <RET>
```

New/Revised Help Frames Option

This option produces a VA FileMan-generated print of all help frames that have been updated during a specified time period.

Cross Reference Help Frames Option

The Cross Reference Help Frames option lists any of the following references to a specified set of help frames:

- **Parents (other help frames that call the specified help frame).**
- **Options (options whose HELP FRAME field references the specified help frame).**
- **Routines (if a developer has entered the routine in the specified help frame's INVOKED BY ROUTINE field).**

Deleting Help Frames (Fix Help Frame File Pointers)

There is no Kernel utility to delete frames, but the menu system does not generate errors if a pointed-to help frame is missing. If a site chooses to delete frames using VA FileMan, they should use the Fix Help Frame File Pointers option afterwards to delete dangling pointers from the OPTION file's Help Frame field.

Assigning Help Frame Editors

An existing help frame can be edited, through the Help Processor options, by the following people:

- **The help frame author.**
- **Any holder of the XUAUTHOR key.**
- **Anyone who has been assigned as an editor to that help frame.**

To assign or de-assign an editor to a given help frame, use the Help Processor's Assign/Unassign Editor options.

Disk Space Concerns

Help frames consume disk space. The amount can be considerable if numerous frames are exported with a package. You can estimate the size of the HELP FRAME file by the Kernel's block count utility.

```
Select Systems Manager Menu Option:  P <RET> rogrammer Options
Select Programmer Options Option:  G <RET> lobal Block Count
Block Count for Global ^DIC(9.2) <RET>
```

Creating and Editing Help Frames

One way to edit help frames from the HELP FRAME file is to use the Display/Edit Help Frames option to display the help frame in question. Then, at the "Select Help System Action:" prompt, you can enter ^E to edit the help frame if you have edit access to the help frame. You have edit access if:

- **You are the help frame's author.**
- **You are assigned as an editor for the help frame.**
- **You are a holder of the XUAUTHOR key.**

Another handy way to edit help frames is within the help frame system as invoked from a package. For example, if the help frames are tied to a package's options, you can use the package, invoke the help frame for each field or option, and then edit that help frame on the spot. To edit a help frame in this manner, enter "^E" at the help frame action prompt. To do this, however, you must have edit access to the help frame as described above.

Namespacing of Help Frames

Like entries in the OPTION or SECURITY KEY files, entries in the HELP FRAME file must be namespaced to avoid overwriting problems.

Help Frame Layout Considerations

When entering the text of help frames, you should keep each line to fewer than 80 characters for proper screen display. Note that the text is displayed "as it stands" and is not processed by VA FileMan's text formatter. That is, the text is not wrapped, and word processing "windows" are not evaluated. Frames are usually 22 lines in length although an end-of-page read is issued to allow a pause if the frame exceeds 22 lines.

If there are only a few lines of text, the Help Processor displays a table at the bottom of the screen of all related frames (those frames that the current frame has keyword links to). The table shows the choices of other frames so the user need not enter the keywords in the text. You can force the table of related frames out of the display by entering enough blank lines so that the frame's length is 20 lines (assuming the display has a page length of 24 lines).

For the Help Processor to identify and highlight keywords, the keywords are entered in the text of the help frame enclosed in square brackets. By convention, keywords in help frames are usually in all capital letters. A square bracket character may be displayed as part of the frame's text by entering two of the characters (e.g., [[or]]).

If the frames are to be printed using the List Help Frames option, the resulting manual will have an organized outline if the frames are linked in a top-down tree structure without any circular connections among the branches.

Linking a Help Frame as Help for an Option or Menu

Once a frame or a series of frames has been created, you can associate them with options by entering the name of the top-level frame in the Help Frame field of the OPTION file. You can use Menu Manager's Edit Options to do this. That way, when a user enters a single question mark in conjunction with the option name, Menu Manager will invoke the associated help frame.

```
Select Systems Manager Menu Option:  M <RET> enu Management
Select Menu Management Option:  Edit options <RET>
Select OPTION to edit:  XQHELP-MENU <RET>  Help Processor
NAME: XQHELP-MENU// ^HELP FRAME <RET>
HELP FRAME: XQHELP <RET>
```

Programmer Tools

Entry and Exit Execute Statements

The HELP FRAME file contains two fields for the entry of M code. Code in the Entry Execute Statement is executed just before the help frame is displayed. Code in the Exit Execute Statement is executed afterwards.

Link to the OBJECT file

The HELP FRAME file contains a pointer to the OBJECT file, a file that is maintained by the Washington DC ISC. It has been established so that images, such as cardiac catheterization films, may be integrated within an educational help system on multimedia workstations.

Callable Entry Points

- **EN^XQH: Display Help Frames**

Usage `D EN^XQH`

Input	**XQH:**	**Help Frame name (the .01 value from the HELP FRAME file).**

Output **none**

Description

EN^XQH displays a help frame. It immediately clears the screen and displays the help frame (unlike the EN1^XQH entry point, which does not clear the screen and offers the user a choice of whether to load the help frame).

• EN1^XQH: Display Help Frames

Usage D EN1^XQH

Input **XQH:** **Help Frame name (the .01 value from the HELP FRAME file).**

Output **none**

Description

EN1^XQH displays a help frame as EN^XQH does, except that it doesn't clear the screen beforehand, and prior to loading the help frame, EN1^XQH invokes end of page handling (i.e., prompting the user "Enter return to continue or '^' to quit"). If the user enters an "^", the help frame is not displayed. If they press <RET>, the help frame is displayed.

• ACTION^XQH4: Print Help Frame Tree

Usage D ACTION^XQH4

Input **XQHFY:** **Help frame name, equal to the .01 field of the desired entry in the HELP FRAME file. Should be set to the NAME of the top level help frame for which a listing is desired.**

XQFMT: **(optional) Specifies the output format. Value of XQFMT can be:**

- **T** **Text of help frames only (default).**
- **R** **Text of help frames, plus a table of related frames and keywords (if any) for each help frame.**
- **C** **Complete listing (text of help frames, table of related frames for each help frame, and internal help frame names).**

Output **none**

Description

ACTION^XQH4 prints out all the help frames in a help frame tree, including a table of contents showing the relationships between help frames and the page of the printout where each help frame is found. Since help frames can be referenced by more than one help frame, any help frame referenced multiple times appears in the table of contents in each appropriate location, but the help text itself is printed only once. You can alter the format of the output with the XQFMT input variable.

Chapter 14 Error Processing

User Interface

When an option you are using encounters an error condition, you are usually returned to the menu system. A message is displayed indicating that an error has occurred. You are then presented with the last menu prompt and may continue.

There are certain error conditions, however, that may prohibit or prevent return to the menu system. In these situations, you will be halted off the system.

System Management

The Error Processing menu handles errors for DSM for OpenVMS, M/SQL, and MSM systems. It provides access to options pertaining to the error trap, displaying, printing, and purging errors. Like the error traps provided by the operating systems, the utility allows the investigation of program execution errors or the examination of system errors by capturing a picture of the environment for later reconstruction.

The %ZTER* routines are called from ERR^ZU to trap errors and store them in the ^%ZTER global, a Manager account global that should be translated so that all errors are included on one report. The XTER* routines are used to format the error report. The error report generator is similar to the one used by DSM for OpenVMS.

Enhanced Error Processing

Enhanced error processing for DSM for OpenVMS sites is supported. For versions of DSM for OpenVMS that support the proposed 95 M standard, Kernel's error trap captures variables in their state at the time errors occur, regardless of how variables may have been NEWed beforehand. Stack levels for the routine call stack are recorded in the error trap in the $STACK variable. When MSM-DOS supports the enhanced error processing in the proposed 95 M standard, a Kernel patch will be issued to support the MSM-DOS implementation.

The descriptions of the Error Processing menu options that follow are arranged in the same order as they appear on the Error Processing menu.

```
SYSTEMS MANAGER MENU ...                                          [EVE]
  Programmer Options ...                                        [XUPROG]
    Error Processing ...                                        [XUERRS]
      P1 Print 1 occurrence of each error for T-1 (QUEUE)
                                              [XUERTRP PRINT T-1 1 ERR]
      P2 Print 2 occurrences of errors on T-1 (QUEUED)
                                              [XUERTRP PRINT T-1 2 ERR]
      Clean Error Trap                                  [XUERTRP CLEAN]
      Error Trap Display                                     [XUERTRAP]
      Interactive Print of Error Messages         [XUERTRP PRINT ERRS]
```

Report the First Occurrence of Each Error

The Print 1 Occurrence of Each Error for T-1 (QUEUE) option lists the first occurrence of each error recorded on the previous day. T-1 represents "Today-1 = Yesterday". You can queue it to run shortly after midnight. If a device is specified, the output is sent to the specified device. If a device is *not* specified, the output is placed in a mail message and sent to the individual who queued the option to run. It should be set to automatically requeue at a 1 D interval.

Report the First Two Occurrences of Each Error

The Print 2 Occurrences of Errors on T-1 (QUEUED) option lists the first two occurrences of each error recorded on the previous day. T-1 represents "Today-1 = Yesterday". It may be queued to run shortly after midnight. If a device is specified, the output is sent to the specified device. If a device is *not* specified, the output is placed in a mail message and sent to the individual who queued the option to run. It should be set to automatically requeue at a 1 D interval.

Cleaning the Error Trap

You can use the Clean Error Trap option to purge the error log. It is locked with the XUPROGMODE security key. You can use the corresponding direct mode utility, ^XTERPUR, in programmer mode. There is also a queuable version, Error Trap Auto Clean [XUERTRP AUTO CLEAN].

Purging is a partial clearing of the ERROR LOG file (#3.075) stored in the ^%ZTER(1, global. This global node should not be deleted directly since potentially important recent errors would be purged. Deletion of the entire ^%ZTER global would be a greater mistake since the standard reference data contained in the ERROR MESSAGES file (#3.076) stored in ^%ZTER(2, would be lost.

You are first prompted for the number of days to leave in the error trap. If you enter a number of days to retain errors, all errors older than the specified number of days are immediately purged:

```
To Remove ALL entries except the last N days, simply enter the
number N at the prompt.  OTHERWISE, enter return at the first
prompt, and a DATE at the second prompt.  If no ending date is
entered at the third prompt, then only the date specified will be
deleted.  If an ending date is entered that range of dates
INCLUSIVE will be deleted from the error log.

Number of days to leave in error trap: 50 <RET>

          DONE
```

If you just press <RET> instead of entering a number of days to retain, you are then prompted for a start date and end date between which to remove errors. Errors in the period you specify will then be purged immediately:

```
Starting Date to DELETE ERRORS from: 1/1 <RET> (JAN 01, 1995)
Ending Date to DELETE ERRORS from: 1/31 <RET> (JAN 31, 1995)
```

The queuable version of this option, Error Trap Auto Clean, can be scheduled to run in the background. By default, it cleans up errors recorded more than 7 days in the past. You can specify a different interval by placing a numeric value (representing the number of days beyond which to purge) in this option's TASK PARAMETERS field.

Error Trap Display

The Error Trap Display option displays errors that have been trapped on the system. The messages for these errors are operating-system dependent. You can use the corresponding direct mode utility, ^XTER, from programmer mode.

The error trap tries to capture a description of the error, the local symbol table, the last global reference, and other sign-on statistics. For DSM for OpenVMS, $ZC calls are used to record IO counts, CPU time, and page faults.

■ Error Trap Display

```
In response to the DATE prompt you can enter:
     'S' to specify text to be matched in error or routine name.

Which date? > T-1 <RET>
1 error logged on 2/9/95
  1)  <ECODETRAP>PRGMODE+5^%ZOSV:2    07:41:52  KDE,KDE  20801D46
      _LTA4523:

No disconnect error

Which error? >  1 <RET>

Process ID:  2020107A  (538972282)          JAN 18, 1992 17:19:21

Username: EXAMPLE            Process Name: DHCP User

UCI/VOL: [KDE,KDE]                      :

$ZA:   2                               $ZB:  94

Current $IO: _LTA4523:       Current $ZIO: LTA_00129420196A

CPU time:   3.17              Page Faults:       1204

Direct I/O:  81              Buffered I/O:         96

$ZE= PRGMODE+5^%ZOSV:2, %DSM-E-ECODETRAP, $ECODE error trap
received

 D @XQZ G OUT"

Last Global Ref: ^XUSEC(0,"CUR",24,2950209.074142)

Which symbol? >
```

Errors may be reported by searching for a date range or character string. Question marks show a count of errors for the selected range. Two question marks exclude disconnects and three include disconnects. A string search could be used to find XQ in all routines or an UNDEF in the definition of all

errors. Once an error is identified, the report generator shows the job number, username, IO value, date/time, UCI/Volume Set, error type, last global reference, and the line of code that caused the error. It then prompts for a listing of variables, ^L to list all or a letter such as X to list those starting with X. The listing may be printed to the screen or to an output device. You can page through the screen listing, one screen at a time, and enter ^Q to quit or ^ to exit at the end of each screen.

A restore feature may be invoked with ^R provided that the user is working in programmer mode. Programmer mode is required as a protection against restoration of variables from within the menu system. To the extent possible, the environment at the time of the error is restored with the routine and local symbol table intact.

```
Which symbol? > ? <RET>

Enter:
     ^Q to EXIT
     '^' to return to the last question
     Leading character(s) of symbol(s) you wish to examine
     $ to get a display of the $ system variables
     ^L to obtain a list of all symbols
     ^R to restore the symbol table and ... and enter direct mode
```

After reviewing the error log, you are given the opportunity to examine the operating system's error log. Now that most DHCP applications record their errors in Kernel's error log, however, there is less need to track DHCP errors in the operating system error log.

```
Do you want to check the OPERATING SYSTEM ERROR TRAP too? NO//
```

Interactive Print of Error Messages

The Interactive Print of Error Messages option provides for an interactive print of the first "n" of occurrences of an error (where "n" is user selectable) over a specified date range.

Direct Mode Utilities

These direct mode utilities can be run from programmer mode. They are not, however, callable entry points; instead, they are provided for convenience.

>D ^XTER

You can call the ^XTER direct mode utility from programmer mode. It is the same as using the Error Trap Display option.

>D ^XTERPUR

You can call the ^XTERPUR direct mode utility from programmer mode. It is the same as using the Clean Error Trap option.

Programmer Tools

Callable Entry Points

• $$EC^%ZOSV: Return Error Message

Usage `S X=$$EC^%ZOSV`

Input **none**

Output **none**

Description

Use this function to return the most recent error message recorded by the operating system.

• ^%ZTER: Kernel Standard Error Recording Routine

Usage `D ^%ZTER`

Input	**%ZT**	**[optional] The %ZT array can be used to identify a global node whose descendants should be recorded in the error log. (When called within the standard Kernel error trap, %ZT is set to record the user's location in the menu system.)**
Output	**%ZTERROR**	**Calls to the error recorder always return this variable. It has the error name and error type as its first and second up-arrow pieces, for example, %ZTERROR=UNDEF^P. While the first piece is always defined since it is retrieved from the operating system, the second piece could be missing if unavailable from the ERROR MESSAGES file (#3.076).**

Description

Kernel sets the error trap in ZU so that all user errors are trapped. In this context, when an error occurs, the optional %ZT input array is set to indicate the user's location in the menu system. Then ^%ZTER is called to record this information in the ERROR LOG file.

The application-specific error trap routine, when it is called as a result of an error, can then use the ^%ZTER entry point to record error information in

the ERROR LOG file if it decides that it needs to. ^%ZTER gathers all available information such as local symbols and last global reference and stores that information in an entry in the ERROR LOG file.

The simple example below shows an application that replaces the standard Kernel error trap with its own error trap. When an error occurs, and the application's error trap routine is called, it calls $$EC^%ZOSV to see what type of error occurred. If an end-of-file (EOF) error occurs, it lets the application continue. Otherwise, it calls ^%ZTER to record the error, and then quits to terminate the application.

NOTE: The recording mechanism of ^%ZTER also functions in the absence of an error. In a debug mode, this would enable a programmer to record local symbols and global structures at predetermined places within code execution for later checking.

■ Error Trap Example

```
ZXGAPP ; 999/NV - sample routine ; 23-FEB-95
   ;;1.0;;
   ;
FILEOPEN ;
   ;
   ; this code resets the error trap routine that is stepped to
   ; when an error occurs
   ;
   S X="ERR^ZXGAPP",@^%ZOSF("TRAP")
   ;
   ; open a file, and read lines from it until End-of-file (EOF)
   ; is reached
   ;
   K %ZIS S %ZIS=""
   S %ZIS("HFSNAME")="MYFILE.DAT",%ZIS("HFSMODE")="RW"
   D ^%ZIS Q:POP
   F  U IO R LINE:DTIME U IO(0) W !,LINE
   ;
FILECLOS ;
   ;
   D ^%ZISC Q
   ;
ERR     ;
   ; this is the application specific error trap
   ;
   I $$EC^%ZOSV["ENDOFILE" G FILECLOS ; continue if EOF error
   D ^%ZTER ; record the error if anything other than EOF
   Q
   ;
```

Part 3: Device Handler

Chapter 15 Device Handler: User Interface

Applications that are designed for the Kernel environment perform output in a consistent manner, using Kernel's Device Handler. This ensures consistency, both for how you are asked to select devices for output, and also for how output is actually performed.

When you respond to the "DEVICE: " prompt, you are using the Device Handler.

Printing to Devices

At the "DEVICE: " prompt, to send output to your terminal, you can simply press <RET>. This tells the device handler to display the report on the home device (that is, on your terminal):

```
DEVICE: <RET>          to direct output to the current terminal.
                       The home device can also be selected by
                       entering H, h, Ø, or HOME.
```

To send output to a printer, enter the name of the printer at the "DEVICE:" prompt:

```
DEVICE: DVNM5 <RET>    to specify a device with the name DVNM5.
```

To select the closest printer, if one is defined, you can simply enter P and press <RET>:

```
DEVICE: P <RET>        to select the closest printer if one is
                       defined.  (This may be disabled when using
                       DECservers or MICOM port contention.)
```

You can enter a question mark to display help about the syntax of the response.

```
DEVICE: ? <RET>
Specify a device with optional parameters in the format
          Device Name;Right Margin;Page Length
                          or
     Device Name;Subtype;Right Margin;Page Length
```

You can enter two question marks to display available printers and other devices connected to the current volume set or "reachable from" the current volume set. You can also ask for a series of help frames under extended help:

```
DEVICE: ?? <RET>
The following information is available:
                    All Printers
                    Printers only on 'ROU'
                    Complete Device Listing
                    Devices only on 'ROU'
                    Extended Help

                Select one (A,P,C,D, or E):
```

You can list all devices. In addition to printers, this list shows other types of devices you can use to handle output. Unusual device types, such as Hunt Groups, SDP devices, and Resource devices are discussed in the Special Devices chapter. An example of a partial printer listing is shown below:

```
                Select one (A,P,C,D, or E): P <RET>

DATASOUTH 220-1(16X218)                 DECNET-TASK-TASK COMM PORT
GENICOM10P 6th Floor 301                GENICOM16P 6th Floor 301
KYOCERA DEV-10P                         KYOCERA DEV-12P
```

Specifying Right Margin and Page Length

Ordinarily, when choosing an output device, you only need to specify the device name. There may be times, however, when you may find it useful to specify the right margin and/or the page length for your output. The syntax to specify margin and page length uses semicolon delimiters. The format is:

DEVICE: Device Name ; Right Margin ; Page Length

The following examples show how to use the additional semicolon-delimited pieces:

DEVICE: DVNM5;80;66	**Use the DVNM5 device with a right margin of 80 columns and page length of 66 lines.**
DEVICE: ;132	**Use the home device, right margin of 132.**
DEVICE: ;;66	**Use the home device and format the output with page breaks at 66 lines.**
DEVICE: ;;9999	**Scroll output on the home device without needing to press <RET> at page breaks.**

Queuing

At the "DEVICE: " prompt, if you enter a device's name, the output goes directly to the device. If the output you're sending is, for example, a long report, this ties your terminal up until the report finishes printing to that device.

You can print output and yet keep your terminal free for other processing by queuing your jobs rather than running them directly. As described in the Task Manager: User Interface chapter, you can queue output by responding to the device prompt with the letter Q. The device prompt is then presented a second time so that you can specify the output device to queue to.

```
DEVICE: Q <RET>
DEVICE: DVNM5 <RET>
REQUESTED TIME TO PRINT: NOW// <RET>
REQUEST QUEUED!
Task number: 856103
```

Alternatively, you can still specify the device first. The Device Handler checks to see if the device is available and, if so, asks you if you want to queue your output. If the device cannot be reached at the current time, Device Handler indicates that the device is busy or unavailable. You can avoid the preliminary availability check by entering Q at the first prompt, as above.

```
DEVICE: DVNM5 <RET>
DO YOU WANT YOUR OUTPUT QUEUED? NO// YES <RET>

REQUESTED TIME TO PRINT: NOW// T@18:00 <RET>  (JUL 11, 1994@18:00)
REQUEST QUEUED!
Task number: 856109
```

Whether you request queuing before or after naming a device, Device Handler then asks you to specify a time for the queued job to run. You can accept the default (NOW) or indicate a later time in the usual format. Queuing sends output to TaskMan for scheduling. Meanwhile, you can continue working on the computer system without a delay.

```
REQUESTED TIME TO PRINT: NOW// T@18:00 <RET>  (JUL 11, 1994@18:00)
REQUEST QUEUED!
Task number: 856109
```

For more information about queuing output, please see the Task Manager: User Interface chapter.

Specifying a Special Subtype

There is an exception to using numbers in the second semicolon piece to indicate a right margin setting. If, instead of a number, you use a letter and then a hyphen in a device specification (for example: P-DEC), the second semicolon piece specifies a terminal type entry from the TERMINAL TYPE file to use for the output. A terminal type entry specifies information about what commands to use with specific printers, such as escape codes.

```
DEVICE: ;P-DEC          if the home device is a video terminal,
                        output would be formatted with page breaks
                        but would scroll without waiting for the
                        user to enter <RET> after a screen display.
```

One form of the subtype request made possible by VA FileMan's print routines is the use of the word SINGLE along with P- or PK-. Appending "-SINGLE" indicates that a pause should occur after the display of each page. If using a slaved device to print the screen display, for example, the next page is displayed only after the user has pressed <RET>:

```
DEVICE: ;P-DEC-SINGLE  if the home device is a video terminal,
                       output would be presented one (single) page
                       at a time, the next page being displayed
                       after the user presses <RET>.
```

If you're not sure which subtype to use, you can enter a partial specification of the subtype in the second piece, and the device handler will let you choose from all matching subtypes. For example, if a dozen subtypes begin with "P-LASER...", you can list them by entering only the beginning of the subtype name (e.g., P-LASER):

```
DEVICE:  LASER;P-LASER <RET>
```

All subtypes beginning with P-LASER are listed; you can then choose a subtype from this list.

When using a subtype as the second semicolon piece of a device specification, you can still specify a right margin and page length to use, but you then do so with the 3rd and 4th semicolon pieces:

```
DEVICE:  LASER;P-LASER-NEW;132;100 <RET>
```

The syntax for the four semicolon piece form of the device specification is:

```
DEVICE:  Device Name ; Subtype ; Right Margin ; Page Length
```

Spool Document Name - An Exception

When you request the spool device at the device prompt, you can use the following formats to specify the spool document name:

```
DEVICE:  Spooler ; Spool Document Name ; Right Margin ; Page Length

DEVICE:  Spooler ; Subtype ; Spool Document Name
```

Although neither right margin nor page length can be specified when including a subtype as the second piece and spool document name as the third, no functionality is lost. The explanation is simple: the spooler only responds to these two terminal type specifications. In other words, identifying a subtype for the spooler does no more than define a margin and page length.

Spool document entries in the SPOOL DOCUMENT file cannot have names beginning with P-, PK-, C-, etc. (one or two letters followed by a hyphen). Because this syntax is the required naming convention for subtypes, you are allowed to specify the document name and the subtype in any order.

For more information about Spool Devices, see the Spooling chapter.

Alternate Syntax for Device Specification

An alternate syntax is available for specifying right margin and page length when responding to the device prompt. Using the alternate format, you can specify pitch, intensity, and quality. The success of specifying these additional attributes, however, depends on whether the corresponding fields have been defined by IRM at your site.

The syntax requires the use of a slash (/) after the last semicolon. You can then use the letters B, L, M, P, and Q in any order, without separating punctuation to delimit the pieces. These codes specify:

B	**Boldface**
L	**Page length**
M	**Margin**
P	**Pitch**
Q	**Quality (can be Q, Q1, or Q2)**

For example, you could specify:

```
DEVICE:  LASER;P-LASER-LANDSCAPE;/M132L100P16BQ2
```

In this example, the margin is set to 132 (M132), the page length to 100 lines (L100), the pitch to 16 (P16), the intensity to bold typeface (B), and the quality set to letter quality (Q2). An absence of the B would indicate normal intensity. The quality settings are Q, Q1, and Q2.

Your IRM needs to confirm that the appropriate code to set the specified printer attributes is set up for the device that you are using. Then, when the Device Handler closes the device, IRM needs to be sure that appropriate reset code is in the Close Execute field so that the characteristics do not stay in effect. If, for example, someone requests a small pitch, subsequent reports will also use the small pitch unless reset in the close execute statement for that device (or altered by the open execute statement of the next device called).

Summary

The Device Handler is a common interface used by all DHCP applications to send output to devices (usually, printers). Once you become familiar with the Device Handler's special features, including queuing, selecting a specific right margin or page length, and selecting a special subtype.

Chapter 16 Device Handler: System Management

The Device Handler makes use of two primary files: the DEVICE file and the TERMINAL TYPE file. Together, these two files control most of the characteristics of devices in Kernel.

DEVICE File

Kernel's DEVICE file stores information about devices on the system. All connected volume Sets/CPUs should make use of a single DEVICE file. Then all information concerning a particular device is stored in just one place, which facilitates device management.

Each CPU has attachment points to which devices may be connected, for example, physical ports. The $I field in the DEVICE file entry identifies this attachment point.

Some devices, like printers, are connected to one CPU via one $I. When using such a device, the Kernel's Device Handler allows the creation and use of multiple DEVICE file entries for the same physical device. Each DEVICE file entry can contain different specifications (font, margin, page length, etc.) to format output. Each entry in the DEVICE file, then, uniquely identifies a set of instructions to send to a particular $I location on a particular CPU.

Each device that Kernel Device Handler needs to communicate with should be set up as an entry in the DEVICE file. The DEVICE file supports a variety of devices, including video display terminals (VDTs), commonly called cathode ray tube devices (CRTs); printers; tape drives; and operating system devices like spool areas and sequential disk processors (SDP).

The DEVICE file is located in the manager's account for common reference from all associated accounts. With TaskMan's help, this information is also available to all associated processors (CPUs) in the local area network. The global locations of the device-related files are:

DEVICE file (#3.5)	**^%ZIS(1,**
TERMINAL TYPE file (#3.2)	**^%ZIS(2,**
DA RETURN CODES file (3.22)	**^%ZIS(3.22,**

The Subtype field of the DEVICE file points to the TERMINAL TYPE file to retrieve a standard set of characteristics that have been defined for vendor devices (for example, Genicom printers or VT320 CRTs).

DEVICE File Fields

The most essential fields to populate or consider populating for DEVICE entries are:

- **NAME**
- **$I**
- **VOLUME SET (CPU)**
- **TYPE**
- **SUBTYPE**

Two fields introduced by Kernel V. 8.0 are:

- **PRE-OPEN EXECUTE**
- **POST-CLOSE EXECUTE**

The DEVICE file has many more fields where additional specific information for particular devices can be entered. Kernel provides a number of options to facilitate creating and editing device types:

```
Device Management ...                                          [XUTIO]
  Device Edit                                                  [XUDEV]
  Edit Devices by Specific Types ...                        [XUDEVEDIT]
     Network Channel Device Edit                         [XUDEVEDITCHAN]
     Host File Server Device Edit                         [XUDEVEDITHFS]
     Magtape Device Edit                                   [XUDEVEDITMT]
     SDP Device Edit                                      [XUDEVEDITSDP]
     Spool Device Edit                                    [XUDEVEDITSPL]
```

VOLUME SET (CPU)

Setting a value in a device's VOLUME SET (CPU) field is optional. If the VOLUME SET (CPU) field is filled in, the device is assumed to be accessible only from the specified CPU. If the field is left blank, the device is assumed to be accessible from any CPU.

In the PDP environment, most devices had a value for the VOLUME SET(CPU) field. An exception was when a port had been reserved on each CPU for the same purpose. For example, the mini-engine or MICOM command port could be accessible by the same $I number, such as 99, on each processor. Device entries made in this way, without a VOLUME SET(CPU) field value, are called generic entries.

In the DSM for OpenVMS environment, where cluster mounting is used, and most devices are set up on all CPUs, all such devices don't need a value for this field.

On MSM-DOS systems, where most devices are set up only on the print server, those devices should have the VOLUME SET (CPU) value set to the print server's volume set name (for example, PSA). Only the few devices that are set up on both the print and computer servers should have the VOLUME SET (CPU) field left blank.

When the VOLUME SET(CPU) field is blank, the Device Handler still maintains the CPU cross-reference to support queuing and other activities. The cross-reference format involves use of periods as delimiters. If the VOLUME SET (CPU) value were BBB, the cross reference for the device with a $I of 75 would be "BBB.75". If the VOLUME SET (CPU) value were null, then ".75" would be the CPU cross-reference.

TYPE

There are twelve types of devices. A device's type is set in the DEVICE file's TYPE field. The twelve types are:

TRM	**Terminal devices such as most CRTs and printers should be associated with a corresponding device entry with a type of TRM.**
OTH	**Other devices that do not fit a particular category should be given a type of OTH.**
MT	**Magtape devices should have a type of MT.**
SDP	**Sequential Disk Processor is a predefined allocated disk space used for sequential processing. With MSM-DOS, the Sequential Block Processor(SBP) is essentially the same as SDP.**
SPL	**Spool device is a predefined allocated disk space similar to SDP. However, access to the spool device can be achieved from multiple users simultaneously.**
BAR	**Bar code reader. This type identifies the device as a barcode reader.**
HFS	**Host File Server is only available on M platforms that exist on a layered system(e.g., DOS, OpenVMS, UNIX, etc.). This type and the associated functionality provides the vehicle to read and write to host level files. Instead of directing reports to a printer, the results could be placed into a DOS, OpenVMS, or UNIX file. This would allow a non-M-based statistical package or spreadsheet to use data produced by the M-based application by simply extracting data from the host file.**
VTRM	**Virtual Terminal Server devices are those that are associated with a dynamically created M port identification($I). A generic device entry with a device type of VTRM can be established for users who log into the system through terminal servers.**

HG Hunt Groups are groups of devices that share a purpose for printing common reports. Printers in the same Hunt Group are usually expected to be in the same general proximity. Reports may be directed to a Hunt Group or to one of its members. If a member is not available, the next available device in the Hunt Group is selected.

RES Resources is a type used for special sequencing of tasks that do not require a particular device.

CHAN Network Channels are high speed devices that use network protocols such as DECNET and TCP/IP.

IMPC Imaging work station device (reserved for future use for DHCP Imaging Project).

This listing of the device type descriptions can also be obtained by entering two question marks at the TYPE field while editing a device.

SUBTYPE

Use this field to select a default terminal type for the device (see below for a discussion of the TERMINAL TYPE file).

SIGN-ON/SYSTEM DEVICE

If set to YES, this field identifies that this entry is the primary device among those device entries that have the same $I with the same VOLUME SET(CPU). Among those device entries that have a common $I and CPU, only one of these entries can have this field set to 'YES'. If none of the common device entries is set to YES, the default device will be identified by the first device on the CPU cross reference. The default device is used when the device handler is invoked with $I as the device to be selected.

QUEUING

You can control the degree of queuing allowed for a device with the QUEUING field. The following settings control queuing for a device:

0	**ALLOWED**	**Jobs can be queued or run directly (default).**
1	**FORCED**	**Queuing is forced, unless disallowed by application.**
2	**NOT ALLOWED**	**Queuing to device is not allowed.**

OPEN PARAMETERS and USE PARAMETERS Fields

Magtape, SDP, and HFS device types use the value of the OPEN PARAMETERS field as the default if the ASK PARAMETERS flag is set. Users would then be prompted for address/parameters. If the ASK PARAMETERS flag is not set and if there is a value in the OPEN PARAMETER field, this value is used when opening the device (or file). The Device Handler also takes information from the USE PARAMETERS field when opening and using such devices as the tape drive.

Each operating system has its own way of specifying parameters. For example, under DSM for OpenVMS, margins are not set with the OPEN command, but instead with the USE command.

PRE-OPEN EXECUTE, POST-CLOSE EXECUTE

These fields can be used to execute a line of M code prior to opening the device and after the device is closed. Note: If you define the variable %ZISQUIT in the PRE-OPEN EXECUTE code, the device open will fail. With this variable, you can use the PRE-OPEN EXECUTE as a screen on whether the device should be opened or not.

OpenVMS-Specific DEVICE Fields

The DEVICE file can store operating system-specific information. Several fields are included, for example, in the DEVICE file to configure terminals and ports on terminal servers as part of an OpenVMS start-up command file. These are LAT Server Node, LAT Server Port, VMS Device Type, and LAT Port Speed. The Kernel Toolkit package distributes a routine, ^XTLATSET, that makes use of these fields.

Device Security

To regulate who may use a particular device, you can use the PASSWORD and SECURITY fields.

The SECURITY field, if populated, should contain a string of characters to compare with a user's File Manager Access Code, DUZ(Ø), when the device is selected. Access is denied to anyone whose DUZ(Ø) does not contain one of the specified characters. As with other uses of DUZ(Ø), the programmer's at-sign will override this restriction.

The PASSWORD field, if populated, forces all users trying to log on to the device to be prompted for the matching password, before entering their access code.

TERMINAL TYPE File

The TERMINAL TYPE file (#3.2) holds vendor-specific code to characterize a terminal type. For example, escape sequences may be entered in the OPEN EXECUTE and CLOSE EXECUTE fields to set pitch or font. Every device in the DEVICE file must be assigned a terminal type, in the SUBTYPE field.

The most common fields to populate for TERMINAL TYPE file entries are:

- **NAME**
- **SELECTABLE AT SIGN-ON**
- **RIGHT MARGIN**
- **FORM FEED**
- **PAGE LENGTH**
- **BACK SPACE**
- **OPEN EXECUTE**
- **CLOSE EXECUTE**

The TERMINAL TYPE file has many more fields where additional specific information for particular terminal types can be entered. Kernel provides the following options to facilitate creating and editing terminal types:

```
Device Management ...                                    [XUTIO]
   Terminal Type Edit                                   [XUTERM]
   Change Device's Terminal Type                      [XUCHANGE]
   List Terminal Types                                  [XULIST]
```

Terminal Type Naming Conventions

The convention for naming terminal types is as follows:

- **C-** **Video terminals (e.g., C-VT100)**
- **PK-** **Printers with keyboards**
- **P-** **Printers without keyboards (e.g., P-DEC)**
- **M-** **Modems**

The general form is limited to two alphabetic characters, a hyphen, and alphanumeric characters.

As mentioned previously, a spool document name may not use this format; this is so that it may be distinguished from a device subtype in a call to the Device Handler. Confusion could arise since either may be used as the second piece of the device specification. The SPOOL DOCUMENT file has

an input transform pattern match that guards against creation of document names in the format of device subtypes.

How Shared Device and Terminal Type Attributes are Used

The DEVICE and TERMINAL TYPE files share attribute fields for right margin, page length, back space and form feed. Currently, when an entry is made in the DEVICE file, choosing a subtype (a field that points to the TERMINAL TYPE file) triggers the setting of the common DEVICE file fields with the values from the Terminal Type fields. Afterwards, if desired, IRM may edit the triggered values to change, for example, the page length of the device entry from 66 to 65.

The device attribute fields that the DEVICE file shares with the TERMINAL TYPE file have been starred out (by placing an asterisk in front of them) to signal future deletion. A future version of Kernel will eliminate them, leaving the TERMINAL TYPE file as the single place determining these attributes for devices.

When a user selects a device by responding to the device prompt with only the first required piece of information, the device identification, Device Handler retrieves parameters to characterize the device, such as right margin, from the DEVICE file. Furthermore, the Device Handler checks the ASK PARAMETERS flag for the selected device and, if the flag is set, prompts the user for associated parameters, presenting DEVICE file characteristics as the default. For terminals and virtual terminals (types TRM and VTRM, respectively), the user is prompted for the right margin. For magtape (MT), sequential disk processor (SDP), and host file server (HFS) devices, they may be prompted for address/parameters with the value of the Open Parameters field (in the DEVICE file) as the default. For more information on MT and SDP devices, see the Special Devices chapter. For more information on HFS devices, see the Host Files chapter.

Terminal Type Information Retained by User

User can change some terminal type attributes of their sign-on device by (1) changing the terminal type during the session with Edit User Characteristics, or (2) selecting a device for direct output. Kernel uses the ^XUTL global (see the Menu Manager: System Management chapter) to hold information about changes made to device characteristics of the home device during a session.

The terminal type established for users at each sign-on is stored according in their NEW PERSON file entries so that, if necessary, it can be used as a default for the next sign-on.

Devices and Sign-On

Device Selection at Sign-On, and Virtual Terminal Devices

Every interactive user must be associated with a device by the Device Handler when they sign on to the DHCP system. The association of device is done by matching the incoming user's $I value with the $I value of an entry in the DEVICE file. On VA's original PDP systems, there were a fixed number of physical ports that terminal users could access the system on, and therefore a fixed number of possible $I values; because of the limited number of possible $I values, it was practical to set up one device entry with a matching $I for each physical port.

With the move to VAX and Alpha platforms, however, the $I of the user was dynamic, with many thousands of $I values possible. The Virtual Terminal device type was created as a way to have one device entry be used for sign-on for multiple incoming $I values. The Device Handler still checks to see if it can assign a device to an incoming process based on an exact match of $I values. If there is no direct match, however, Device Handler checks to see if the *first part* of the user's $I value matches the $I value of a virtual device entry. This way, a virtual device with a $I value of "_LTA" can service all incoming processes whose $I values *start* with the string "_LTA".

Virtual devices do not need a value in the VOLUME SET (CPU) field; they should have SIGN-ON/SYSTEM DEVICE field set to YES, however, to speed up the sign-on device selection process.

Virtual Devices for DSM for OpenVMS Systems: Processes on DSM for OpenVMS systems that use LAT (Local Area Transport) usually have $I values beginning with the prefix "_LTA", concatenated with an integer value and a colon, for example "_LTA8456:". A single virtual terminal device entry whose $I value is "_LTA" will service all such processes. Other common device prefixes on DSM for OpenVMS systems that could be used for virtual terminal device entries include "_TNA" for telnet devices and "_RTA" for remote processes using the "SET HOST" command.

Virtual Devices for MSM-DOS Systems: On MSM-DOS systems, the $I value for a port is always numeric. But using MSM's $ZDEVICE function, Device Handler attempts to retrieve a port and server name for a process. For terminals connected to terminal servers, $ZDEVICE returns (depending on site configuration) a value that begins with "PORT_". When attempting to select a virtual device, if $ZDEVICE returns a different value than $I value of the process, Device Handler matches the return value from $ZDEVICE (rather than $I) with the $I field of DEVICE file entries. In this case, a single virtual terminal device entry whose $I value is "PORT_" will service all such processes.

Terminal Type Selection at Sign-On

Besides needing a device assigned at sign-on, users also need a terminal type. As described in the Sign-On/Security: System Management chapter, Kernel can usually determine the correct subtype without needing to prompt the user, by querying the terminal, and matching the returned string (if any) with return codes for terminals stored in the DA RETURN CODES file.

If the user is prompted to enter a terminal type, they will need to choose one. The list of terminal types they can choose from is screened by the field in the TERMINAL TYPE file called SELECTABLE AT SIGN-ON. Users can only choose from entries with this field set to YES. This stops users from choosing inappropriate terminal types. The setting of this field does not prevent terminal types from being chosen by the DA return code method, however. Make sure that all terminal types appropriate for sign-on have SELECTABLE AT SIGN-ON set to YES.

If the Sign-On/Security system cannot supply even a default, the Device Handler makes a selection according to the sign-on device's subtype.

Managing the Display Attributes (DA) Return Codes

```
Device Management ...                                        [XUTIO]
   DA Return Code Edit                                  [XU DA EDIT]
```

The DA RETURN CODES file, #3.22, stores entries for the codes returned by different terminals after the Kernel asks for their display attributes at sign-on. This file then maps Kernel terminal types to terminal's return codes. This mapping allows sites to set up mappings for new terminals or to map different terminals to a common type. For example, a site could map all codes returned by all DEC VT type terminals to a single C-VT102 type terminal type.

The DA RETURN CODES file is a small static file managed by the DA Return Code Edit option. You can use the DA Return Code Edit option to automate the population of the DA RETURN CODES file. When you select this option, the terminal you are using is queried and you are shown the terminal's DA code response. You are then asked for the terminal type and description for this return code. Enter the terminal type name for the terminal you are using. The option updates the DA RETURN CODES file, and all terminals responding with this code will be recognized at sign-on. You can quickly populate the DA RETURN CODES file by using this option from several different types of terminals.

Kernel pre-populates the DA RETURN CODE file with a set of standard Terminal Type entries. You may need to add more entries as needed to handle all terminals at your site.

Troubleshooting

```
SYSTEM MANAGER MENU                                                  [EVE]
  Device Management...                                             [XUTIO]
    Loopback Test of Device Port                               [XUTLOOPBACK]
    Send Test Pattern to Terminal                                  [XUTTEST]
    Out of Service Set/Clear                                       [XUOUT]
```

Kernel provides several options to aid with troubleshooting device problems.

Loopback Test of Device Port

Use this option to test an RS-232 serial data line when using a loopback connection on the line. First, disconnect the data line from the device it is attached to (if any). Then, tie pins 2 and 3 of the RS-232 serial data line together. This is a loopback connection; data sent down pin 2 (transmit) will loop back up pin 3 (receive). The Loopback Test of Device Port option sends the letters of the alphabet down the data line one at a time, and attempts to read them back. If both lines are intact, you should see "ABCDEFGHIJKLMNOPQRSTUVWXYZ" print on the terminal from which you are testing the data line.

Send Test Pattern to Terminal

Use this option to send a simple test pattern to a device. This is an easy way to verify whether a device is connected to the system. It lets you choose how many lines of the test pattern to send, and then sends that number of lines to the device. You can confirm on the device end exactly how many lines of the test pattern you receive, which can be useful when troubleshooting printer handshaking problems.

Out of Service Set/Clear

You can use this option to set a device out of order. It asks you the date on which to put the device out of order. From that date forward, the Device Handler will not allow any jobs to use the device (users will get a message that the device is out of order). To clear the out of order status, use this option again and delete the out of order date.

Device Identification and Cross-references

Devices may be selected in several ways from the "DEVICE:" prompt. Besides the NAME (.01) field, three other attributes, MNEMONIC, LOCAL SYNONYM and $I may also be used to select devices. When LOCAL SYNONYM is used, the Device Handler searches the local CPU for a match. The same LOCAL SYNONYM value, such as PRINTER, can thus be used to identify several devices, one per CPU.

When editing devices through VA FileMan, two additional fields may be used for lookup: VOLUME SET(CPU) and SIGN-ON/SYSTEM DEVICE. You can separate these values with a period delimiter, as follows:

CPU	**All devices matching CPU.**
CPU.$I	**All devices matching the CPU and $I.**
SYS	**All SIGN-ON DEVICES.**
SYS.CPU	**All SIGN-ON DEVICES matching CPU.**
SYS..$I	**All SIGN-ON DEVICES matching $I.**
SYS.CPU.$I	**All SIGN-ON devices matching CPU and $I.**

For example, to display all sign-on devices on CPU "BBB", you could do:

```
Select DEVICE NAME: SYS.BBB <RET>
```

To display all sign-on devices whose $I begins with "_LTA" you could do:

```
Select DEVICE NAME: SYS._LTA <RET>
```

The following global listing shows the cross references for a device with a $I value of 99 and an internal entry number of 251. It is a Sign-On/System Device and has a VOLUME SET(CPU) value of AAA.

```
^%ZIS(1,"G","SYS.AAA.99",251) = ""
^%ZIS(1,"CPU","AAA.99",251) = ""
^%ZIS(1,"C","99",251) = ""
```

If this device is a virtual terminal with a $I of _LTA and established as a Sign-On/System Device but not given a VOLUME SET(CPU) value, the cross reference structure would be as follows:

```
^%ZIS(1,"G","SYS._LTA",251) = ""
^%ZIS(1,"CPU","._LTA",251) = ""
^%ZIS(1,"C","_LTA",251) = ""
```

Chapter 17 Device Handler: Programmer Tools

The Device Handler provides a common user interface and programmer API for using output devices. This chapter describes the Device Handler's programmer API.

The ZIS* series of routines becomes the Device Handler when the Kernel installation process (the ZTMGRSET routine) saves them in the Manager's account as %ZIS* routines. A separate set of ZIS* routines is distributed for each operating system.

Callable Entry Points

• ^%ZIS: Standard Device Call

All input variables are optional. Non-namespaced variables that are defined and later killed by ^%ZIS include %A, %E, %H, %X, and %Y.

If device selection is successful, characteristics of the output device are returned in a number of different variables. If selection is unsuccessful, ^%ZIS returns the POP output variable with a positive number. So, checks for an unsuccessful device selection should be based on the variable POP as a positive number.

Usage `D ^%ZIS`

Input

%ZIS: **(optional) The %ZIS variable is defined as a string containing one or more single-character flags that act as input specifications. The functions of each of the flags that may be included in the string are described below. If the %ZIS variable contains:**

M RIGHT MARGIN
The user will be prompted with the right margin query.

N NO OPENING
The Device Handler will return the characteristics of the selected device without issuing the OPEN command to open the device.

P CLOSEST PRINTER
The closest printer, if one has been defined in the DEVICE file, will be presented at the default response to the device prompt.

Q QUEUING ALLOWED
The job may be queued to run at a later time. Note that there is no automatic link between the Device Handler and the Task Manager. If queuing is allowed, just before the Device Handler is called, the application routine must set the variable %ZIS to a string that includes the letter "Q".

```
Example: S %ZIS="MQ" D ^%ZIS
```

If the user selects queuing, the Device Handler will define the variable IO("Q") as an output variable, to indicate that queuing was selected. If queuing is selected, the application should set the needed TaskMan variables and call the TaskMan interface routine ^%ZTLOAD. For further details on how to call the TaskMan interface, refer to the Task Manager: Programmer Tools chapter.

0 DON'T USE IO(0)
The Device Handler will not attempt to use IO(0), the home device at the time of the call to ^%ZIS.

D DIRECT PRINTING
If the selected device is unavailable and belongs to a hunt group, the Device Handler will not route the output to another hunt group member. Unavailability will simply be handled in the usual way (by returning a positive number in POP).

L RESET IO("ZIO")
If %ZIS contains L, the IO("ZIO") output variable will be reset with the static physical port name (for example, the port name from a terminal server). It is useful when the $I of the M implementation does not represent a physical port name.

%ZIS("A"): (optional) Use to replace the default device prompt.

%ZIS("B"): (optional) If %ZIS is defined, HOME is presented as the default response to the device prompt. Use %ZIS("B") to replace this default with another response. S %ZIS("B")="" if you do not want to display any default response.

%ZIS("HFSMODE"): (optional) Use to pass the host file access mode to %ZIS. A value of "RW" represents Read/Write access, "R" represents Read Only access, "W" represents Write access, and "A" represents Append mode.

Example:
```
S  %ZIS("HFSMODE")="R"
```

%ZIS("HFSNAME"): (optional) Use to pass the name of a host file to %ZIS.

Example:
```
S  %ZIS("HFSNAME")="MYFILE.DAT"
```

%ZIS("IOPAR"): (optional) Use this variable to pass open command parameters to the Device Handler. If defined, the value of this variable is used instead of any value specified in the OPEN PARAMETER field of the DEVICE file. The Device Handler uses the data from either this variable or from the OPEN PARAMETERS field whether or not the device type is TRM.

On some M systems, Right Margin is an open parameter. Therefore, any value for Right Margin in the DEVICE file, TERMINAL TYPE file, or user response may be ignored when this variable is used.

To set open parameters for the tape drive device, a device with $I=47 and device name of MAGTAPE, the following code could be used:

```
S %ZIS("IOPAR")="(""VAL4"":0:2048)"
S IOP="MAGTAPE" D ^%ZIS
```

Note that the specific parameters you pass may not be functional for all operating systems. Use of this feature should thus be limited to local development efforts.

%ZIS("IOUPAR"): (optional) Use this variable in the same way as %ZIS("IOPAR"), but for parameters to the Use (rather than Open) command. Any Use parameters specified in the DEVICE file will be overridden.

Example:
```
S %ZIS("IOUPAR")="NOECHO"
S IOP="C72" D ^%ZIS
```

%ZIS("S"): (optional) Use this input variable to specify a device selection screen. The string of M code this variable is set to should contain an IF statement to set the value of $T. Those entries that the IF sets as $T=0 will not be displayed or selectable. Like comparable VA FileMan screens, %ZIS("S") should be set to sort on nodes and pieces, without using variables like ION or IOT. As with

VA FileMan, the variable "Y" may be used in the screen to refer to the internal entry number of the Device. Also, the M naked indicator is at the global level ^%ZIS(1,Y,0). An example to limit device selection to spool device types (SPL) only might be coded as follows:

Example:
```
S %ZIS("S")="I $G(^(""TYPE""))=""SPL"""
```

IOP: **(optional) Use IOP to specify the output device. There is no user interaction when IOP is defined to specify an output device; the device name (.01 field) is the usual value of IOP. You can also set IOP to Q and P. (The value of IOP must not be $I).**

You can request queuing by setting IOP="Q". The user is then asked to specify a device for queuing. To pre-select the device, set IOP="Q;device": the device specified after the semicolon is selected and IO("Q") is set.

You can request the closest printer, as specified in the DEVICE file, by setting IOP="P" or IOP="p". If there is not a closest printer associated with the home device at the time of the call, device selection fails and POP is returned with a positive value.

You may now also pass the internal entry number (ien) of the desired device through IOP. For instance, to select a device with an ien of 202, you can set IOP to an accent grave character followed by the ien value of 202 before the call to ^%ZIS. The following example illustrates the above call:

Example:
```
S IOP="`202" D ^%ZIS
```

Using the ien rather than device name may be useful when applications have the desired device stored as a pointer to the DEVICE file rather than as free text.

Output

IO: **If a device is successfully opened, IO is returned with the device $I value of the selected device. If an abnormal exit occurs, POP is returned with a positive numeric value and IO is returned as null. Because the returned value of IO can be changed, since December 1990 programmers have been advised to check for a positive value in POP rather for IO equal to null when determining if an abnormal exit occurred.**

IO(0): **HOME DEVICE**
Contains the $I value of the home device at the time of the call to the Device Handler. Since it is defined at the time of the call, there is obviously no restoration after the call.

IO(1,$I): **OPENED DEVICES**
This array contains a list of devices opened for the current job by the Device Handler. The first subscript of this array is "1". The second subscript is the $I value of the device opened. The data value is null. The Device Handler sets, kills, and checks the existence of IO(1,IO).

Note: This array should not be altered by applications outside of the Kernel.

IO("DOC"): **SPOOL DOCUMENT NAME**
If output has been sent to the spool device, this variable holds the name of the spool document that was selected.

IO("HFSIO"): **HOST FILE DEVICE IO**
This is defined by the Device Handler when a user queues to a file at the host operating system level (of a layered system) and selects a file name other than the default. This host file system device variable should have the same value as that stored in the variable IO. If IO("HFSIO") exists when the TaskMan interface is called, the interface will save IO("HFSIO") and IOPAR so that the scheduled task opens the appropriate host file.

IO("Q"): **OUTPUT WAS QUEUED**
If queuing is allowed (%ZIS["Q") and an output device for queuing is selected, this variable is returned with a value of 1: IO("Q")=1. Otherwise it will be undefined.

IO("S"): **SLAVED DEVICE**
When a slaved printer is selected, the Device Handler uses this variable to save the subtype specification for the home device so that the appropriate close printer logic may be executed with X ^%ZIS("C").

IO("SPOOL"): **SPOOLER WAS USED**
The existence of this variable indicates that output was sent to the spool device. It will exist temporarily, during spooling, and is killed upon normal exit.

IO("ZIO"): **TERMINAL SERVER PORT**
If %ZIS["L", both physical port and server names are returned in IO("ZIO") under DSM for OpenVMS and

MSM-DOS. This information is useful on M implementations where the value of $I does not represent a port on a terminal server.

IOBS: BACKSPACE
The code for backspace, usually $C(8), is returned in this variable. This code is used to write a backspace with W @IOBS.

IOCPU: CPU INDICATOR
If the selected device is on another CPU, this variable will be returned with the other CPU reference, obtained from the VOLUME SET (CPU) field in the DEVICE file. IOCPU is used by TaskMan as an indicator of where the job should ultimately be run.

IOF: FORM FEED
This variable is used to issue a form feed when writing its value with indirection; that is, W @IOF.

IOHG: HUNT GROUP
If the selected device is a member of a hunt group, this variable will return the hunt group name. If not, IOHG is returned as null.

IOM: RIGHT MARGIN
The right margin is commonly set to either 80 or 132 columns.

ION: DEVICE NAME
This variable returns the device name (.01 field) as recorded in the DEVICE file.

IOPAR: OPEN PARAMETERS
This variable returns any Open Parameters that may have been defined for the selected device, for example, a magnetic tape drive. If the open parameters variable has not been defined, IOPAR is returned as null.

IOUPAR: USE PARAMETERS
This variable returns any Use Parameters that may have been defined for the selected device. If the use parameters variable has not been defined, IOUPAR is returned as null.

IOS: DEVICE NUMBER
The DEVICE file internal entry number for the selected device.

IOSL: **SCREEN/PAGE LENGTH**
The number of lines per screen or page is defined with this variable. The page length of a printing device is usually 66 lines. The screen length of a display terminal is usually 24 lines.

IOST: **SUBTYPE NAME**
This variable returns the name (.01 field) of the selected device's subtype as recorded in the TERMINAL TYPE file.

IOST(0): **SUBTYPE NUMBER**
This variable returns the internal entry number of the selected device's subtype as recorded in the TERMINAL TYPE file.

IOT: **TYPE OF DEVICE**
The DEVICE file holds an indication of Type for all devices. IOT returns the value of the device type (for example, TRM for terminal, VTRM for virtual terminal, and HFS for host file server).

IOXY: **CURSOR POSITIONING**
This variable returns the executable M code that allows cursor positioning, given the input variables DX and DY. The column position is passed in DX and the row position is passed in DY.

Note: The system special variables $X and $Y are not necessarily updated.

POP: **EXIT STATUS**
When the Device Handler is called, POP is the variable that indicates the outcome status. If device selection is successful, POP is returned with a value of zero (POP=0). Abnormal exit returns a positive number in the variable POP.

There are three general conditions for abnormal exit upon which the POP output variable is returned as positive. The first case is one in which a device is not selected. The second concerns unavailable devices. The third situation arises when a device is identified but is unknown to the system.

The first condition of no device selection will be met if the user types an up-arrow (^) or times out at the device prompt. Exceeding the timed read at the right margin or address/parameters prompts will have the same result.

The second condition, unavailability, is met if the Device Handler cannot open the selected device. If the device is a member of a hunt group, all members of the group may be busy. The selected device may also have existed on another computer but queuing was not requested or perhaps not permitted (%ZIS had not contained Q).

Finally, the selected device may not exist in the DEVICE file. A device name may have been used that is not found as a .01 field entry. If the device is selected with P for the closest printer, the CLOSEST PRINTER field in the DEVICE file may be null.

If the exit is abnormal, returning POP with a positive value, the following output variables will be restored with their values before the call to the Device Handler (before D ^%ZIS): ION, IOF, IOSL, IOBS, IOST(0), IOST, IOPAR, IOUPAR, IOS, and IOCPU. Note that if IOF had been null before the call, it is returned with the pound sign as its value (IOF="#"). For backward compatibility, IO is currently returned as null (IO=""). However, the returned value of IO may change in future Kernel versions.

Description

Device selection can be done as in the following sample routine. This is a simplified example; the process of issuing form feeds is not shown. See the Form Feed section of the Special Device Issues chapter for a discussion of form feeds.

```
SAMPLE   ;SAMPLE ROUTINE
   ;
   S %ZIS="QM" D ^%ZIS G EXIT:POP
   I $D(IO("Q")) D  Q
   .S ZTRTN="DQ^SAMPLE",ZTDESC="Sample Test routine"
   .D ^%ZTLOAD D HOME^%ZIS K IO("Q") Q
DQ U IO W !,"THIS IS YOUR REPORT"
   W !,"LINE 2"
   W !,"LINE 3"
   D ^%ZISC
EXIT     S:$D(ZTQUEUED) ZTREQ="@" K VAR1,VAR2,VAR3 Q
```

The variable IOP may be defined to pass a string to the Device Handler so that no user interaction will be required for device selection information. The following is the general format for defining IOP:

```
S IOP=[Q[;]][DEVICE NAME][;SUBTYPE][;SPOOL DOCUMENT NAME][;RIGHT
   MARGIN[;PAGE LENGTH]]
```

If the SPOOL DOCUMENT NAME is included, then the RIGHT MARGIN and PAGE LENGTH will be ignored. Therefore, use the following format if a spool device is desired:

```
S IOP=[Q[;]][DEVICE NAME][;SUBTYPE][;SPOOL DOCUMENT NAME]
```

The following shows how a device named "RXPRINTER" in the DEVICE file (#3.5) can be opened without user interaction:

```
S IOP="RXPRINTER" D ^%ZIS Q:POP
```

When setting IOP, you may include the right margin:

```
S IOP=ION_";"_IOM  or  S IOP=";120"
                or
S IOP="RXPRINTER;120"
```

In the example above, ION is the local variable that contains the name of the device to be opened and IOM contains the value of the desired right margin.

The IOP variable can be set to force queuing by starting the string with "Q":

```
SET IOP="Q;"_ION_";"_IOM  ... etc.
```

In order to force queuing and prompt the user for a device:

```
SET IOP="Q" D ^%ZIS Q:POP
```

A spool document name may be passed to the Device Handler (see the Spooling chapter for more information):

```
S IOP=DEVNAM_";"_IO("DOC") D ^%ZIS Q:POP
                or
S IOP="SPOOL;"_IO("DOC")
                or
S IOP=DEVNAM_";"_IOST_";"_IO("DOC")
                or
S IOP="SPOOL;P-OTHER;MYDOC"
```

In the example above, DEVNAM contains the name of the device to be opened. IO("DOC") contains the spool document name, and IOST contains the name of the desired subtype. "SPOOL" is the actual name of a device entry that corresponds to the spool device, "P-OTHER" is the desired subtype, and "MYDOC" is the name of the spool document.
Finally, IOP can now be used to select a device by the device's internal entry number (ien). To select a device with an ien of 202, set IOP to an accent grave character followed by the ien value of 202:

```
S IOP="`202" D ^%ZIS
```

Multiple Devices and ^%ZIS

Beyond the home device, the ^%ZIS entry point is not designed to open more than one additional device at a time.

For interactive users, the home device should already be open and defined in the Kernel environment. ^%ZIS should only be used to open one additional device at a time for interactive users. For a task, you can use ^%ZIS to open one additional device beyond the task's assigned device.

Beginning with Kernel V. 8.0, there are three entry points to support using more than one additional device simultaneously: OPEN^%ZISUTL, USE^%ZISUTL, and CLOSE^%ZISUTL. These "multiple device" entry points are described later in this chapter.

Host Files and ^%ZIS

Beginning with Kernel V. 8.0, there is a new API for working with host files. Although it is possible to use the ^%ZIS entry point to manipulate host files, the new host file API (in ^%ZISH) offers more robust host file functionality. See the Host Files chapter for more information on using the new host file API.

• HLP1^%ZIS: Display Brief Device Help

Usage `D HLP1^%ZIS`

Input **none**

Output **none**

Description

Use this entry point to display brief help about device selection. There are no input variables.

While invoking the Help Processor involves a straightforward call in the production account (the EN^XQH or EN1^XQH calls), it is a more complex matter in the Manager account where ^%ZIS resides. Hence, this call is provided.

• HLP2^%ZIS: Display Device Help Frames

Usage `D HLP2^%ZIS`

Input **none**

Output **none**

Description

You can display extended help about device selection by calling this entry point. The Help Processor is invoked to display a series of help frames. There are no input variables.

While invoking the Help Processor involves a straightforward call in the production account (the EN^XQH or EN1^XQH calls), it is a more complex matter in the Manager account where ^%ZIS resides. Hence, this call is provided.

• HOME^%ZIS: Reset Home Device IO Variables

Usage `D HOME^%ZIS`

Input **none**

Output

IO:	**Device $I.**
IO(0):	**Home device at the time of the call to ^%ZIS.**
IOBS:	**Backspace code.**
IOF:	**Form Feed code.**
IOM:	**Right Margin length.**
ION:	**Name of last selected input/output device from the DEVICE file.**
IOS:	**Internal entry number of last selected input/output device from the DEVICE file.**
IOSL :	**Screen or Page Length.**
IOST:	**Subtype of the selected device.**
IOST(0):	**Subtype internal entry number.**

IOT: Type of device, such as TRM for terminal.

IOXY: Executable M code for cursor control.

Description

Use the HOME^%ZIS entry point to set the key IO variables to match the characteristics of the home device. HOME^%ZIS performs the same function as the obsolete CURRENT^%ZIS entry point. Developers have been advised that Kernel V. 8.0 is the last version of Kernel to support CURRENT^%ZIS.

HOME^%ZIS, beyond updating the set of variables for the home device, also updates the active right margin system setting for the home device, by executing ^%ZOSF("RM") based on the home device's IOM value.

• RESETVAR^%ZIS: Reset Home Device IO Variables

Usage `D RESETVAR^%ZIS`

Input none

Output (various) RESETVAR^%ZIS outputs the same variables as the HOME^%ZIS function. See HOME^%ZIS for the complete list of output variables.

Description

A call to this function sets the key IO variables to match the characteristics of the home device. It is similar to HOME^%ZIS, except that the active right margin setting for the home device is not changed; that is, ^%ZOSF("RM") is not executed. Use this call in situations where you want to update the IO* variables for the home device but do not want to update the active right margin setting.

• $$REWIND^%ZIS: Rewind Devices

Usage `S X=$$REWIND^%ZIS(io,iot,iopar)`

Input	**io:**	**The $IO representation of the device to be rewound, in the same format as IO, which is returned by ^%ZIS.**
	iot:	**The "Type" of device to be rewound, in the same format as IOT, which is returned by ^%ZIS.**
	iopar:	**The "Open Parameters" for the selected device, in the same format as IOPAR which is returned by ^%ZIS.**
Output	**return value:**	**1 if the device was rewound successfully; 0 if the device was not rewound successfully.**

Description

Programmers can use this extrinsic function to rewind special devices. These devices may be of the following types:

- **Magtape**
- **Sequential Disk Processor**
- **Host File Server**

Example

```
S Y=$$REWIND^%ZIS(IO,IOT,IOPAR)
```

• ^%ZISC: Close Device

Usage `D ^%ZISC`

Input **IONOFF:** **No form feed**
The existence of this variable instructs the Device Handler not to issue a form feed when closing the device.

Output **None**

Description

Use the ^%ZISC entry point to close a device opened with a call to the ^%ZIS entry point.

Do not issue a form feed when calling ^%ZISC. The Device Handler takes care of issuing a form feed if necessary: that is, if $Y>0, indicating the cursor or print head is not at the top of form. To prevent the Device Handler from issuing this form feed, as appropriate for continuous printing of labels, for example, define the IONOFF variable before calling ^%ZISC.

Before the ^%ZISC entry point existed, close logic was executed with the command X ^%ZIS("C"). Developers have been advised that X ^%ZIS("C") will no longer be supported and that the ^%ZISC entry point should be used instead. In the current version of Kernel, the ^%ZIS("C") node only holds a call to the ^%ZISC routine. Versions of Kernel following V. 8.0 will not export ^%ZIS("C").

• PKILL^%ZISP: Kill Special Printer Variables

Usage D PKILL^%ZISP

Input **none**

Output **none**

Description

Use PKILL^%ZISP to kill printer-specific Device Handler variables. All output variables defined by the PSET^%ZISP entry point are killed.

• PSET^%ZISP: Set Up Special Printer Variables

Usage D PSET^%ZISP

Input	**IOST(0)**	**Pointer to the TERMINAL TYPE entry for the printer in question, as set up by the Device Handler.**
Output	**IOBAROFF:**	**Bar code off.**
	IOBARON:	**Bar code on.**
	IOCLROFF:	**Color off.**
	IOCLRON:	**Color on.**
	IODPLXL:	**Duplex, long edge binding.**
	IODPLXS:	**Duplex, short edge binding.**
	IOITLOFF:	**Italics off.**
	IOITLON:	**Italics on.**
	IOSMPLX:	**Simplex.**
	IOSPROFF:	**Superscript off.**
	IOSPRON:	**Superscript on.**
	IOSUBOFF:	**Subscript off.**
	IOSUBON:	**Subscript on.**

Description

Use PSET^%ZISP to define a set of variables that toggle special printer modes. The corresponding fields in the TERMINAL TYPE file entry for the terminal type in question must be correctly set up, however; that is where PSET^%ZISP retrieves its output values.

To toggle a printer mode with one of PSET^%ZISP's output variables, write the variable to the printer using indirection, as follows:

```
D PSET^%ZISP
W @IOBARON
```

• ENDR^%ZISS: Set Up Specific Screen Handling Variables

Usage `D ENDR^%ZISS`

Input

IOST(0): **Internal entry number of the selected device's subtype as recorded in the TERMINAL TYPE file.**

X: **Use this variable to select the ENS^%ZISS screen handling variables to define. It should be a semicolon-delimited list of the variables to define. For example:**

```
S X="IORVON;IORVOFF;IOUON;IOUOFF"
```

If more than 255 characters are needed to define the variable X, make two or more calls to ENDR^%ZISS, each with a partial list of the parameter settings for X.

%ZIS: **(optional) If you define %ZIS="I", the output array IOIS is created. The format of IOIS is as follows: IOIS(ASCII value of first character followed by remaining characters)=output variable; (e.g., IOIS("27[C")=IOCUF). Not every screen handling variable has a corresponding IOIS node. Also, only the nodes in the IOIS array that correspond to screen handling variables specified in the X input variable will be created.**

Output **A subset of the variables returned by ENS^%ZISS are returned by ENDR^%ZISS, depending on what screen handling variables are requested in the input variable X.**

Description

Use ENDR^%ZISS to set up specific screen handling variables and other terminal type attributes. Unlike the ENS^%ZISS entry point, which sets up all screen handling variables, you specify which ones to set up with ENDR^%ZISS.

• ENS^%ZISS: Set Up Screen Handling Variables

Used for screen management. This entry point sets up screen handling variables and other terminal type attributes.

Usage `D ENS^%ZISS`

Input

IOST(0): **Internal entry number of the selected device's subtype as recorded in the TERMINAL TYPE file.**

%ZIS: **(optional) If you define %ZIS="I", the output array IOIS (mapping escape codes sent by input keys to input keys) is created. See below for a description of the IOIS nodes created.**

Output **The output variables are listed below. Note that not all characteristics are possible on all terminal types. Also note that two of the parameters are used with indirection, IOEFLD and IOSTBM (marked with * below). Furthermore, IOSTBM requires the setting of IOTM and IOBM as input variables for the top and bottom margins.**

IOARM0	**Auto repeat mode off.**
IOARM1	**Auto repeat mode on.**
IOAWM0	**Auto wrap mode off.**
IOAWM1	**Auto wrap mode on.**
IOBOFF	**Blink off.**
IOBON	**Blink on.**
IOCOMMA	**Keypad's comma.**
IOCUB	**Cursor backward.**
IOCUD	**Cursor down.**
IOCUF	**Cursor forward.**
IOCUON	**Cursor on.**
IOCUOFF	**Cursor off.**
IOCUU	**Cursor up.**
IODCH	**Delete character.**
IODHLB	**Double high/double wide bottom.**
IODHLT	**Double high/wide top.**
IODL	**Delete line.**
IODWL	**Double wide length.**
IOECH	**Erase character.**
IOEDALL	**Erase in display entire page.**
IOEDBOP	**Erase in display from beginning of page to cursor.**
IOEDEOP	**Erase in display from cursor to end of page.**
IOEFLD	**Erase field (*use through indirection, e.g., W @IOEFLD).**
IOELALL	**Erase in line entire line.**
IOELBOL	**Erase in line from beginning of line to cursor.**

IOELEOL	**Erase in line from cursor to end of line.**
IOENTER	**Keypad's Enter.**
IOFIND	**Find key.**
IOHDWN	**Half down.**
IOHOME	**Home cursor.**
IOHTS	**Horizontal tab set.**
IOHUP	**Half up.**
IOICH	**Insert character.**
IOIL	**Insert line.**
IOIND	**Index.**
IOINHI	**High intensity.**
IOINLOW	**Low intensity.**
IOINORM	**Normal intensity.**
IOINSERT	**Insert key.**
IOKP0	**Keypad 0.**
IOKP1	**Keypad 1.**
IOKP2	**Keypad 2.**
IOKP3	**Keypad 3.**
IOKP4	**Keypad 4.**
IOKP5	**Keypad 5.**
IOKP6	**Keypad 6.**
IOKP7	**Keypad 7.**
IOKP8	**Keypad 8.**
IOKP9	**Keypad 9.**
IOIRM0	**Replace mode.**
IOIRM1	**Insert mode.**
IOKPAM	**Keypad application mode on.**
IOKPNM	**Keypad numeric mode on.**
IOMC	**Print screen.**
IOMINUS	**Keypad's minus.**
IONEL	**Next line.**
IONEXTSC	**Next screen.**
IOPERIOD	**Keypad's period.**
IOPF1	**Function key 1.**
IOPF2	**Function key 2.**
IOPF3	**Function key 3.**
IOPF4	**Function key 4.**
IOPREVSC	**Previous screen.**
IOPROP	**Proportional spacing.**
IOPTCH10	**10 Pitch.**
IOPTCH12	**12 Pitch.**
IOPTCH16	**16 Pitch.**
IORC	**Restore cursor.**
IOREMOVE	**Keypad's Remove.**
IORESET	**Reset.**
IORI	**Reverse index.**
IORLF	**Reverse line feed.**
IORVOFF	**Reverse video off.**
IORVON	**Reverse video on.**

IOSC	**Save cursor.**
IOSGR0	**Turn off select graphic rendition attributes.**
IOSELECT	**Keypad's Select.**
IOSTBM	**Set top and bottom margins (*use through indirection, e.g., W @IOSTBM; IOTM and IOBM must be defined as the top and bottom margins.)**
IOSWL	**Single wide length.**
IOTBC	**Tab clear.**
IOTBCALL	**Clear all tabs.**
IOUOFF	**Underline off.**
IOUON	**Underline on.**

IOIS **The IOIS array is created as follows:**

```
IOIS(escape_code)=KEYNAME
```

where escape_code is the escape code generated by pressing the key KEYNAME on the selected terminal, and KEYNAME can be one of the following:

COMMA	**KP5**
DO	**KP6**
ENTER	**KP7**
FIND	**KP8**
HELP	**KP9**
INSERT	**MINUS**
IOCUB	**NEXTSCRN**
IOCUD	**PERIOD**
IOCUF	**PF1**
IOCUU	**PF2**
KP0	**PF3**
KP1	**PF4**
KP2	**PREVSCRN**
KP3	**REMOVE**
KP4	**SELECT**

• GKILL^%ZISS: Kill Graphic Variables

Used for screen management, to kill graphic variables used in screen handling. All output variables set up by the GSET^%ZISS call are killed.

Usage `D GKILL^%ZISS`

Input **none**

Output **none**

• GSET^%ZISS: Set Up Graphics Variables

Usage `D GSET^%ZISS`

Input	**IOST(0)**	**Terminal Type.**
Output	**IOBLC:**	**Bottom left corner.**
	IOBLC:	**Bottom left corner.**
	IOBRC:	**Bottom right corner.**
	IOBT:	**Bottom "T".**
	IOG1:	**Graphics on.**
	IOG0:	**Graphics off.**
	IOHL:	**Horizontal line.**
	IOLT:	**Left "T".**
	IOMT:	**Middle "T", or cross hair (+).**
	IORT:	**Right "T".**
	IOTLC:	**Top left corner.**
	IOTRC:	**Top right corner.**
	IOTT:	**Top "T".**
	IOVL:	**Vertical line.**

Description

Used for screen management, this function sets up graphic variables for screen handling. Graphics on/off is a toggle that remaps characters for use as graphics. Not all terminals need remapping (like DataTree consoles, for example) since they already have the high range of ASCII codes.

Example

```
; write a horizontal line
D GSET^%ZISS
W IOG1
F I=1:1:20 W IOHL
W IOG0
D GKILL^%ZISS
```

• KILL^%ZISS: Kill Screen Handling Variables

Usage `D KILL^%ZISS`

Input **none**

Output **none**

Description

For screen management, use this function to kill graphic variables used in screen handling. Only the variables set up by the ENS^%ZISS (and ENDR^%ZISS) calls are killed by this call.

• CLOSE^%ZISUTL: Close Device with Handle

One of three functions that support using multiple devices at the same time (see also: OPEN^%ZISUTL and USE^%ZISUTL).

Usage `D CLOSE^%ZISUTL(handle)`

Input **handle:** **The handle of a device opened with an OPEN^%ZISUTL call.**

Output **none**

Description

Use CLOSE^%ZISUTL to close a device opened with the OPEN^%ZISUTL function. When you close a device with CLOSE^%ZISUTL, the IO variables are set back to the home device's and the home device is made the current device.

• OPEN^%ZISUTL: Open Device with Handle

One of three functions that support using multiple devices at the same time (see also: USE^%ZISUTL and CLOSE^%ZISUTL).

Usage `D OPEN^%ZISUTL(handle,[valiop],[.valzis])`

Input

handle:	**A unique free text name to associate with a device you want to open.**
valiop:	**(optional) Output device specification, in the same format as the IOP input variable for the ^%ZIS entry point. The one exception to this is passing a value of null: this is like leaving IOP undefined. With ^%ZIS, on the other hand, setting IOP to null specifies the home device. To request the home device, pass a value of "HOME" instead.**
.valzis:	**(optional) Input specification array, in the same format (and with the same meanings) as the %ZIS input specification array for the ^%ZIS entry point. Must be passed by reference. Please see the documentation of the ^%ZIS function for more information.**

Output

IOF
IOM
IOSL
IO
IO(0)
IO("Q")
IO("S")
IO("DOC")
IO("SPOOL")
IO("ZIO")
IO("HFSIO")
IO(1,$I)
IOST
IOST(0)
IOT
ION
IOBS
IOPAR
IOUPAR
IOS
IOHG
IOXY
POP

OPEN^%ZISUTL returns all the same output variables as the ^%ZIS entry point. OPEN^%ZISUTL serves as a "wrapper" around the ^%ZIS entry point, providing additional management of IO variables that ^%ZIS does not (principally to support opening multiple devices simultaneously).

For more information on these output variables, please see the documentation for ^%ZIS.

Description

Use OPEN^%ZISUTL when you expect to be using multiple output devices. This entry point (as well as its two companion entry points, USE^%ZISUTL and CLOSE^%ZISUTL) makes use of handles to refer to a device. A handle is a unique string identifying the device.

The three ^%ZISUTL entry points are essentially wrappers around the ^%ZIS entry point. They provide enhanced management of IO variables and the current device, especially when working with multiple open devices.

Example

```
ZXGTMP  ; ISC-SF/doc %ZISUTL sample ;11-oct-94
        ;;1.0;;
EN      ;
        K A6AZIS S A6AZIS("A")="Enter the printer to output first 40
chars in each line: "
        D OPEN^%ZISUTL("PRT1","",.A6AZIS) Q:POP
        K A6AZIS S A6AZIS("A")="Enter the printer to output chars 41
to end of line: "
        D OPEN^%ZISUTL("PRT2","",.A6AZIS) I POP D CLOSE^%ZISUTL("PRT1
") Q
        S I=""  F  S I=$O(^TMP($J,"DOC",I)) Q:I']""  S X=^(I) D
        .D USE^%ZISUTL("PRT1") U IO W $E(X,1,40),!
        .D USE^%ZISUTL("PRT2") U IO W $E(X,41,$L(X)),!
        D CLOSE^%ZISUTL("PRT1"),CLOSE^%ZISUTL("PRT2")
        Q
```

• USE^%ZISUTL: Use Device with Handle

One of three functions that support using multiple devices at the same time (see also: OPEN^%ZISUTL and CLOSE^%ZISUTL).

Usage `D USE^%ZISUTL(handle)`

Input	**handle:**	**The handle of a device opened with an OPEN^%ZISUTL call.**
Output	**IO variables:**	**When you use a device with USE^%ZISUTL function, the IO variables for that device are restored.**

Description

Use USE^%ZISUTL to restore the IO variables for a device opened with the OPEN^%ZISUTL function and to make that device the current device.

Chapter 18 Host Files

User Interface

Host file server (HFS) devices allow you to send output to a file maintained by your computer's operating system, rather than to a printer. You can send your output to an HFS device, if such a device type has been established on the system. Depending upon how IRM defines the HFS device, you may be prompted for a host file name and for an input/output operation:

```
DEVICE: HFS <RET> DISK FILE
HOST FILE NAME: TMP.TMP// <RET>            INPUT/OUTPUT OPERATION: ? <RET>
Enter one of the following host file input/output operation:
               R = READONLY
               N = NEWVERSION
              RW = READ/WRITE
```

Not all input/output modes are available on all systems. The possible modes for input/output operation work as follows:

APPEND	**Data from a write operation will be appended to the file.**
MIXED	**Both reading and writing are allowed for the specified file.**
NEWVERSION	**A new file will be created with a higher version number; this file can be used for writing only.**
READ	**Reading is allowed from the specified file; writing is not allowed.**
READONLY	**Reading is allowed from the specified file; writing is not allowed.**
READ/WRITE	**Both reading and writing are allowed for the specified file; if a write operation is performed, output is appended to the file.**
WRITE	**Writing is allowed; output can be sent to the specified file.**

System Management

To provide access to host files through the Device Handler, set up device entries of type HFS.

There are three fields in an HFS device entry that act as flags for what a user must enter when they use an HFS device. The fields are:

- **ASK PARAMETERS: If this field is set to YES, the user must enter the correct M open parameters to open the device. This should be set to NO if the device is accessible to non-IRM users. If it is set to YES, the default value is the current value of the Open Parameters field.**
- **ASK HFS I/O OPERATION: If this field is set to YES, the user can choose what mode the file should be opened in (for example, read or write). If it is set to NO, files are opened in write mode. This should be set to NO if the device is accessible to non-IRM users, assuming that all such users would only need to write host files.**
- **ASK HOST FILE: Field is set to YES, the user can choose what file will be opened. If it is set to NO, the default file name built into the device entry is always used. This should be set to NO in most cases if the HFS device is accessible to non-IRM users, since host files can proliferate if many users are able to create many files, and also because an HFS device opens up access to the host operating system (and the potential for overwriting vital files).**

Host File Server Device Edit

```
Device Management...                                        [XUTIO]
   Edit Devices by Specific Types...                     [XUDEVEDIT]
      Host File Server Device Edit                     [XUDEVEDITHFS]
```

The Host File Server Device Edit option lets you to edit Host File Server device attributes using a ScreenMan form.

DSM for OpenVMS HFS Device Setup

DSM for OpenVMS requires the name of the host file to be part of the device $I and not part of the parameter list.

HFS I/O Operation Modes for DSM for OpenVMS

NEWVERSION	**A new file will be created with a higher version number; this file can be used for writing only.**
READONLY	**Reading is allowed from the specified file; writing is not allowed.**
READ/WRITE	**Both reading and writing are allowed for the specified file; if a write operation is performed, output is appended to the file.**

Host File Server Device for DSM for OpenVMS

```
                  Name:   HFS
                    $I:   TMP.TMP
                  Type:   HFS
        Ask Parameters:   NO
         Ask Host File:   NO
Ask HFS I/O Operation:   NO
      Open Parameters:   NEW
```

MSM-DOS HFS Device Setup

Micronetics' MSM-DOS reserves the $I of 51 through 54 for Host File Server Units. The actual name of the file to be referenced must be specified as part of the parameter list. The OPEN PARAMETERS field of the DEVICE file should be used to hold this information. When using the full path names to the file, a backward slash "\" is used under DOS as opposed to the forward slash "/" used under UNIX. The OPEN PARAMETERS field should contain the file name along with parameters and enclosing parentheses as it would appear in an M open command. Or it may just contain the file name without parameters, parentheses, or additional quotes. If the OPEN PARAMETERS contains the file name, the device handler defaults to write mode.

■ Host File Server Device for MSM

```
                  Name:   HFS1
                    $I:   51
                  Type:   HFS
        Ask Parameters:   NO
         Ask Host File:   NO
Ask HFS I/O Operation:   NO
      Open Parameters:   ("C:\MSM\MYDATA.DAT":"W")
```

■ HFS I/O Operation Modes for MSM-DOS

READ	**Reading is allowed from the specified file; writing is not allowed.**
WRITE	**Writing is allowed; output can be sent to the specified file.**
MIXED	**Both reading and writing are allowed for the specified file.**
APPEND	**Data from a write operation will be appended to the file.**

Programmer Tools

The traditional method of working with HFS files prior to Kernel V. 8.0 was to use the device handler entry point (^%ZIS). Using several input variables, you could open a host file (given a host file device entry in the DEVICE file). For example:

```
S %ZIS("HFSNAME")="ARCHIVE.DAT"
S %ZIS("HFSMODE")="W"
S IOP="HFS" D ^%ZIS Q:POP
U IO D...
```

Kernel V. 8.0 provides a new set of entry points for working with host files. The new host file entry points are:

- **CLOSE^%ZISH** — **Close host file opened by OPEN^%ZISH.**
- **$$DEL^%ZISH** — **Delete host file.**
- **$$FTG^%ZISH** — **Copy lines from a host file into a global.**
- **$$GATF^%ZISH** — **Append records from a global to a host file.**
- **$$GTF^%ZISH** — **Copy records from a global into a host file.**
- **$$LIST^%ZISH** — **Retrieve a list of files in a directory.**
- **$$MV^%ZISH** — **Rename host file.**
- **OPEN^%ZISH** — **Open host file (bypass device handler).**
- **$$PWD^%ZISH** — **Retrieve name of current directory.**
- **$$STATUS^%ZISH** — **Return end-of-file status.**

The following definitions apply for the new host file entry points:

Path: **Full path specification up to, but not including, the filename. This includes any trailing slashes or brackets. If the operating system allows shortcuts, you can use them. Examples of valid paths include:**

DOS	**c:\scratch**
UNIX	**/home/scratch/**
VMS	**USER$:[SCRATCH]**

To specify the current directory, use a path of null ("").

Filename: **Filename of the file only. Do not include device or directory specifications.**

Access mode: **Access mode when opening files. It can be one of the following codes:**

"**R**" **Read, use the file for reading only.**

"W" **Write, use the file for writing. If the file exists, it is truncated to a length of 0 first. If the file doesn't exist, it is created.**

"A" **Append, use the file for writing but start writing at the end of the current file. If the file doesn't exist, it is created.**

• CLOSE^%ZISH: Close Host File

Format `D CLOSE^%ZISH(handle)`

Input **handle:** **Handle used when file was opened with OPEN^%ZISH call.**

Output **none**

Description

Use this entry point to close a host file that was opened with the OPEN^%ZISH entry point.

Example

```
D OPEN^%ZISH("OUTFILE","USER$:[ANONYMOUS]","ARCHIVE.DAT","W")
Q:POP
U IO F I=1:1:100 W I,": ",ARRAY(I),!
D CLOSE^%ZISH("OUTFILE")
```

• $$DEL^%ZISH: Delete Host File

Format `S Y=$$DEL^%ZISH(path,arrname)`

Input

path: **Full path, up to but not including the filename.**

arrname: **Fully resolved array *name* containing the files to delete as subscripts at the next descendant subscript level. For example, to delete two files, FILE1.DAT and FILE2.DAT, set up the array as:**

```
ARRAY("FILE1.DAT")=""
ARRAY("FILE2.DAT")=""
```

and pass the array name "ARRAY" as the arrname parameter. Wildcard specifications cannot be used with this function.

Output

return value: **1=success for all deletions.**
0=failure on at least one deletion.

Description

Use this function to delete host files. You can delete one or many host files, depending on how you set up the array whose name you pass as the second input parameter.

Example

```
K FILESPEC
S FILESPEC("TMP.DAT")=""
S Y=$$DEL^%ZISH("\MYDIR\",$NA(FILESPEC))
```

• $$FTG^%ZISH: Load File into Global

Format

```
S Y=$$FTG^%ZISH(path,filename,global_ref,inc_subscr
  [,ovfsub])
```

Input

path: **Full path, up to but not including the filename.**

filename: **Name of the file to open.**

global_ref: **Global reference to write host file to, in fully resolved (closed root) format. This function does not kill the global before writing to it.**

At least one subscript must be numeric. This will be the incrementing subscript, i.e., the subscript that $$FTG^%ZISH will increment to store each new global node. This subscript need not be the final subscript. For example, to load into a word processing field, the incrementing node is the second-to-last subscript; the final subscript is always zero.

inc_subscr: **Identifies the incrementing subscript level. For example, if you pass ^TMP(115,1,1,0) as the global_ref parameter and pass 3 as the inc_subscr parameter, $$FTG^%ZISH will increment the third subscript (e.g., ^TMP(115,1,x)), but will write nodes at the full global reference (e.g., ^TMP(115,1,x,0)).**

ovfsub: **[optional] Name of subscript level at which overflow nodes for lines (if any) should be stored. Overflows occur if a line is greater than 255 characters. Further overflows occur for every additional 255 characters. The default subscript name at which overflows are stored for a line is "OVF".**

Output

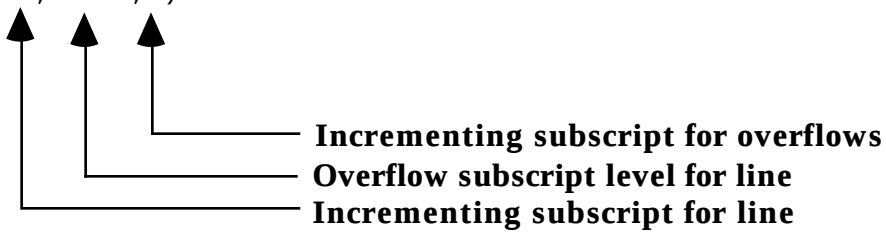
return value **1 = success; 0 = failure.**

Description

Use this function to load a host file into a global. Each line of the host file becomes the value of one node in the global. You do not need to open the host file before making this call; it is opened and closed by $$FTG^%ZISH.

If a line from a host file exceeds 255 characters in length, the overflow(s) are stored in overflow nodes for that line, as follows:

```
^TMP($J,35,0)="1st 255 of host file line..."
^TMP($J,35,"OVF",1)="next 255 chars of host file line..."
^TMP($J,35,"OVF",2)="next 255 characters of line etc."
```



Example

```
S Y=$$FTG("USER$:[COMMON]","MYFILE.DAT",$NA(^MYGLOBAL(612,1,0)),2)
```

• $$GATF^%ZISH: Copy Global to Host File

Format `S Y=$$GATF^%ZISH(global_ref,inc_subscr,path,filename)`

Input **Same as input for $$GTF^%ZISH.**

Output **Same as output for $$GTF^%ZISH.**

Description

Use this function in the same way as the $$GTF^%ZISH entry point; the one difference is that if the file already exists, $$GATF^%ZISH appends global nodes to the existing file rather than truncating the existing file first.
For more information, see the description of $$GTF^%ZISH.

• $$GTF^%ZISH: Copy Global to Host File

Format `S Y=$$GTF^%ZISH(global_ref,inc_subscr,path,filename)`

Input

global_ref: **Global to read lines from, fully resolved in closed root form.**

inc_subscr: **Identifies the incrementing subscript level. For example, if you pass ^TMP(115,1,1,0) as the global_ref parameter, and pass 3 as the inc_subscr parameter, $$GTF will increment the third subscript (e.g., ^TMP(115,1,x)), but will read nodes at the full global reference (e.g., ^TMP(115,1,x,0)).**

path: **Full path, up to but not including the filename.**

filename: **Name of the file to open.**

Output

return value: **1=success; 0=failure.**

Description

Use $$GTF^%ZISH to write the values of nodes in a global (at the subscript level you specify) to a host file. If the host file already exists, it is truncated to length 0 before the copy. You do not need to open the host file before making this call. The host file is opened (in Write mode) and closed by $$GTF^%ZISH.

Example

```
S Y=$$GTF($NA(^MYGLOBAL(612,1,0)),2,"USER$:[COMMON]","MYFILE.DAT")
```

• $$LIST^%ZISH: List Directory

Format `S Y=$$LIST^%ZISH(path,arrname,retarrnam)`

Input

path: **Full path, up to but not including any filename. For current directory, pass the null string.**

arrname: **Fully resolved array *name* containing file specifications to list at the next descendant subscript level.**

For example, to list all files, set one node in the named array, at subscript "*", equal to null. To list all files beginning with "E" and "L", using the array ARRAY, set the nodes:

```
ARRAY("E*")=""
ARRAY("L*")=""
```

and pass the name "ARRAY" as the arrname parameter. You can use the asterisk wildcard in the file specification.

retarrnam: **Fully resolved array *name* to return the list of matching filenames. You should ordinarily kill this array first (it is not purged by LIST^%ZISH).**

Output

return value: **1=success; 0=failure.**

retarrnam: **$$LIST^%ZISH populates the array named in the third input parameter with all matching files it finds in the directory you specify. It populates the array in the format:**

```
ARRAY("filename1")=""
ARRAY("filename2")=""
  (etc.)
```

Description

Use this function to return a list of file names in the current directory. The list is returned in an array in the variable named by the third parameter.

Example

```
K FILESPEC,FILE
S FILESPEC("L*")="",FILESPEC("P*")=""
S Y=$$LIST^%ZISH("","FILESPEC","FILE")
```

• $$MV^%ZISH: Rename Host File

Format `S Y=$$MV^%ZISH(path1,filename1,path2,filename2)`

Input

path1:	**Full path of the original file, up to but not including the filename.**
filename1:	**Name of the original file.**
path2:	**Full path of renamed file, up to but not including the filename.**
filename2:	**Name of the renamed file.**

Output

return value:	**1=success; 0=failure.**

Description

Use this function to rename a host file. The function performs the renaming, regardless of the underlying operating system, by first copying the file to the new name/location and then deleting the original file at the old name/location.

Example

```
S Y=$$MV^%ZISH("","TMP.DAT","","ZXG"_I_".DAT")
```

• OPEN^%ZISH: Open Host File

Format `D OPEN^%ZISH(handle,path,filename,mode)`

Input	**handle:**	**Unique name you supply to identify the opened device.**
	path:	**Full path, up to but not including the filename.**
	filename:	**Name of the file to open.**
	mode:	**Mode to open file: "W" for write, "R" for read, "A" for append.**
Output	**POP:**	**A value of 0 means the file was opened successfully; a positive value means the file was not opened.**
	IO:	**Name of the opened file in the format to use for M USE and CLOSE commands.**

Description

OPEN^%ZISH opens a host file without using the device handler. You can USE the device name returned in IO. You can then READ and WRITE from the opened host file (depending on what access mode you used to open the file).

To close the host file, use the CLOSE^%ZISH entry point with the handle you used to open the file.

Example

```
D OPEN^%ZISH("FILE1","USER$:[ANONYMOUS]","ARCHIVE.DAT","A")
Q:POP
U IO F I=1:1:100 W I,": ",ARRAY(I),!
D CLOSE^%ZISH("FILE1")
```

• $$PWD^%ZISH: Retrieve Current Directory

Format S Y=$$PWD^%ZISH

Input **none**

Output

return value:	**Returns the string representing the current directory specification, including device if any. If a problem occurs while retrieving the current directory, the null string is returned.**

Description

Use this function to retrieve the name of the current working directory.

Example

```
S Y=$$PWD^%ZISH()
```

• $$STATUS^%ZISH: Return End-of-File Status

Format S Y=$$STATUS^%ZISH

Input **none**

Output

return value:	**1 if end-of-file has been reached; 0=if end-of-file has not been reached.**

Description

$$STATUS^%ZISH returns the current end-of-file status. If end-of-file has been reached, $$STATUS^%ZISH returns 1. Otherwise, it returns 0.

Example

```
D OPEN^%ZISH("INFILE","USER$:[ANONYMOUS]","ZXG.DAT","R")
Q:POP
U IO F I=1:1 R X:DTIME Q:$$STATUS^%ZISH  S ^TMP($J,"ZXG",I)=X
D CLOSE^%ZISH("INFILE")
```

Chapter 19 Spooling

User Interface

Spooling privileges may be granted by IRM to users who prepare and manage reports. By sending your output to the spooler, rather than to a printer, you can benefit in several ways. Since spooling saves the output on-line in a holding area, you can easily print multiple copies of the report at a later time. Spooling is also a good way to store the results of a time-consuming calculation, such as a complex VA File Manager report. By queuing to the spooler, a report involving intensive processing can be done at night when the system is relatively free. Output can then be printed during the day when the printer can be attended. Finally, when using the spooler, report processing can run to completion without printer problems interfering.

Sending Output to the Spooler

If you have been given the authority to spool, you can send output to the spooler by responding to the "DEVICE:" prompt with the name of the spool device. Devices used for spooling are commonly named SPOOL or SPOOLER.

If you don't have spooling privileges and you try to use the spool device, the spooler issues a message that authority has not been granted, as below:

```
DEVICE: SPOOL <RET>

     You aren't an authorized SPOOLER user.
```

To send output to the spooler with a customized right margin of 96 and page length of 66, you can use the following syntax:

```
DEVICE: SPOOL;96;66 <RET>
```

After requesting the spool device, you are usually prompted for a spool document name, as shown below. The prompt is not issued, however, if the spool device has been set up to generate the spool document name itself.

```
DEVICE: SPOOL <RET>

Select SPOOL DOCUMENT NAME:
```

To skip the SPOOL DOCUMENT NAME prompt, you can specify the spool document name at the "DEVICE:" prompt by entering the name in the second semicolon piece. A name entered here is not used if the spooler is set up to

generate names itself, however. Because of the format used, the Device Handler knows that a spool document name, rather than a device subtype, is being specified. Subtypes begin with one or two letters followed by a dash (e.g., P-DEC), while spool document names cannot.

```
DEVICE: SPOOL;MYDOC

DEVICE: SPOOL;P-OTHER80;MYDOC
```

If the computing environment is composed of several networked processors, you may need to specify where spooling should take place. The spooler on the current CPU should be chosen unless the output is queued.

```
DEVICE: SPOOL <RET>
  1   SPOOL AAA
  2   SPOOL BBB
Choose 1-2>
```

If the output is queued, you can choose a spooler on another CPU and a time to schedule the job to run.

```
DEVICE: Q <RET>
DEVICE: SPOOL BBB <RET>
```

■ Summary of Spooler Parameters at Device Prompt

```
DEVICE:  Spooler

DEVICE:  Spooler;Right Margin;Page Length

DEVICE:  Spooler;Subtype

DEVICE:  Spooler;Spool Document Name

DEVICE:  Spooler;Subtype;Spool Document Name
```

Retrieving Spooled Documents

After a spool document has been created, you can retrieve the output by using options on the Spooler Menu. This menu is distributed as part of the Kernel's Common menu, a menu available to all users. Specifically, the Spooler Menu is in your User's Toolbox (to quickly reach the Toolbox, or any other option on the Common menu, you can enter a quotation mark plus the menu text or synonym):

```
Select Primary Menu Option: "TBOX <RET>

Select User's Toolbox Option: SP<RET>ooler Menu

Select Spooler Menu Option: ? <RET>

          Allow other users access to spool documents
          Browse a Spool Document
          Delete A Spool Document
          List Spool Documents
          Make spool document into a mail message
          Print A Spool Document
```

The List Spool Documents option lists any documents that you have created. Other users cannot read or print these documents unless you have authorized them to with the "Allow other users access to spool documents" option, also on the Spooler menu.

To delete spool documents, you can use the Delete A Spool Document option. Since there is a limit on the amount of spool space that any one user may consume, you may need to delete old spool documents to free up space for new ones. If you attempt to create a new document when the space limits have been exceeded, the spooler issues a message about the need to delete some documents.

Old documents are deleted automatically, on a schedule as determined by IRM. The "life span" of a spool document is a site parameter that IRM controls. IRM should inform you of the life span of spooled documents so that you are not surprised when old documents are purged.

Browsing a Spool Document

With the Browse a Spool Document option, you can view spool documents with VA FileMan's Browser. The Browser allows you to view spool documents on your terminal screen, letting you scroll backward and forward through the report, and also letting you perform simple searches within the report. In the past, when you sent a report to the terminal screen, you could only go in one direction, forward, by pressing <RET> after each screenful of text. For more information on using the Browser, please see the VA FileMan User Manual (starting with version 21).

Printing Spool Documents

You can print spool documents with the Print A Spool Document option. Before selecting an output device, you are prompted for the number of copies to print. If you have been granted the ability to print to multiple devices, you can send your output to several devices for simultaneous printing. If this privilege has been granted to you, the device prompt is displayed again after you choose the first printer. Entering a null response to the second device prompt tells the spooler not to use any more additional printers.

To save users the time and trouble of despooling their documents, IRM can set up a spool device for auto-despooling. If you invoke such a spool device, the spool document is sent to one or more printers when the spooling process has completed. After automatic printing, the spool document remains available for reprinting as necessary (it is not automatically deleted upon despooling).

Making Spool Documents into Mail Messages

You can also be granted the ability to make spool documents into mail messages. If so, the corresponding option on the Spooler Menu will be available. You can use it to make documents into regular mail messages that can then be edited, copied, or forwarded just like other messages. After the text has been moved into a mail message, the spool document is deleted. The deletion is to allow space for new spool documents.

If you plan to make a document into a message, you should do the original output to the spool device with an appropriate margin and page length for a MailMan message. Since MailMan breaks incoming text lines at about the 75th character, a right margin of 75 may be desirable. Indicating that page breaks should not be inserted during the spooling process may also be desirable. Otherwise, the VA FileMan window command |TOP| is inserted into the text at the beginning of each page. While this automatic formatting is an advantage when printing spool documents, it is a disadvantage when creating a mail message. Page breaks will not be inserted when indicating a page length of 99999 lines, or a number greater than the document's total. So when you know your spool document will end up as a MailMan message, a suitable margin and page length request might be:

```
DEVICE: SPOOL;75;99999 <RET>
```

To turn the spool document into a MailMan message, once your spool document completes, go to the Spooler Menu and select the appropriate option, as illustrated below:

```
Select Primary Menu Option: ^Spooler Menu <RET>

Select Spooler Menu Option: Make <RET> spool document into a mail
message
```

If the number of lines in the document exceeds 500, you are asked whether the transfer process should be queued. This prompt is provided for the your convenience since queuing of a time-consuming process is usually preferred. After using the option, you can find your messages by reviewing recently delivered mail in your IN basket.

System Management

Spool Document Storage

Spool document identification is stored in the SPOOL DOCUMENT file (#3.51) in the ^XMB global. This file is for internal use by the Kernel's spooler and should not be directly manipulated by IRM. It holds identifying information such as the name of the spool document and the line count totals. The document's text is stored in another file, the SPOOL DATA file (#3.519), in the ^XMBS global. If the spool document is made into a mail message, the text is moved into the MESSAGE file (#3.9), the ^XMB global, and the corresponding entry in the SPOOL DOCUMENT file is deleted.

When initially creating a spooled document, output is sent to the operating system's spooling area (as defined in the spool device). The Kernel's spooler moves the output into ^XMBS when the operating system's spooling process is complete. The status of the document (a field in the SPOOL DOCUMENT file) is then changed from Active to Ready and the document can be accessed by the user. Thus, except during spooling, the operating system's spool area should be empty.

Overflowing Spool Document Storage

When the output is moved from the operating system's spool area into the ^XMBS global, the lines are counted. If, during the count, the user's maximum line limit is reached, the transfer process is halted and a notification message is appended to the transferred text. The entry in the SPOOL DOCUMENT file is also marked as incomplete. The ^XMBS global is thus protected from growth expansion that could overflow the disk storage area.

The Kernel spooler cannot, however, count the lines of output as they are sent to the operating system's spool area. If the user's line limit is not exceeded before initiating the report, the Kernel permits sending of an unlimited amount of output to the operating system's spooler. This should be considered by IRM when granting spooling privileges. Users who are allowed to spool should be trained accordingly.

Users need to anticipate the results of a process they send to the spooler. If they are not sure what to expect, they should be instructed to test the process by sending it directly to an output device. If unexpected results such as an endless loop or meaningless sort should occur, they can interrupt and cancel the process. Users should also be advised about appropriate use of processing time. Methods of efficient VA FileMan searching and sorting should be used when invoking the spooler (just as when printing directly). For example, as described in the VA FileMan documentation, the first sort-by field should be a

cross-referenced field when possible and search criteria should be specified with the most likely conditions first.

Granting Spooling Privileges

Options on the Spool Management menu can be used to grant spooling privileges to users.

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Spool Management ...                                        [XU-SPL-MGR]
  Edit User's Spooler Access                               [XU-SPL-USER]
```

The ability to invoke the spooler at the device prompt is controlled by one flag. Another flag can enable the use of more than one device when despooling. A third flag permits the conversion of a spool document into a mail message. These three flags are user-specific and are stored in the NEW PERSON file (#200). As mentioned earlier, the user-oriented spooler options are distributed as part of the Common menu, a menu available to all users. If IRM has chosen to lock the Spooler Menu or remove it from the Common menu, access to the options will need to be reestablished for users who are allowed to spool.

```
Select Spool Management Option: ED <RET> it User's Spooler Access

Select NEW PERSON NAME: ADVANCED,USER <RET>
ALLOWED TO USE SPOOLER: YES// <RET>
MULTI-DEVICE DESPOOLING: YES// <RET>
CAN MAKE INTO A MAIL MESSAGE: YES// <RET>
```

Managing Spool Documents

The remaining options on the Spool Management menu are also found on the user-oriented Spooler Menu. They are provided on the Spool Management menu simply for convenience to IRM to access any spool document on the system. The only provision for access to all spool documents is holding the XUMGR key. Together, these options along with the XUMGR key permit IRM to view, print, or delete anyone's documents.

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Spool Management ...                                        [XU-SPL-MGR]
  Delete A Spool Document                                [XU-SPL-DELETE]
  List Spool Documents                                     [XU-SPL-LIST]
  Print A Spool Document                                  [XU-SPL-PRINT]
```

Spooler Site Parameters

```
SYSTEMS MANAGER MENU ...                                        [EVE]
   Spool Management ...                                      [XU-SPL-MGR]
      Spooler Site Parameters Edit                          [XU-SPL-SITE]
```

The Spool Management menu also has an option for setting the spooler site parameters (system-wide defaults for the spooler). The initial settings are defined when installing the Kernel but can be edited afterwards.
The spooler site parameters control the total number of documents a user may create and the total number of lines for all documents. When the limits are reached, the user cannot create new documents.

The effects of the three spooler site parameter fields are as follows:

MAX SPOOL LINES PER USER	**This field holds the MAX number of lines of spooled output a user is allowed. If the user has more than this number, then they will not be permitted to spool any more until some of their spool documents are deleted. This only controls allowing the creation of new spool documents and doesn't terminate a job that is running that has gone over the limit. Recommended value 9999.**
MAX SPOOL DOCUMENTS PER USER	**This field limits the number of spool documents that any user may have on the system. Recommended value 10-100.**
MAX SPOOL DOCUMENT LIFE-SPAN	**This field controls the number of days that a spooled document will be allowed to remain in the spooler before deletion by the XU-SPL-PURGE option that needs to be setup to run in the background.**

Purging Spool Documents

```
PARENT OF QUEUABLE OPTIONS                    [ZTMQUEUABLE OPTIONS]
  Purge old spool documents                          [XU-SPL-PURGE]
```

A spool document is automatically deleted when its life span (in days) is reached. The purge is carried out by the Purge old spool documents option. This option is listed on the Parent of Queuable Options menu along with others that should not be invoked interactively but should be scheduled to run through Task Manager.

Defining Spool Device Types

The DEVICE file (#3.5) entries for spooler device types make use of information about the underlying operating system's spooling mechanism. Examples for several operating systems are provided below.

MSM-DOS

Under MSM-DOS, disk space needs to be allocated to enable spooling. Contiguous maps must be set aside as spool space since space is otherwise assumed to be available for the database (routines and globals). On MSM-DOS, a map consists of 512 blocks; each block contains 1024 bytes. Spool space may thus be located at specific addresses on disk. The addresses are not used by the Kernel spooler, however, since MSM-DOS simply searches for the first available disk location for spooling. MSM-DOS reserves the $I of 2 for the spool device.

An example of an entry in the DEVICE file is shown below. Note that when defining a subtype for the spool device, a simple one such as P-OTHER may be used since the only operative parameters are right margin and page length. The VOLUME SET (CPU) field may be left blank if spool space can be accessed from any CPU. But if spool space has been allocated on only one CPU, the name of that CPU should be entered in the VOLUME SET (CPU) field.

■ **Spool Device for MSM-DOS**

```
            Name:   SPOOL
              $I:   2
            Type:   SPL
         Subtype:   P-OTHER
```

DSM for OpenVMS

DSM for OpenVMS uses an OpenVMS directory for spooling. As indicated in the DHCP Cookbook for VAX sites, the directory should be established with full privileges for System, Owner, Group, and World. The directory specifications are used as the $I value.

■ **Spool Device for DSM for OpenVMS**

```
            Name:   SPOOL
              $I:   VA1$:[SPOOLER]
            Type:   SPL
         Subtype:   P-OTHER
```

Spool Device Edit

The Spool Device Edit option lets you edit Spool device attributes, using a ScreenMan form.

```
Device Management...                                          [XUTIO]
   Edit Devices by Specific Types...                       [XUDEVEDIT]
      Spool Device Edit                                  [XUDEVEDITSPL]
```

Note that the type of data entered in the $I and OPEN PARAMETERS fields depends on the type of M system you are using and the mode of access. Refer to your M system manuals for further details. Examples are provided on the previous pages in the Defining Spool Device Types section.

Auto-despooling

For convenience, spool devices may be defined to ensure that despooling takes place automatically, without user interaction. If the Auto Despool flag is set, one copy of the spooled output is sent to each device named in the Despool Devices multiple. Having the output automatically despooled saves users the time and trouble of logging on and printing a spool document that may have been created the previous evening. Documents are not deleted upon despooling; they remain available to the user for subsequent printing.

```
Select Device Handler Option: DE<RET> vice Edit

Select DEVICE NAME: SPOOL <RET>
NAME: SPOOL// ^AUTO D<RET> ESPOOL
AUTO DESPOOL: 1<RET> YES
Select DESPOOL DEVICES:
```

Generating Spool Document Names

Spool devices may be set up to generate the name that will identify the spool document. If this flag is set in the DEVICE file, users of that device will not be prompted to enter the spool document name. Also, if the flag is set, any user- or programmer-defined name [in IO("DOC")] is ignored. The generated name consists of the first 15 characters of the spool device's name, an underscore (_), and the internal entry number of the spool document in the SPOOL DOCUMENT file.

```
NAME: SPOOL// ^GEN<RET> ERATE SPL DOC NAME
GENERATE SPL DOC NAME: YES<RET>
```

Programmer Tools

In order for an application to spool reports, the application must call the device handler to open the spool device. If the application fails to close the device, the spool document will not be accessible. The application should close the spool device by using D ^%ZISC. Furthermore, queuing to the spooler requires that the application invoke ^%ZTLOAD with the proper variables defined.

The input variable ZTIO can be set to identify how the device should be opened. If incorrectly set up, the queued task could fail to send results to the spooler. If you have any doubt about how to set ZTIO, you should leave it undefined. ^%ZTLOAD can define ZTIO with the appropriate parameters from symbols left in the current partition following the last call to the device handler.

Note: The following code samples are NOT complete. They do not contain code to issue form feeds between pages of output. See the Form Feed section of this chapter for the details of issuing form feeds.

■ Sending Output to the Spooler (and Pre-defining ZTIO)

```
SAMPLE  ;SAMPLE ROUTINE
        ;
        S %ZIS="QM" D ^%ZIS G EXIT:POP
        I $D(IO("Q")) D  D ^%ZTLOAD D HOME^%ZIS K IO("Q") Q
        .S ZTRTN="DQ^SAMPLE",ZTDESC="Sample Test routine"
        .S ZTIO=ION_";"_IOST
        .I $D(IO("DOC"))#2,IO("DOC")]"" S ZTIO=ZTIO_";"_IO("DOC") Q
        .I IOM S ZTIO=ZTIO_";"_IOM
        .I IOSL S ZTIO=ZTIO_";"_IOSL
DQ      U IO W !,"THIS IS YOUR REPORT"
        W !,"LINE 2"
        W !,"LINE 3"
        D ^%ZISC
EXIT    S:$D(ZTQUEUED) ZTREQ="@" K VAR1,VAR2,VAR3 Q
```

■ Allowing Output to go the Spooler (without Pre-defining ZTIO)

```
SAMPLE  ;SAMPLE ROUTINE
        ;
        S %ZIS="QM" D ^%ZIS G EXIT:POP
        I $D(IO("Q")) D  Q
        .S ZTRTN="DQ^SAMPLE",ZTDESC="Sample Test routine"
        .D ^%ZTLOAD D HOME^%ZIS K IO("Q") Q
DQ      U IO W !,"THIS IS YOUR REPORT"
        W !,"LINE 2"
        W !,"LINE 3"
        D ^%ZISC
EXIT    S:$D(ZTQUEUED) ZTREQ="@" K VAR1,VAR2,VAR3 Q
```

Chapter 20 Special Device Issues

This chapter discusses the following special devices and device issues:

- **Browser Device**
- **Form Feeds**
- **Hunt Groups**
- **Magtape**
- **Network Channel Device Support**
- **Resources**
- **SDP**
- **Slave Printers**

Browser Device

User Interface

VA FileMan's new Browser allows you to view reports on your terminal screen, letting you scroll backward and forward through the report, and also letting you perform simple searches within the report. In the past, when you sent a report to the terminal screen, you could only go in one direction, forward, by pressing <RET> after each screenful of text.

If the Browser has been installed at your site and set up as a device, you can now use the Browser to view any report that asks you for an output device.

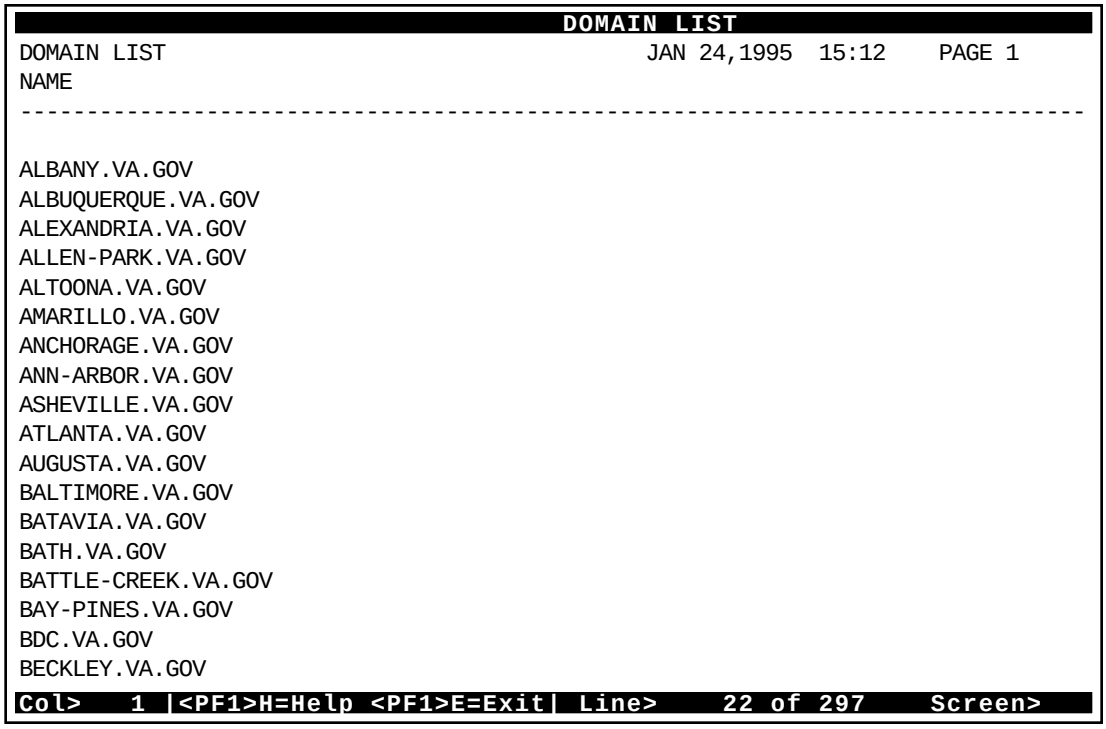
To send a report to the BROWSER device, at any device prompt, enter BROWSER as the device. You may not want to send huge reports to the BROWSER, however, since the report must complete before you can view its output in the Browser.

For information on using the Browser and on Browser commands, please see the *VA FileMan User Manual* (starting with version 21).

■ Sending a Report to the Browser Device

```
Select VA FileMan Option: Print File Entries <RET>

OUTPUT FROM WHAT FILE: DOMAIN <RET>
SORT BY: NAME// <RET>
START WITH NAME: FIRST// <RET>
FIRST PRINT FIELD: NAME <RET>
THEN PRINT FIELD: <RET>
HEADING: DOMAIN LIST// <RET>
DEVICE: HOME// BROWSER <RET> HFS/CRT

...one moment...
```

Report Displayed in Browser Device



System Management

Starting with VA FileMan V. 21, you can set up VA FileMan's Browser as a device that users can send their output to.

When a user sends output to a Browser device, the Browser device performs the following steps:

1. **Output is sent to a host file.**
2. **When the output completes, the host file is closed.**
3. **The contents of the host file are read back into a scratch global.**
4. **The host file is deleted.**
5. **The Browser is called, which displays the data in the global to the user, through the Browser interface.**
6. **When the user exits the Browser, the scratch global is deleted.**

This provides a quick way to generate a report and view the report through the scrollable Browser, potentially saving paper and wear and tear on printers.

To support the Browser device, you need to set up a special terminal type (P-BROWSER), and a special device type (BROWSER). The following page lists sample entries of the special Browser terminal type and device entries for the MSM and DSM operating systems.

The Browser Device tests the current terminal to see whether it supports (a) a scrolling region, and (b) reverse indexing. If the terminal doesn't support these features, the Browser Devices issues a message saying that it is not selectable from the current terminal. Also, in order for the check ($$TEST^DDBRT) to work properly, the user must already be in the Kernel menu system or must have set up programmer variables through the ^XUP entry point. Otherwise, the test will always fail.

Storing Host Files in a Specific Directory

By default, the temporary host files created by the Browser device are stored in the current default directory. You can optionally specify a path to a specific directory to store the temporary host files. Make sure the directory you specify exists on all nodes/CPUs where users can sign on. On DOS systems, don't specify the root directory (since there is a limit on the number of files a DOS root directory can hold). Finally, make sure you change both the OPEN PARAMETERS and POST-CLOSE EXECUTE fields in the Browser DEVICE file entry to specify the directory (replace DD with, for example, D:\BROW\DD).

DSM for OpenVMS Browser Device, TERMINAL TYPE Entry

```
NAME: P-BROWSER                          SELECTABLE AT SIGN-ON: NO
  RIGHT MARGIN: 80                       FORM FEED: #
  PAGE LENGTH: 99999                     BACK SPACE: $C(8)
  OPEN EXECUTE: D OPEN^DDBRZIS
  CLOSE EXECUTE: D CLOSE^DDBRZIS
  DESCRIPTION: Browser Device
```

DSM for OpenVMS Browser Device, DEVICE File Entry

```
NAME: BROWSER                            $I: DDBR.TXT
  ASK DEVICE: YES                        ASK PARAMETERS: NO
  SIGN-ON/SYSTEM DEVICE: NO              QUEUING: NOT ALLOWED
  LOCATION OF TERMINAL: HFS/CRT          ASK HOST FILE: NO
  ASK HFS I/O OPERATION: NO              MARGIN WIDTH: 80
  FORM FEED: #                           PAGE LENGTH: 99999
  BACK SPACE: $C(8)                      OPEN PARAMETERS: NEW:DELETE
  POST-CLOSE EXECUTE: D POST^DDBRZIS
  SUBTYPE: P-BROWSER                     TYPE: HOST FILE SERVER
  PRE-OPEN EXECUTE: I '$$TEST^DDBRT S %ZISQUIT=1 W $C(7),!,"Browser
not selectable from current terminal.",!
```

MSM Browser Device, TERMINAL TYPE Entry

```
NAME: P-BROWSER                          SELECTABLE AT SIGN-ON: NO
  RIGHT MARGIN: 80                       FORM FEED: #
  PAGE LENGTH: 99999                     BACK SPACE: $C(8)
  OPEN EXECUTE: D OPEN^DDBRZIS
  CLOSE EXECUTE: D CLOSE^DDBRZIS
  DESCRIPTION: Browser Device
```

MSM Browser Device, DEVICE File Entry

```
NAME: BROWSER                            $I: 51
  ASK DEVICE: YES                        ASK PARAMETERS: NO
  SIGN-ON/SYSTEM DEVICE: NO              QUEUING: NOT ALLOWED
  LOCATION OF TERMINAL: HFS/CRT          ASK HOST FILE: NO
  ASK HFS I/O OPERATION: NO              MARGIN WIDTH: 80
  FORM FEED: #                           PAGE LENGTH: 99999
  BACK SPACE: $C(8)
  OPEN PARAMETERS: ("DD"_DUZ_".DAT":"M")
  POST-CLOSE EXECUTE: X "N X S X=$ZOS(2,""DD""_DUZ_"".DAT"")" D
POST^DDBRZIS
  SUBTYPE: P-BROWSER                     TYPE: HOST FILE SERVER
  PRE-OPEN EXECUTE: I '$$TEST^DDBRT S %ZISQUIT=1 W $C(7),!,"Browser
not selectable from current terminal.",!
```

Form Feeds

User Interface

Most users would prefer to see their printouts without any extra blank pages before or after. Most prefer to see their reports printed on a fresh page instead of starting in the middle of the previous printout. The printing of labels should also be accomplished without unnecessary form feeds. If a printer is generating extra pages, you should contact IRM to remedy the problem.

System Management

If a particular device does not need a form feed between reports, IRM should set the SUPPRESS FORM FEED AT CLOSE field to YES in the device's DEVICE file entry. Label printers, for example, should have this flag set. This procedure prevents the Device Handler from issuing a form feed:

```
Select Systems Manager Menu Option: DE <RET> vice Handler
Select Device Handler Option: DE <RET> vice Edit

Select DEVICE NAME: LABEL PRINTER <RET>
NAME: LABEL PRINTER// ^SUP <RET> PRESS FORM FEED AT CLOSE
SUPPRESS FORM FEED AT CLOSE: Y <RET>
```

The Device Handler also checks the TERMINAL TYPE file to see if form feeds have been suppressed for that terminal type. It checks for the existence of the IONOFF variable. So, for certain terminal types such as laser printers, IRM can set this "no form feed" variable in the corresponding terminal type's Close Execute field. (IONOFF may also be set by the calling program to suppress form feeds.)

```
Select Systems Manager Menu Option: DE <RET> vice Handler
Select Device Handler Option:  T <RET> erminal Type Edit
Select TERMINAL TYPE NAME: P-DEC-LABEL <RET>
NAME: P-ZPK80// ^CLOSE EXECUTE <RET>
CLOSE EXECUTE: S IONOFF="" <RET>
```

Programmer Tools

The Device Handler has a method for issuing a form feed at the point when it closes the device. The purpose for this utility is to eliminate unnecessary page feeds at the beginning or end of a report. Extra page feeds result when an application package issues its own form feed at the beginning of a report and then VA FileMan issues another pair, one at the beginning and one at the end. An additional problem is laser printers that also generate an extra form feed to clear the print buffer.

When closing a device, ^%ZISC checks the value of $Y to determine the cursor or print head's vertical line location. If $Y is greater than zero, the Device Handler writes a form feed (W @IOF) to reset the value of $Y to zero. Application packages, therefore, should not issue any form feeds when calling the Device Handler to open or close a device.

VA FileMan has already removed its initial form feed. For the benefit of those who use VA FileMan without the Kernel and its Device Handler, VA FileMan continues to issue a form feed at the end when the device is closed. Since this procedure resets the $Y special variable to zero, the Device Handler does not send an additional form feed when VA FileMan is used with the Kernel.

Device Handler also checks for the existence of the IONOFF variable when closing the device. Application programmers may thus use the IONOFF variable to suppress form feeds by setting it just before calling ^%ZISC to close the device.

How to Check if Current Device is a CRT

You should use the following code to test if the current device is a CRT (if it returns false, the current device is a CRT; if it returns true, you should assume that the current device is a printer):

```
I $E(IOST,1,2)'="C-"
```

Guidelines for Form Issuing Form Feeds

In most cases, a form feed before the first page is only needed for reports to CRTs. When directing reports to a printer, do not issue an initial form feed before the first page; it is not needed. However, you should print the heading (if used) on the first page. You do need to issue a form feed between pages, regardless of whether the report is directed to a CRT or to a printer.

The following summarizes the current guidelines for issuing form feeds for CRTs and printers:

CRTs

1. **Issue the initial form feed before the first page of a report as before.**
2. **Print a heading on the first page if headings are used.**
3. **Print the lines of the report while checking the value of the vertical position ($Y).**
4. **If there is no more data to process, then GO TO STEP 9.**
5. **If the value of the vertical position plus a predetermined number to serve as a buffer exceeds the screen length, prompt the user to press <RET> to continue.**
6. **A time-out at the read or an up-arrow (^) response to the continue prompt represents a request to terminate the display. GO TO STEP 9.**
7. **If the user presses <RET> in response to the prompt, issue a form feed followed by a heading (if used).**
8. **GO TO STEP 3.**
9. **The application should terminate the display of the report.**
10. **END**

PRINTERS

1. **Do not issue a form feed before the first page of a report.**
2. **Print a heading on the first page if headings are used.**
3. **Print the lines of the report while checking the value of the vertical position ($Y).**
4. **If there is no more data to process, then GO TO STEP 7.**
5. **If the value of the vertical position plus a predetermined number to serve as a buffer exceeds the page line limit, issue a form feed.**
6. **GO TO STEP 3.**
7. **The application should terminate the printout of the report.**
8. **END**

The sample routines on the following page provide two examples of how to output a report following current guidelines for form feeds. In the examples, a series of three vertical dots indicates omitted information.

■ Issuing Form Feeds Following Current Guidelines

```
ROU      ;SAMPLE ROUTINE
         S IOP="DEVNAM" D ^%ZIS G EXIT:POP
         I $D(IO("Q")) S ZTRTN="DQ^ROU",ZTDESC="SAMPLE REPORT" D
^%ZTLOAD,HOME^%ZIS Q
.
.
.
DQ       ;SAMPLE REPORT
         S (END,PAGE)=0
         U IO D @("HDR"_(2-($E(IOST,1,2)="C-"))) F  Q:END  D
         .W !,....
         .W !,...
         .D HDR:$Y+5>IOSL Q
               .
               .
               .
         D ^%ZISC Q
HDR      ;SAMPLE HEADER
         I $E(IOST,1,2)="C-" W !,"Press RETURN to continue or '^' to
exit: " R X:DTIME S END='$T!(X="^") Q:END
HDR1     W @IOF
HDR2     S PAGE=PAGE+1 W ?20,"SAMPLE HEADING",?(IOM-10),"PAGE:
",$J(PAGE,3)
```

■ Alternate Approach Following Current Guidelines

```
ROU      ;SAMPLE ROUTINE
         S IOP="DEVNAM" D ^%ZIS G EXIT:POP
         I $D(IO("Q")) S ZTRTN="DQ^ROU",ZTDESC="SAMPLE REPORT" D
^%ZTLOAD,HOME^%ZIS Q
.
.
.
DQ       ;SAMPLE REPORT
         S (END,PAGE)=0
         U IO F  Q:END  D
         .D HDR:$Y+5>IOSL Q
         .W !,....
         .W !,...
               .
               .
               .
         D ^%ZISC Q
HDR      ;SAMPLE HEADER
         I PAGE,$E(IOST,1,2)="C-" W !,"Press RETURN to continue or
'^' to exit: " R X:DTIME S END='$T!(X="^") Q:END
HDR1     W:'($E(IOST,1,2)'="C-"&'PAGE) @IOF
HDR2     S PAGE=PAGE+1 W ?20,"SAMPLE HEADING",?(IOM-10),"PAGE:
",$J(PAGE,3)
```

Hunt Groups

User Interface

Hunt groups are a set of printers that IRM can set up to share print workload. If one printer in the hunt group is busy, output is directed to another one that is free. This results in less waiting time for printouts, and helps prevent any particularly long print job on one printer from holding up other output directed to the same printer. Hunt group devices are defined by IRM and would typically include similar printers located near each other. Users may send to the hunt group device or to any member of the hunt group with the same effect.

```
DEVICE: PRTGROUP <RET>
```

The Device Handler indicates which hunt group device was selected so the user knows where to look for the output. If all members of the hunt group are busy, Device Handler displays a message with this information.

System Management

Hunt group devices can be created and managed with options on the Device Handler menu. The hunt group device type (HG type) makes use of a multiple field that holds the names of printers and other devices that can be used interchangeably.

```
Select Systems Manager Menu Option: DE<RET>vice Handler

Select Device Handler Option: HU<RET>nt Group Manager

Select Hunt Group Manager Option: ? <RET>

          Edit Hunt Groups
          Delete Hunt Groups
          List Hunt Groups
          Print Hunt Groups and Associated Devices
```

If the user attempts to select a hunt group device (HG type), the Device Handler uses the B cross reference of the multiple field to find the device with the lowest internal entry number in the DEVICE file. If this device cannot be opened, the Device Handler finds the hunt group member with the next higher internal number and tries again. The Device Handler discontinues the search after all hunt group members have been tried.

If the user attempts to select a printer or other device that happens to be a member of a hunt group, the Device Handler attempts to open the device. If

unsuccessful, the Device Handler identifies the name of the associated hunt group device from the AHG whole-file cross reference and proceeds as described above.

Hunt Group Device on Another CPU

If the user sends output to a device that is a member of a hunt group and is located on another CPU, the hunt group search will not succeed since the Device Handler only attempts to open devices that are on the local CPU. If the user has *queued* to the device, however, TaskMan would have the Device Handler begin searching for a device on the local CPU. If the Device Handler was unsuccessful, TaskMan would send the job to the other CPU's task global to be scheduled for processing and the effort would eventually succeed.

Queuing to a Hunt Group Device

If the user had queued to a hunt group member on the local CPU, TaskMan again calls the Device Handler to search the local CPU and, if the Device Handler is unable to open a device, returns the job to TaskMan for processing on the local CPU. (The Device Handler returns POP with a non-zero value, IO="", and IO(CPU)=destination CPU.) Note that, at this point, the job would be placed into the IO queue and thereafter be processed as if it was not a hunt group member.

Using a Hunt Group Device without Using the Hunt Group

Using a device as if it was not a member of a hunt group is possible with the use of the "D" flag in the Device Handler input variable ^%ZIS. See the documentation for the ^%ZIS call in the Device Handler: Programmer Tools chapter for more information.

Magtape

System Management

```
Device Management...                                          [XUTIO]
     Edit Devices by Specific Types...                     [XUDEVEDIT]
        Magtape Device Edit                               [XUDEVEDITMT]
```

The Edit Devices by Specific Types [XUDEVEDIT] option lets you edit specific types of devices using ScreenMan.

Values entered in a Magtape device's SUBTYPE, FORM FEED, BACK SPACE, MARGIN WIDTH, and PAGE LENGTH fields may not be significant to a given application. The value of the data entered may be arbitrary. On the other hand, if the application plans to copy the output to a printer, the characteristics may need to be similar to that of the printer.

If an application intends to use these fields, be cautious about the type of data that is entered. When sent to the tape unit, some control codes will initiate tape movement or cause tape markers to be written to the mounted tape.

Data entered in the $I and OPEN PARAMETERS fields depends on the type of M system you are running, the type of tape unit, and the desired format. Examples of the type of data required in these fields is provided in Device Handler: System Management chapter. For further details, refer to your specific M implementation manuals.

Network Channel Devices

System Management

Network channel devices are typically high speed channel devices such as DECnet and TCP/IP. Currently, this Network Channel Device Support exists under the DSM for OpenVMS operating system. In most cases, these devices are used for specialized purposes rather than for general output. For example, network mail could use such devices to move enormous amounts of mail through high speed communication channels.

The use of network channel devices requires at least two processes on each end of the communication channel, a server and a client, which can then exchange information:

Server Process	**One process must be available at all times. It can be actively running or triggered to run at a given moment. This process is commonly known as a server. The server waits until another process makes a request to exchange information.**
Client Process	**The other process is known as the client.**

The two processes may be hosted by two CPUs using network protocols.

Because of the split between server and client, two DEVICE file entries are needed in order to use network channel devices. Examples of the device definition entries are given below to illustrate the client and server setups for both Decnet (DSM for OpenVMS) and TCP/IP:

■ Decnet Device Entries

```
NAME: DECNET-SERVER                    $I: SYS$NET
  LOCATION OF TERMINAL: DECNET SERVER PORT
  MARGIN WIDTH: 255                     FORM FEED: #
  PAGE LENGTH: 999                      BACK SPACE: $C(8)
  SUBTYPE: P-DECNET                     TYPE: NETWORK CHANNEL


NAME: DECNET-CLIENT                    $I: VAXA::"TASK=STARTSRV"
  ASK DEVICE: NO                       ASK PARAMETERS: NO
  LOCATION OF TERMINAL: DECNET TEST PORT
  MARGIN WIDTH: 255                    FORM FEED: #
  PAGE LENGTH: 999                     BACK SPACE: $C(8)
  GLOBAL TO LOCK/UNLOCK: ^ZZ("TEST") SUBTYPE: P-DECNET
  USE TIMEOUT ON OPENS: NO             TYPE: NETWORK CHANNEL
```

■ TCP/IP Device Entries

```
NAME: TCP/IP-SERVER                    $I: 1025
  ASK DEVICE: YES                      ASK PARAMETERS: NO
  LOCATION OF TERMINAL: TCP/IP CHANNEL
  MARGIN WIDTH: 132
  FORM FEED: #                         PAGE LENGTH: 64
  BACK SPACE: $C(8)                    OPEN PARAMETERS: TCPCHAN
  SUBTYPE: P-OTHER                     USE TIMEOUT ON OPENS: NO
  TYPE:  NETWORK CHANNEL

NAME: TCP/IP-CLIENT                    $I: 1025
  ASK DEVICE: YES                      ASK PARAMETERS: NO
  LOCATION OF TERMINAL: TPC/IP CHANNEL
  MARGIN WIDTH: 132
  FORM FEED: #                         PAGE LENGTH: 64
  BACK SPACE: $C(8)
  OPEN PARAMETERS: (TCPCHAN,ADDRESS="200.6.0.2")
  SUBTYPE: P-OTHER                     OPEN TIMEOUT: 10
  TYPE: NETWORK CHANNEL
```

Network Channel Device Edit

```
Device Management...                                          [XUTIO]
     Edit Devices by Specific Types...                     [XUDEVEDIT]
        Network Channel Device Edit                     [XUDEVEDITCHAN]
```

The Network Channel Device Edit option allows you to edit network channel device attributes.

When editing network channel devices, note that the contents of fields SUBTYPE, FORM FEED, BACK SPACE, MARGIN WIDTH, and PAGE LENGTH are not necessarily needed for using NETWORK CHANNEL devices. However, these fields are provided in case the application calling the Device Handler is not able to distinguish between a printer and a Network Channel device when sending output.

The timeout on the M open command may not be applicable with network channel devices. Therefore, it may be necessary to answer NO to the USE TIMEOUT ON OPENS field. Refer to the appropriate DSM for OpenVMS manual for more information about device timeout applicability.

For devices using DECNET, data is not required for the OPEN PARAMETERS field. However, the OPEN PARAMETERS are needed for Network Channel devices that use TCP/IP. For the client device set up, this field stores the remote internet address of the host to connect to. Refer to the DSM for OpenVMS manuals for more details. Examples of the type of data required for these fields is provided in the System Management section on Network Channel Devices.

■ Edit a Network Channel Device

```
NAME: DECNET-TASK-TASK                                          PAGE 1 OF 1
_____________________________________________________________________________

NAME: DECNET-TASK-TASK              LOCATION OF TERMINAL: COMM PORT
  $I: SYS$NET                            VOLUME SET(CPU): ISC
TYPE: NETWORK CHANNEL               SIGN-ON/SYSTEM DEVICE: YES

SUBTYPE: P-DECNET                       FORM FEED: #
                                      BACK SPACE: $C(8)
                                    MARGIN WIDTH: 255
                                     PAGE LENGTH: 256

            ASK DEVICE: NO   USE TIMEOUT ON OPENS:
       ASK PARAMETERS: NO           OPEN TIMEOUT:
      OPEN PARAMETERS:
GLOBAL TO LOCK/UNLOCK:
```

The GLOBAL TO LOCK/UNLOCK field stores the name of a global to lock/unlock when opening and closing a Network Channel device. This is important, especially if the application expects that only one client at a given time is able to open the Network Channel. Unlike other types of devices (except spool devices), network channels can be opened simultaneously. This can be controlled with the use of the GLOBAL TO LOCK/UNLOCK field. Network mail is an example of an application that requires that only one client to access a network channel at a given time. Therefore, a global name needs to be entered in this field for network channel devices to be set up for network mail. The selection of the global name should be made judiciously so that other applications are not affected.

Resources

System Management

A resource is a type of device that can only be used by tasks. They cannot be used for input or output (I/O). As such, they are not available for user selection at the device prompt. The purpose of a resource is to provide a mechanism of limiting the number of concurrent jobs that can run at any one time.

When creating a task, a task can request the resource as an input variable for the call. The resource itself, as defined in the DEVICE file, has a field called RESOURCE SLOTS that determines how many jobs can simultaneously own it as a resource.

Device Handler and Task Manager work together to provide resource device functionality. The RESOURCE file (#3.54), stored in the translated ^%ZISL global, regulates processing and is for internal use only. The NAME field holds the $I of the resource device. Other fields hold information on jobs currently using the resource, information that is cleared when the resource is closed.

The RESOURCE file supports processing by maintaining a count of the number of available "slots." The ability to open and close resources is accomplished by decrementing and incrementing this count.

Limiting Simultaneous Running of a Particular Task

Resources make it possible for you to control the number of a particular kind of non-I/O task that runs at any one time. If you have a particular job and you want no more than three running versions of it at any one time, you can queue the job (through the ^%ZTLOAD interface) to a resource that had a RESOURCE SLOT setting of 3.

Running Sequences of Tasks

Resources also make it possible to run non-I/O tasks in sequential order. Non-I/O tasks ordinarily can run simultaneously because they do not compete for the ownership of I/O devices. If you instead queue such tasks to the same resource, and the resource has a RESOURCE SLOT setting of 1, Task Manager will run the tasks one at a time and in the order queued. In this way, the results of one process can be used by another. This sequential processing might be appropriate, for example, for the processing of physician orders or other nested tasks involving code execution.

An additional enhancement to resource devices, called SYNC FLAGs, allows Task Manager to run the next task waiting for a resource only if the previous

task using that resource has completed successfully. You can use SYNC FLAGs to ensure that subsequent jobs run only if previous jobs have completed successfully.

Creating Resource Devices

```
SYSTEMS MANAGER MENU ...                                          [EVE]
   Device Management ...                                        [XUTIO]
      Resource Device Edit                                [XUDEVEDITRES]
```

The Resource Device Edit option provides a facility for editing resource devices. An application package that uses a resource should include in its installation instructions the way the new resource should be defined in the DEVICE file. IRM can then create one or more resource-type (RES) entries.

```
NAME:  ZZRES                        $I:  ZZRES
   LOCATION OF TERMINAL:  NA        RESOURCE SLOTS:  1
   TYPE:  RESOURCE
```

The installation instructions should indicate the number of resource slots. Sequential processing should use a value of 1. The name and $I should probably use the same value and be namespaced according to DHCP conventions.

Programmer Tools

Queuing to a Resource

You can only use resources through calls to ^%ZTLOAD. They cannot be directly manipulated (except by TaskMan). To use a resource, you need to set the ZTIO input variable to the name of the resource. For example:

```
S ZTIO="ZZRES",ZTRTN="tag^routine",ZTDTH=$H
S ZTDESC="First task in a series"
D ^%ZTLOAD
```

Since the name of the resource is part of the call, application developers must include installation procedures so that IRM will be able to create the resources using the correct names and other attributes.

You can optionally use a SYNC FLAG when queuing to a Resource type device. Using a SYNC FLAG helps to ensure that sequential tasks queued to a resource only run if the preceding task in the series has completed successfully. For more information on using SYNC FLAGs, see the Task Manager: Programmer Tools chapter.

SDP

User Interface

Users can send output to Sequential Disk Processor (SDP) device types for temporary storage. Such devices are designed exclusively for temporary storage (for writing to and then reading from) to facilitate, for example, moving routines from one location to another. VA FileMan makes explicit reference to the SDP device type. FileMan uses SDP to hold output so that multiple copies can then be read back to a specified printing device.

```
DEVICE: SDP <RET>

NUMBER OF COPIES: 2 <RET>
OUTPUT COPIES TO
DEVICE:
```

Depending on the way IRM has defined SDP devices, the user may be prompted for address/parameters and may need instruction about responding.

```
DEVICE: SDP<RET>  DISK  ADDRESS/PARAMETERS: NEW// <RET>
```

System Management

SDP device types are so named because they originally made use of the sequential disk processor on DSM-11 systems. The method for storing output sent to SDP device types may be different, though, depending on the underlying operating system.

SDP Device Edit

The Edit Devices by Specific Types [XUDEVEDIT] option lets you edit specific types of devices using ScreenMan.

```
Device Management...                                          [XUTIO]
   Edit Devices by Specific Types...                       [XUDEVEDIT]
      SDP Device Edit                                    [XUDEVEDITSDP]
```

The SDP Device Edit option allows you to edit Sequential Disk Block Processor device attributes, using a ScreenMan form.

To enable minimal use of SDP with VA FileMan, an SDP device entry should be defined with a subtype so that page length and right margin will be available to format output.

However, it may be helpful to enter information for an SDP device's SUBTYPE, FORM FEED, BACK SPACE, MARGIN WIDTH, and PAGE LENGTH fields. The reason is that the SDP device can be used as a spooling device before the output is directed to a printer. This is especially useful when using VA FileMan for multiple copies of output. When VA FileMan prompts for a device, the user may select an SDP device. In this case, VA FileMan prompts the user next for a printer device. As VA FileMan sends the output to the SDP device, the output is formatted according to the characteristics (MARGIN WIDTH, PAGE LENGTH, etc.) of the SDP device. Then VA FileMan copies the output from the SDP device to the printer with no format changes. Therefore, the characteristics of the SDP device should be set up to match those of the printer to be used or, at a minimum, according to the desired output format.

Example setups for MSM-DOS and DSM for OpenVMS are provided below.

DSM for OpenVMS

DSM for OpenVMS uses an OpenVMS file for SDP. It should be established with full privileges for System, Owner, Group, and World. The $I value will hold the file specifications.

The OPEN PARAMETERS field for SDP must be set to NEW so that a new version will be created rather than appending to a document of the same name. The version number limit should be set (manually) so that only a few are retained. For example:

```
$ SET FILE SDP.DAT/VER=3 <RET>
```

■ SDP Device for DSM for OpenVMS

```
                Name:   SDP
                  $I:   SDP.DAT
                Type:   SDP
             Subtype:   P-DEC
     Open Parameters:   NEW
```

MSM-DOS

Micronetics' MSM-DOS has Sequential Block Processor (SBP) which is very similar to SDP. MSM-DOS reserves the $I of 59 through 62 for SBP devices.

The format of the value for the OPEN PARAMETERS field for devices on MSM systems is currently (Opt1:Opt2:Opt3:Opt4:Opt5), where

Opt1 is block offset of SBP space.

Opt2 is starting block number of SBP space.

Opt3 is volume group number of SBP space.

Opt4 is read terminator (usually not needed).

Opt5 is record format (S for stream format, V for variable format).

For example, the OPEN PARAMETERS value on a system might be:

```
(0:25088:0::"V")
```

To find the starting block number of the SBP space allocated on your system, use the MSM utility D ^SBP. If your system does not have any SBP space allocated, you will need to allocate some SBP space. For more information on SBP space, see the *MSM User's Guide*.

■ SDP Device for MSM-DOS

```
                Name:   SDP
                  $I:   59
                Type:   SDP
             Subtype:   P-OTHER
     Open Parameters:   needed
```

Slaved Printers

User Interface

If your terminal has an auxiliary printer port with a printer directly attached, you can send output normally destined for the CRT terminal directly to a printer. Output for the terminal is redirected from the host computer through the terminal's auxiliary port to the printer. Such printers are commonly called slaved printers or slaved devices.

If slaved printing is available from your terminal, you can send a printed report to your slaved printer, by entering the device name that corresponds to your slaved printer like this:

```
DEVICE:  SLAVELA50 <RET>
```

You can consult your local IRM to find out if slaved printing devices are available.

System Management

There are two modes of slaved printing. One is called auto print mode (also known as copy print mode); the other is printer controller mode (also known as transparent print mode).

When auto print mode is toggled on, output is displayed on the terminal as well as printed on the printer. Special escape sequences and control characters such as those that are normally used to adjust fonts/pitches are not passed to the printer; those used for actions like carriage return, line feed, and form feed are passed on to the printer, however.

When printer controller mode is on, output is only printed on the printer; nothing is displayed on the terminal. All escape sequences and control characters are passed to the printer. This mode is preferable to auto print mode, especially when compressed mode printing is desired.

The following are the escape sequences used to toggle the slaved printing modes for DEC VT220/VT320 terminals:

- **Auto print mode on:** `ESC [?5i`
- **Auto print mode off:** `ESC [?4i`
- **Printer controller mode on:** `ESC [5i`
- **Printer controller mode off:** `ESC [4i`

Device and Terminal Type File Entries

To use a slaved printer through the Device Handler, two DEVICE file entries along with corresponding TERMINAL TYPE file entries must be made: one for the home device, and the other for the slaved printer.

One pair of Device/Terminal Type entries is needed to describe the home (i.e., CRT) terminal attributes including the codes to open and close the printer port. The OPEN PRINTER PORT and CLOSE PRINTER PORT fields of the TERMINAL TYPE file can be used to store the appropriate codes.

Another pair of DEVICE/TERMINAL TYPE entries is needed to describe the attributes of the slaved printer including escape codes to adjust fonts/pitches. The OPEN EXECUTE and CLOSE EXECUTE fields of the TERMINAL TYPE file can be used to hold such codes. Additionally, the device entry for the slaved printer must have a value of 0 (zero) entered into the $I field. This $I value identifies the DEVICE file entry as one for a slaved device.

One example is provided below of the setup for a home device, and two examples are provided of the setup for slaved printers.

■ Home Device Example (VT320)

DEVICE Entry:

```
NAME: LAT DEVICE                          $I: _LTA
  ASK DEVICE: YES                         ASK PARAMETERS: NO
  VOLUME SET(CPU): KDE                    SIGN-ON/SYSTEM DEVICE: YES
  LOCATION OF TERMINAL: DECSERVER         MARGIN WIDTH: 80
  FORM FEED: #,$C(27,91,50,74,27,91,72) PAGE LENGTH: 24
  BACK SPACE: $C(8)                       SUBTYPE: C-VT320
  TYPE: VIRTUAL TERMINAL
  TIMED READ (# OF SECONDS): 400
```

TERMINAL TYPE Entry:

```
NAME: C-VT320                             SELECTABLE AT SIGN-ON: YES
  FORM FEED: #,$C(27,91,50,74,27,91,72) RIGHT MARGIN: 80
  PAGE LENGTH: 24                         BACK SPACE: $C(8)
  DESCRIPTION: Digital Equipment Corporation VT-320 video
  OPEN PRINTER PORT: W *27,"[5i"
  CLOSE PRINTER PORT: W *27,"[4i"
```

Slaved Printer Example: DEC LA50

DEVICE Entry:

```
NAME: SLAVELA50                          $I: 0
  ASK DEVICE: YES                        ASK PARAMETERS: YES
  SLAVED FROM DEVICE: TRM
  LOCATION OF TERMINAL: SLAVE DEVICE FOR LA50
  MARGIN WIDTH: 132                      FORM FEED: #
  PAGE LENGTH: 64                        SUBTYPE: P-LA50
  TYPE: TERMINAL
```

TERMINAL TYPE Entry:

```
NAME: P-LA50                             RIGHT MARGIN: 132
  FORM FEED: #                           PAGE LENGTH: 64
  OPEN EXECUTE: W *27,"[4w"              CLOSE EXECUTE: W *27,"[0w"
  DESCRIPTION: LA50 132 COL/16.5 CPI
```

Slaved Printer Example: Epson LQ870

DEVICE Entry:

```
NAME: SLAVELQ870                         $I: 0
  ASK DEVICE: YES                        ASK PARAMETERS: YES
  SLAVED FROM DEVICE: TRM
  LOCATION OF TERMINAL: SLAVE DEVICE FOR LQ870
  MARGIN WIDTH: 132                      FORM FEED: #
  PAGE LENGTH: 64                        SUBTYPE: P-LQ870
  TYPE: TERMINAL
```

TERMINAL TYPE Entry:

```
NAME: P-LQ870                            RIGHT MARGIN: 132
  FORM FEED: #                           PAGE LENGTH: 64
  OPEN EXECUTE: W *15                    CLOSE EXECUTE: W *18
  DESCRIPTION: EPSON LQ870 PRINTER--CONDENSED
```

Use of Slaved Printer: Processing Steps

The device handler manages output to slaved printers using the following steps:

1. **Execute the OPEN PRINTER PORT code of the home device's terminal type.**
2. **Execute the OPEN EXECUTE code of the slaved printer's terminal type.**
3. **When the application closes the device, execute the CLOSE EXECUTE code of the slaved printer's terminal type.**
4. **Execute the CLOSE PRINTER PORT code of the home device's terminal type.**

Queuing to Slaved Printers

If queuing to a slaved device is desired, then the SLAVE FROM DEVICE field of the DEVICE file must be used. This field is a pointer to the DEVICE file. Data must be entered in this field for the entry for the slaved printer. This data should point to the home device entry unless the slaved printer is attached to a terminal on a terminal server (i.e., a virtual terminal).

If queuing to a slaved device is being performed from a virtual terminal, then a third device entry must be established that fully describes the home device with a type of TRM. This device should be entered into the SLAVE FROM DEVICE field.

Note that when queuing to a slaved device from a terminal on a terminal server, the user must be fully logged off the computer system and logged off the port by the time the queued task is scheduled to run.

Part 4: Task Manager

Chapter 21 Task Manager: User Interface

The Kernel Task Manager module (abbreviated TaskMan or TM) allows you to run tasks such as VA FileMan prints and sorts in the background and lets you continue working without interruption.

Creating Tasks

DHCP runs in a multiprocessing environment, which means the computer can work on more than one job at a time. Each job the computer works on consumes a part of the computer's resources. Initially, you have only one job, your interactive terminal session, with which to do your work. TaskMan, however, allows you to claim more of the computer's resources by allowing you to schedule additional jobs to run in the background.

Background Jobs

You can queue additional tasks to run through TaskMan. Once started, these additional tasks (called background tasks) can run at the same time as the foreground jobs and without further dialogue with the people who started them. Appropriate use of background tasks can cut your frustration by reducing the amount of time you must wait for the computer to do lengthy, repetitious work that does not need human intervention. Every task queued to run in the background reduces time spent waiting and also uses the computer's resources more efficiently.

Queuing Output

Most users use TaskMan by queuing reports, labels, and other kinds of output. Because output involves no dialogue once it has begun and because it requires you to wait while it prints, it makes an ideal candidate for queuing. You can queue most output when the computer asks you to select a device to send the output to. The series of prompts and responses to queue a job to a device usually looks something like this:

```
DEVICE:  Q <RET>UEUE TO PRINT ON

DEVICE:  <answer with name of output device>

Requested time to print:  NOW// <RET>

Request queued.
```

After you answer this series of prompts, the output is queued for TaskMan to start at the requested time, and you can continue with other work while TaskMan prints the output. When many tasks need the same device at the same time, TaskMan runs them in order based on the time they were requested.

Other Sources of Tasks

An application can create other kinds of tasks without your interaction. The application might offer to queue other kinds of work like large filing or complex data analysis jobs. Sometimes applications queue tasks without asking. For example, the delivery of MailMan messages is performed by a job running as a task. If that task is not running when someone uses the MailMan options, MailMan automatically uses their foreground job to queue the task without asking them. Although people may knowingly or unknowingly queue these other kinds of tasks, output remains the most common kind of work to queue.

Working with Tasks

```
System Command Options ...                                    [XUCOMMAND]
User's Toolbox ...  "TBOX"                                   [XUSERTOOLS]
   TaskMan User                                                [ZTMUSER]
```

TaskMan also allows you to examine or modify your own tasks. You can do this by using the TaskMan User option, located in the User's Toolbox menu on your common menu. This option lets you monitor or manipulate one task at a time.

Selecting Tasks

When you choose the TaskMan User option, it first asks you to select a task to work with. TaskMan displays the "Select TASK:" prompt. If you enter a single question mark, you get some general help about the option; if you enter two question marks, you can get a list of every task that you have queued to run. Typically, you would enter two question marks at this prompt so that you can get a listing of your individual tasks, listed by task number. You then choose a task from the list of tasks to work with. Using the TaskMan User option looks like:

```
Select User's Toolbox Option: TASKMAN USER <RET>


Select TASK: ?? <RET>


Please wait while I find your tasks...searching...finished!

------------------------------------------------------------------------------
1: (Task #161325) ZTSK2^XMA02, Queued print for SMITH,JANE.  Device VER$LW.
   KRN,KDE.  From TODAY at 14:22,  By you.  Scheduled for TODAY at 20:00
------------------------------------------------------------------------------
2: (Task #161776) ZTSK^DIP4, DEVICE LIST.  Device VER$LW.  KRN,KDE.
   From TODAY at 14:22,  By you.  Scheduled for TODAY at 22:00
------------------------------------------------------------------------------

End of listing.  Press RETURN to continue: <RET>


Select TASK: 161776 <RET> DEVICE LIST

               Taskman User Option

                    Display status.
                    Stop task.
                    Edit task.
                    Print task.
                    List own tasks.
                    Select another task.

               Select Action (Task # 161776):
```

You can select tasks either by task number or list number. In the list of tasks, the list number is at the left hand side of the each task listing, and is followed by the task number for each task (in parentheses). The rest of the information helps identify where the task came from and what it will do.

Tasks in the Task List

You can only select tasks that are still in TaskMan's task list. When a task finishes running, it usually removes itself from the task list. So you shouldn't get a listing of every task you've run in the last year! Tasks that do not clean up their entries usually get cleaned out by TaskMan several days after they complete. You should only have to select tasks that are still actively waiting to start, that are currently running, or that encountered some kind of problem while running.

Display Status of Tasks

Once you've selected a task to work with, you can ask to see the status of that task, using the Display Status option. TaskMan uses a task's status to try to explain how soon the task will run and why. The possible normal statuses for a task include:

- **Scheduled for <date and time>.**
- **Being inspected by Task Manager.**
- **Waiting for a partition.**
- **Being prepared.**
- **Currently running.**
- **Completed <date and time>.**

TaskMan can only guess whether a task is currently running, so take that particular status with a grain of salt.

One of the following messages may show up if the task needs some system resource not currently available:

- **Waiting for hunt group(s) <list of hunt groups>.**
- **Waiting for device <name of device>.**
- **Waiting for the link to <name of CPU> to be restored.**

When you display the status of a task waiting for a device, TaskMan shows you how many tasks are in line for that device ahead of your task. Additional statuses exist for tasks that have encountered some kind of problem. For each situation it lists a different explanation of the problem. For example, if

you use the Stop Task option to stop a task, its status shows up as "Stopped by you."

Stopping Tasks

Under certain conditions, you may want to stop a task. The TaskMan User menu allows you to do this through the Stop Task option. Your ability to stop a task depends on the task's status, however. If the task has already been stopped, if it is finished, or if it encountered a problem while running and you try to stop it, Stop Task tells you that the task has already stopped. If the task has not yet started running, on the other hand, you can always stop it. If the task has started running, the Stop Task option will succeed in stopping it only if the programmer who wrote the task has designed the task to be stopped by a user. At any rate, it does not cause any problems if you try to stop a running task.

To stop a task, use the Stop Task option. Once you stop a task, it remains in the TASKS file until you edit it to run again or until TaskMan purges it from the Task list.

Editing Tasks

The Edit Task option lets you edit a task's output device, description, and run time.

The task must be unscheduled before it can be edited. The Edit Task option asks if it's OK to unschedule the task. To edit the task, answer YES. But once the task is unscheduled, it won't run unless you reschedule it by finishing each step of editing the task.

Note: You can't edit a task that is already running.

Once the task is unscheduled, you can update the following task settings:

- **When the task should start.**
- **Which device it should use (and whether a device is needed).**
- **What the description of the task should be.**

Once you've had a chance to modify these three settings, you're asked whether the task should be rescheduled as shown. If you answer YES, the task is updated to reflect the changes you specified. If you answer NO, however, no settings are changed, but the task remains unscheduled (and will not run until you use Edit Task to reschedule it).

■ Editing a Task

```
Before you edit the task I'll make sure it's not scheduled, okay?
      YES//  <RET>
  Task ready for editing.

  Currently, this task requests output device VER$LW.
  Do you want to change the output device for this task? NO//Y <RET>
  Select Task's Output Device (^ for none): P236 <RET>

  When should this task run?:  AUG 16, 1994@22:00// <RET>

  Task's purpose: DEVICE LIST// <RET>

161776: DEVICE LIST.  P236.  Next run time: AUG 16, 1994@22:00.

  Shall I reschedule this task as shown? YES// <RET>
  Task rescheduled.
```

Listing and Printing Tasks

You can use the "List own tasks" action to review your tasks. This option displays the same list as that given when you enter two question marks at the "Select Task:" prompt.

The "Print task" option lets you print out the description of the task that you have currently selected.

Selecting Another Task

Once in the TaskMan User menu, you can choose to work with a different task by using the Select Another Task option. Enter another task number to work with a different task. If you're not sure what task you want to work with, you can get a list of all of your tasks by entering two question marks.

Summary

Most output in DHCP is performed by creating tasks that run in the background. Once you become familiar with Task Manager's queuing system, you can increase productivity by using some of Task Manager's special features, including listing your future tasks, displaying a task's status, stopping a running task, and editing a future task's run time and output device.

Chapter 22 Task Manager System Management: Overview

The Kernel's TaskMan module provides a standardized system for initiating and managing background processing. Since TaskMan handles all background processes, system managers have a unified set of controls that apply to all background processes on their systems.

Most of Task Manager's processing does not involve interaction with users, rendering its operation virtually invisible. The explanations that follow provide information about the operation of Task Manager.

Task Manager's Division of Labor

TaskMan uses a three-step system to start and manage background processing:

1. **Queuers**

   First, foreground jobs cannot directly start any background jobs. Instead, they call the TaskMan API (Application Programming Interface) to file requests in the TASKS and Schedule files. The program code calling the TaskMan API is called a Queuer. The TASKS file is VA FileMan-compatible. The Schedule file is not VA FileMan compatible. Their structure is described in the Troubleshooting Information section of the Task Manager System Management: Operation chapter.

2. **The Manager**

   Second, a TaskMan program called the Manager runs at all times in the background. The Manager monitors the Schedule files: as needed, it initiates background jobs (called submanagers) to perform the work requested by the foreground jobs.

3. **Submanagers**

   Third, each background job request is picked up by a TaskMan process called the Submanager. The Submanager is the job that actually runs each task. Submanagers handle contention for partitions and I/O devices by running the waiting tasks in order, first the oldest tasks and then the more recent ones.

Queuers

Tasks run by TaskMan begin with code in an application software package that decides to perform some work in the background. This code is a queuer. Most applications in DHCP respond to a user's request to queue some output, but other decisions may be involved. Two commonly used queuers are programs that create report output (by using the TaskMan API) and options that are scheduled through the OPTION SCHEDULING file.

Programs that Use the TaskMan API

One commonly used queuer is an application's call to the TaskMan API to queue tasks. In this process the queuer defines the task and its environment. Applications are not allowed to do direct manipulation of the ^%ZTSCH and ^%ZTSK globals.

The TaskMan API consists of entry points that allow programmers to create, manipulate, and inquire about tasks. The most widely used entry point, ^%ZTLOAD, lets programmers queue tasks, which involves creating and scheduling them. First, an application sets the variables that ^%ZTLOAD needs to define the desired task. In turn, ^%ZTLOAD uses that information to create an entry in the TASKS file. ^%ZTLOAD then sets up a simple cross reference to the new task in the Schedule file, thereby finishing the queuing process.

After queuing the task, ^%ZTLOAD quits, returning control back to the queuer and leaving the next step in the process to the Manager routines.

Option Scheduling through the OPTION SCHEDULING File

Another commonly used queuer is the OPTION SCHEDULING file. Menu Manager and TaskMan work together to allow certain options to be run as TaskMan tasks. These special options may be scheduled to run just once, or they may be set up to run over and over based on a rescheduling cycle. Such cycles may even include running the task whenever the computer system boots up.

The Manager

For tasks to run, at least one CPU in a configuration needs to run a Manager. Only one Manager process needs to run per CPU; the site determines how many CPUs should be configured to run a Manager. The Manager's job is to route the tasks created by queuers. It normally runs at all times in the manager UCIs. It repeats the same loop of code all day long: every two seconds it looks for overdue tasks, every fifteen seconds it checks the environment and performs some cleanup.

The environment check allows the system manager to control the Manager even at its busiest. All of the commands to which the Manager responds (described later) take effect here, between every task processed.

The Manager looks for overdue tasks in the schedule list, comparing the current time to the start time of the tasks listed. If an overdue task is found, the Manager removes it from the schedule list and inspects it. If the task is defined with a complete task record, the Manager places it in a list of tasks ready to run. The Manager places a task on one of several different lists depending on whether the task needs ownership of a currently unavailable I/O device. As its final step in processing each overdue task, the Manager checks the number of Submanagers available to process tasks and starts up new submanagers, if needed. The Manager uses the JOB command (or %SPAWN if the Manager is running in a DCL context on a DSM for OpenVMS system).

The only variation on this scheme happens when the Manager finds a task bound for a different volume set. Depending on the system configuration, such tasks may need to be run by the Manager running on that other volume set. In this case, the current volume set's Manager copies the task over to the volume set on which the task should run and marks it as moved in the current TASKS file. In this process, the task is assigned a new task number, and the Manager on that other volume set handles the task from there. If during this process the Manager discovers that the link between the two volume sets has dropped, it saves the task in a list of tasks waiting for that volume set and checks periodically to see whether it has been restored. When the link recovers, the Manager will send, in sequence, all the waiting tasks to the other volume set.

The Manager never actually runs the task but merely places it in a list as a task now available to be run by a Submanager.

Submanagers

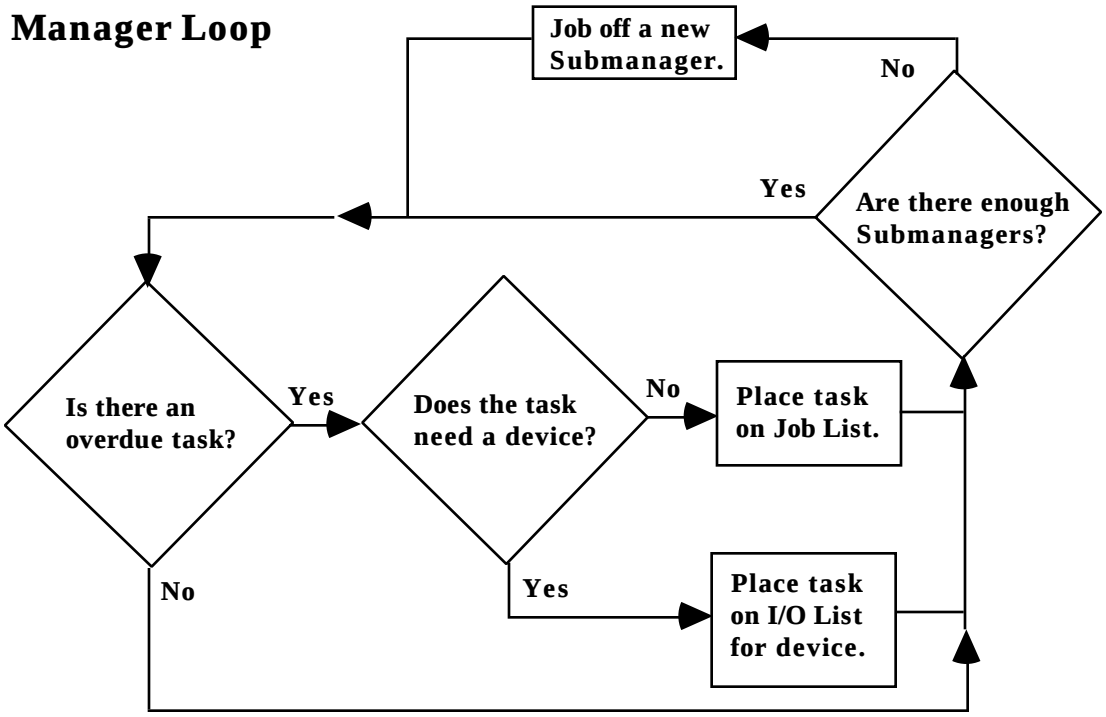
Submanagers are the processes that actually run tasks. A Manager starts Submanagers whenever more are needed to handle the current workload of tasks, and they only last as long as they are needed. Submanagers loop back and forth between finding new tasks to run and running them.

To run each task, the Submanager first removes the task from the list of waiting tasks it resides on (for example, the Job or the I/O list). Then it looks up the task's entry in the TASKS file, unloading all of the information about the task. If the task needs a device, the Submanager calls the Device Handler to get ownership of it and issues a USE command for it. Then the Submanager sets up the partition for the task: it sets the priority, cleans out unwanted variables, sets up requested variables, prints a page header on the device if one was requested, etc. Next, the Submanager starts the task running at the task's entry point. The Submanager uses a DO command and runs the task's entry point in its own partition. When the task finishes, the Submanager cleans up after the task: it closes the output device, performs any commands left for it by the task, etc.

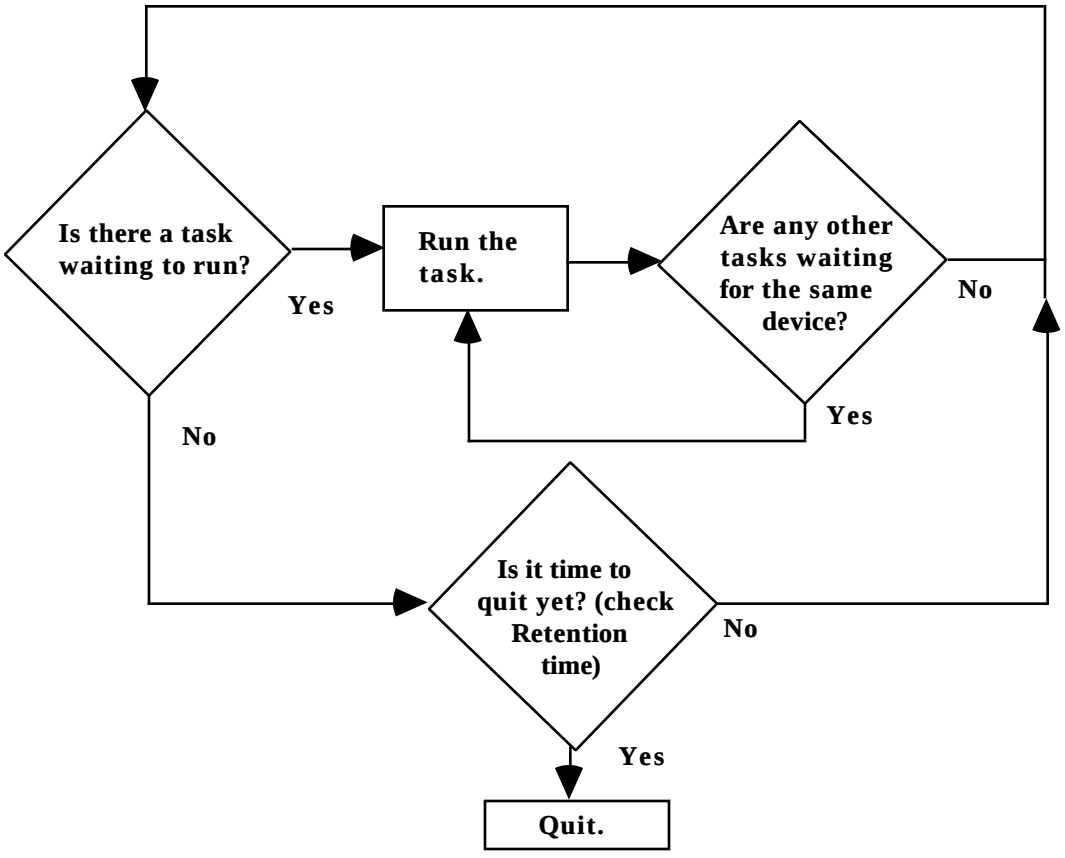
Running completely without user interaction, each task performs the work it was created to do and then quits, returning control to the Submanager that started it. The task may leave instructions for its Submanager, such as to requeue the task so that it runs again later or to delete the task's entry from the TASKS file, but the task itself finishes before the Submanager continues.

After Submanagers have run all available tasks, they wait a while before quitting. This period, called Submanager retention time, allows the Submanager to keep its partition open for new tasks for a while so that the Manager need not start a new Submanager. Every time a new task shows up during retention time, the Submanager starts its main loop over again, returning to retention again only after all new tasks have been run. When the Submanagers eventually reach the end of their retention time, they quit.

Manager Loop



Submanager Loop



Task Manager's Files

The two central files that facilitate task processing are:

- **TASKS file (#14.4)**
- **Schedule file (*not* VA FileMan-compatible)**

TaskMan is configured by three configuration files:

- **VOLUME SET (#14.5)**
- **UCI ASSOCIATION (#14.6)**
- **TASKMAN SITE PARAMETERS (#14.7)**

These files and the TaskMan routines fall within TaskMan's namespace (ZTM), and numberspace. TaskMan user interface routines have been moved to the XUTM namespace beginning with Kernel V. 8.0 (they were previously in the ZTM namespace).

TaskMan also relies upon software components outside of its direct control. As an integral part of the Kernel, TaskMan accesses several files controlled by other Kernel modules and calls many software entry points within the package as a whole. TaskMan's main external relation, however, is with DHCP application packages through the queuers and the tasks they use.

TaskMan Globals: ^%ZTSCH and ^%ZTSK

^%ZTSCH holds the Schedule file, and ^%ZTSK the TASKS file. Every environment controlled by a single Manager needs each of these globals in its library UCI. % globals are used to make these files accessible to all the UCIs in that environment so a single Manager's influence spans all of those UCIs. When the environment spans volume sets, ^%ZTSCH and ^%ZTSK are translated across the volume sets included. They are never replicated because TaskMan updates them so frequently.

The ^%ZTSK global is mostly defined by VA FileMan (beginning with Kernel V. 8.0), but the ^%ZTSCH is not. Historically these globals were not VA FileMan-compatible. Now the inquire, search, and print capabilities of VA FileMan can be used to study the TASKS file. At present, all edit access to these globals is restricted to the TaskMan options that edit the tasks in various ways. The Troubleshooting Information section of the Task Manager System Management: Operation chapter contains a description of the structure of ^%ZTSCH and ^%ZTSK.

The Schedule File

The Schedule file holds all of the lists and nodes that TaskMan uses to manage itself and to schedule tasks. Some of these lists are:

- **Schedule List (or Time Queue)**
- **Waiting List (or IO Queue)**
- **Job List**
- **Compute Server Job List (or C List)**
- **Link List**
- **Status List**
- **Run Node**
- **Taskman Error Log**
- **Error Screens**

The Schedule file's function is split between identifying the status of active tasks and of TaskMan itself. For more information on these lists, please see the Task Manager System Management: Management and Troubleshooting chapter.

Most of the lists in the Schedule file describe tasks, as follows:

- **Schedule List sorts all scheduled tasks by time, according to when they are supposed to begin running.**
- **Waiting List stores each task whose running was delayed because its I/O device was busy.**
- **Job List holds those tasks that can begin running immediately.**
- **Link List stores tasks whose running is delayed because of a dropped link to another volume set.**
- **Task List describes all actively running tasks.**
- **Compute Server Job List describes all tasks waiting to start on a compute server (cross-CPU queuing).**

The role of tracking the status of TaskMan itself is split between lists of information and individual nodes and flags. The Status List is where the Manager keeps track of its current condition; it is a list because IRM may choose to run more than one Manager in the same TaskMan environment. The Run Node is a place where TaskMan stamps the current time; this node reveals when TaskMan stops running. The Taskman Error Log is a simple list in which TaskMan stores each error that occurs either within TaskMan itself or within the tasks that it runs. The Error Screens are screens that can be established by IRM to prevent the recording of certain errors.

These lists and nodes, as well as others not described here, are the primary data structures that TaskMan uses to schedule and run tasks.

The TASKS File (#14.4)

The TASKS file, unlike the Schedule file, contains the tasks themselves.

Every task run by TaskMan is described by an entry in the TASKS file. Each entry is subscripted by a unique internal number, and ^%ZTSK(-1) always equals the number of the most recently created task. The lists and nodes in ^%ZTSCH store the tasks' numbers that are scheduled to run. Each task's entry consists of a ^%ZTSK(task #, 0) node that contains most of the essential information about the task, several decimal nodes (.1, .2, .25, and .26) that store the remainder of the critical information, and a number of storage nodes under ^%ZTSK(task#,.3) that store the names and values of parameters that TaskMan creates for the task. Left unchecked, this file tends to grow. The various means of controlling this growth are discussed later in the Task Manager System Management: Operation chapter.

Other Files

The Schedule and TASKS files, taken together, describe all the information about tasks on the system. A few more files are needed, however, to describe everything about how tasks are managed on the system.

The following three files are stored in ^%ZIS:

- The VOLUME SET file (#14.5) describes the computer system's volume sets and how they are configured into TaskMan environments.
- The UCI ASSOCIATION file (#14.6) lists all the UCIs on the system and which volume sets they belong to. In more complicated systems, it is also used to describe how the UCIs in different environments correspond with one another.
- The TASKMAN SITE PARAMETERS file (#14.7) lets the system manager divide up the environments by both CPU and volume set. This allows a fine degree of control over such parameters as priority, partition size, and retention time.

Taken together, these files give IRM precise and powerful control over TaskMan's behavior.

Other minor pieces of information are scattered throughout other Kernel files, especially the DEVICE (#3.5) and OPTION SCHEDULING (#19.2) files.

System Configuration Terminology

TaskMan operates close to the level of the system architecture. It must be capable of starting tasks in all the environments within a computer system. This means it must know about those environments; consequently, the options, routines, files, and documentation somehow must refer to that architecture.

One problem presented by system configuration is terminology. Such system architecture features as UCIs, directories, volume sets, and namespaces are not part of the ANSI M standard, so different vendors use different terminology. Although it would be ideal for the Kernel to use a universal terminology, none exists. For historical reasons, the Kernel has settled on a terminology based on that of DSM-11 that includes the following terms:

UCI	**User Class Identifier. This is roughly equivalent to a "directory" or an "account". In DataTree terms, this is a very restricted kind of "namespace". A UCI refers to the environment limited to a particular set of routines and globals.**
Manager UCI	**Roughly equivalent to a "system UCI" or a "library UCI". This is where the vendor's system management routines are kept, and where all %-namespaced routines and globals reside. The TaskMan Manager routines must run in the manager UCI because it sometimes offers privileges with respect to the JOB command that other UCIs do not.**
Volume Set	**Roughly equivalent to a DSM for OpenVMS "volume set", or an MSM "volume group". This is a collection of UCIs that share a common manager UCI. In M terms, they are distinguished by having the same second comma-piece in an extended global reference or JOB command. This is the critical definition, since this is what affects how TaskMan starts background jobs.**
CPU	**Also known as a "node" or "computer", this designates a source of computing power and partitions. Although a CPU was once difficult to distinguish from a volume set (because they were usually paired), the advent of mounted volume sets and the DSM for OpenVMS cluster configurations has made the distinction significant both for controlling TaskMan's behavior with parameters and for sending tasks to specific CPUs.**

Mounted Volume Set A volume set may share CPU power with another. Every volume set added after the first is said to be mounted on the first, while the original is referred to as the primary or system volume set. Mounted volume sets may lack manager UCIs of their own, in which case TaskMan treats them as special extensions of the primary volume set. If they have a manager UCI, TaskMan treats them as separate volume sets unrelated to their primary volume set.

The Task Manager chapters that follow make use of this terminology.

Task Manager Security Key

The TaskMan module comes with one security key, ZTMQ. The ZTMQ security key doesn't completely lock any options. Instead, it affects the behavior of three options:

- **Dequeue Tasks [XUTM DQ]**
- **Requeue Tasks [XUTM REQ]**
- **Delete Tasks [XUTM DEL]**

Those who use these options without holding this key can manipulate only their own tasks. Only the holder of the ZTMQ key can use these options to manipulate any task on the system.

Chapter 23 Task Manager System Management: Configuration

This chapter discusses the many issues surrounding the configuration of Task Manager.

Defining Task Manager Environments

The part of configuring TaskMan for a system that requires the most creativity is deciding how to divide the system's UCIs, volume sets, and CPUs into Task Manager environments. A TaskMan environment is the collection of UCIs from which entries can be made directly into a given Manager's TASKS and Schedule files and that are within that Manager's reach. This requires looking at the system in terms of queuing and starting tasks. There are a number of options available. Many different configurations are possible.

One type of configuration has CPUs sharing the same volume set. Since this type of environment shares a single volume set among multiple CPUs, they also share a single TASKS and Schedule file. However, the reach of Managers may not span CPUs. Therefore, you must decide which CPUs in that environment run Managers, or whether some of them should rely on the other CPUs to run their tasks for them. Alpha clusters in VA are typically configured with Managers on only one or a few CPUs.

A different configuration allows you to limit the number of places TaskMan runs. In this scenario, you pick certain CPUs to run TaskMan and give them Managers and files to do the job. To have background processing support, the remaining volume sets need to be able to queue to one of the Managers on the system. This entails translating the TASKS and Schedule files of that Manager so they are visible to the unsupported volume set. To tell TaskMan that the one volume set runs no tasks but is instead supported by the other, you must configure the VOLUME SET file as described later in this section. VA's 486 Cookbook configuration (MSM-DOS) uses this style.

Another possible configuration is to allow tasks to run everywhere, which requires that you place Managers within reach of every UCI and that you define your TaskMan environments accordingly. Under this configuration every CPU needs its own Manager, and its own TASKS and Schedule files. The classic example of this configuration in DHCP was the PDP network.

One other configuration to keep in mind, of course, is to have a stand-alone environment disconnected from the rest of the computer system. Such environments make excellent test areas for programmers. They are configured the same regardless of the configuration of the main system.

Configuring Task Manager

TaskMan's three configuration files must be setup to properly reflect your system's layout. The three files are:

- **TASKMAN SITE PARAMETERS**
- **VOLUME SET**
- **UCI ASSOCIATION**

There are three options on the Edit TaskMan Parameters menu, one to edit each of the three configuration files.

Because the TASKMAN SITE PARAMETERS file allows you to define parameters such as TaskMan Job Limit separately for each CPU on your system, you are able to optimize TaskMan's behavior individually for each CPU.

You no longer need to stop and then restart TaskMan in order to change the TASKMAN JOB LIMIT on a CPU. Cross references on the relevant fields locate every TaskMan on your system and inform them that they need to update their TaskMan parameter information. So, within a minute or so of making the changes, TaskMan on that CPU should be operating with the new value.

TaskMan's Reach

The key issue that defines TaskMan's configuration is its "reach," those places where TaskMan can start background jobs. TaskMan's reach extends to:

- **All UCIs a Submanager can access directly after using the Kernel's UCI switching facilities.**
- **All other Managers to whose TASKS and Schedule files a given Manager can write using extended global reference.**
- **All UCIs on print servers with link access to the current volume set.**

TaskMan's reach does not include other sites on a wide area network, because they can't be accessed through either UCI switching or through extended global reference. There are ways to simulate such a reach through the use of Servers, however. For purposes of TaskMan configuration, we generally think in terms of the reach of a single Manager, which can only run tasks in the UCIs it can reach.

TASKMAN SITE PARAMETERS File

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Task Manager ...                                              [XUTM MGR]
   Taskman Management Utilities ...                           [XUTM UTIL]
      Edit Taskman Parameters ...                  [XUTM PARAMETER EDIT]
         Site Parameters Edit                                  [XUTM BVPAIR]
```

System managers must enter one set of site parameters into the TASKMAN SITE PARAMETERS file for each Manager that runs in a different volume set/CPU. This set of parameters tells each Manager how it should process tasks. The parameters are organized both by volume set and by CPU. This allows a mounted volume set to be treated differently from the primary volume set, for example, even though they share a CPU. It also allows two CPUs that share a volume set to be treated differently if one is more powerful than the other.

BOX-VOLUME PAIR

This field identifies a volume set and the CPU on which it is available. It contains the name of a volume set concatenated to the CPU ("box") name: first the volume set name and then the CPU name. For example, if the volume set name is "KRN" and the name of the CPU (e.g., box) is "ISC6A1", then the box-volume pair would be "KRN:ISC6A1".

For systems on which each CPU tends to have a unique volume set, and vice versa, you may enter just the volume set name (e.g., "PSA" or "AAA"). This field's value for the current process can be found by doing GETENV^%ZOSV and checking the fourth ^-piece of Y. Since the volume set and CPU are identified, the TaskMan site parameters can be tuned for each specific volume set and CPU affected. Systems running Managers on more than one CPU need one entry for each CPU where a manager is running.

LOG TASKS?

Set this field to YES to make tasks log in and out through the sign-on log the way interactive users do. How to set this is up to the individual site; it does consume space and resources.

DEFAULT TASK PRIORITY

This field sets the default priority assigned to tasks. If this value is too low, tasks won't be processed quickly enough to keep up with the rate of creation. If this value is set too high, tasks may interfere with other jobs by consuming too many resources.

This value is overridden only for special options, devices, and tasks. Application developers may define tasks with a specific priority that overrides the value of this field. The DEVICE file field PRIORITY AT RUN TIME overrides both of these if it is set. The PRIORITY field of the OPTION file overrides all three of these sources if you are scheduling a queueable option. Since most tasks won't involve special priorities, this default sets the value for almost all tasks on your system.

The value for this field can be an integer from 1 to 10. This scale represents increasing priorities, with 10 corresponding to the highest priority value available for the operating system you are on. Here is how the Kernel priority scheme translates to the operating system-specific schemes:

KERNEL	DSM	MSM
10	4	HIGH
9	4	HIGH
8	3	HIGH
7	3	HIGH
6	2	HIGH
5	2	LOW
4	1	LOW
3	1	LOW
2	0	LOW
1	0	LOW

The DEFAULT TASK PRIORITY setting for MSM-DOS systems should be 5. This ensures that tasks will run with a priority of LOW, and avoid conflicting with MSM-DOS's DDP servers, which run with a priority of HIGH.

TASK PARTITION SIZE

This field is used to assign partition sizes for tasks. The value from this field is plugged directly into the JOB command used to create new submanagers. If this field is left blank, all tasks receive the operating system's current default value. This field should only be used by system managers who thoroughly understand how their vendor's version of M handles partition sizes with the JOB command.

SUBMANAGER RETENTION TIME

This number determines how many seconds submanagers should wait while looking for new tasks. The purpose of this field is to reduce the number of JOB commands needed to process a site's tasks. By keeping old submanagers around to run new tasks, new process creation is significantly reduced.

TASKMAN JOB LIMIT

If there are more active processes on the system than the number stored in this field, TaskMan will not create new submanagers to handle tasks. Task processing will be left to existing submanagers until the number of processes falls back below this number. This number should be slightly lower than the MAX SIGNON ALLOWED field of the VOLUME SET multiple in the KERNEL SYSTEM PARAMETERS file so that the system manager still has room to sign on when TaskMan is using its greatest number of partitions.

TASKMAN HANG BETWEEN NEW JOBS

This field sets a delay between the creation of new submanagers, in seconds. Such a delay is necessary only for systems with a high cost of new process creation (such as OpenVMS systems). For such systems, this delay spaces out the use of the JOB command to avoid slowing users' response time when the Manager needs to JOB off many new processes in rapid succession.

For systems that create new processes cheaply, this delay is unnecessary. This delay also becomes less important when a high Submanager retention time is used since higher retention times reduce the likelihood that TaskMan will need to create new processes.

Be sure not to combine a high TASKMAN HANG BETWEEN NEW JOBS value with a low SUBMANAGER RETENTION TIME value, since that increases the number of jobs per day TaskMan has to start and may cause busy systems to fall behind. The number should be the lowest value that prevents the problem and can be left blank for systems with efficient JOB commands.

MODE OF TASKMAN

This field determines how each CPU (box-volume pair entry) should process tasks. You can set it to one of four values:

- **General Processor: The G type should be selected when the TASKS and Schedule files are seen by only one volume set. For example, VA's Alpha clusters have several CPUs, but each of them runs on the same volume set. The Manager on a G type runs tasks created on the same volume set, and tasks from any other volume set that explicitly requests the G type's volume set. The G type sends tasks from another volume set that didn't explicitly request its volume set back to the originating volume set, however.**

  To transfer tasks to a G type, TaskMan uses extended global references to copy the task to the destination TASKS and Schedule files and then removes the task from its own side. Submanagers started on a G-type processor process tasks in the Partition Waiting List and the Busy Device Waiting List.

- **Print Server: The P type should be selected when multiple volume sets map to the same TASKS and Schedule files, and you want to run the Manager on the volume set/CPU in question. For example, in VA's 486 configurations, each Print and Compute Server runs on a separate volume set, but shares a common TASKS file and Schedule file across volume sets.**

  Like the G type, the Manager on a P type runs tasks created on the same volume set and tasks from any other volume set/CPU that explicitly request the P type's volume set/CPU. Unlike the G type, however, the P type also runs tasks from other volume sets that did not make an explicit volume set request. Tasks are transferred to a P type in the same way as to a G type, and Submanagers behave the same.

- **Compute Server: The C type should be selected when multiple volume sets map to the same TASKS and Schedule files (as with the P type), but when the volume set/CPU in question runs users (not tasks). The Manager will not start on a C type. Tasks that explicitly request to run on a C type are transferred to it by being placed in the Link Waiting List; a Submanager is then jobbed across to the C type volume set/CPU. Submanagers started on a C type only process tasks in the Link Waiting List for their volume set.**

- **Other Non-TaskMan: Neither the Manager nor the Submanager will run on O types. Tasks sent from or to an O type are rejected.**

Because of the field's crucial role in guiding TaskMan's behavior, the field is required.

VAX DSM ENVIRONMENT FOR DCL

This field only has meaning to DSM for OpenVMS systems. It is set to the OpenVMS username of the DSM environment manager account. Setting it to this username causes the Manager to use %SPAWN to SUBMIT submanagers to run. This method requires that certain DCL command files exist, along with a TASKMAN OpenVMS user account and directory (see the section in this chapter, Running TaskMan with a DCL Context, for descriptions of the needed setups).

If the field is empty, the Manager starts submanagers with the JOB command instead.

LOAD BALANCE ROUTINE

If you are running multiple Managers (one per node), use this field to set up load balancing between the Managers on each node. It should be set to the name of an extrinsic function that returns a load rating for the node. For more information on load balancing, see Multiple Manager and Load Balancing in this chapter.

VOLUME SET File

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Task Manager ...                                              [XUTM MGR]
   Taskman Management Utilities ...                           [XUTM UTIL]
      Edit Taskman Parameters ...                  [XUTM PARAMETER EDIT]
         Volume Set Edit                                      [XUTM VOLUME]
```

TaskMan knows about a system's configuration from the values entered into the VOLUME SET file using the Volume Set Edit option. The information stored in this file strongly affects TaskMan's behavior. If you inaccurately describe your system, you will usually notice very quickly as TaskMan begins processing tasks in a consistently incorrect way.

You need to make one entry in this file for each volume set that tasks can be queued to or from. These entries are only used when:

- **A Manager is running on the volume set and must look up information about its own environment.**
- **The volume set is a required volume, in which case every Manager must check access to it when they start up.**
- **A task needs to run on the volume set, in which case the Manager must look up how to get the task there.**

VOLUME SET

This field should be set to the name of a volume set. It is used in extended global references to reach this volume set and may be used in UCI-switching software to move Submanagers between UCIs. If you are unsure how your volume sets are named, you can look at the value of ^%ZOSF("VOL") in the volume set in question.

TYPE

This field is used to help resolve where tasks should run; it should properly identify the type of the volume set. Typically it should be set to the same value as the MODE OF TASKMAN field for all box-volume pairs associated with this volume set, in the TASKMAN SITE PARAMETERS file. This field must be filled in for all volume sets. This field can have the following values:

- **GENERAL PURPOSE VOLUME SET**
- **PRINT SERVER**
- **COMPUTE SERVER**
- **OTHER NON-TASKMAN VOLUME SET**

These values have the same meanings as the equivalent values for the MODE OF TASKMAN field in the TASKMAN SITE PARAMETERS file, described above in the Where Tasks Run section of this chapter. GENERAL PURPOSE VOLUME SET for volume sets is the rough equivalent of the MODE OF TASKMAN value GENERAL PROCESSOR for box-volume pairs.

The FILE SERVER value has been removed; volume sets for file servers should be set to a TYPE of OTHER NON-TASKMAN VOLUME SET.

INHIBIT LOGONS?

Setting this field to YES causes TaskMan to notify Sign-on that logons are now prohibited and to enter a PAUSE state (stopping processing of tasks) until logons are allowed again. Under ordinary circumstances, system managers should leave this field as null or NO.

LINK ACCESS

This field should always be set to null or YES for the usual kinds of configurations used in DHCP. Answer NO to tell TaskMan that this volume set cannot be accessed by other volume sets using the local network links. Tasks that request a volume set without link access are rejected by TaskMan. Such volume sets are usually PC workstations linked into the larger network. They can access the core computers, but cannot be accessed themselves. Some system managers may wish to have a completely isolated computer for testing. They may cut it off from the rest of the world by making entries for all the other volume sets and setting this field to NO for each of them. This explicitly tells TaskMan it can't reach the other volume sets.

OUT OF SERVICE?

Ordinarily, this field should be set to null or NO. However, answer YES to temporarily prevent tasks from being sent to this volume set. While a volume set is out of service, tasks that would normally be sent across the local links to it are either saved until it is back in service or sent to a replacement volume set (depending on whether one is specified). If no replacement volume set is specified, then once the volume set is back in service the waiting tasks will be transferred across in order. Please note that while this field was used in previous versions to make a 486 system identify its compute servers and print servers, beginning with TaskMan 7.1 the Type field takes over that function.

REQUIRED VOLUME SET?

Answer YES if TaskMan cannot run without this volume set being accessible. If a volume set serves files that the rest of the system needs, it should be a required volume set. So long as a volume set is required, the normal mechanisms of handling links that are out of service or dropped are bypassed for that volume set. Instead, TaskMan waits until the required link is restored before processing any more tasks. The answer to this field must be decided on a case-by-case basis, depending on what files the volume set serves and whether it holds any other resources needed by the rest of the local network. A null value equates to NO.

TASKMAN FILES UCI

This field should be set to the name of the UCI that holds the ^%ZTSCH and ^%ZTSK globals (usually the manager UCI). The answer should not contain a comma and volume set name (e.g., "MGR,PSA"), just the UCI name (e.g., "MGR"). This field is required.

TASKMAN FILES VOLUME SET

This field should be set to the name of the volume set that holds ^%ZTSCH and ^%ZTSK. This field is most frequently used when:

- **Mounted volume set(s) lack manager UCIs of their own.**
- **A DSM for OpenVMS site is not using cluster-mounted databases (the field is needed to reference standalone systems.)**

A null value means this volume set holds its own TaskMan files, which is usually the case.

REPLACEMENT VOLUME SET

This field should be set to the name of a volume set to which tasks can be sent if this volume set is unavailable. A replacement volume set should be essentially equivalent in features to the current one, since tasks that would normally run on the current one will be running on the replacement volume set instead. For many volume sets, no other volume set is equivalent, and tasks should wait for the link to be restored rather than run elsewhere. If tasks that need this volume set should wait, leave the field blank.

DAYS TO KEEP OLD TASKS

This number stored in this field is used by the XUTM QCLEAN option to decide which tasks to delete. The decision only affects inactive tasks, as explained in the discussion of the XUTM QCLEAN option below. Values in this field cannot inadvertently cause TaskMan to delete scheduled or running tasks. If the field contains no value, XUTM QCLEAN keeps the last seven days' tasks. A value of 0 here keeps your file very clean.

UCI ASSOCIATION File

```
SYSTEMS MANAGER MENU ...                                          [EVE]
Task Manager ...                                                [XUTM MGR]
   Taskman Management Utilities ...                            [XUTM UTIL]
      Edit Taskman Parameters ...                  [XUTM PARAMETER EDIT]
         UCI Association Table Edit                              [XUTM UCI]
```

There are two different kinds of entries made into the UCI ASSOCIATION file using this option.

Entries with only the first two fields (FROM UCI and FROM VOLUME SET) filled in identify the valid UCIs on the system for TaskMan. Every kind of DHCP site needs one entry of this kind for each UCI to which tasks can be queued or from which tasks are created. DSM for OpenVMS sites and MSM-DOS sites only need to fill in the first two fields (FROM UCI and FROM VOLUME SET). For examples, please see sample configurations for both types of systems elsewhere in this chapter.

Entries with all four fields completed collectively build a UCI Association Table. An example of a system that needed to be configured with a complete UCI Association Table was VA's PDP networked system.

A complete UCI Association table tells TaskMan which UCI to use for tasks that must switch volume sets in order to reach an I/O device. This situation arises when an I/O device is located in a different volume set than the volume set where the task was created. In such situations, the Manager knows exactly where the task originated and knows to which volume set it must be moved, but it does not know in which UCI on that volume set it should run the task. A UCI ASSOCIATION TABLE entry supplies the missing information by linking equivalent UCIs together. When building a full UCI association table, you may omit entries where the UCIs on both volume sets have the same name because TaskMan assumes that same-named UCIs are equivalent if no entry is present.

FROM UCI

This field should be set to the name of a UCI on your system. For entries requiring only two fields, this catalogues all the UCIs on your system (and there should be an entry for each). For four-field entries, this represents a UCI from which tasks are being transferred in order to reach their I/O device. Enter only the UCI name (e.g., "VAH"). Do not include the volume set name (e.g., "VAH,ROU").

FROM VOLUME SET

This field should be set to the name of the volume set that holds the UCI identified in the entry's FROM UCI field. For four-field entries, this represents the volume set from which tasks are being transferred in order to reach their I/O device. Every volume set listed in this field should be described in the VOLUME SET file.

TO VOLUME SET

This field is only used for entries that build a UCI Association Table. For such entries, it should be the name of the volume set to which tasks are being transferred in order to reach their I/O devices.

TO UCI

As with TO VOLUME SET, this field is only used for entries that build a UCI Association Table. For such entries, it should be the name of the UCI to which tasks will be transferred whenever they must be moved from the UCI on the first volume set to the second volume set in order to reach their I/O devices. As with the From UCI field, the volume set name should not be included.

Sample Configuration 1: Standardized VA DSM for OpenVMS Configuration

Sites that run Managers on their satellites should make the appropriate TASKMAN SITE PARAMETERS file entries for each satellite and adjust their TaskMan Job Limit to reflect each satellite's individual capacity.

■ VOLUME SET

VOLUME SET	You need one entry, for ROU
TYPE	GENERAL PURPOSE VOLUME SET
INHIBIT LOGONS?	Blank or NO
LINK ACCESS?	Blank or YES
OUT OF SERVICE?	Blank or NO
REQUIRED VOLUME SET?	YES
TASKMAN FILES UCI	MGR
TASKMAN FILES VOLUME SET	Leave this blank
REPLACEMENT VOLUME SET	Leave this blank
DAYS TO KEEP OLD TASKS	Up to you; may leave blank

■ UCI ASSOCIATION

FROM UCI	2 entries: VAH and MGR
FROM VOLUME SET	ROU for both
TO VOLUME SET	Blank
TO UCI	Blank

■ TASKMAN SITE PARAMETERS

BOX-VOLUME PAIR	ROU:662A1 Your answer should be the volume set name concatenated with the ":" concatenated with the name of the CPU.
LOG TASKS?	Blank or NO (unless TaskMan is running in a DCL context, in which case set to YES)
DEFAULT TASK PRIORITY	7
TASK PARTITION SIZE	Blank
SUBMANAGER RETENTION TIME	600
TASKMAN JOB LIMIT	2-5 lower than Max Signons
TASKMAN HANG BETWEEN NEW JOBS	1
MODE OF TASKMAN	GENERAL PROCESSOR
VAX DSM ENVIRONMENT FOR DCL	Blank
OUT OF SERVICE	Blank
LOAD BALANCE ROUTINE	Blank

Sample Configuration 2: Standardized VA MSM-DOS (486) Configuration

For the Print Server

Some sites may choose to have multiple interchangeable print servers. Aside from having multiple entries for print servers instead of just one, the only resulting configuration difference is that each print server should list as its replacement the one next in line (PSB for PSA, PSC for PSB, etc.). The final print server should have the first as its replacement. This arrangement allows TaskMan to respond immediately to the loss of one of the print servers by rerouting to the next one.

■ VOLUME SET

VOLUME SET	PSA
TYPE	PRINT SERVER
INHIBIT LOGONS?	Blank or NO
LINK ACCESS?	Blank or YES
OUT OF SERVICE?	Blank or NO
REQUIRED VOLUME SET?	Blank or NO
TASKMAN FILES UCI	MGR
TASKMAN FILES VOLUME SET	Blank
REPLACEMENT VOLUME SET	Blank
DAYS TO KEEP OLD TASKS	Up to you; may be blank.

■ UCI ASSOCIATION

FROM UCI	2 entries: VAH and MGR
FROM VOLUME SET	PSA
TO VOLUME SET	Blank
TO UCI	Blank

■ TASKMAN SITE PARAMETERS

BOX-VOLUME PAIR	PSA
LOG TASKS?	Blank or NO
DEFAULT TASK PRIORITY	4
TASK PARTITION SIZE	Blank
SUBMANAGER RETENTION TIME	50
TASKMAN JOB LIMIT	2-5 lower than Max Signons
TASKMAN HANG BETWEEN NEW JOBS	Blank or 0
MODE OF TASKMAN	PRINT SERVER
VAX DSM ENVIRONMENT FOR DCL	Blank
OUT OF SERVICE	Blank
LOAD BALANCE ROUTINE	Blank

When TaskMan runs on a print server, it automatically knows that any task that can run on one print server can be run on any other print server.

For the Compute Servers

Entries for compute servers are required in the TASKMAN SITE PARAMETERS file; this allows cross-CPU tasking where some tasks get sent from the print server to the compute server.

■ VOLUME SET

VOLUME SET	1 entry per compute server: CSA, CSB, etc.
TYPE	COMPUTE SERVER
INHIBIT LOGONS?	Blank or NO
LINK ACCESS?	Blank or YES
OUT OF SERVICE?	NO or blank
REQUIRED VOLUME SET?	Blank or NO
TASKMAN FILES UCI	MGR
TASKMAN FILES VOLUME SET	Blank
REPLACEMENT VOLUME SET	Blank
DAYS TO KEEP OLD TASKS	Up to you -- may be blank.

■ UCI ASSOCIATION

FROM UCI	2 entries per compute server: VAH and MGR
FROM VOLUME SET	Name of compute server.
TO VOLUME SET	Blank
TO UCI	Blank

■ TASKMAN SITE PARAMETERS

BOX-VOLUME PAIR	1 entry per compute server: CSA, CSB, etc.
LOG TASKS?	Blank or NO
DEFAULT TASK PRIORITY	4
TASK PARTITION SIZE	Blank
SUBMANAGER RETENTION TIME	5
TASKMAN JOB LIMIT	Not used
TASKMAN HANG BETWEEN NEW JOBS	Blank or 0
MODE OF TASKMAN	COMPUTE SERVER
VAX DSM ENVIRONMENT FOR DCL	Blank
OUT OF SERVICE	Blank
LOAD BALANCE ROUTINE	Blank

For the File Servers

No entries for file servers are required in the UCI ASSOCIATION or TASKMAN SITE PARAMETERS files. Although entries for file servers in the VOLUME SET file aren't needed to guide TaskMan running on those volume sets (because file servers don't run TaskMan), such entries can be used to make TaskMan hibernate if the file servers ever become unavailable. This prevents TaskMan from generating errors and discarding tasks when file servers go down. To do this, simply make an entry for each file server and set the REQUIRED VOLUME SET? field to YES.

Note that the TASKMAN FILES UCI value of "MGR" is a dummy value. Although file servers do not run TaskMan, and hence have no TASKMAN FILES UCI, this is a required field.

Note also: if you are an MSM-DOS site using the performance monitoring software in Kernel Toolkit V. 7.3 or greater (a separate package from Kernel), there is a different recommended configuration for the file servers. Please see the documentation from Kernel Toolkit for information on how to set up the UCI ASSOCIATION, TASKMAN SITE PARAMETERS, and VOLUME SET files on MSM-DOS file servers when running performance monitoring.

■ VOLUME SET

VOLUME SET	1 entry per file server: FSA, FSB, etc.
TYPE	FILE SERVER
INHIBIT LOGONS?	Blank or NO
LINK ACCESS?	Blank or YES
OUT OF SERVICE?	Blank or NO
REQUIRED VOLUME SET?	YES
TASKMAN FILES UCI	MGR
TASKMAN FILES VOLUME SET	Blank
REPLACEMENT VOLUME SET	Blank
DAYS TO KEEP OLD TASKS	Blank

Mixed Systems

Mixing different systems together in the same local network (for example, allowing a stand-alone 486 machine running MSM to interact with a DSM for OpenVMS cluster) has become easier than it used to be. As long as all of the environments are described correctly in whatever VOLUME SET file each environment sees, nothing special need be done.

One situation may arise if the systems do not share the ^%ZIS global, in which case they will be looking at different copies of the three configuration files. If you keep the files in synchronization, they will be able to send tasks back and forth between systems.

Mounted Volume Sets

Mounted volume sets that possess their own manager UCIs should be treated like any other independent TaskMan environment. Those without their own manager UCI need the support of their primary volume set, which is obtained by adjusting the VOLUME SET file entry of the mounted volume set. Here is a sample volume set, MNT, mounted on the primary volume set PRI, whose manager UCI is MGR:

■ VOLUME SET

```
VOLUME SET                      MNT
INHIBIT LOGONS?                 Blank or NO
LINK ACCESS?                    Blank or YES
OUT OF SERVICE?                 Blank or NO
REQUIRED VOLUME SET?            Blank or NO
TASKMAN FILES UCI               MGR
TASKMAN FILES VOLUME SET        PRI
REPLACEMENT VOLUME SET          Blank
DAYS TO KEEP OLD TASKS          Up to you (may be blank).
TYPE                            GENERAL PURPOSE VOLUME SET
```

Manager Startup

You may want to configure your system so that, on CPUs where the Manager should run, a Manager starts up every time the CPU starts up. Otherwise, you will need to manually start up the Manager each time you start up those nodes that should run the Manager.

For most sites, only one Manager is needed to cover each environment. Therefore, this section focuses on starting up only a single Manager.

Neither the Manager nor the Submanagers will start up on a box-volume pair of the wrong type, so pay attention to how you fill in the MODE OF TASKMAN field of the TASKMAN SITE PARAMETERS file. If you want the Manager to start, you must make sure this field is set to either a Print Server or a General Processor.

Getting the Manager to start up when the system does is accomplished in several ways, depending on your type of operating system.

MSM-DOS Systems

On MSM-DOS systems you must edit the SYSGEN parameters to specify ZTMB as a routine to be invoked automatically upon system startup.

DSM for OpenVMS Systems

On DSM for OpenVMS systems, you need to edit the DSM site-specific installation and startup command procedure, usually called SYS$STARTUP:DSM$INSTALL_SITE.COM. The command file should start up a Manager if the current node is a Manager node:

```
$ NODE = "''F$EDIT(F$GETSYI("SCSNODE"),"COLLAPSE,TRIM,UPCASE")'"
$ IF NODE .EQS. "662A1" THEN SUBMIT/USER=DMANAGER SYS$MANAGER:SYS_TASKMAN.COM
```

The above example assumes you are using DHCP's *VAX DSM Systems Guide* (cookbook) setup, and your DSM environment manager username is DMANAGER. If you are running TaskMan from a DCL context, you would use a different username (see the section at the end of this chapter entitled Running TaskMan with a DCL Context).

Also in the above example, the name SYS_TASKMAN.COM refers to the command file that starts up Task Manager (see the *VAX DSM Systems Guide* (cookbook) for an example of this command file.) This DCL command file runs START^ZTMB and then quits. Sites running TaskMan with a DCL context may need to use a different command file (again, see the section at the end of this chapter entitled Running TaskMan with a DCL Context).

Multiple Managers and Load Balancing

Task Manager supports the running of multiple Manager processes (but only one Manager process should run per CPU). Running multiple managers is probably useful only at large sites; at a large site, doing this can enable tasks to be processed more quickly than if only one CPU runs a manager. An added bonus with multiple managers is that if one CPU running a manager becomes unavailable, Manager(s) will still run on the other CPU(s), with no further re-configuration required.

Configuration for Multiple Managers

Each node that runs a Manager must have its own entry (box-volume pair) in the TASKMAN SITE PARAMETERS file.

Each CPU must share access to a common ^%ZTSK and ^%ZTSCH global, and have access to the same devices. This setup is called redundant print servers. Because of this, all CPUs must run the same M implementation (MSM-DOS or DSM for OpenVMS).

Starting Up, Pausing, and Stopping Multiple Managers

You will need to start a Manager on each CPU where a Manager should run. Whatever steps you follow to start a single Manager, you will need to repeat for any additional nodes on which you want to run additional Managers.

The options that place Task Manager in a wait state and stop Task Manager are not CPU-specific; they will affect all running Managers across the system.

Load Balancing

The LOAD BALANCE ROUTINE field in the TASKMAN SITE PARAMETERS file holds the name of a function that returns a CPU's load rating. This field is only useful if you are running multiple Managers.

To use load balancing, enter a routine name in the LOAD BALANCE ROUTINE field for each participating CPU's BOX-VOLUME PAIR entry. Kernel supplies load balancing routines for MSM-DOS and DSM for OpenVMS:

M Implementation	Kernel Load Balance Routine
MSM version 4	$$MSM4
DSM for OpenVMS	$$VXD

It is all right to run multiple managers without using load balancing; it's also all right if load balancing is set up and only one manager is running (that manager automatically takes all jobs itself). If one manager's CPU has LOAD BALANCE ROUTINE filled in, and another running manager's CPU doesn't, the managers will act as if no load balancing is taking place. In short, the only ramification from various combinations of managers with the LOAD BALANCE ROUTINE field filled in or not filled in is that load balancing might not take place.

The load balancing routine must be an extrinsic function that returns a value between 1 and 256. 1 means a CPU with no capacity for more work and 256 means a CPU that is idle. The VXD algorithm is:

Capacity left= Available jobs - Active jobs - (4 * Computable jobs).

Each CPU performing load balancing compares its current CPU capacity with that of the other nodes running Managers. If the current CPU has a lower rating than the other CPUs, it puts itself in a BALANCE state and waits to let the other CPUs take up the load before running more jobs itself.

Submanagers should have a low retention time, e.g., 30 seconds, when using load balancing. This helps load balancing retain a true picture of the load on each Manager node.

Monitor Taskman Option

On a system where multiple managers are running, the Monitor Taskman option shows a combined view of the operation of multiple managers.

If the current node (the one where you are running the Monitor Taskman option) has a lower rating than other nodes, Monitor TaskMan will show that the current node is in a BALANCE state.

Device Handler's Influence on TaskMan

Certain DEVICE file fields strongly affect TaskMan's behavior. System managers should keep these effects in mind as they configure their systems' devices.

VOLUME SET (CPU)

If this field is not filled in, TaskMan considers this device to be available from all volume sets. If it is filled in, TaskMan makes sure all tasks that need this device start on the designated volume set.

TYPE

Any tasks that must wait for HFS- or SPL-type devices are rescheduled for ten minutes in the future, instead of being placed in a list of waiting tasks. This is because these lists are checked through repeated opens, which may contaminate the output of these two special types of devices.

PRIORITY AT RUN TIME

This field overrides the default priority that system managers can establish for tasks using the Site Parameters Edit option on the Edit TaskMan Parameters menu.

TASKMAN PRINT A HEADER PAGE?

If this field is set to YES for the device being opened by the Submanager, a header page is printed. The header page distributed with TaskMan is very simple, and system managers may substitute their own locally written header pages. To do this, you must rename your header page routine as ^%ZTMSH, the name of the one distributed with TaskMan.

Whenever you install new versions of the Kernel, it overwrites ^%ZTMSH with the default copy, so you should maintain your local version by doing the following:

- **Keep your local header page routine saved somewhere under a local name.**
- **After each Kernel install, re-save the locally named copy as ^%ZTMSH.**

On the following page, an example shows an alternative to the default header page distributed with the Kernel.

Customized Header Page Routine

```
%ZZTMSH     ;SEA/RDS-Local: Sample Header Page ;3/9/92 11:17 ;
         ;;1.0;Local;;
         ;
LOCAL    ;Print The Local Header Page
         ;
B        ;build text lines
         S X1=$P($G(^VA(200,DUZ,0)),U) I X1="" S X1="name unknown"
         S X2=$P($G(^VA(200,DUZ,5)),U,2) I X2="" S X2="unlisted mail
stop"
         S X3=$P($G(^VA(200,DUZ,.13)),U,2) I X3="" S X3="unlisted
phone number"
         S ZZLINE1=$$FORMAT(" "_X1_"  ("_X2_")  "_X3_"",IOM)
         S ZZLINE2=$$FORMAT(" "_ZTDESC_" ",IOM)
         S ZZLINE3=$$FORMAT(" "_ION_"  "_$$HTE^XLFDT($H)_"",IOM)
         ;
D        ;display each line three times
         F X=1:1:3 W !,ZZLINE1
         W ! F X=1:1:3 W !,ZZLINE2
         W ! F X=1:1:3 W !,ZZLINE3
         Q
         ;
FORMAT(ZZTEXT,ZZIOM)  ;local extrinsic function
         ;input: text to be formatted, and margin width
         ;output: text filled out to margin width -3 with *characters
         N ZZ1,ZZFILLED
         S ZZ1=ZZIOM-3-$L(ZZTEXT)\2
         S $P(ZZFILLED,"*",ZZ1*2+1)=""
         S $P(ZZFILLED,"*",ZZ1+1)=ZZTEXT
         I $L(ZZFILLED)+3-ZZIOM S ZZFILLED=ZZFILLED_"*"
         Q ZZFILLED
```

Customized Header Page

```
********** MARSHALL,RICK  (ISC)  FTS 764-2283 **********
********** MARSHALL,RICK  (ISC)  FTS 764-2283 **********
********** MARSHALL,RICK  (ISC)  FTS 764-2283 **********

********************** SAMPLE TASK **********************
********************** SAMPLE TASK **********************
********************** SAMPLE TASK **********************

********** LAT DEVICE  Jun 30, 1992@14:34:01 ************
********** LAT DEVICE  Jun 30, 1992@14:34:01 ************
********** LAT DEVICE  Jun 30, 1992@14:34:01 ************
```

Running TaskMan with a DCL Context

When run from a DCL context, TaskMan runs as a privileged OpenVMS user. The Manager runs as a job that originates from a node-specific OpenVMS batch queue and, by default, submits new submanagers to the same queue as needed.

One advantage to running TaskMan from a DCL context is that it allows jobs to be queued to specific CPUs. When a program calls ^%ZTLOAD, it can request that the job run on a specific CPU/node in your cluster (via the ZTCPU input variable). Unless you are running TaskMan in a DCL context (on DSM for OpenVMS systems only), this request will probably fail (and possibly cause the task not to run). When TaskMan runs with a DCL context, however, the Manager can submit the job as a new submanager to a given CPU's TaskMan batch queue.

Another reason to run TaskMan from a DCL context is to run the VAX/Alpha Performance Monitor (VPM). VPM is part of Kernel Toolkit (a package separate from Kernel). VPM requires that TaskMan run in a DCL context at OpenVMS sites.

Multiple DSM environments on the same CPU can each run TaskMan in a DCL context. Although TaskMan in each DSM environment shares the same account, directory, DCL command files, and batch queue, jobs will run in the environment specified in each environment's VAX DSM ENVIRONMENT FOR DCL site parameter.

Required OpenVMS Setup

These instructions show how to set up TaskMan to run with a DCL context. These instructions only apply to DSM for OpenVMS systems.

1. **Establish an OpenVMS user with the name TASKMAN that runs the TaskMan jobs.**

   The TASKMAN account requires the following OpenVMS privileges: TMPMBX, NETMBX, OPER and CMKRNL. It also requires ACL privileges to run in environment manager (DSM/MANAGER) mode (if ACL protection has been implemented for DSM).

   Set the TASKMAN account's "Prclm" parameter to at least 2. The account's default directory also must contain the following DCL command files (see the examples of these command files elsewhere in this section):

   - **ZTMWDCL.COM**
   - **ZTMSWDCL.COM**

Both Manager and Submanager processes run under this OpenVMS account. For security reasons, this account should have a password.

2. **Set up a batch queue on each node with a name of TM$<nodename> and a job limit that equals or exceeds the TaskMan job limit + the number of ZTM jobs, as follows:**

```
$ INIT/QUEUE/BATCH/OWN=TASKMAN/JOB=nnn/ON=<node>:: TM$<node>
$ START/QUEUE/ON=<node>::TM$<node>
```

   This queue needs to be monitored regularly. If the queue stops, all TaskMan jobs will be suspended.

3. **The system startup command file needs to define a logical that points to TASKMAN's home directory as follows:**

```
$ DEFINE/SYS DHCP$TASKMAN <diskname>:[<directory>]
```

   Because this command is placed in the system startup file, it won't take effect until the system is rebooted. To avoid rebooting the system, you can also issue the above command yourself, directly from the DCL command line. To issue this command yourself, you should be certain your account has the appropriate privileges, however.

4. **Change the site parameter in the TASKMAN SITE PARAMETERS file called VAX DSM ENVIRONMENT FOR DCL. Set it to the name of the DSM for OpenVMS environment manager account. This causes the Manager to use %SPAWN to SUBMIT submanagers with a DCL context, rather than using the JOB command.**

5. **If you have been starting TaskMan Manager(s) from the DCL command file SYS$STARTUP:DSM$INSTALL_SITE.COM, continue to do so. The OpenVMS account that starts up DSM should be privileged enough (that is, with SYSPRV enabled) to start up TaskMan in a DCL context.**

   Otherwise, if you manually start up managers, you will need to do so from either the TASKMAN OpenVMS account, or a OpenVMS account with both OPER and SYSPRV privileges. Then, D ^ZTMB to start a Manager.

   In either case, TaskMan (ZTMB) looks at the VAX DSM ENVIRONMENT FOR DCL site parameter and starts submanagers using either the JOB command or %SPAWN as set by the site parameter.

6. **Shut down Task Manager and restart as described in the How to Restart TaskMan when Running in a DCL Context section.**

How to Restart TaskMan when Running in a DCL Context

To manually restart TaskMan when TaskMan is running in a DCL context, you can either:

- **Sign-in as OpenVMS user TASKMAN and DO RESTART^ZTMB.**
- **Sign in from an OpenVMS account that has the OPER and SYSPRV privileges and DO RESTART^ZTMB. This submits the Manager to run under the username TASKMAN.**

In either case, however, do not use the Restart Task Manager option in the Kernel menus; it is not compatible with TaskMan in a DCL context.

■ ZTMWDCL.COM Command File

```
$!--------------------------------------------------------------------
$! ZTMWDCL.COM - Run TaskMan in a DCL Context
$! * KERNEL 8.0 *
$!
$! P1 is the environment that TaskMan should start in.
$!       P2 = null to START and 1 to RESTART
$!
$ entry="START"
$ if p2 .eq.  1 then entry="RESTART"
$ dsm/E='p1/man 'entry^%ZTM0
$!
```

■ ZTMSWDCL.COM Command File

```
$!--------------------------------------------------------------------
$! ZTMSWDCL.COM - Start Submanager with a DCL Context
$! * KERNEL 8.0 *
$! p1 is the VAX DSM environment
$! p2 is the UCI to start.
$! p3 is the VOL to start.
$ dsm/E='p1/UCI='p2/VOL='p3 START^%ZTMS
$!
```

Example of OpenVMS User TASKMAN on ALPHA AXP Systems

```
Username: TASKMAN                           Owner:
Account:                                    UIC:    [50,20]
([DEV,TASKMAN])
CLI:      DCL                               Tables: DCLTABLES
Default:  USER$:[TASKMAN]
LGICMD:   LOGIN
Flags:  DisCtlY Restricted DisWelcome DisReport
Primary days:   Mon Tue Wed Thu Fri
Secondary days:                     Sat Sun
No access restrictions
Expiration:            (none)    Pwdminimum:  6   Login Fails:     0
Pwdlifetime:        180 00:00    Pwdchange:  19-NOV-1992 14:12
Last Login: 20-NOV-1992 10:34 (interactive), 20-NOV-1992 10:44 (non-
interactive)
Maxjobs:         0  Fillm:       100  Bytlm:        64000
Maxacctjobs:     0  Shrfillm:      0  Pbytlm:           0
Maxdetach:       0  BIOlm:       100  JTquota:       1024
Prclm:           2  DIOlm:       100  WSdef:          512
Prio:            4  ASTlm:       300  WSquo:         1024
Queprio:         0  TQElm:        10  WSextent:     16384
CPU:        (none)  Enqlm:       300  Pgflquo:      50000
Authorized Privileges:
  CMKRNL TMPMBX OPER NETMBX
Default Privileges:
  CMKRNL TMPMBX OPER NETMBX
Identifier                      Value           Attributes
  DSM$MANAGER_KDAMGR            %X8001001A
```

Example of OpenVMS TASKMAN Queue

```
ISC6A1$ SH QUE/FULL TM$ISC6A1 <RET>

 Batch queue TM$ISC6A1, available, on ISC6A1::
   /BASE_PRIORITY=4 /JOB_LIMIT=50 /OWNER=[DEV,TASKMAN]
 /PROTECTION=(S:E,O:D,G:R,W:W)

ISC6A1$
```

■ Example of ACL Setup for TaskMan

Note: Providing ACL access for the TaskMan user account is only necessary if ACL protection has been implemented for DSM for OpenVMS.

```
>D ^ACL <RET>

Environment Access Utilities

    1.  ADD/MODIFY USER                (ADD^ACL)
    2.  DELETE USER                    (DELETE^ACL)
    3.  MODIFY ACTIVE AUTHORIZATIONS   (^ACLSET)
    4.  PRINT                          (PRINT^ACL)

Select Option > 1 <RET>  ADD/MODIFY USER

VMS User Name:   >   TASKMAN <RET>

ACCESS MODE    VOL       UCI       ROUTINE
-----------    ---       ---       -------

No access rights for this user.

Access Mode ([M]ANAGER, [P]ROGRAMMER, or [A]PPLICATION):   > M <RET>

USER           ACCESS MODE    VOL       UCI       ROUTINE
----           -----------    ---       ---       -------

TASKMAN        MANAGER

OK to file?   <Y>   Y <RET>

Identifier DSM$MANAGER_KDAMGR granted to user TASKMAN.  Modifications
have been made to the VMS rights database.  These changes will take
effect the next time TASKMAN logs in to VMS.

VMS User Name:   > <RET>

OK to activate changes now?   <Y>   Y <RET>

Creating access authorization file:  KDA$:[DMANAGER]DSM$ACCESS.DAT.
    Press RETURN to continue <RET>

Environment Access Utilities

    1.  ADD/MODIFY USER                (ADD^ACL)
    2.  DELETE USER                    (DELETE^ACL)
    3.  MODIFY ACTIVE AUTHORIZATIONS   (^ACLSET)
    4.  PRINT                          (PRINT^ACL)

Select Option >
```

Chapter 24 Task Manager System Management: Operation

This chapter describes how to operate Task Manager. This chapter discusses:

- **Taskman Management Menu**
- **Taskman Management Utilities**
- **TaskMan Direct Mode Utilities**
- **Scheduling Options**
- **The Taskman Error Log**
- **Troubleshooting**

TaskMan Management Menu

The Taskman Management menu [XUTMGR] is the main point of entry into the TaskMan options. It contains the following options:

Schedule/Unschedule Options
One-time Option Queue
Taskman Management Utilities ...
List Tasks
Dequeue Tasks
Requeue Tasks
Delete Tasks
Print Options that are Scheduled to run
Cleanup Task List
Print Options Recommended for Queueing

The TaskMan Management Utilities submenu and the option scheduling options are discussed later in this chapter. The options for listing, dequeuing, requeuing, deleting, and cleaning up tasks are discussed first.

List Tasks [XUTM INQ]

```
SYSTEMS MANAGER MENU ...                                          [EVE]
Taskman Management ...                                        [XUTM MGR]
   List Tasks                                                 [XUTM INQ]
```

Beginning with Kernel V. 8.0, the TASKS file (in ^%ZTSK) is VA FileMan compatible, i.e. you can use VA FileMan to print out information about a task. However, the List Task option also provides a way to examine tasks in the TASKS file. The List Tasks option allows you to choose between several useful ways of selecting tasks. When you choose this option, it presents you with this menu:

```
                    List Tasks Option

                         All your tasks.
                         Your future tasks.
                         Every task.
                         List of tasks.
                         Unsuccessful tasks.
                         Future tasks.
                         Tasks waiting for a device.
                         Running tasks.

                    Select Type Of Listing:
```

Several choices only appear on the list when there are tasks in those collections to be displayed. Remember, the TASKS file can be CPU/volume set-specific. This means that the option can only display tasks from the TASKS file on the current volume set/CPU.

Holders of the ZTMQ key see a slightly different list of selections. Instead of "All your tasks" and "Your future tasks" they see "All of one user's tasks" and "One user's future tasks." These two selections are generic versions of those available to normal users: they allow the holder to see any user's tasks and start by prompting the holder for the user whose tasks should be shown. Other than that, they are identical to the selections used by normal users, described below.

"All your tasks" displays every task in the TASKS file on the current volume set/CPU that you created. If you have no tasks scheduled, the option gives you the message: "You have no tasks in this volume set's TASKS file."

"Your future tasks" displays those tasks you created that are currently scheduled to run. If there are none, the option tells you.

"Every task" lists every task in the TASKS file.

"List of tasks" allows you to list one or more tasks by task number. You can specify individual tasks separated by commas along with ranges of tasks using a hyphen.

"Unsuccessful tasks" lists three kinds of tasks: those that were rejected by the Manager's validation process, those that encountered an error while they were running, and those that were unscheduled through the Dequeue Tasks option.

"Future tasks" shows all tasks that are in the Schedule List or the Waiting List. It does not show the tasks that are in the Job List. In other words, it shows all tasks that are scheduled to run but not those that are currently being run or those that are ready to be run. "Future Tasks" is not offered by the List Tasks option if the Schedule List and Waiting List are empty (an unlikely occurrence at most sites).

"Tasks waiting for a device" shows just the Waiting List, which can be a useful way of isolating problem printers. If there are no tasks currently waiting for output devices to become available, the List Tasks option won't show this choice.

"Running tasks" shows tasks that are currently running. The Troubleshooting section has a discussion of how TaskMan knows a task is running.

Although each choice shows a different set of tasks, the format for the output is the same. Here is a sample display from "All your tasks":

```
All tasks that you created...

2572: ALIVE^%INDEX, %INDEX of 1 routine.  Device QMS-17P.  VAH,KXX.
      From TODAY at 10:55,  By you.  Scheduled for TODAY at 12:05

End of listing.  Press RETURN to continue:
```

In the upper left-hand corner of each entry is the task number. What follows the task number is either an option name (e.g., XUTM QCLEAN) or a routine entry point (e.g., ERROR^ZTMZT) depending on whether the task was a queued routine or a queued option. This is generally followed by a description of the task. The device to which the task was queued (if any), along with the account in which the task was/is scheduled to run, complete the first line. The next line contains the time the task was created followed by an identification of the creator. In the case of tasks that requeue themselves, this date and time represents when the task was last requeued.

When the creator's DUZ number is not listed in the NEW PERSON file, the phrase "USER #" followed by the DUZ is substituted. Finally, the status of the task is shown. (See the Troubleshooting section for a list and description of the status messages.)

Dequeue Tasks [XUTM DQ]

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                      [XUTM MGR]
   Dequeue Tasks                                             [XUTM DQ]
```

This option allows you to unschedule a task so that the task still exists in the TASKS file but is no longer in the Schedule, Waiting, or Job List. The process of unscheduling a task is called "dequeuing". This option allows you to dequeue any one task or range of tasks. A task that you dequeue has a status of NOT QUEUED in a List Tasks display.

The option first prompts you for the task number. Entering one question mark gets you a short explanatory message, but entering two puts you in the List Tasks option to find the task you are interested in dequeuing. When you leave the List Tasks option, you automatically return to the task number prompt.

If you enter the number of a nonexistent task, List Tasks tells you and then prompts you for another task number. If you enter the number of a task that does exist, the option displays the task and asks you if you are sure. Answering "no" returns you to the task number prompt, whereas a "yes" dequeues the task and then returns you to the task number prompt.

You can also enter a list of tasks to be dequeued. The list can include single tasks separated by commas and ranges of tasks consisting of two numbers separated by a hyphen. After you enter the list, you are asked if you want to know the actual number of tasks in the list. You are then asked if you want a display of the actual tasks that are about to be dequeued.

Only holders of the ZTMQ key can dequeue any task. Others can only dequeue their own tasks as identified by their DUZ.

Requeue Tasks [XUTM REQ]

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                      [XUTM MGR]
   Requeue Tasks                                            [XUTM REQ]
```

A benefit of the Dequeue Tasks option is that it is completely non-destructive. If you dequeue a task and subsequently change your mind, you can use the Requeue Tasks option to requeue the task exactly the way that it was. You can also use this option to change some of the details of a task that is already queued.

As with XUTM DQ, you are first prompted for a Task Number with the same help available. Here, you may only enter a single task, not a range. The task

is then displayed, and you are asked for a new run time with the default being either the original or current run time (whichever applies). The next question is "Do you wish to requeue this task to a device?", with the default depending on whether the task originally requested an output device. If you answer "yes," the option asks you to specify an output device using the original output device (if there was one) as a default. The option also allows you to adjust the task's priority.

The task is requeued according to your specifications. Requeuing involves completely dequeuing the task so that your task does not run twice, making the changes you requested, and placing the task back on the Schedule List. Notice that the task is not dequeued until after you specify the changes you want to make. If you want to modify a task that may start running soon, it is usually a good idea to dequeue it first.

The ZTMQ key affects this option in two ways. Those who do not hold the key are limited to requeuing only their own tasks. Also, they are not prompted to change the priority.

Delete Tasks [XUTM DEL]

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                      [XUTM MGR]
   Delete Tasks                                             [XUTM DEL]
```

This option has the same structure as the Dequeue Tasks option. The only difference is that where dequeuing a task just removes it from the lists (unschedules it), the Delete Tasks option also deletes the task from the TASKS file. When you have deleted a task, there is no reference to that task anywhere in TaskMan's files.

Only holders of the ZTMQ key can delete any task. Others can only delete their own tasks as identified by their DUZ.

Cleanup Task List [XUTM TL CLEAN]

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                      [XUTM MGR]
   Cleanup Task List                                    [XUTM TL CLEAN]
```

You can use this option to remove a task entry from a task list for a job that is no longer running. This might happen when a process is forcibly exited, but TaskMan still believes the task is running. You can use this option to tell TaskMan which tasks you forcibly exited. TaskMan then removes those tasks from its list of running tasks.

TaskMan Management Utilities

A submenu on the Taskman Management menu, called TaskMan Management Utilities, provides several options to set up, monitor, and modify the TaskMan environment. The Taskman Error Log submenu is discussed later in this chapter. This section describes the other options.

The Taskman Management Utilities menu contains the following options:

Monitor Taskman
Check Taskman's Environment
Edit Taskman Parameters ...
Restart Task Manager
Place Taskman in a WAIT State
Remove Taskman from WAIT State
Stop Task Manager
Taskman Error Log ...
Clean Task File
SYNC flag file control

Monitor TaskMan [XUTM ZTMON]

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                       [XUTM MGR]
   Taskman Management Utilities ...                         [XUTM UTIL]
      Monitor Taskman                                      [XUTM ZTMON]
```

The monitor gives you a screenful of information about the current state of TaskMan and offers you several ways to get more information. The monitor focuses on the current state of the Manager itself and on the contents of the Schedule file.

As you use this option, you will acquire an intuitive understanding of how these lists should look and behave when your system is healthy. Spending the time using this option to get that intuition will save you troubleshooting time by helping you to notice problems sooner.

The Run Node

The first section of the Monitor TaskMan screen reports whether the Manager is currently running on your machine and, if so, whether or not it is being delayed. This is accomplished by comparing Task Manager's Run Node to the M $HOROLOG variable. Under normal circumstances they should be within 15 seconds of each other, though certain conditions can cause a difference of up to two minutes. Any difference greater than that, however, is a sign that the Manager is being delayed, typically by a problematic device or a recurring error. Of course, the Manager is also likely to fall behind if the

system is saturated to the point where all of the jobs on the system are slow. The last line of the first section evaluates the difference and guesses at the Manager's current condition. The $HOROLOG values are translated into an external format for your convenience in understanding the values.

■ Sample Monitor TaskMan Screen

```
Checking TaskMan.  Current $H=54180,45147  (MAY 04, 1989 @12:32:27)
                     RUN NODE=54180,45145  (MAY 04, 1989 @12:32:25)

  TaskMan is current.

  Checking the Status List:
       TaskMan job 4 status 54180,45145^RUN^Main Loop.
       There are 3 idle submanagers

  Checking the Schedule List:
       TaskMan has 29 tasks in the Schedule List.
       None of them are overdue.

  Checking the IO Lists:  Last TM scan: 54180,45146^_LTA9995:
   Device: _LTA9995: is not available, and there are 7 tasks waiting.

  Checking the Job List:
       There are no tasks waiting for partitions.
       For KDE:ISC6V2 there are 2 tasks.   Not responding


  Checking the Task List:
       There are 5 tasks currently running.

  Enter monitor action: UPDATE//
```

The Status List

The Status List is where each Manager periodically reports its current status. The job number of the Manager is reported both for ease of location on a system status report and also to distinguish between multiple Managers (if there are more than one). Under normal circumstances, the Manager removes its entry from the Status List when it shuts down, but if a Manager stops abnormally (e.g., RJD or FORCEX) its entry is usually left on the list. The list is updated and cleaned out whenever a new Manager is started or restarted.

The status of a Manager consists of three parts. The first part is a date and time that should equal the RUN node's date and time, and like that node it should be close to the current $HOROLOG. The second part is the Manager's state, and the third part describes special circumstances. The Manager can be in one of five states at any given time: BALANCE, ERROR, PAUSE, RUN, or WAIT. RUN is the normal state, with a description of "Main Loop."

The Manager's status is the most important piece of information the monitor gives, and it should always be the first thing checked when troubleshooting problems. The possible state messages are described in detail in the section on Troubleshooting in this chapter.

The Schedule List

The Schedule List always shows the number of tasks currently scheduled to run and checks the times for which they are scheduled to determine whether any of them should already have started. When many tasks are queued to run at the same time, it is not unusual for the Manager to be a little late in sending off the last few.

When most of the tasks on the Schedule List are overdue, however, the Manager is probably having problems keeping up. This is not a normal condition. If the problem is not a recurring error or a difficult output device, the most likely culprit is your default setup in the TASKMAN SITE PARAMETERS file. Another possible problem is that TaskMan is trapping many errors or trying to access a very slow link between volume sets. If the problem is error trapping, the Status List should regularly show the Manager in an Error state. Remember also that if the machine is saturated, all of the jobs on the system, including the Manager, will run slowly.

The IO List

The IO List first shows the last time ($H) a submanager checked the list and the last device checked. The check generally shows how many tasks are waiting for each device in the IO List. The occasional remark "Allocated" means that a submanager has already noticed that the device is available and has allocated the device to a task using the Device Allocation List. Devices should only be allocated for a short time before the Submanager opens the device, making it unavailable.

Understanding how the IO List works can make this particular check very useful. Submanagers handle the Device IO Lists. Unusual behavior in these lists usually points to device or Submanager problems.

There are three fundamental things to look for with this check:

- **When a device becomes available, the Submanagers should notice and start a task running on that device. If the Submanagers don't do this, it is probably time to start looking for problems with the Submanagers.**
- **When a device is allocated, a Submanager should quickly make it unavailable. If this fails to occur, the Submanagers may be having problems. There can be extenuating circumstances, such as the system being very slow, that explain these occurrences.**
- **When many tasks are backed up waiting for the same device, sometimes it is just because that device is busy. Sometimes, however, the device is off-line or out of paper.**

The Job List

The Job List is where tasks wait for partitions, so if many tasks are backed up here you know the Submanagers aren't picking them up. This can be caused by a slow system, TaskMan reaching its job limit, or TaskMan assigning tasks a priority that is too low for them to run. Systems that are too busy will back up in this list, not the Schedule List. The Compute Server Job List is checked here and will let you know about tasks waiting to run on other CPUs and if the submanagers are not starting.

The Task List

The Task List is where TaskMan keeps track of the tasks it has started running. Entries are set into this list when the Submanagers start their tasks and are cleared when the tasks quit or cause errors to be trapped. Killing a task by forcing its process to exit in the middle of execution (using such vendor-specific tools as RJD, RESJOB, FORCEX, KILLJOB, etc.) doesn't give the Submanager a chance to clear the task from the Task list, so the Task List can become inaccurate. If you frequently kill jobs but want to keep your Task List accurate, you will need to manually remove the obsolete entries. The exit action of the Kill off a user's job option [XURESJOB] will help you identify and remove from the list of running tasks those you have forcibly exited.

The Monitor Action Prompt

After summarizing the status of the Manager and the principal lists of the Schedule file, the monitor offers you a choice of actions. They are displayed if you enter ? at the "Enter monitor action:" prompt:

```
Enter <RET> to update the monitor screen.
Enter ^ to exit the monitor.
Enter E to inspect the TaskMan Error File.
Enter S to see a system status listing.
Enter ? to see this message.
Enter ?? to inspect the tasks in the monitor's lists.
```

These actions attempt to bring together those utilities used most often in response to seeing a monitor screen. Updating is the most commonly used choice since you often want to watch how the lists change over time. The TaskMan Error File needs to be easily accessible, not only in case the Manager enters an Error state, but also if a task that should take a long time to run leaves the Job List but never shows up in the Task List. This usually means the task hit an error and quit, which may be confirmed or disproved by a quick glance at the Taskman Error Log. The System Status Report can be used to verify that tasks, Submanagers, and the Manager are indeed running as the monitor suggests.

Some actions at the Monitor Action prompt are not accessible when monitoring TaskMan from the manager's account (using the direct-mode utility D ^ZTMON) .

Inspecting the Tasks in the Monitor's Lists

If you are in a non-library account, you can directly inspect the contents of the various lists. Do this by entering two question marks at the "Enter monitor action:" prompt. You get the following list of choices:

```
  Help For Monitor TaskMan Option

       Schedule List.
       Waiting Lists.
       One Waiting List.
       Job List.
       Task List.
       Link Lists.

  Select Type Of Listing:
```

These listings use the same format as that of the List Tasks option, and show you the contents of the lists at the time you look at them. The One Waiting List listing prompts you to select a device, and the help for that prompt lets you see those devices that have tasks waiting. Many of these lists change

very quickly. So, it is not unusual to enter the help with the intention of seeing the task that was shown by the main screen to be in the Job List, only to be told by the help software that the Job List is now empty. These kinds of experiences are simply part of troubleshooting TaskMan.

While these monitor actions are useful, there are still times when you must leave the monitor to follow up on information you saw there. For example, you may want to check the list of unsuccessful tasks or to list a specific task; both these actions require using the List Tasks option.

Taken as a whole, the checks that make up the monitor can save you a lot of time in trying to evaluate TaskMan's status. The example shown here is of a healthy and not very busy Manager. Monitors at sites usually show considerably more activity, especially in the Waiting Lists.

Check Taskman's Environment [XUTM CHECK ENV]

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                      [XUTM MGR]
   Taskman Management Utilities ...                        [XUTM UTIL]
      Check Taskman's Environment                       [XUTM CHECK ENV]
```

This option presents two screens of information about TaskMan's environment on the current CPU. The first screen performs all of the checks that the Manager does whenever it starts, restarts, or encounters an error. The second screen shows what values the Manager is using for its definition variables. This information can be very useful in pinpointing startup problems, in verifying that the Manager is using the information you want it to use and in getting a general feel for how you have defined your system's task management.

This first screen goes through each step that the Manager goes through when it starts or restarts and reports the results. If your Manager is failing to start, this screen should identify any problem with the environment.

The second screen reports more information about the current TaskMan environment. The first group of four items identifies the current TaskMan operating environment. The next group of items reports the values of some Kernel site parameters that are important to TaskMan. These parameters, as well as all the other parameters that TaskMan uses, are described in detail in the TaskMan Parameters Menu section of this chapter. The last four items show if logons are being inhibited and how many partitions TaskMan currently has to work with. These values show how busy your system is, as well as how busy it can become. Their importance is also described in the discussion of parameters.

Check TaskMan's Environment, First Screen

```
Checking Task Manager's Environment.

Checking TaskMan's globals...
     ^%ZTSCH is defined!
     ^%ZTSK is defined!
     ^%ZTSK(0) is defined!
     ^%ZIS(14.5,0) is defined!
     ^%ZIS(14.6,0) is defined!
     ^%ZIS(14.7,0) is defined!

Checking the ^%ZOSF nodes required by TaskMan...
     All ^%ZOSF nodes required by TaskMan are defined!

Checking the links to the required volume sets...
     There are no volume sets whose links are required!

Checks completed...TaskMan's environment is okay!

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

Check TaskMan's Environment, Second Screen

```
Here is the information that TaskMan has:
     Operating System:  VAX DSM(V6)
     Volume Set:  ROU
     Cpu-volume Pair:  ROU:ISC6V0
     TaskMan Files UCI and Volume Set:  MGR,ROU

     Log Tasks?  N
     Default Task Priority:
     Submanager Retention Time: 30
     Taskman Hang Between New Jobs: 1
     TaskMan running as a type: GENERAL
     TaskMan is using VAX DSM environment: KDAMGR
     TaskMan is using '$$VXD for load balancing


     Logons Inhibited?:  N
     Taskman Job Limit:  35
     Max sign-ons: 40
     Current number of active jobs: 25
End of listing.  Press RETURN to continue:
```

Restart Task Manager [XUTM RESTART]

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                      [XUTM MGR]
   Taskman Management Utilities ...                        [XUTM UTIL]
      Restart Task Manager                                [XUTM RESTART]
```

The Manager generally starts automatically when your system comes up. If the Manager crashes or is stopped, you can use this option to restart it. The option first checks the RUN node and calculates whether it thinks the Manager is currently running. If this option believes the Manager is running, it will ask you if you are sure you want to restart another Taskman; you must answer YES to start the Manager. If XUTM RESTART thinks the Manager has stopped, it will ask you for confirmation before jobbing out a new Manager. If XUTM RESTART believes the Manager to be active when you know for sure that it has failed, you can invoke XUTM STOP to prove to XUTM RESTART that the Manager really has stopped. Then you will be able to restart it.

Place Taskman in a WAIT State [XUTM WAIT]

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                      [XUTM MGR]
   Taskman Management Utilities ...                        [XUTM UTIL]
      Place Taskman in a WAIT State                        [XUTM WAIT]
```

The WAIT state (as described in the Troubleshooting section of this chapter) is a condition in which the Manager does nothing but wait for you to release it. Putting a stop to the Manager's activities without actually shutting down the Manager can often be very useful. For example, with the Manager in a WAIT state, you can look at the tasks after they are queued but before the Manager has a chance to validate them. This can help you isolate problems caused by the queuing process from those caused by the validation process. Another time you may want to create a WAIT state is before restarting a manager that has stopped. This prevents the Manager from processing any tasks when it first starts up; the Manager will check out its environment and then WAIT for your command to continue. This option gives you a way to switch the Manager's activities on and off without having to completely shut down and restart the Manager.

When you select the XUTM WAIT option, you are also prompted with the question "Should active submanagers shut down after finishing their current tasks?". If you answer "yes," the Submanagers on the current volume set/CPU will quit when they finish a task instead of recycling. If you answer "no," the Manager enters a WAIT state and the Submanagers continue with their business. If you also want to keep the Submanagers from searching the Waiting List and the Job List for tasks, you need to explicitly say so at this prompt. This inhibition of the Submanagers' recycling remains in effect

either until you remove the WAIT state or until a new Manager starts or restarts, whichever comes first.

Remove Taskman from WAIT State [XUTM RUN]

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                       [XUTM MGR]
   Taskman Management Utilities ...                         [XUTM UTIL]
      Remove Taskman from WAIT State                         [XUTM RUN]
```

This option simply undoes the effects of XUTM WAIT, allowing the Manager to process tasks and allowing the Submanagers to recycle (if recycling had been inhibited).

Stop Task Manager [XUTM STOP]

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                       [XUTM MGR]
   Taskman Management Utilities ...                         [XUTM UTIL]
      Stop Task Manager                                      [XUTM STOP]
```

The Stop Task Manager option gives you a clean way to stop the Manager from within the menu system. This option also asks if you want the Submanagers to shut down when they finish what they are doing. Note that the WAIT state takes precedence. While the Manager is in a WAIT state, not even XUTM STOP affects it until after you invoke XUTM RUN to release it from the WAIT state; after it is released, it shuts down. This option should always be used to shut down TaskMan, rather than simply killing the TaskMan process, which can leave the TaskMan globals in an improper state and even lose tasks.

SYNC Flag File Control [XUTM SYNC]

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                       [XUTM MGR]
   Taskman Management Utilities ...                         [XUTM UTIL]
      SYNC flag file control                                 [XUTM SYNC]
```

With this option, for any SYNC FLAG entry, you can ZAP it (remove it from the file and delete all waiting tasks with the same SYNC FLAG). You can also choose START NEXT (which resumes running the series of tasks associated with that SYNC FLAG). This is useful when one task in a series of tasks that is synchronized with SYNC FLAG fails.

Clean Task File [XUTM CLEAN]

The TASKS file grows every time a new task is queued. While the SAC requires applications to delete their tasks' entries when they complete, it is possible that older applications may not do this. Other tasks abort with errors; still others are rejected. The result is that ^%ZTSK is always growing. Options are available that clean up the ^%ZTSK global.

```
SYSTEMS MANAGER MENU ...                                          [EVE]
Taskman Management ...                                        [XUTM MGR]
   Taskman Management Utilities ...                           [XUTM UTIL]
      Clean Task File                                        [XUTM CLEAN]
```

In unusual circumstances, you may need to clean the ^%ZTSK global manually. Kernel provides an option called Queuable Task Log Clean Up (discussed below) to regularly clean up the TASKS file in the background.

Only rarely will you not be able to rely on the queued cleanup to perform this function. However, when necessary, you can use the interactive Clean Task File (XUTM CLEAN) option. First, XUTM CLEAN asks you if you are sure you want to clean out the old entries from the TASKS file. If you respond that you are, the option asks you how far back you want to keep old entries. The default is to keep old entries going back a week and to delete the older ones. After you provide this value, the option queues a task to do the cleanup. XUTM CLEAN cannot be queued.

Queuable Task Log Cleanup [XUTM QCLEAN]

The Queuable Task Log Cleanup option, XUTM QCLEAN, resides on the menu ZTMQUEUABLE OPTIONS. This option allows you to purge all of the entries for tasks that are no longer queued (for whatever reason) and to purge the TaskMan error log. It is very useful to be able to queue the cleanup to run automatically each night; XUTM QCLEAN has been distributed to provide this feature. XUTM QCLEAN should not be run interactively; indeed, it is not available from any of TaskMan's menus. To queue this option, use Schedule/Unschedule Options to queue it to run.

The date XUTM QCLEAN starts purging the TASKS file is controlled by the parameter DAYS TO KEEP OLD TASKS in the VOLUME SET file. A value of seven days is recommended. XUTM QCLEAN does not need an output device; so, you can leave that field blank. Once set up, the task automatically runs periodically, cleaning out inactive task entries that are older than the time period specified in the DAYS TO KEEP OLD TASKS parameter. If you want to run this on all of your machines, create an entry in the OPTION SCHEDULING file for each machine on which you want to run it.

TaskMan Direct Mode Utilities

You can use TaskMan's direct mode utilities from both the Manager and Production UCIs. Developers may not call them from applications, however.

>D ^ZTMB: Start TaskMan

This utility can be used to start TaskMan for the first time since system startup. As part of this startup, any tasks scheduled to begin at system startup are fired off.

>D RESTART^ZTMB: Restart TaskMan

This utility restarts TaskMan. RESTART^ZTMB, unlike ^ZTMB, does not fire off the startup tasks and should be used whenever the startup tasks have already been initiated. The Restart TaskMan option uses this entry point.

>D ^ZTMCHK: Check TaskMan's Environment

This utility provides the same functionality as the Check Taskman's Environment option but from programmer mode.

>D RUN^ZTMKU: Remove TaskMan From A WAIT State

This utility provides the same functionality as the Remove TaskMan From A WAIT State option but from programmer mode.

>D STOP^ZTMKU: Stop TaskMan

This utility provides the same functionality as the Stop TaskMan option but from programmer mode.

>D WAIT^ZTMKU: Place TaskMan In A WAIT State

This utility provides the same functionality as the Place TaskMan In A WAIT State option, but from programmer mode.

>D ^ZTMON: Monitor TaskMan

This utility provides the same functionality as the Monitor Taskman option, but from programmer mode.

Scheduling Options

TaskMan lets you, the site manager, schedule options that run regularly as tasks. Menu Manager and TaskMan work together to give you this ability. All you have to do is tell TaskMan which option you want to queue and how you want to queue it.

Which Options to Queue

The first requirement for queuing regards the option type. Only the run, print, and action types of options can be queued. The second requirement is that the option (if a run or action type) must not involve user input! There is nothing to prevent you from queuing an option of the wrong type or from queuing one that prompts the user for input, but doing so results in a failed task. You must be conscious of the nature of the task when you consider creating one that performs an option. If the option itself won't run in the background, then queuing it is pointless. Even options that themselves queue tasks probably cannot be queued, because most of these ask the user for an output device or a run time.

Application packages may make recommendations for scheduling of options. This is a great help to site managers. Recommendations for scheduling Kernel options can be found the *Kernel Installation Guide* and the *Kernel Technical Manual*.

Parent Of Queuable Options [ZTMQUEUABLE OPTIONS]

Some options that are intended to be queued are not intended to be run interactively, so placing such options on a user menu could cause problems. Parent Of Queuable Options, a menu-type option, has no parent in the menu tree and is intended to be used as the parent of all such options.

Printing Options Recommended to Run and Scheduled to Run

```
SYSTEMS MANAGER MENU ...                                          [EVE]
Taskman Management ...                                        [XUTM MGR]
  Print Options Recommended for Queueing[XUTM BACKGROUND RECOMMENDED]
  Print Options that are Scheduled to run     [XUTM BACKGROUND PRINT]
```

The first of these print options displays all options in the OPTION SCHEDULING file that are recommended for scheduling by the option's developer. The second of these print options lists all currently scheduled options on your system. By comparing these two reports, you can see if any options recommended for scheduling are not scheduled on your system (and vice-versa).

Schedule/Unschedule Options [XUTM SCHEDULE]

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                      [XUTM MGR]
   Schedule/Unschedule Options                          [XUTM SCHEDULE]
```

This option is a straightforward VA ScreenMan edit option, and allows you to schedule and unschedule options. After you select the option to schedule, you are prompted for information about the task you want to set up. You can edit the following six fields in the OPTION SCHEDULING file:

- **QUEUED TO RUN AT WHAT TIME**
- **DEVICE FOR QUEUED JOB OUTPUT**
- **QUEUED TO RUN ON VOLUME SET**
- **RESCHEDULE FREQUENCY**
- **TASK PARAMETERS**
- **SPECIAL QUEUEING**

The cross references on these fields make calls to TaskMan's API to update the TASKS file and ^%ZTSCH.

Note: in order to queue a task, its SCHEDULING RECOMMENDED field must be set to YES.

QUEUED TO RUN AT WHAT TIME

To queue an option, select the option and enter a time at least two minutes in the future into the QUEUED TO RUN AT WHAT TIME field. When you enter a time (and date) for the task to run, the task is immediately put on the Schedule List for that time.

How to Delete a Regularly Scheduled Task: Deleting a scheduled task is as simple as entering the "@" character at the QUEUED TO RUN AT WHAT TIME field. TaskMan then searches the current TASKS file for the task that corresponds to the entry in the OPTION SCHEDULING file and deletes it.

If your system has multiple copies of the TaskMan globals, you must use Schedule/Unschedule Options on the same CPU/volume set where your task originated, when you delete the task. Otherwise, the future task in the TASKS file will not be found (and deleted) when you enter an "@" character in the QUEUED TO RUN AT WHAT TIME field.

How to Requeue a Regularly Scheduled Task: Requeuing merely involves placing a new value in the QUEUED TO RUN AT WHAT TIME field. When you do this, the currently scheduled task is deleted (exactly as described above when deleting a scheduled task). Then, a new task is created at the new time to replace the previously scheduled task.

If your system has multiple copies of the TaskMan globals, you must use Schedule/Unschedule Options on the same CPU/volume set where your task originated, when you requeue the a task. Otherwise, the existing future task in the TASKS file will not be found (and deleted) when you enter a new time in the QUEUED TO RUN AT WHAT TIME field.

DEVICE FOR QUEUED JOB OUTPUT

This field is where you can give the task an output device. For print (Report) type options this is obviously mandatory; for run or action types you need to consider if the option needs an output device. Modifying this value for an already-scheduled task merely causes a direct change to the currently scheduled task.

Tasks with an output device are assigned a process name of "Task ####" where #### is the task number; tasks with no output device are assigned a process name of "BTask ####" (with B meaning background).

QUEUED TO RUN ON VOLUME SET

This field lets you designate a volume set or CPU for the task other than your current one. This field is only useful for options that do not have a device selected because most devices are tied to a CPU and thus the task must run on the CPU that has that device.

Modifying this value for an already-scheduled task merely causes a direct change to the currently scheduled task.

Running a task on each CPU for a given option may at times be useful (consider XQBUILDTREEQUE). In such cases, make multiple entries in the OPTION SCHEDULING file, and use the QUEUED TO RUN ON VOLUME SET field to specify the CPU/volume set where each scheduled task should run.

If you leave the DEVICE FOR QUEUED JOB OUTPUT field blank, the task that performs the option runs without a device (or tries to). If you also leave the QUEUED TO RUN ON VOLUME SET field blank, the task runs on the current CPU without a device. If you fill in both fields, TaskMan uses the value of the QUEUED TO RUN ON VOLUME SET field (unless overridden by the VOLUME SET(CPU) field in the DEVICE file entry of the selected device).

RESCHEDULING FREQUENCY

Whenever a task starts running an option, it looks to see what is in the RESCHEDULE FREQUENCY field. If the field is blank, the option does not reschedule itself. If you have filled in this field, the task uses the value you placed in the field to figure out when you want it to run next. Then it updates the QUEUED TO RUN AT WHAT TIME field to reflect the new scheduled time. When this field is updated, the next task in the sequence is scheduled.

If you change the existing value in the RESCHEDULE FREQUENCY field, the new increment is used beginning after the next time the option runs.

There are several formats you can use in this field: every n seconds, hours, days, or months (incremental); on a particular day of the month; or a list of times every n months. See the next page for a list of the code formats for the RESCHEDULING FREQUENCY field.

For the incremental scheduling frequencies (every n seconds, hours, days, or months), the increment is added to the scheduled date and time in the QUEUED TO RUN AT WHAT TIME field to determine when the task should run next. If the incremented time is in the past, however, TaskMan keeps adding the increment until a future time is reached, only then does it reschedule the task. This capability is part of Kernel beginning withV. 8.0, and represents a change from previous versions.

TASK PARAMETERS

The TASK PARAMETERS field provides a way to pass data to a scheduled option. TASK PARAMETERS holds a string that is passed to scheduled jobs through the ZTQPARAM variable. Ideally, the developer of an option that uses the TASK PARAMETERS string should describe the format and meaning of the string in the option's DESCRIPTION field.

SPECIAL QUEUEING

Use this field to designate an option that should run every time that TaskMan is started (but not restarted). By setting this field to S for STARTUP, you designate an option to run every time TaskMan is started.

If you want to the run the startup option on multiple CPUs, make multiple entries in the OPTION SCHEDULING file, and use the QUEUED TO RUN ON VOLUME SET field to specify what CPU/volume set each should run on.

Option Scheduling Code Formats

Code	Frequency
nS	Every n seconds.
nH	Every n hours.
nD	Every n days.
nM	Every n months.
day[@time]	Day of week (see below for day codes).
D[@time]	Every weekday.
E[@time]	Every weekend day (Sat,Sun).
nM(entry[,entry[,...]])	Every n months, at each entry in the parameter list; the entries in the parameter list (for every n months only) can be:

Entry Format	Frequency
dd[@time]	Day of month, e.g., 15.
nday[@time]	Nth day of week in month, e.g., 1W,3W.
L[@time]	Last day of month.
Lday[@time]	Last specific DAY in month, e.g., LM,LT,LW...

day codes used in Option Scheduling Code Formats

M: Monday
T: Tuesday
W: Wednesday
R: Thursday
F: Friday
S: Saturday
U: Sunday

Examples

Code	Frequency
12H	Every 12 hours.
14D	Every 14 days.
1M(1,15)	First and 15th of the month.
1M(L@23:45)	Last day of the month at 11:45 pm.
1M(LS)	The last Saturday of the month.
3M(15@12:00,L@12:00)	Noon (on the 15th and last days), every 3 months.
W@4pm	Each Wednesday at 4 pm.
D	Each weekday.

Problems With Scheduled Options

Once an option has been put on a schedule, it stays on that schedule unless one of the following happens:

- **You delete the task.**
- **The running task aborts while setting up the next task in the sequence; the schedule sequence is broken.**
- **You dequeue the task that is scheduled to run the option. You must either requeue the task or use Schedule/Unschedule Options to start the cycle over.**
- **You change the value in the RESCHEDULE FREQUENCY field. The new increment is used beginning after the next time the option runs.**
- **You change the value in the QUEUED TO RUN AT WHAT TIME field. The currently scheduled task will be unscheduled and a new one will be scheduled for the time you specify.**

Another peculiarity in this process involves using a monthly scheduling frequency. What should happen if on January 31st you queue an option and give it a monthly scheduling frequency? Other months lack a 31st day. In this situation, the task pretends there is a 31st day in every month. To avoid this, you can use the new RESCHEDULING FREQUENCY code 1M(L@time).

One-time option queue [XU OPTION QUEUE]

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                      [XUTM MGR]
   One-time Option Queue                                [XU OPTION QUEUE]
```

To run the option at a special time one day without affecting its established schedule, use the One-time Option Queue option. It queues a task to run once, without affecting the option's normal schedule in any way. This lets you handle the condition where you have an option queued to run periodically and you would like to queue it once to run at an irregular time without affecting its normal periodic schedule.

The Taskman Error Log

The Manager and Submanagers keep track of all errors caused by their own software or by the tasks they start. They log their own errors in two places: the ERROR LOG file (#3.075) and the Taskman Error Log. Those errors caused by tasks are also recorded in the entries of the tasks themselves and can be seen with any of the various task listing options (List Tasks, TaskMan User, etc.). Just as there are options to display and purge the ERROR LOG file, there are options to do the same for the Taskman Error Log.

When the XUTM QCLEAN option cleans tasks from the TASKS file, it also cleans any corresponding entries in the Taskman Error Log since it is hard to make sense of an error log entry without the task data.

Kernel strongly recommends that you report new errors to your ISCs and follow up to ensure expeditious patching. If you do this, over time the number of errors occurring on your system will diminish. This also improves the value of the various error logging systems as indicators of significant events deserving investigation.

Allocation and store errors are often not logged in the Kernel's ERROR LOG file because the process of logging errors is complicated and usually requires the use of local variables. Local variables take up space and there is no excess space when these errors occur. However, TaskMan makes its simple entries in the Taskman Error Log prior to calling the Kernel error logging utility. So, these errors are often recorded in the TaskMan log, but not the Kernel's. You are encouraged to carefully monitor both places.

Show Error Log

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                       [XUTM MGR]
   Taskman Management Utilities ...                         [XUTM UTIL]
      Taskman Error Log ...                                [XUTM ERROR]
         Show Error Log                                 [XUTM ERROR SHOW]
```

This option displays the errors currently stored in the Taskman Error Log, showing the date and time that the error occurred in a readable format and showing the error message. After the listing, the option gives the number of errors in the error log.

Errors stored in the Taskman Error Log historically are also cross-referenced to the TASKS file, linking tasks to the errors they cause.

Clean Error Log Over Range of Dates

```
SYSTEMS MANAGER MENU ...                                          [EVE]
Taskman Management ...                                         [XUTM MGR]
   Taskman Management Utilities ...                           [XUTM UTIL]
      Taskman Error Log ...                                  [XUTM ERROR]
         Clean Error Log Over Range Of Dates    [XUTM ERROR LOG CLEAN
                                                                 RANGE]
```

After prompting for a "First date to purge:" and a "Final date to purge:", this option removes the entries for all errors that occurred on and between the two dates. It prints the number of entries removed. If the first date is not earlier than the final date, no entries are removed.

Use this option to delete all but recent errors that deserve your attention. It is better to resolve specific kinds of errors as you encounter them. However, if there is a period during which you cannot resolve them fast enough to keep the log clean, this option will help you focus on the recent ones.

Purge Error Log Of Type of Error

```
SYSTEMS MANAGER MENU ...                                          [EVE]
Taskman Management ...                                         [XUTM MGR]
   Taskman Management Utilities ...                           [XUTM UTIL]
      Taskman Error Log ...                                  [XUTM ERROR]
         Purge Error Log Of Type Of Error     [XUTM ERROR PURGE TYPE]
```

With this option you can delete from the Taskman Error Log all entries for an error of a specific type. In fact, this option uses the M contains operator ([); so, it removes every error whose message contains your input as a substring. For example, you can remove every error that occurred in a certain routine or even every error whose message contains a "Q." After performing the purge, the option shows you how many entries were removed.

This option is the best way to keep the log clean. As you resolve certain kinds of errors and prevent them from happening again, you can remove all errors of that kind from the log. This leaves behind only those errors you have not resolved, helping you focus on the problems that remain.

Delete Error Log

```
SYSTEMS MANAGER MENU ...                                          [EVE]
Taskman Management ...                                         [XUTM MGR]
   Taskman Management Utilities ...                            [XUTM UTIL]
      Taskman Error Log ...                                    [XUTM ERROR]
         Delete Error Log                              [XUTM ERROR DELETE]
```

This option completely deletes all errors in the Taskman Error Log. If the error log is cleaned and purged as described above, you will rarely need to use this option.

Error Screens

At times you may wish not to trap a certain type of error, but merely to count them because you are already aware of the error and can do nothing to prevent it. At other times you may not even want to count the error because it is inevitable or harmless. An error screen is a string of characters that is compared with the error message of every error TaskMan traps. Any trapped error whose message contains the screen is screened out. You decide for each screen whether the error is counted or completely ignored. In either case the error is not recorded in either the Kernel ERROR LOG file or the Taskman Error Log. If a running task encounters a screened error, the Submanager still notes the error in the record for that task.

TaskMan gives you four options with which to manage your error screens. They are List, Add, Edit, and Remove Error Screens.

List Error Screens

```
SYSTEMS MANAGER MENU ...                                          [EVE]
Taskman Management ...                                         [XUTM MGR]
   Taskman Management Utilities ...                            [XUTM UTIL]
      Taskman Error Log ...                                    [XUTM ERROR]
         List Error Screens                       [XUTM ERROR SCREEN LIST]
```

This option lists in a simple table the screens you have established and the number of errors that have been screened out by each.

Add Error Screens

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                       [XUTM MGR]
   Taskman Management Utilities ...                         [XUTM UTIL]
      Taskman Error Log ...                                [XUTM ERROR]
         Add Error Screens                    [XUTM ERROR SCREEN ADD]
```

With this option you can enter a screen and specify whether the errors should be counted. If there are already similar screens in place (e.g., entering SYN when SYNTAX is already established) you will be so informed, shown the similar screens, and prompted for confirmation before being asked about the count. Entering two question marks at the "Enter Screen To Apply:" prompt displays the list of error screens.

Edit Error Screens

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                       [XUTM MGR]
   Taskman Management Utilities ...                         [XUTM UTIL]
      Taskman Error Log ...                                [XUTM ERROR]
         Edit Error Screens                   [XUTM ERROR SCREEN EDIT]
```

If you want to reset the counter on a screen or change your mind about whether or not the screen counts its errors, use this option. You must type in the exact screen you wish to edit. Again, entering two questions marks displays the list of error screens currently in place.

Remove Error Screens

```
SYSTEMS MANAGER MENU ...                                        [EVE]
Taskman Management ...                                       [XUTM MGR]
   Taskman Management Utilities ...                         [XUTM UTIL]
      Taskman Error Log ...                                [XUTM ERROR]
         Remove Error Screens               [XUTM ERROR SCREEN REMOVE]
```

When you type in a screen at the prompt for this option, the screen is removed for you. If there are any similar screens, the option asks whether you wish to remove them also. Again, entering two question marks displays the list of error screens.

Troubleshooting

The information given in this section may not be used by application developers in their code. It is provided to help site managers troubleshoot problems with tasks and TaskMan. Consider this section a reference to TaskMan's global structure and messages.

The Schedule File

^%ZTSCH holds the non-VA FileMan-compatible Schedule file, which consists of independent lists and nodes. This is where Task Manager processes tasks. This structure is not supported for use by application software. All task manipulation must be done through approved options and entry points. These structures must be free to change from version to version to easily adapt and meet the changing needs of DHCP. On the following pages is an example of a global that contains one of each type of node used by Task Manager:

The initial node was used to create ^%ZTSCH before Task Manager was active, so that the global type and protection could be assigned.

^%ZTSCH(next run time, task #) stores the Schedule List. The task # corresponds to an entry in the TASKS file, and the next run time is computed from the value in the sixth ^-piece of the entry's 0 node (and is the total number of seconds contained in the next run time's $H translation). If the Schedule List entry equals a device name, the entry was not created through the Programmer Interface.

^%ZTSCH("C") stores the Compute Server Job List (C list). This list holds tasks that are ready to be run by Submanagers on specific compute servers. A Submanager cross-volume set jobbed to a compute server only runs tasks under this list for the compute server on which it is running, and does not process the Device Waiting List or the Job List. The volume set, next run time, task #, and device $IO are stored here.

^%ZTSCH("DEV") stores the Device Allocation List. This list is used by TaskMan to coordinate its allocation of devices to tasks. The presence of a node indicates that TaskMan has already allocated this device to a specific task that has not yet gained ownership of it. It tells TaskMan not to give the device to another task. When the task for whom the allocation node was established gains ownership of the device or fails due to possession by some interactive job, the node is killed off. The $H value is used in case the task fails to remove its own node for some reason; after two minutes TaskMan kills the node on its next idle loop.

^%ZTSCH("ER") stores the Taskman Error Log.

^%ZTSCH("ES") stores the Error Screens.

^%ZTSCH("IDLE"), the idle node, is used to ensure that the Manager's idle loop activities are spaced out correctly in case multiple Managers are being run in the same environment.

^%ZTSCH("IO") stores the Device Waiting List. The device $IO value is the value for the task's device and should not be the $IO of a spool or host file device. The run time subscript (the total number of seconds contained in the run time's $H translation) prioritizes the tasks that should have started the longest time ago. The Submanagers use the top node to space out access to the list, and the last device so that only one Submanager at a time is checking the list, and so that checks that find all devices still busy are followed by a short waiting period before the list is checked again.

^%ZTSCH("JOB") stores the Job List. This list holds tasks that are ready to be run by Submanagers. The run time is the total number of seconds contained in the run time's $H translation, and task # and device $IO are what you would expect.

^%ZTSCH("LINK") stores the Link Lists. The LINK node itself is only present when a link is down. It is used to time the checks that occur every fifteen minutes. The second level nodes should always be present with the current information on each of the CPUs and volume sets.

^%ZTSCH("LOAD", load rating), the Load node, is used to balance the CPU load among the various Managers that work out of the current TASKS and Schedule files. It identifies the CPU that most recently checked its rating and decided to run. Managers more loaded (a lower rating) than this one wait to allow this Manager to pick up more of its share of the load.

^%ZTSCH("LOADA") stores the Load List. This list records the ratings for all the CPUs with Managers processing this TASKS file. The first ^-piece, which flags the Managers that decide to wait to balance the load, is used to tell the Submanagers on those CPUs that they, too, should wait.

^%ZTSCH("LOGRSRC") flags whether submanagers should log resources for the capacity management software. This node is set for every volume set whenever the LOG RESOURCE USAGE? field of the KERNEL SYSTEM PARAMETERS file is edited. A cross-reference keeps the ^%ZTSCH("LOGRSRC") node in synchronization with the Log Resource Usage? field.

^%ZTSCH("NO-OPTION"), if set, stops the submanagers from running any scheduled options. This is for the KIDS install process.

^%ZTSCH("RUN"), the Run node, is where the Manager periodically stamps the current time, leaving a way to determine whether it is currently active. Invoking the XUTM STOP option removes this node.

■ ^%ZTSCH Global Structure

```
^%ZTSCH= ""
^%ZTSCH(next run time, task #)= ""
^%ZTSCH(next run time, task #)= (D1) device IOP value
^%ZTSCH("C", volume set)= count
^%ZTSCH("C", volume set, next run time, task #) = device $IO
^%ZTSCH("DEV", device $IO)= $H when device was allocated for a specific
                            ==>task
^%ZTSCH("ER")= "A1" or ""
^%ZTSCH("ER", $H when error happened)= error message
^%ZTSCH("ER", $H when error happened, 0)= context of error
^%ZTSCH("ES", error screen, 0)= ""
^%ZTSCH("ES", error screen, 1)= screened errors count
^%ZTSCH("IDLE")= $H when the Manager's idle loop checks were last performed
^%ZTSCH("IO")= $H when device waiting list was last checked without finding
               ==> an available device^ $IO of last device tried
^%ZTSCH("IO", device $IO)=device type
^%ZTSCH("IO", device $IO, run time, task #)= ""
^%ZTSCH("JOB", run time, task #) = device $IO
^%ZTSCH("LINK")= "" or $H when dropped link was last checked
^%ZTSCH("LINK", volume set)= 1 if link has dropped
^%ZTSCH("LINK", volume set, next run time, task #)= ""
^%ZTSCH("LOAD", load rating) = cpu ^ $H when rating was checked
^%ZTSCH("LOADA", cpu) = whether TM should wait ^ load rating ^ $H
                     ==>when rating was checked ^ $J of Manager
^%ZTSCH("LOGRSRC") = ""

^%ZTSCH("NO-OPTION")=""
^%ZTSCH("RUN")= $H when Manager last checked in
^%ZTSCH("STARTUP", UCI, option #)= $H when option was first queued for
                                   ==>startup
^%ZTSCH("STATUS", $J of Manager)= $H when Manager last checked in [1] ^
                                  ==>status [2] ^ description of status [3]
^%ZTSCH("STOP")= ""
^%ZTSCH("SUB")= count of Submanagers waiting for tasks
^%ZTSCH("TASK", task #)= (A2) entry point [1] ^ (A3) routine [2] ^ (A4)
                      ==>option # [3] ^ (A5) option name [4] ^ (C6)
                      ==>description [5] ^ device name [6] ^ (E1) UCI [7] ^
                      ==>(C3) creation time [8] ^ (C1) creator DUZ or (C2)
                      ==>creator name [9] ^ $J of running task [10] ^ $H
                      ==>when task actually started running [11]
^%ZTSCH("UPDATE", $J of Manager)= $H when the Manager last updated its
                                  ==>parameters
^%ZTSCH("WAIT")= ""
```

^%ZTSCH("STARTUP", UCI, option #) holds the Startup List. This list holds the internal number of all options that are specially queued to run every time the Manager starts up. The $HOROLOG value reflects when the option was placed on this list.

^%ZTSCH("STATUS", $J of Manager) holds the Status List. This list holds the periodically updated entries for each Manager active on your machine and reflects each Manager's own perception of its current state.

^%ZTSCH("STOP"), the Stop Node, prevents Submanagers from running. While it is present, Managers won't start new Submanagers, Submanagers

waiting for tasks quit immediately, and those currently running tasks quit as soon as the tasks finish.

^%ZTSCH("SUB"), the Sub Node, counts the number of Submanagers waiting for new tasks. It is updated regularly by Submanagers as they run tasks. The Manager uses this value to decide whether to JOB out new Submanagers and adjusts its value during the idle loop whenever it believes it to be inaccurate.

^%ZTSCH("TASK", task #), the Task List, holds the tasks TaskMan believes are currently running. Since entries are cleaned up when tasks quit or encounter errors, those that are forcibly exited by the system manager are left on the list even though they are not running. The Manager clears the list whenever the system starts up, and the system manager may manually remove inaccurate entries by using the exit action of the Kill off a users' job option [XURESJOB]. The task data stored at each node allows TaskMan to list the tasks even when they clean out their TASKS file records when they start instead of when they quit.

^%ZTSCH("UPDATE", $J of Manager), the Update Node, records when the Manager last updated its local information about the site parameters. This node is killed whenever the Manager should update (for example, site parameters are changed).

^%ZTSCH("WAIT"), the Wait Node, puts the Manager into a WAIT state.

The TASKS File

^%ZTSK holds this partially-VA FileMan-compatible file of tasks. It is structured with a descriptor node followed by sequential entries. The data dictionary for this file is 14.4, TASKS. It is a read-only file. The TASKS file has no cross references, not even a top-level B cross reference, and its descriptor node is updated by the purge option (XUTM QCLEAN).

Each entry itself contains a zero node and several decimal nodes followed by a number of storage nodes. Like the Schedule file, the TASKS file is not available for direct manipulation or examination by application software. Site managers, however, can print out information on entries in the TASKS file using VA FileMan.

The diagram on the next page describes the nodes 0 through .26 for each entry in the TASKS file.

TASKS File Layout

```
^%ZTSK(task #, 0)= (A2) entry point [1] ^ (A3) routine [2] ^ (C1) creator
                ==>DUZ [3] ^ (E1) requested UCI [4] ^ (C3) creation time
                ==>[5] ^ (B4) next run time [6] ^ (A1) type of task [7] ^
                ==>(A4) option number [8] ^ (A5) option name [9] ^ (C2)
                ==>creator name [10] ^ (C4) creation UCI [11] ^ (C5)
                ==>creation volume set [12] ^ (C6) description [13] ^ (E2)
                ==>requested volume set [14] ^ (E3) priority [15] ^ (E4)
                ==>partition size (or 3 for non-DSM 11 systems) [16] ^ (E5)
                ==>Submanager retention time [17] ^
^%ZTSK(task #, .01)= (E8) original destination UCI ^ (E9) original
                ==>destination volume set
^%ZTSK(task #, .02)= (E6) current destination UCI ^ (E7) current destination
                ==>volume set
^%ZTSK(task #,.03)= Description [1]
^%ZTSK(task #, .1)= (B1) status [1] ^ (B2) last updated [2] ^ (B3) status
                ==>notes [3] ^ (B5) cycle type [4] ^ (B6) schedule [5] ^
                ==>(B7) holidays [6] ^ (B8) # runs remaining [7] ^ (B9)
                ==>remember finished task until [8] ^ (B10) linked task [9]
                ==>^ (B11) stop flag [10] ^ (B12) forget flag [11] ^
^%ZTSK(task #, .12, +$H when error happened, seconds past midnight)= (B3)
                ==>status notes (full error message for trapped errors)
^%ZTSK(task #, .2)= (D1) device IOP value [1] ^ (D2) $IO value [2] ^ (D3)
                ==>device type [3] ^ (D4) device subtype [4] ^ (D5) device
                ==>%IS modifiers [5] ^ (D6) host file address [6] ^
                ==>sync flag [7]
^%ZTSK(task #, .21)= (D8) device file entry # [1] ^
^%ZTSK(task #, .25)= (D7) device parameters [1] ^
^%ZTSK(task #, .26)= (D10) hunt group name [1] ^
^%ZTSK(task #, .26, $IO values of hunt group members)= ""
```

The remaining nodes of each entry are used to pass variables to the task. If the task has been manipulated only using Task Manager's Programmer Interface, then the entries look like this:

```
^%ZTSK(task #, .3, "name")= (F2) value of saved variable
^%ZTSK(task #, .3, "array(", node #)= (F2) value of saved variable
^%ZTSK(task #, .3, "array", node #)= (F2) value of saved variable
```

The distinguishing characteristic here is the fact that the variables to be passed are all subscripted under the .3-node.

Task Status Codes

This section lists the various codes that may be found in the first ^-piece of the .1 node, the text displayed for that code by the List Tasks option, and the meaning of that code. These codes are set into the tasks at every point in processing where the status changes, along with a time stamp and an explanation where necessary.

Several of the codes correspond to the status of the Schedule File entry for the task. If all applications used the Programmer Interface, the status code would always agree with the task's real status. In fact, many applications still directly manipulate ^%ZTSCH and ^%ZTSK, and they often neglect to update the status codes. Whenever the Schedule file disagrees with the status code, the Schedule file is correct. This is the reason many of the codes listed below have multiple meanings.

Status codes 1 through 6 represent one of two common paths a task takes through TaskMan. The other common path replaces code 3 with A, where the task's device is not immediately available.

Status Code	Description
0	**Incomplete or still being created.**
1	**Scheduled for <date and time>.** **TaskMan uses this status in every option and entry point that schedules a task.** **If the task fails or errors out and TaskMan cannot trap the error, this status has a different meaning: "Stopped irregularly while scheduled."**
2	**Being inspected by Task Manager.** **The Manager sets this status when the time comes for a task to run. As it removes the task from the Schedule file, it sets this code into the task.**
3	**Waiting for a partition.** **When the Manager places a task in the Job list of the Schedule file, it gives the task this code.** **If the task fails or errors out, and TaskMan cannot trap the error, this status has a different meaning: "Stopped irregularly while waiting for a partition."**

(continued)

Status Code	Description
4	**Being prepared.** **The Submanager gives a task this code when it removes the task from the Job list or Busy Device Waiting list in order to run it.**
5	**Currently running.** **The Submanager gives a task this status just before it starts the task at its entry point.** **If the task fails or errors out, and TaskMan cannot trap the error, this status has a different meaning: "Started running <date & time> and stopped irregularly."**
6	**Completed <date and time>.** **The Submanager gives a task this status after the task quits.**
A	**Waiting for device <device name or $I> or hunt group <hunt group name>.** **The Manager or the Submanager gives a task this status when it places the task in the Busy Device Waiting list.** **If the task fails or errors out and TaskMan cannot trap the error, this status has a different meaning: "Stopped irregularly while waiting for a device."**
B	**Rejected. <rejection message>.** **The Manager or the Submanager gives a task this status if it fails one of the basic validation tests. (The rejection messages are contained in the next section.)**
C	**Error <date and time>. <error message>.** **The Submanager gives a task this status if it traps an error after starting the task. The error message records the vendor-specific $ZE text.**

(continued)

Status Code	Description
D	**Stopped by user.** **The Manager or the Submanager gives a task this status if, when TaskMan removes the task from the Schedule file for processing, it finds that the user has asked the task to stop. The Submanager also assigns this status if, just before starting the task, it finds the stop request has been made. Finally, the Submanager gives a task this status if the task uses the ZTSTOP output variable to report that it stopped in response to a user's request. (For an explanation of ZTSTOP, see the description of $$S^%ZTLOAD in the Task Manager: Programmer Tools chapter.)**
E	**Interrupted while running.** **At startup, the Manager gives this status to any task listed in the Task list of the Schedule file as still running.**
F	**Unscheduled by <user name or "you">.** **The Dequeue Tasks [XUTM DQ] and TaskMan User [XUTM USER] options and the DQ^%ZTLOAD entry point use this status for tasks they unschedule.**
G	**Waiting for the link to <volume set name> to be restored.** **The Manager uses this status for tasks that would have been transferred to a different TaskMan environment and deleted from this one, if the local area network link to the remote environment were functioning properly.** **If the task fails or errors out, and TaskMan cannot trap the error, this status has a different meaning: "Stopped irregularly while waiting for a link."**
H	**Edited without being scheduled.** **The Requeue Tasks [XUTM REQ] and TaskMan User [XUTM USER] options and the REQ^%ZTLOAD entry point use this status when edited tasks are not subsequently rescheduled.**

(continued)

Status Code	Description
I	**Discarded by Task Manager because its record was incomplete.** **The Manager or the Submanager uses this status for tasks listed in the Schedule file that lack critical information in the corresponding TASKS file entries.**
J	**Currently being edited.** **This status has been set aside for possible use in future versions of TaskMan.**
K	**Created without being scheduled.** **The ^%ZTLOAD entry point uses this status for tasks when the application passes ZTDTH="@". The Kernel Toolkit utility ^%ZTMOVE uses this value for the tasks it creates to transfer routines between volume sets manually.**
L	**Preparing this task caused the Submanager an error <date and time>. <error msg>.** **The Submanager uses this status when it traps an error after claiming a task but before starting it.** **The Manager does not yet record a corresponding status for the analogous situation. Tasks that never start, that are left with a status of 2, have usually caused the Manager an error while it tried to examine them.**
M	**Waiting for a partition on a compute server.** **The Manager gives a task this code when it places the task in the Compute Server Job List.** **If the task fails or errors out, and TaskMan cannot trap the error, this status has a different meaning: "Stopped irregularly while waiting for a partition on a compute server."**

Task Rejection Messages

Under certain conditions TaskMan can avoid trapping obvious errors by checking the tasks themselves for internal consistency. Whenever it finds tasks with bad data, it rejects them. This involves unscheduling them, setting their status codes to "B", and adding a brief explanatory message. These messages can help identify bugs in application queuing software, in the local system configuration, or in TaskMan itself.

BAD DESTINATION UCI

The Manager rejects a task for this reason under three different conditions:

- **If the task is bound for the Manager's own volume set, whatever value has been passed for the destination UCI must be a valid UCI on the current volume set. If ^%ZOSF("UCICHECK") rejects the UCI, TaskMan rejects the task.**
- **If the task is bound for a different volume set and the destination UCI is not listed in the UCI ASSOCIATION file (#14.6) under that volume set, the UCI must be accepted as a valid UCI on the current volume set so TaskMan can use File 14.6 to determine where the task should run. If ^%ZOSF("UCICHECK") rejects the UCI, TaskMan rejects the task.**
- **If the task is bound for a different volume set and that volume set's link is down and its replacement volume set is the current volume set, TaskMan rejects the task.**

BAD DESTINATION VOLUME SET

Every task's destination volume set must be listed in the VOLUME SET file.

BAD IO DEVICE <$I>

If a port goes bad while many tasks wait for it in the Busy Device Waiting list, TaskMan traps an error whenever the port is tested for availability. When the Submanager traps such an error, it rejects every task waiting for that device.

INVALID OUTPUT DEVICE

The Manager performs a look-up on the devices that tasks request. If the ^%ZIS call indicates that the device does not exist, then TaskMan rejects the task.

INVALID ROUTINE NAME

If a task's entry point is in a %-routine, the Manager tests for that routine's existence in the library UCI. If the routine does not exist there, TaskMan rejects the task.

NO DESTINATION UCI

When older applications bypassed the Programmer Interface, they sometimes scheduled tasks without specifying the destination UCI. The Manager rejects all such tasks.

NO LINK ACCESS TO VOLUME SET

If the VOLUME SET file entry for a task's destination volume set indicates there is no link access to that volume set, the task is rejected.

NO ROUTINE AT DESTINATION

If a task's entry point is in a non-%-routine, then the check for the routine's existence is done by the Submanager prior to starting the task.

TaskMan State Messages

When the Manager does not run, all background processing grinds to a halt. For this reason, the Manager's condition is of vital importance to system managers. When problems are detected with background processing at a site, checking the Manager's condition should be the first step. The Manager periodically records its state in the Status List. The Monitor TaskMan [XUTM ZTMON] option displays this list near the top of the screen. The various states and their meanings are described below.

BALANCE State

The Manager lists itself in this state if other Managers (that are processing the same files) appear to have more CPU capacity available than the current Manager. While in the BALANCE state, the Manager does not process any tasks or start any new Submanagers. The Manager removes itself from the BALANCE state when it appears to have at least as much CPU capacity as the active Manager. In general, when many Managers are working out of the same TASKS and Schedule files, most of them will be in the BALANCE state at any given time, with only the one or two least loaded Managers actually processing tasks.

For more information about TaskMan load balancing, please see the Task Manager Load Balancing section of the Task Manager System Management: Configuration chapter.

ERROR State

The Manager lists itself in this state after trapping errors. On some systems the process of recording an error is slow, so the presence of a distinct state helps identify the source of delay to the system manager. A troubleshooter who sees this state for TaskMan should immediately check the TaskMan Error list to see what kind of error is being recorded. Because TaskMan's code is structured as a series of nested loops, it can very easily generate thousands of errors a day under certain conditions.

PAUSE States

The PAUSE state means that some external condition is preventing the Manager from processing tasks. The description always indicates the cause. While in the PAUSE state, the Manager waits until the problem is resolved, checking once every 60 seconds. The pause states are as follows:

The following required ^%ZOSF nodes are undefined: <list of nodes>:
When the Manager starts, restarts, or recovers from a trapped error, its first order of business is to drop through some setup code that checks TaskMan's environment. If any critical ^%ZOSF nodes are missing, it enters a PAUSE state and waits until the system manager restores the nodes.

Required link to <volume set name> is down:
The other key check in the setup code is to ensure that all volume sets listed in the VOLUME SET file as required can actually be reached. The Manager tests each required link and enters the PAUSE state if any tests cause an error. The Manager remains in the PAUSE state, periodically testing the links, until they are restored.

Logons Inhibited:
When the system manager sets the INHIBIT LOGONS? field of the VOLUME SET file, TaskMan enters a PAUSE state and waits until the flag is cleared.

No Signons Allowed:
The system manager may use the software switch to stop logons, which places TaskMan in the PAUSE state.

RUN States

The RUN state indicates that the Manager is going about its business in a relatively normal manner, managing background tasks on your system.

- **Start: the Manager sets this value before and after executing the setup code at system startup.**
- **Setup: the Manager identifies when it executes the setup code to test its environment.**
- **Restart: the Manager sets this value after executing the setup code during a restart.**
- **Main Loop: this should be the Manager's usual state. This indicates the Manager is executing the main loop that checks the environment, processes the Schedule list, and performs idle loop activities when appropriate.**
- **TaskMan Job Limit Reached: when the total number of processes on the Manager's CPU exceeds the TaskMan Job Limit given in the VOLUME SET file, the Manager may continue to process the Schedule list but may not start any new Submanagers.**

WAIT State

While in the WAIT state, the Manager does not react to changes in its environment. It does not process tasks, enter PAUSE states, or even stop after the option Stop TaskMan has been used.

You have two options (described above) that let you create or undo the WAIT state. TaskMan cannot enter this state on its own; it can only be initiated manually. This is essentially a tool for you to tightly control the processing of tasks on your machines. The description for this state always reads "TaskMan Waiting".

Chapter 25 Task Manager: Programmer Tools

The TaskMan API consists of several callable entry points and an extrinsic variable. Use of these calls makes the creation, scheduling, and monitoring of background processing from within applications straightforward.

Programmers must avoid directly setting information into TaskMan's globals to queue tasks. In fact, the SAC specifies that TaskMan's calls be used. The structure of the globals is not static; there is no commitment to support their current structure in the future.

For more information on why and when to use Task Manager to perform queuing, please see the Task Manager: Overview chapter earlier in this section.

How to Write Code to Queue Tasks

Writing code to queue a task is not difficult; however, the coding must be done carefully and systematically. If you think of it in two parts, it will be easier to write. These two parts are the queuer and the task. Usually, both pieces of code should be planned together since they interact heavily.

- **The Queuer: Some code must invoke ^%ZTLOAD to create and schedule the task. This code is the queuer. The most complex part of a queuer is determining which variables must be passed on to the task.**

  In one type of queuer, the program application makes its own calls to ^%ZTLOAD to queue tasks. In the other common type of queuer, scheduled options, an option is scheduled to run as a task through the OPTION SCHEDULING file; Task Manager itself takes care of the queuing.

- **The Task: Some code must perform the actual work in the background. Sometimes the task shares code with an equivalent foreground activity. However, remember that a queued task runs under special conditions that must be considered. For example, no interactive dialogue with the user is possible.**

Queuers

As mentioned above, there are two common types of queuers: application code that itself acts as the queuer by calling ^%ZTLOAD; and options that are scheduled (in which case, Task Manager itself acts as the queuer).

Calling ^%ZTLOAD to Create Tasks

One common way to create tasks is to call Task Manager's main entry point, ^%ZTLOAD. You can use ^%ZTLOAD interactively, or non-interactively. For more information on queuing tasks with ^%ZTLOAD, see the description of ^%ZTLOAD later in this chapter.

Calling EN^XUTMDEVQ to Create Tasks

Beginning with Kernel V. 8.0, the new EN^XUTMDEVQ entry point encapsulates the logic to handle both direct printing and queuing in a single call.

Creating Tasks Using Scheduled Options

You can also create options that you ask the sites to schedule on a regular basis. In this case, Task Manager itself (rather than application code) acts as the queuer. Site managers use Task Manager to queue options and can schedule these options to run again and again on some specified schedule.

You should be careful because this creates a great possibility for confusion. Obviously some options cannot be scheduled, in the same way that some routines cannot be queued. When you create options that should be scheduled, you should:

- **Indicate whether an option can be scheduled through Task Manager and, if so, the recommended frequency of scheduling. Do this using the DESCRIPTION field of the option.**
- **Indicate the format of data to pass to the scheduled option via the TASK PARAMETERS field, if the option uses such data. Do this using the DESCRIPTION field of the option.**
- **Set the SCHEDULING RECOMMENDED field of the option to YES. This will make the option show up in a Kernel report that lists all options on the system that should be scheduled.**
- **Consider using a name for the option that reflects the fact that it is intended to be run only by Task Manager, if you create such an option.**
- **Give the option a parent (that is, attach it to a menu). This prevents the option from being deleted by Kernel's Delete Unreferenced Options (XQ UNREF'D OPTIONS) purge option. If the option cannot be used**

interactively, make sure that it is not attached to a menu that will be part of a user's menu tree. Instead, attach it to a menu that is not on any user's menu tree. An example is Kernel's ZTMQUEUABLE OPTIONS. It is not in any user menu tree. If you don't want to create your own menu to be a parent of queuable options, you are allowed to attach your option to Kernel's ZTMQUEUABLE OPTIONS option and export ZTMQUEUABLE OPTIONS through KIDS' USE AS LINK FOR MENU ITEMS action.

When you create options that queue tasks but that cannot be scheduled themselves, you should be especially clear in documenting this so that site managers will not try to schedule them.

Queued options differ from other tasks in only a few ways:

- **They may have an entry and exit action and may set XQUIT in the entry action to avoid running.**
- **They may run on a scheduling cycle as defined by the system manager.**
- **They are designed explicitly for the system manager to use, since the option used to schedule options is available only to system managers.**
- **They can be better documented than normal tasks because the OPTION file entry provides a place for a permanent description of the task's purpose and behavior (the DESCRIPTION field).**
- **If the option is scheduled regularly, data can be passed to your task from the OPTION SCHEDULING file's TASK PARAMETERS field; the data is made available to the task at run time in the ZTQPARAM variable. The variable is only defined if an entry is made in the TASK PARAMETERS field when the task is scheduled. The format that is expected of information entered in the TASK PARAMETERS field should be described in the option's DESCRIPTION field.**

You should describe scheduling recommendations and the format, if any, for the TASK PARAMETERS field (as well as in the option's DESCRIPTION field) in your package installation guide for all the queuable options, since options are usually set on their schedules shortly after installation.

Tasks

This section describes information about Tasks. It applies whether the queuer that queued the task was a call to ^%ZTLOAD, or Task Manager itself was running the task because it was scheduled in the OPTION SCHEDULING file.

When you write a task, you create an entry point that TaskMan can call to perform the work. The Submanager calls the entry point you specify to run the task. The Submanager does more than pass your task a few parameters, however; it creates an entire specialized environment for the task, according to your specifications. Then the Submanager calls your entry point, at which point your task begins running. When your task quits, control passes back to the Submanager.

The interface between tasks and Submanagers determines the special problems you must solve and the features you have available to do so. This interface consists of two parts: the environment and tools that the Submanagers guarantee to the tasks, and the responsibilities of the tasks themselves.

Key Variables and Environment When Task is Running

All DHCP processes run in a guaranteed environment, with standard variables and devices available to the software. The guaranteed environment for tasks differs from that of foreground processes in some ways, however. This reflects the differences between the foreground and background, and the special services provided by TaskMan. The Submanagers guarantee tasks the following variables and other features:

- **DT: While this usually designates the date when a user signs on, here it contains the date when the task first began running (in FileMan format, of course).**
- **DUZ(: The entire DUZ array (except DUZ("NEWCODE")), as defined at the time of your call to the Programmer Interface, is always passed to your task. If DUZ was not properly set up at that time, then it is set to 0. If DUZ(0) was not properly set up, then the Submanager attempts to look it up using your DUZ variable; if the lookup fails, it sets DUZ(0)="". The Submanager does the same thing with DUZ(2).**
- **IO*: All of the IO variables describing the output device that you receive are passed to you. If you request no output device, then IO, IO(0), and ZTIO will all equal "".**
- **ZTDESC: This contains the free-text description of your task that you passed to the Programmer Interface.**
- **ZTDTH: This contains the date and time (in $HOROLOG format) that you wanted your task to begin running. Because delays from a number of sources can make your task begin late, this variable may be useful.**

- **ZTIO: This contains your original output device specifications. Because the Device Handler allows the use of Hunt Groups, it is possible that your task may get an output device other than the one you requested. In such cases, however, the device will be one that the site manager considers equivalent.**
- **ZTQUEUED: This variable is always defined when your task begins, and is only defined for background tasks. Many queued routines can run either in the foreground or in the background. The only reliable way to determine which situation is currently the case is using the M code:**

```
IF $D(ZTQUEUED)
```

- **ZTSK: Every task is passed its internal number so that it can make use of the Programmer Interface.**
- **Destination: Using ZTUCI, ZTIO, and ZTCPU, you can request a specific UCI on a specific volume set and CPU node where your task should run. The location you request is where the Submanager will call your entry point. Remember that the SAC does not protect the TaskMan namespaced input variables to your task (e.g., ZTIO, ZTSK, etc.), however. The Submanagers guarantee their values to the tasks, but once you begin running their values may change. For example, the utilities you call may alter these variables, or your own code may. If your task needs to know these values throughout its execution, you should load them into your own namespaced variables, which you can then protect.**
- **Device: If you request an IO device for your task then, when the task starts, the device will be open. The Submanager will even issue the USE command for you and after your task completes, it will properly close the device for you. If you leave it open when you are finished with it, the Submanager will be able to recycle the device more efficiently for use with other tasks.**
- **Error Trap: The Submanager always sets an error trap before calling your task. This way, if your task errors out, the Submanager can record that fact in the system error log, in Task Manager's error log, and in the entry for your task in the TASKS file.**
- **Priority: Your task will begin running with the priority specified if you request one.**
- **Saved Variables: The Submanager passes any variables that the queuer saved using ZTSAVE. These act as input variables.**
- **Tools: The task can rely upon the following tools to assist it in meeting its responsibilities (as described below): $$S^%ZTLOAD, ZTSTOP, ZTQUEUED, ZTREQ, KILL^%ZTLOAD, ^%ZTLOAD, the Device Handler, resource devices, and SYNC FLAGs.**

Checking for Stop Requests

You should write tasks in such a way that your tasks honor stop requests. Since Kernel V. 7.0, users have been able to call the TaskMan User option to stop tasks that they started. A task should periodically check whether it has been asked to stop and should gracefully shut down when asked. This involves four steps:

1. **To check for a stop request, the task may execute the following code:**

   ```
   IF $$S^%ZTLOAD
   ```

   If this evaluates to TRUE, the user has asked the task to stop. This check should occur periodically throughout the task; not so often as to increase significantly the task's CPU usage, but often enough that the response time satisfies the users. For example, a report printout might check once per page, while a massive data compilation might check once every hundred or even thousand records. Very short tasks may choose not to check at all.

2. **The task may need to perform some internal flagging or cleanup. Stop requests from a user rarely come at ideal moments in the overall algorithm of the task, and the task may need to perform some work to prepare to quit.**

3. **The task needs to notify the Submanager that it responded to the user's request to stop, so that the Submanager can notify the user. The task should use the following code to do so:**

   ```
   SET ZTSTOP=1
   ```

   The ZTSTOP flag is processed by the Submanager when the task quits. Don't kill this variable if you wish to pass it back to the Submanager.

4. **The task should then quit. Depending on how deeply within loops these stop request checks are made, it may take some processing to work out of all loops and quit on short notice. The code may need to be adjusted to allow for this kind of exit.**

In the end, checking for stop requests benefits not only the developer, by satisfying your users, but also the users themselves by making them feel more in control, and the system managers by freeing them up from stopping tasks for users.

Purging the Task Record

According to the SAC, tasks have a responsibility to remove their own records from the TASKS file when they complete. This serves two purposes. First, it helps keep the TASKS file small which makes TaskMan more efficient. Second, because any tasks that cause errors will never reach the final commands to delete the task's record, such tasks will remain in the TASKS file after they complete. This greatly assists system management staff in identifying and troubleshooting problem tasks.

You have two methods to delete TASKS file entries:

- **ZTREQ output variable**
- **KILL^%ZTLOAD entry point**

The recommended method, simpler than the other, is to use the ZTREQ output variable to instruct the Submanager to delete your task's record after it finishes running. Do this with the following line of M code:

```
S ZTREQ="@"
```

Because the Submanager does not get this variable back until after your task quits, you can set ZTREQ anywhere within the task and still ensure your task does not delete its record if it errors out. Note that if you kill off the variable before the task quits, the Submanager does not delete your task.

The other method is to call KILL^%ZTLOAD to delete the task's record. This solution has two disadvantages. First, the ZTSK input variable to KILL^%ZTLOAD needs to equal the task number of the task to delete, which may not be the case if the task has called other utilities. The task can solve this problem by saving off ZTSK at the beginning and restoring it prior to calling KILL^%ZTLOAD. Second, you must place the call at the end of the task, just prior to quitting, to ensure the record remains if the task encounters an error. This causes problems for tasks that lack a single exit point, but you can solve this by writing a new entry point for the task that does the main body of the task, performs the deletion, and then quits.

Checking For Background Execution: ZTQUEUED

When you share code for both foreground and background processing, you often need the code to behave differently under the two situations. The only reliable way to test whether the code is running in the background is to check if the ZTQUEUED variable is defined. It will only be defined if the current running job is a task. You can check for its existence, and therefore whether the code is truly running in the background, with the following M statement:

```
IF $D(ZTQUEUED)
```

Post-Execution Commands: ZTREQ

Tasks can make the Submanager execute a certain limited set of commands after the tasks complete. Use the ZTREQ output variable to describe these post-execution commands.

The use of ZTREQ to delete a task's record has already been discussed above. ZTREQ can also be used to edit and/or reschedule the task.

- **To reschedule the task to run again immediately:**

  ```
  S ZTREQ=""
  ```

- **To requeue a modified version of your task:**

  Use ZTREQ to specify how to modify the existing task to run again. By optionally setting any of the various ^-pieces of ZTREQ, you can modify that aspect of how the rescheduled task will run. The purpose and format of each ^-piece roughly corresponds to the input variables of REQ^%ZTLOAD listed below:

ZTREQ piece	Equivalent REQ^%ZTLOAD variable
1	ZTDTH
2	ZTIO
3	ZTDESC
4^5	ZTRTN

  All of these ^-pieces in ZTREQ are optional; only set the pieces that affect parameters you want to change. Note, however, that in the case of leaving piece 2 null, the task uses the same device that your task initially requested, which is not necessarily the device that it actually got. If the system manager uses hunt groups, your task may use a task other than the one it requested. To reschedule the task to run on the device your task currently has, you must build up the ZTIO value using your IO variables.

- **To edit the task without actually rescheduling it:**

  Set ^-piece 1 to "@", and set the other pieces to the values you want. This is equivalent to setting ZTDTH="@", as described in the REQ^%ZTLOAD entry point below. Remember, however, to include at least one ^ in ZTREQ to do this, since if ZTREQ="@" the task will be deleted.

Remember that ZTREQ is not an input parameter that you pass to the Submanager; it is an output parameter from your task. The Submanager does its best to honor your request, but if the request is impossible, then there is no way for you to find out. For example, if you specify that the Submanager should requeue your task, then it attempts to do so; if it finds

that your task has been deleted, there is no way for the Submanager to let you know. When the Submanager cannot honor your request, it ignores it.

Calling ^%ZTLOAD Within a Task

Tasks can use all of the standard TaskMan API calls. There is no reason a task shouldn't itself call the TaskMan API to do requeuing, deletion, or any of the other standard calls. The only way such calls are special is that they have many of the variables they need to pass already defined for them by the Submanager.

You should be careful to avoid interference from these pre-defined variables: sometimes the Submanager passes you the value you will need for the API call, but sometimes you will need a different one. For example, from within a task that has an IO device, to call ^%ZTLOAD to queue a task without an IO device, you should set ZTIO (to ""), because the input variable passed in by the Submanager may still be defined. With a little care, these kinds of problems can easily be anticipated and prevented.

Calling the Device Handler (^%ZIS) Within a Task

The main Device Handler entry point (^%ZIS) by itself is not designed to open more than one I/O device beyond the already-open home device. Within a task, you are free to open one additional device (beyond the home device) using ^%ZIS. If you need to open more than one device concurrently within a task, however, you should use Kernel's multiple device entry points (OPEN^%ZISUTL, USE^%ZISUTL, and CLOSE^%ZISUTL).

Writing Two-step Tasks

A situation you often need to consider is how to deal with tasks that take a long time to generate and then print a report. If you write this as a single task that requests the IO device it will eventually use, then during the entire generation time the device sits idle, unused so far by the task but unavailable to any other tasks.

If you write the task to start without a device, and to call ^%ZIS to open the device when the report is ready, two different problems occur. First, if the device is heavily used by tasks, then this task may never get a chance to open the device: TaskMan will keep it busy with other tasks. Second, if the task does manage somehow to grab the device away from TaskMan, it interferes with the fair distribution of resources, potentially running ahead of other tasks that have been waiting longer.

One way around this problem is to queue the task to a spool device. Spool devices are always available, which solves the problem of tying up a device.

Some system managers discourage use of spoolers, however, because of the possibility for disk crashes resulting from users who send excessively large reports to the spooler.

Another solution involves splitting the task into two tasks, the first of which runs without a device, generates the report data in ^XTMP, and queues the second task; and the second of which runs with the IO device and prints the report.

Using SYNC FLAGs to Control Sequences of Tasks

You can use SYNC FLAGs together with resource type devices when queuing through ^%ZTLOAD, as a mechanism to ensure sequential processing of a series of tasks. The mechanism also ensures that subsequent tasks in the series will not run if a previous task errors out or completes unsuccessfully.

A SYNC FLAG is a unique, arbitrary free text name you use as an identifying flag. You use SYNC FLAGs in conjunction with resource devices; when paired with a particular resource device, the pairing is called a SYNC FLAG pair.

The SYNC FLAG pair ties all tasks that have requested the same SYNC FLAG and the same resource together. If a task in a group of tasks is running, all other tasks queued with the same SYNC FLAG pair have to wait until the running task has completed. If one task in the series doesn't finish successfully, then all other tasks using the same SYNC FLAG pair will wait.

To build a series of tasks, you need to choose a resource device and queue the entire series of tasks in the same order that they should run, through ^%ZTLOAD. Use the ZTIO parameter to queue all tasks in the series to the same resource device. Use the ZTSYNC parameter to use the same SYNC FLAG for each task in the series. TaskMan then runs the series of tasks in the same order that they were queued.

The SYNC FLAG pair uniquely identifies one group of tasks using one resource device. Task Manager builds a SYNC FLAG pair by concatenating the requested resource (from the ^%ZTLOAD ZTIO input variable) with the name of the SYNC FLAG (from the ^%ZTLOAD ZTSYNC input variable).

In any given task in the series of tasks, you indicate that the task completed successfully by killing the ZTSTAT variable or setting it to 0. Otherwise, no subsequent tasks will be able to run.

The following describes how using SYNC FLAG pairs ensures sequential processing of a series of tasks:

1. **When a task is queued through ^%ZTLOAD, if the ZTSYNC is defined, then the SYNC FLAG defined by ZTSYNC is saved with that task.**

2. **When TaskMan is ready to start the task, after it is able to allocate the resource device to which it was queued, it checks whether the SYNC FLAG pair (Resource_SYNC FLAG) exists in the TASK SYNC FLAG file.**

3. **If the SYNC FLAG pair does not exist in the TASK SYNC FLAG file, TaskMan creates an entry for the SYNC FLAG pair in the TASK SYNC FLAG file and starts the task.**

   If, on the other hand, the SYNC FLAG pair already exists in the TASK SYNC FLAG file, then any task requiring the same SYNC FLAG has to wait until the corresponding entry in the TASK SYNC FLAG file is deleted.

4. **If the task was able to start, the variable ZTSTAT is set to "1" in the running task.**

   To indicate success (e.g., that the series of tasks should continue), you must kill ZTSTAT or set it to zero. In this case, when your task completes, the SYNC FLAG pair for that task will be cleared.

   To indicate failure (e.g., that the series of tasks should not continue) leave ZTSTAT set to 1.

5. **When the task completes, TaskMan checks to see the value of ZTSTAT. If ZTSTAT is set to 0 or not defined, TaskMan deletes the SYNC FLAG pair entry in the TASK SYNC FLAG file. This allows any future tasks in the series to run.**

   If, on the other hand, ZTSTAT is left with a positive value, the task is assumed to have had some kind of error. In this case the value of ZTSTAT is saved in the STATUS field of the SYNC FLAG pair entry, and the entry in the TASK SYNC FLAG file is not deleted. Subsequent jobs in the series are prevented from running.

If the task errors out, the SYNC FLAG pair entry is also left in the TASK SYNC FLAG file, preventing subsequent jobs in the series from running. TaskMan puts a message in the STATUS field, saying that the task stopped due to an error.

Callable Entry Points

• EN^XUTMDEVQ: Run a Task (Directly or Queued)

Usage `D EN^XUTMDEVQ(ztrtn,ztdesc,.ztsave[,.%zis])`

Input

ztrtn: **The entry point EN^XUTMDEVQ will DO to start the job. Specify it as "LABEL^ROUTINE" or "^ROUTINE" or "ROUTINE".**

ztdesc: **Task description, up to two hundred characters describing the task, with the package name at the front.**

.ztsave: **Pass by reference. Set up this array in the same format as the ZTSAVE input array is set up for the ^%ZTLOAD Task Manager entry point. The array you set up in ztsave is passed directly as ZTSAVE to Task Manager if the user chooses to queue the job.**

.%zis: **(optional) Pass by reference. String containing input specifications for the Device Handler. Set up the array in the same way as the %ZIS array is set up for the ^%ZIS Device Handler entry point. The array you set up in the %zis parameter is passed directly as %ZIS to the Device Handler.**

All %ZIS subscripts from the regular ^%ZIS call ("A", "B", "HFSMODE", etc.) can be passed in the %zis input array.

Output **none**

Description

The new EN^XUTMDEVQ entry point encapsulates the logic to handle both direct printing and queuing in a single call.

EN^XUTMDEVQ calls ^%ZIS to query the user for device selection. The user can choose a device on which to run the job directly or choose to queue the job.

After calling ^%ZIS, EN^XUTMDEVQ looks to see if the queuing was chosen. If so, EN^XUTMDEVQ uses the values from the ztrtn, ztdesc, and ztsave input parameters to queue the job to the chosen device. If the user did not choose to queue, EN^XUTMDEVQ runs the job directly using the ztrtn input

parameter. EN^XUTMDEVQ thus provides a simple way to facilitate both queuing and running a job directly.

If the variable IOP is defined before calling EN^XUTMDEVQ, it will have the same effect as it does if defined before a ^%ZIS call.

If the variables ZTPRI or ZTKIL are defined before calling EN^XUTMDEVQ, they will have the same effect as they do if defined before an ^%ZTLOAD call. Other ^%ZTLOAD input variables have no effect, however.

You do not need to "USE IO" in the routine specified in the ztrtn input parameter; IO will be the current device, whether the job is queued or run directly. Also, you do not need to pass "Q" in the top level of the %zis input array; if the top level of the array does not contain "Q", "Q" will be appended to it (to allow queuing).

■ Sample Report Using EN^XUTMDEVQ

```
ZZYZOPT  ;ISC-SF/doc
   ;;1.0;;
EN ;
   N ZZEN K X,DIC S DIC=9.6,DIC(0)="AEMO" D ^DIC
   Q:+Y'>0  S ZZEN=+Y
   ;
   K ZTSAVE S ZTSAVE("ZZEN")=""
   D EN^XUTMDEVQ("P^ZZYZOPT","Print from BUILD File",.ZTSAVE)
   Q
P  ;
   ; code for printout
   ;
   W !,"Here goes the body of the report!"
   W !,"ZZEN = ",ZZEN
   Q
```

• ^%ZTLOAD: Queue a Task

^%ZTLOAD is the main entry point used to create and schedule tasks (commonly referred to as "queuing"). Queuing tells TaskMan to use a background partition to DO a certain entry point at a certain time, with certain other conditions established as described by the input parameters.

Usage D ^%ZTLOAD

Input

ZTRTN: **The entry point TaskMan will DO to start the task. You can specify it as "LABEL^ROUTINE" or "^ROUTINE" or "ROUTINE".**

ZTDESC: **Task description, up to two hundred characters describing the task, with the package name at the front. While not required, use of this variable is recommended.**

ZTDTH: **(optional) Start Time when TaskMan should start the task. It must be a date and time in VA FileMan or $HOROLOG format. Setting it to "@" will cause the task to be created but not scheduled. If ZTDTH is not set, ^%ZTLOAD asks the user for the start time.**

ZTIO: **(optional) The I/O device the task should use. If ZTIO is null, no device is used. If undefined, the current I/O variables will be used to select a device. ZTIO should only be used when the current I/O variables do not describe the needed device. If you do not need a device for a job, set ZTIO="". ZTIO accepts the same I/O formatting string as IOP. (See the Device Handler section.)**

ZTUCI: **(optional) UCI the task should use. The current UCI is used if ZTUCI is undefined.**

ZTCPU: **(optional) Volume Set:CPU. Specifies the name of the volume set and CPU on which the task should run. The volume set can be passed in the first :-piece, and the CPU in the second. Neither piece of information is required, and either can be passed without the other (if the CPU alone is passed, it must still be preceded by a ":", e. g., :ISC6A1). If the volume set is not passed,**

TaskMan will run the task on the volume set it came from or on a print server. If the CPU is not passed, TaskMan will run the task on the CPU where TaskMan resides. Any volume set and/or CPU specified by the task's I/O device takes precedence over the same information passed here.

Note: On DSM for OpenVMS systems, specifying which CPU a job should run on only works if you are running TaskMan from a DCL context. If you specify the CPU, but are not running TaskMan from a DCL context, the job may not run correctly. At MSM-DOS sites, there is no special setup needed.

ZTPRI: **(optional) The CPU priority the task should receive. It should be an integer between 1 (low) and 10 (high). The site's default for tasks is used if this is undefined.**

ZTSAVE(): **(optional) Input parameters array: an array whose nodes specify input parameters to the task beyond the usual set all tasks receive. There are four kinds of nodes this array can have:**

- **ZTSAVE("variable") can be set equal to null or to a value; if null, the current value of that variable is copied for the task, otherwise the variable is created with the value assigned [for example, ZTSAVE("PSIN")=42]. The variable can be local or global, and it can be a variable or an individual array node.**
- **ZTSAVE("open array reference") can be set to null to declare a set of nodes within an array to be input parameters to the task [for example, ZTSAVE("^UTILITY($J,")].**
- **ZTSAVE("namespace*") can be set to null to save all local variables in a certain namespace [for example, ZTSAVE("LR*")].**
- **ZTSAVE("*") can be used to save all local variables. Non-namespaced variables (esp. %, X, Y, etc.) may or may not be saved. Saving individual variables is more efficient. ZTSAVE nodes are saved just as**

they are typed, so special variables like $J have one value when used to save the variables, and a different value when used to restore them for the task.

ZTKIL: (optional) KEEP UNTIL: Set this to the first day the Task File Cleanup can delete this task. It should be a date and time in FileMan or $HOROLOG format. Use of this variable is recommended when ZTDTH equals "@".

ZTSYNC: (optional) Name of a SYNC FLAG. Using SYNC FLAGS allows Task Manager to run the next task in a series of tasks only if the preceding task in the series completed successfully.

You can choose any name for a SYNC FLAG. You should namespace the name, however, and make it no longer than 30 characters in length.

To use SYNC FLAGs, the task must be queued to a device of type resource (through the ZTIO parameter). For complete information on how to use SYNC FLAGs, see the section in this chapter on TaskMan SYNC FLAGs.

Output ZTSK: (Usually returned) The task number assigned to a task, returned whenever a task is successfully created. It can be used as an input parameter to the other TaskMan application mode entry points. Note that if a task is queued to a volume set other than the one where it was created, it is usually assigned a new task number when it is moved.

If ZTSK is not defined after calling ^%ZTLOAD, either ZTRTN was not set up or the user canceled the creation when prompted for a start time. If a task is not created and if ^%ZTLOAD is being called by a foreground job, then ^%ZTLOAD will display a message to the user indicating that the task has been canceled.

Note: ZTSK is not a system variable. It is killed and manipulated in many places. If the software needs to remember a task number, ZTSK should be set into some properly namespaced variable the application can protect.

ZTSK("D"): START TIME (usually returned) contains the task's requested start time in $HOROLOG format. It is returned whenever ZTSK is returned, and gives you a way to know the start time a user requests.

Description

The ^%ZTLOAD entry point as used in code behaves consistently, so most queuers strongly resemble one another. The queuer can be written so that it is either interactive with the user or so that it is not interactive. The standard variations on this structure deserve attention.

Interactive Use of ^%ZTLOAD

The DHCP Standards and Conventions require that anywhere you let a user pick the output device you also let the user choose to queue the output.

Often one part of the queuer is a call to ^%ZIS (the Device Handler). When you set up the parameters for your call, include a "Q" in the variable ^%ZIS so the Device Handler will let the user pick queuing. After the Device Handler call (and after you check POP to ensure that a valid device was selected), you can check $DATA(IO("Q")) to see whether the user chose to queue to that device. If so, then you must queue the printout you were about to do directly, and your software should branch to the code to set up the task. A sample of the code for this kind of print queuer looks something like this:

```
SELECT   ;select IO device for report
         S %ZIS="Q" D ^%ZIS
         I POP D CANCEL Q
         I $D(IO("Q")) D QUEUE Q
         D PRINT,^%ZISC Q
         ;
QUEUE    ;queue the report
         S ZTRTN="PRINT^ZZREPORT"
         S ZTDESC="ZZ Application Daily Report 1"
         S ZTSAVE("ZZRANGE")=""
         D ^%ZTLOAD
         I $D(ZTSK)[0 W !!?5,"Report canceled!"
         E  W !!?5,"Report queued!"
         D HOME^%ZIS Q
```

The code to set up the task after the call to ^%ZIS has four steps. First, it sets the ^%ZTLOAD input variables to define the task. Second, it calls ^%ZTLOAD to queue the task. Third, it checks $DATA(ZTSK)#2 to find out whether a task was really queued and provides appropriate feedback. Fourth, it calls HOME^%ZIS to reset its IO variables.

Note that this queuer did not define ZTIO. Print queuers can take advantage of the fact that they directly follow a ^%ZIS call that sets up all the IO variables they need. Under these conditions, the queuer code can rely on ^%ZTLOAD to identify the task's IO device from the IO variables, thus saving the programmer the work of building the correct ZTIO string.

Notice also that when queuing output, we need not call ^%ZISC to close the IO device because when the user chooses to queue output the Device Handler does not open the device. Thus all we need to do here is reset our IO variables with a HOME^%ZIS call.

As usual in these kinds of queuers, we did not define ZTDTH, but instead let ^%ZTLOAD ask the user when the report should run.

Finally, notice that we tell the task to begin at PRINT, the same tag used by the trigger code to start the foreground print when the user chooses not to queue. Under most circumstances, print queuers can use most of the same code for their tasks that the foreground print uses.

Non-interactive Use of ^%ZTLOAD

Under certain conditions, queuers must create and schedule their tasks with no interaction with the user. Examples include queuers operating out of tasks or queuers that need to run without the users' knowledge. Only two items must be changed from interactive queuers to make non-interactive queuers work:

1. **ZTDTH must be passed to ^%ZTLOAD, and must contain a valid date/time value.**

2. **If the code to queue the task does not follow a call to ^%ZIS, you must define ZTIO yourself. Either set it, or allow it to be built from the current I/O variables (if those I/O variables describe the proper device).**

After the call to ^%ZTLOAD, you may (or may not) want to issue feedback messages.

Queuing Tasks Without an I/O Device

Certain tasks need no IO device. These include primarily tasks that rearrange large amounts of data but produce no report, such as filing and compiling tasks. Two different kinds of non-IO tasks exist: those that can run concurrently, and those that must run sequentially.

Queuers for concurrent non-IO tasks need only set ZTIO to null, and TaskMan will run the task, with no IO device.

For sequential non-IO tasks, queuers must set ZTIO to the name of a resource type device. TaskMan will then ensure that the tasks run single file, one after the other in order by requested start time. Applications that need sequential non-IO tasks should instruct system managers in the Package Installation Guide to create a resource device with the desired characteristics so that these queuers can safely queue their tasks to them. Such devices should be namespaced by the package that uses them. SYNC FLAGs can also be used to allow the next task in a series to start only if the previous task in the series completed successfully. For more information see the section in this chapter on TaskMan SYNC FLAGs.

• DQ^%ZTLOAD: Dequeue a Task

Usage `D DQ^%ZTLOAD`

Input	**ZTSK:**	**The number of the task to unschedule. This task must currently be defined in the TASKS file or the call will fail.**
Output	**ZTSK(0):**	**Returned as 1 if the task was unscheduled successfully; otherwise returned as 0.**

Description

Use DQ^%ZTLOAD to unschedule tasks. Unscheduling a task ensures that, after the call, it is not scheduled or waiting for a device, computer link, or partition in memory. Unscheduling is guaranteed to be successful as long as the task is currently defined in the TASKS file. Note, however, that unscheduling a task that has already started running does not stop the task in any way.

• ISQED^%ZTLOAD: Return Task Status

Usage `D ISQED^%ZTLOAD`

Input

ZTSK: **Task number of the task to look up. The task must be currently defined on the volume set to be searched, or the look-up fails.**

ZTCPU: **(optional) The volume set TaskMan should search for the task being looked up. If not passed, TaskMan searches the current volume set. Unlike ^%ZTLOAD's ZTCPU input variable, this one doesn't accept a second :-piece specifying the CPU. It only specifies a volume set to search.**

Output

ZTSK(0): **ZTSK(0) is returned as 1 if task ZTSK is currently scheduled or waiting on volume set ZTCPU. It is returned as 0 if the task is not. It is returned as null ("") if the look-up was unsuccessful.**

ZTSK("E"): **(sometimes returned) The error code, returned when some error condition prevented a successful look-up. The codes and their values are:**

- **IT** **The task number was not valid (0, negative, or non numeric).**
- **I** **The task does not exist on the specified volume set.**
- **IS** **The volume set is not listed in the VOLUME SET file.**
- **LS** **The link to that volume set is not available.**
- **U** **An unexpected error arose (e.g., disk full, protection, etc.).**

ZTSK("D"): **(sometimes returned) The date and time the task was scheduled to start, in $HOROLOG format. It is returned only if ZTSK(0) equals 0 or 1.**

ZTSK("DUZ"): **(sometimes returned) Holds the DUZ of the user who created the task. It is returned only if ZTSK(0) equals 0 or 1.**

Description

ISQED^%ZTLOAD returns whether a task is currently pending. Pending means that the task is either scheduled, waiting for an I/O device, waiting for a volume set link, or waiting for a partition in memory. It also returns the DUZ of the task's creator and the time the task was scheduled to start.

• KILL^%ZTLOAD: Delete a Task

Usage `D KILL^%ZTLOAD`

Input **ZTSK:** **Task number of the task to delete.**

Output **ZTSK(0):** **Set to 0 if the requested task number is invalid; otherwise, set to 1 to indicate successful deletion of the task.**

Description

Use this entry point to delete a task. When a task is deleted by KILL^%ZTLOAD, the task referenced by ZTSK will not be defined in the volume set's Task file. If the task was pending, it will not start, but if it had already started running, the effects of deleting its record are unpredictable.

Note that tasks can delete their own records through the use of the ZTREQ output variable.

• REQ^%ZTLOAD: Requeue a Task

Usage D REQ^%ZTLOAD

Input

ZTSK: The task number of the task to edit. It must be defined on the current volume set for the edit to succeed. It is strongly recommended that this task not be currently running.

ZTDESC: (optional) New description for the task. It should describe the task and name the package that created the task.

ZTDTH: (optional) New start time for the task. Pass this as a date and time in VA FileMan or $HOROLOG format. If not passed, the original start time is used again. If passed as "@", the task will not be rescheduled.

ZTDTH may also be passed as a rescheduling code. This code is a number followed by an "S" (seconds), an "H" (hours), or a "D" (days). This code represents an interval of time (e.g., "60S" is sixty seconds) that is added to the current time (for seconds or hours) or the original start time (for days) to produce the new start time.

ZTIO: (optional) New I/O device for the task. It is used to set IOP, and can take all of IOP's format specification strings. If ZTIO is set to "@", the task is rescheduled for no I/O device. If it is not passed, the originally requested I/O device is used.

ZTRTN: (optional) Entry point for TaskMan to use for the task. If it is not passed, the original entry point is used.

ZTSAVE: (optional) Input parameters array: an array whose nodes specify input parameters to the task beyond the usual set all tasks receive. Set up in the same format as the ZTSAVE input variable for the ^%ZTLOAD entry point.

Output

ZTSK(0): Returned as 1 if the task is defined, 0 if it is not or if ZTDTH was passed in a bad format.

Description

Use REQ^%ZTLOAD to unschedule, edit, and reschedule a task. Unscheduling ensures the task is not pending but does not stop it from running. Editing is limited to the entry point, start time, description, and I/O device. Rescheduling is optional. However, if the task is not rescheduled, it is vulnerable to the Task File Cleanup option. The entire procedure is referred to as requeuing. Because this procedure does not involve stopping a running task, it is possible to wind up with the same task running in two different partitions if the algorithm is not designed carefully. This is not supported by TaskMan; so, programmers should use requeuing very carefully. Queuing a new task is usually a better way to accomplish the same goals.

Note that tasks may reschedule themselves through use of the ZTREQ output variable.

• $$S^%ZTLOAD: Check for Task Stop Request

Usage `I $$S^%ZTLOAD([message])`

Input	**message:**	**[optional] Allows you to leave a message for the creator of the task.**
Output	**return value:**	**1 if the creator of the task has asked the task to stop; 0 otherwise.**

Description

Use within a task to determine if the task has been asked to stop. Using the $$S^%ZTLOAD() function in longer tasks is highly recommended. Tasks should test $$S^%ZTLOAD to check if the user who queued the task has requested that the task be stopped. If the task has been asked to stop, it should set the local variable ZTSTOP to 1 before quitting. This will alert the Submanager to set the task's status to STOPPED instead of FINISHED, to give the user feedback that the task has obeyed their request.

You can use the optional message parameter to inform the user of the progress of a job. It is displayed when the task is listed by one of the many options that list tasks.

• STAT^%ZTLOAD: Task Status

Usage D STAT^%ZTLOAD

Input

ZTSK: The task number to look up. It must be defined on the current volume set.

Output

ZTSK(0): Returned as 1 if the task is defined, 0 if not.

ZTSK(1): Numeric status code from 0 to 5 indicating the status of the task.

ZTSK(2): Status text describing the status of the task. Its value corresponds with the status code in ZTSK(1). The possible values and their meanings are as follows:

ZTSK(1)=0 and ZTSK(2)="Undefined" means the task does not exist on this volume set.

ZTSK(1)=1 and ZTSK(2)="Active: Pending" means the task is scheduled, waiting for an I/O device, waiting for a volume set link, or waiting for a partition in memory.

ZTSK(1)=2 and ZTSK(2)="Active: Running" means the task has started running.

ZTSK(1)=3 and ZTSK(2)="Inactive: Finished" means the task quit normally after running.

ZTSK(1)=4 and ZTSK(2)="Inactive: Available" means the task was created without being scheduled or was edited without being rescheduled.

ZTSK(1)=5 and ZTSK(2)="Inactive: Interrupted" means the task was interrupted before it would have quit normally. Causes may include bad data, user intervention, hard error, and many other possibilities.

Description

Use STAT^%ZTLOAD to look up tasks and retrieve their current status. The status of a task returned by STAT^%ZTLOAD is expressed in the general

terms of whether the task ran, is running, or will run. ZTSK(1) and (2) return the code and text of the current status. This status is an abstraction based on the more complex system used by TaskMan.

An active task is one that is either expected to start or is currently running. An inactive task will not start in the future without outside intervention; this may be because it has already completed, was never scheduled, or was interrupted. Note that the "running" status is not based on direct examination of the system tables but is inferred from TaskMan's information about the task.

When interpreting the output of STAT^%ZTLOAD, consider that:

- **If a task is transferred to another volume set, it becomes undefined on the original volume set.**
- **A status of "running" is a guess.**
- **"Finished" does not necessarily mean the task accomplished what it set out to do.**
- **An interrupted task may or may not run correctly if edited and rescheduled.**

• $$TM^%ZTLOAD: Check if TaskMan is Running

Usage `S X=$$TM^%ZTLOAD`

Input **none**

Output

return value:	**If TaskMan is running on the current volume set, 1 is returned; otherwise, 0 is returned.**

Description

Determines if TaskMan is running. Use this function if you need to know the status of TaskMan.

Part 5: KIDS

Chapter 26 KIDS System Management: Installations

The Kernel Installation and Distribution System (KIDS) is a new module in Kernel V. 8.0. Previously, packages were exported using a utility called DIFROM, and installed by running INIT routines that the DIFROM utility created. KIDS is the replacement for DIFROM, and introduces significant revisions to the package distribution and installation processes. This chapter introduces KIDS, and describes some of the changes to the package export process.

The following definitions apply throughout the KIDS documentation:

Transport Global	**An exported package, stored in a global. KIDS exports a package based on its definition in a build entry. The transport global also contains the build entry and the PACKAGE file entry (if any) for a given package.**
Build entry	**An entry in the BUILD file that defines the parts of a package to export. Also known as a build.**
Component	**An element of one of the following types: template (print, sort, and input); form; function; bulletin; help frame; routine; option; security key; and protocol.**
Distribution	**A host file system (HFS) file containing transport global(s). If a distribution contains multiple transport globals, KIDS treats them as a single installation when installing from the distribution.**
Package	**A cohesive set of files, data, and components that together form a set of computing activities related to a functional area.**

KIDS = Distribution and Installation

As indicated by its name, KIDS supports two major functions: distribution and installation. The distribution portion of KIDS allows developers to:

- **Define the contents of a package in a build entry.**
- **Create transport globals from build entries.**
- **Export transport globals by creating distributions.**

The installation portion of KIDS allows sites to:

- **Load transport globals from KIDS distributions.**
- **Load transport globals from KIDS PackMan messages.**
- **Print out the contents of loaded transport globals before installing them.**
- **Compare the contents of loaded transport globals to the current system before installing them.**
- **Install loaded transport globals.**

KIDS brings two new files into Kernel: the BUILD file and the INSTALL file. KIDS still makes use of the existing PACKAGE file, but its role in exporting and installing packages is diminished.

Build Entries and the BUILD File

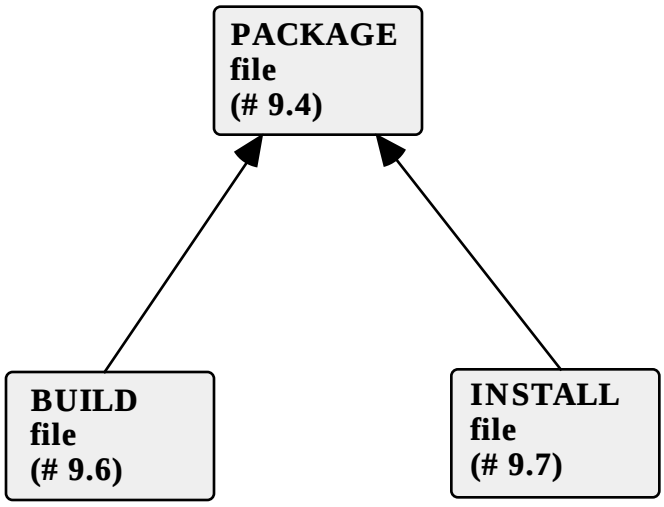
Build entries, stored in the BUILD file, are where developers define a package. This build entry defines the set of files, data, components, installation questions, national package information, pre- and post-install routines, and other settings that comprise the exported package.

Package components are no longer tied to namespace, as they were previously with DIFROM and the PACKAGE file. Developers can select any components available on the current system and include them in their build entries as package components.

The format of the .01 field of a build entry must be the package name concatenated with a space, and then a version number. This means that there is a separate entry for every version of a package that a developer exports.

Also, a package's build entry is sent to installing sites as part of the package; after an installation, the site can examine the build entry to see the package definition.

■ **KIDS File Diagram**



The INSTALL File

The INSTALL file stores a record of each installation a site performs. The INSTALL file allows KIDS to store a separate installation entry for each installation. A new version of a package no longer overwrites the installation information of a previous version, and developers' installation history no longer overwrites the sites' installation history. The national PACKAGE file is now static at its top level.

The three main items recorded in the INSTALL file for each installation are the installing site's answers to installation questions, any installation output, and the installation's timing information.

Changes in the Role of the PACKAGE File

The PACKAGE file still plays a role in installations with KIDS, albeit a diminished one. KIDS provides a link from the build entry of a package to the PACKAGE file, so that developers can link a package to a PACKAGE file entry.

The top level of a PACKAGE file entry for a package now stores static package information. The only part of the PACKAGE file entry that installations update automatically now is the VERSION multiple. Patch installations will update the PATCH APPLICATION HISTORY multiple, which is within the VERSION multiple. Most other fields have been designated for removal at the top level of the PACKAGE file. The PACKAGE file now stores mainly static package information that is not version specific, as well as the patch history of the package.

The New Transport Mechanism: Distributions

Distributions are the mechanism KIDS uses to export packages. They are more flexible than the previous mechanism (INIT routines).

Distributions are usually in the form of an HFS file. The developer creates transport globals from build entries. KIDS stores transport globals in a global. KIDS can write the global (in a format readable only by KIDS) to an HFS file; the HFS file is the distribution. The HFS file can then be distributed by a variety of methods, including FTP (file transfer protocol), diskette, and tape.

One advantage to using distributions over INIT routines is that there is no limit to the size of a package you can export. Another advantage is that during installations, you no longer have to overwrite a package's existing routines with the new routines before running the installation.

Alternatively, a KIDS distribution can be sent via a PackMan message in MailMan. But transporting packages as host files, especially large ones, avoids showing down MailMan.

Two Kinds of Distributions

KIDS supports two kinds of distributions. The first type is a standard distribution. This type of distribution contains transport globals for what are traditionally thought of as packages, including files, data, and all components. A standard distribution can contain one or more transport globals. If there is more than one transport global, KIDS treats each one as a single installation unit.

The second type of distribution is a global distribution. This type of distribution contains one transport global only, and that transport global can export M globals only.

The transport globals in both types of distributions also contain the corresponding build entry, and (if linked to a PACKAGE file entry) the corresponding PACKAGE file entry.

What Happens to DIFROM?

With the release of Kernel V. 8.0 and VA FileMan V. 21.0, developers should no longer use the DIFROM entry point to export packages. Developers should now use KIDS. The DIFROM method is still supported, but only for the support of sites that use standalone VA FileMan (VA FileMan without Kernel). Refer to VA FileMan V. 21.0's Programmer Manual for more information on using DIFROM.

Installing Standard Distributions

As noted previously, KIDS supports two types of distributions: standard and global. This section describes how KIDS installations work when installing standard distributions.

Installation Sequence

KIDS installs standard distributions in three phases: Loading transport globals from the distribution; answering installation questions for each transport global; and installing each transport global in the distribution.

Phase 1: Loading Transport Globals from a Distribution or PackMan Message

1. **Using the Load a Distribution option, the installer chooses the HFS file to load distributions from. If loading from a PackMan message, choose the message and invoke the INSTALL/CHECK MESSAGE PackMan option.**

2. **For each transport global, KIDS makes an entry in the INSTALL file for the transport global.**

3. **KIDS loads transport globals from distribution into ^XTMP.**

4. **KIDS runs the environment check for each transport global (if unsuccessful, the process quits here; the developer may or may not kill INSTALL file entries and transport globals from ^XTMP.)**

5. **The installer can print the contents of the transport global, compare the contents to the current system, and verify checksums of the transport global.**

Phase 2: Answering Installation Questions for Transport Globals in a Distribution

1. **Using the Install Package(s) option, the installer selects a distribution to install by choosing an entry from the INSTALL file.**

2. **KIDS runs the environment check for the first transport global; the environment check can allow KIDS to install the transport global, cancel installation of the transport global, or cancel installation of all transport globals in the distribution.**

3. **The installer answers pre-installation questions for the first transport global.**

4. **The installer answers standard KIDS questions for the first transport global.**

5. **The installer answers post-installation questions for the first transport global.**

6. **The installer repeats steps 2-5 for the remaining transport globals, if there are any more transport globals to process.**

7. **The installer chooses a device for the installation to run on. The installer can queue the installation or run it directly; entering an up-arrow aborts the installation.**

Phase 3: KIDS Installation of Packages

1. **KIDS disables any options and protocols the site has asked to be disabled for this install.**

2. **KIDS waits for the time period (from 0 to 60 minutes) the site specifies, if they chose to disable options and protocols.**

3. **KIDS suspends the running of queued options by TaskMan for this install, if the site chooses to do so.**

4. **The pre-install routine is run for the first transport global.**

5. **All components are installed for the first transport global.**

6. **The post-install routine is run for the first transport global.**

7. **KIDS repeats steps 4-6 for any remaining transport globals to install in the distribution.**

8. **Options and protocols that were disabled for this install (if any) are re-enabled.**

9. **Queued options are removed from suspense (if the site chose to suspend queued options).**

The Installation Menu

The KIDS Installation Menu contains the following options:

```
Installation ...                                  [XPD INSTALLATION MENU]
           **> Locked with XUPROGMODE
 Load a Distribution                              [XPD LOAD DISTRIBUTION]
 Print Transport Global                               [XPD PRINT INSTALL]
 Compare Transport global to Current System      [XPD COMPARE TO SYSTEM]
 Verify Checksums in Transport Global               [XPD PRINT CHECKSUM]
 Install Package(s)                                  [XPD INSTALL BUILD]
 Restart Install of Package(s)                     [XPD RESTART INSTALL]
 Unload a Distribution                           [XPD UNLOAD DISTRIBUTION]
```

Loading a Standard Distribution

The first step in installing a standard distribution is to load the transport globals from the Distribution. The Load a Distribution option:

- **Lists what transport globals are contained in the distribution and asks you if you want to continue.**
- **Creates entries in the INSTALL file for each transport global in the distribution that passed its environment check.**
- **Loads transport globals from the distribution (HFS file) into the ^XTMP global (if you answer YES to continue).**
- **Runs the environment check routine for each transport global. If a transport global doesn't pass its environment check, KIDS may purge it from ^XTMP; otherwise, the transport global stays in ^XTMP. KIDS tells you the result of each environment check.**
- **Checks the version number of the incoming package against any existing package of the same name at the site. If the incoming version number is not greater than the existing version, KIDS aborts the installation for the transport global in question.**
- **Echoes the name of the first transport global to pass environment check (i.e., "Use *transport globalname* to install this Distribution"). The name of the first transport global to pass its environment check is the name you use to install the distribution, in the next phase.**

Loading a distribution is the first of three phases to install a package. The second phase is answering installation questions, including scheduling the installation; the final phase is the actual running of the installation.

When loading from a PackMan message, load the distribution using the INSTALL/CHECK MESSAGE PackMan option in MailMan. For KIDS PackMan messages, this option through MailMan is equivalent to the Load a Distribution option.

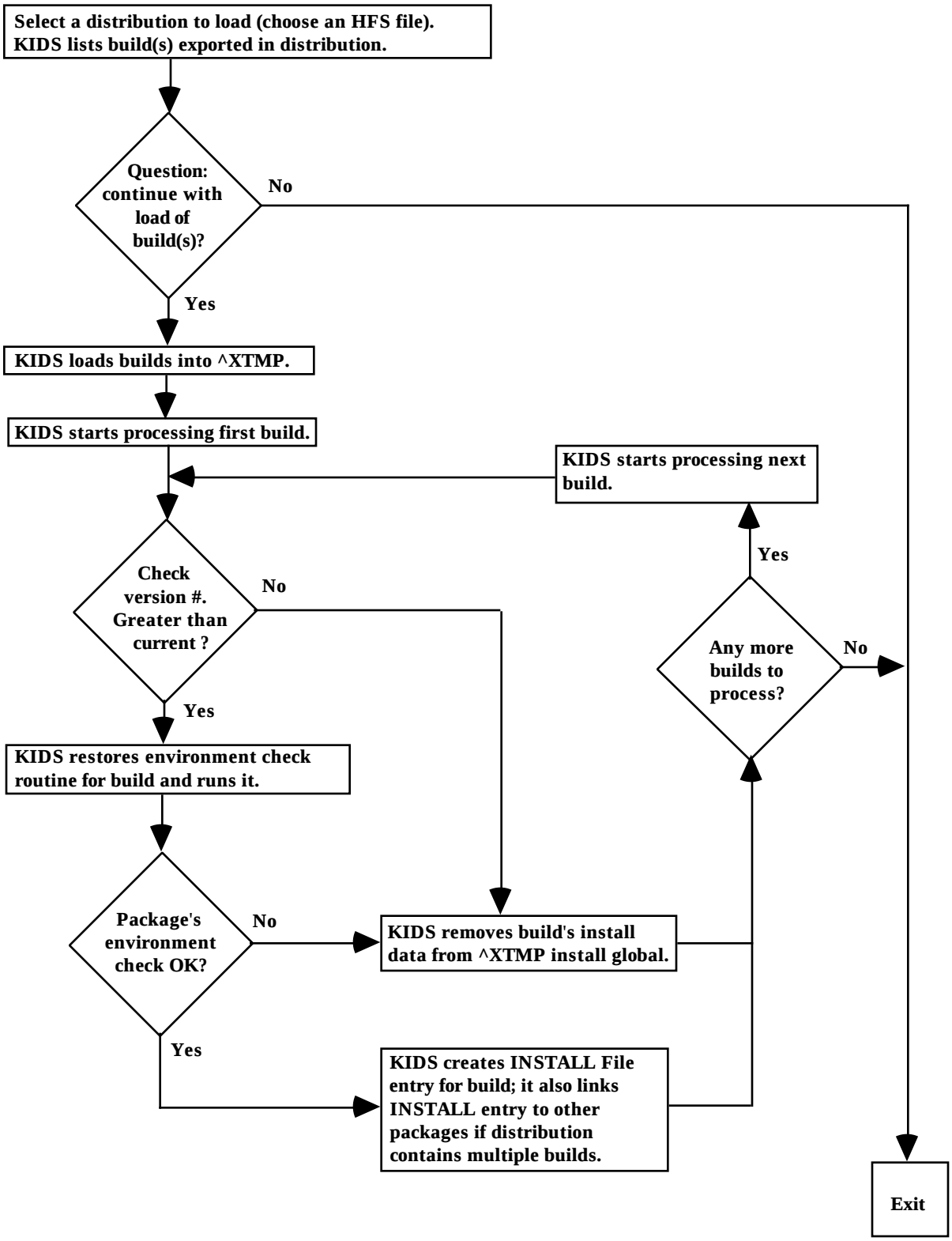
■ Loading from a Distribution

```
Select Installation Option: LOad a Distribution <RET>
Enter a Host File: ZXG_EXPT.DAT <RET>

Distribution saved on Oct 13, 1994@09:29:08
Comment: TEST ZXG PKGS

This Distribution contains Transport Globals for the following
package(s):
     ZXG DEMO 1.0
     ZXG TEST2 1.0

Want to Continue with Load? YES// <RET>
Loading Distribution...

Will first run the Environment Check Routine, ZXGDEMEN

Will first run the Environment Check Routine, ZXGTSTEN

Use ZXG DEMO 1.0 to install this Distribution.


Select Installation Option:
```

When the Distribution is Split Across Diskettes

Distributions may come in a single host file (as above); alternatively, they may come on diskettes, with the host file split up among the diskettes. If you are installing from a distribution that is spread across diskettes, the Load a Distribution option will ask you for subsequent diskettes (e.g., "Insert the next diskette, #2, and Press the return key", etc.). Insert the appropriate disk and press return, and continue until the distribution is loaded.

Loading Transport Globals from a Distribution



Printing Loaded Transport Globals

Once you have loaded transport globals from a standard distribution onto your system, you can print out the definitions of the transport globals, using the Print Transport Global option. This way, you can see every component exported in each transport global, before you install them.

■ Printed Transport Global

```
PACKAGE: ZXG DEMO 1.0                                                    PAGE 1
------------------------------------------------------------------------------
NATIONAL PACKAGE:
DESCRIPTION:

ENVIRONMENT CHECK : ZXGENV
PRE-INIT ROUTINE : ZXGPRE
POST-INIT ROUTINE: ZXGPOS
------------------------------------------------------------------------------

ROUTINE:
   ZXGC00                                          SEND TO SITE
   ZXGC01                                          SEND TO SITE
   ZXGC02                                          SEND TO SITE
   ZXGCMOVE                                        SEND TO SITE
   ZXGCTEST                                        SEND TO SITE
   ZXGCTW1                                         SEND TO SITE
   ZXGCWE                                          SEND TO SITE
   ZXGCXMP1                                        SEND TO SITE
   ZXGCXMPL                                        SEND TO SITE
   ZXGDEMO                                         SEND TO SITE
   ZXGKC                                           SEND TO SITE
   ZXGLMSG                                         SEND TO SITE
   ZXGLOAD                                         SEND TO SITE
   ZXGTMP                                          SEND TO SITE

INSTALL QUESTIONS:
     SUBSCRIPT: PRE1
DIR(0)=YA^^
DIR("A")=Do you want to run the pre-install conversion?
DIR("B")=YES
DIR("?")=Answer YES to run the pre-install conversion, NO to skip it...
```

Comparing Loaded Transport Globals to the Current System

When you have loaded transport global(s) from a standard distribution onto your system, you can also compare a transport global to the matching package already installed on your system (if any), using the Compare Transport Global to Current System option. This way, you can compare the package you're about to install with the current version of the package on your system.

When this option finds differences, it notes the change by displaying the differences between the current package and the transport global on two lines, one line labeled * OLD * and the other * NEW *.

Note that pointers are converted to free text when exporting VA FileMan entries, so these converted free pointers show up as differences when using the compare feature.

■ Comparison Sample

```
Compare ZXP 1.0 to current site
------------------------------------------------------------------------


 Routine: ZUVXD


 File # 3.2 Data Dictionary


 File # 3.2 Data

* OLD *   ^%ZIS(2,9,8) =
$C(27)_"[A"^$C(27)_"[B"^$C(27)_"[C"^$C(27)_"[D"^3^^$C(27)_"[L"
* NEW *   ^%ZIS(2,9,8) =
$C(27)_"[A"^$C(27)_"[B"^$C(27)_"[C"^$C(27)_"[D"^3
* OLD *   ^%ZIS(2,44,13) = ^$C(26)^^^^$J("",X)_$C(27,93,($X+32-X))
* NEW *   ^%ZIS(2,44,13) = ^$C(26)^^^^
* OLD *   ^%ZIS(2,60,8) =
$C(27)_"[A"^$C(27)_"[B"^$C(27)_"[C"^$C(27)_"[D"^3^^$C(27)_"[L"
* NEW *   ^%ZIS(2,60,8) =
$C(27)_"[A"^$C(27)_"[B"^$C(27)_"[C"^$C(27)_"[D"^3
* OLD *   ^%ZIS(2,90,0) = C-DATATREE^1
* NEW *   ^%ZIS(2,90,0) = C-DATATREE
* ADD *   ^%ZIS(2,93,21) = ^


HELP FRAME


BULLETIN
```

Verifying Checksums in a Transport Global

You can verify the checksums for a loaded transport global in advance of installing from it, using the Verify Checksums in Transport Global option. This option verifies all checksums of routines in the transport global, reporting any discrepancies. In the future, the ability to verify checksums will be extended to other KIDS components besides routines.

Recovering from an Aborted Distribution Load

If you encounter an error while loading a distribution (using the KIDS option to load a distribution from the export medium into the ^XTMP global), you will be unable to re-load the distribution until you clear out what was stored during the aborted load attempt.

To clear out the previously loaded distribution, use the Unload a Distribution option. To unload a distribution, enter the name of the first transport global that was loaded when you loaded the distribution. The entries in the INSTALL file for all transport globals in the distribution will be removed, and the transport globals themselves will be purged from the ^XTMP global.

Once you delete entries in the INSTALL file and entries in the ^XTMP global with the Unload a Distribution, you should be able to reload the distribution in question.

Running the Installation

Once you've loaded the transport global(s) from a standard distribution, you can install them. Do this using the INSTALL PACKAGE(S) option.

When you load a distribution, KIDS tells you which transport global name to use to install the distribution (e.g., "Use PACKAGE 1.0 to install this Distribution"). This will always be the first transport global to successfully load from the distribution. When you use the INSTALL PACKAGE(S) option, select the transport global name reported when you loaded the original distribution. Once you've done that, you can answer the installation questions for each transport global in the distribution.

Processing Each Transport Global

When you select a distribution to install, the INSTALL PACKAGE(S) option processes the installation questions for each transport global in the distribution. For each transport global, you're asked:

- **Pre-Install questions.**
- **Standard KIDS Questions.**
- **Post-Install Questions.**
- **Whether to disable any options or protocols. If you answer YES, all incoming options and protocols are disabled. You can also add to the list of options and protocols to disable. All scheduled options on the system are also disabled. Finally, you are asked a time period (from 0 to 60 minutes) to delay after disabling options and protocols, but before starting the installation. This is to allow users already in (disabled) options time to exit the options before the installation starts.**
- **Whether to install routines on other CPUs (if you are an MSM site).**

Scheduling the Installation

The final question you're asked when using the INSTALL PACKAGE(S) option to load packages is what device to run the installation on. Your choices at the DEVICE prompt are:

- **Run the installation directly by selecting a device without queuing. The installation runs immediately, on the device you specify.**
- **Queue the installation.**
- **Up-arrow out. This aborts the installation of the distribution.**

When the Installation is Queued

If you queued the installation, you can look up the installation task in TaskMan. A KIDS installation task looks like:

```
--------------------------------------------------------------------------------
3: (Task #1179950) EN^XPDIJ, KIDS install.  Device VER$LW.  KRN,KDE.
   From TODAY at 16:24,  By you.  Scheduled for TODAY at 22:00
--------------------------------------------------------------------------------
```

You can cancel a queued installation (before it has started) by deleting the task.

Re-Answering Installation Questions

If you queued an installation, you can re-answer installation questions, if you so choose, using the Install Package(s) option. To be able to re-answer the questions, however, you need to locate the task that was queued for the installation and delete it first. Once you delete the installation's queued task, you can re-answer the install questions. When you re-answer questions, your answers from the previous time come up as default responses.

Also, if you up-arrow out of an installation after answering its installation questions, your responses will again be used as the defaults the next time you try to install.

Information Stored in the INSTALL File

KIDS exports the definition of a package in the BUILD file. KIDS records installations of packages in the INSTALL file. The installation records in the INSTALL file provide a record of the start time, timing for each checkpoint, and completion time (if any) for an installation.

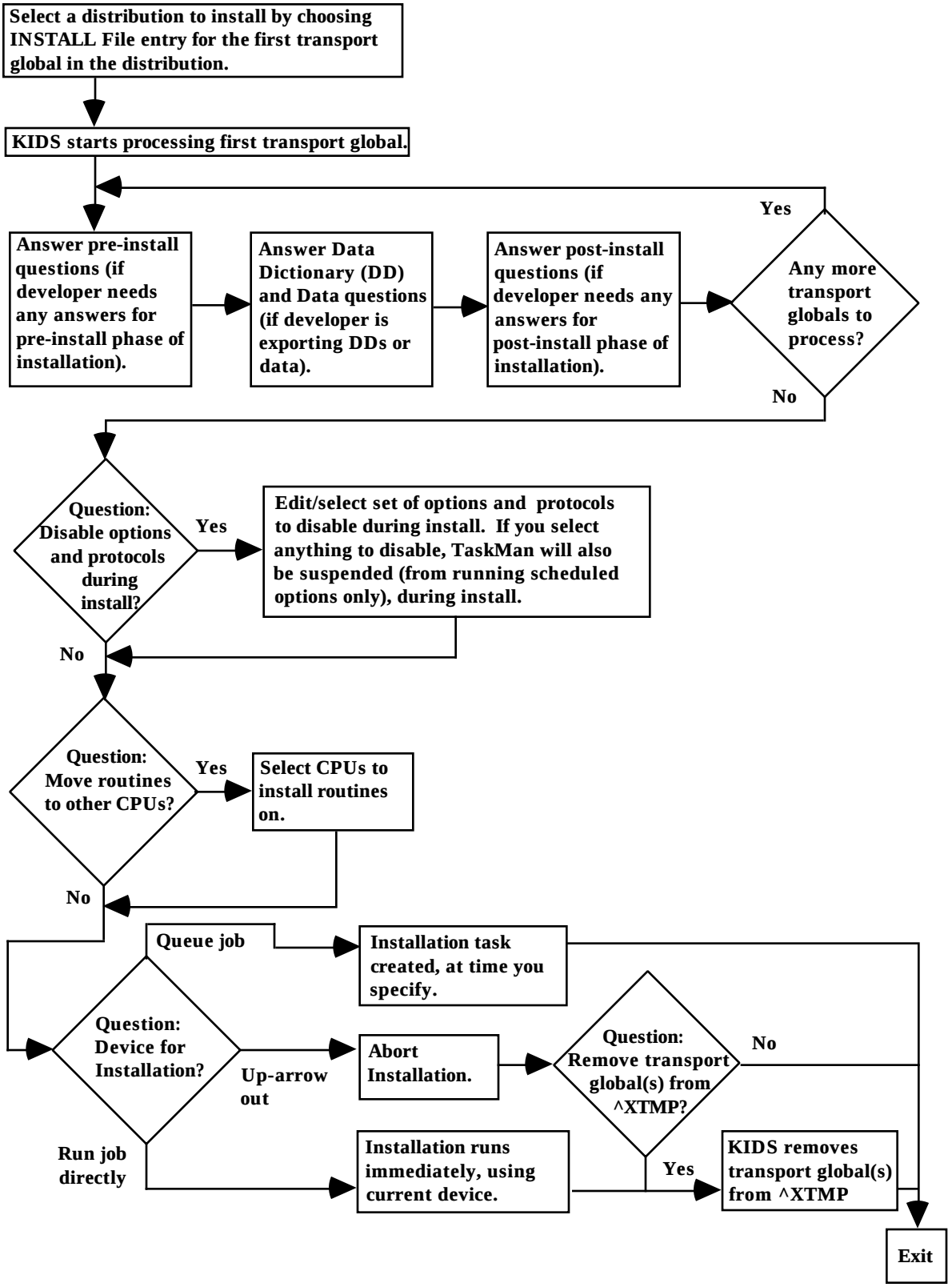
When an installation aborts, the contents of the INSTALL file determine where the install will start up again when you use the RESTART option (checkpoint information is stored in the INSTALL file).

As well as being sent to the installation's principal device, all output from the installation is also stored in the INSTALL file, in the MESSAGES word processing field.

The installation questions (and your answers to them) are stored in the INSTALL ANSWERS multiple of the INSTALL file.

You can print entries from the INSTALL file with the Install File Print option.

■ Answering Installation Questions for a Distribution

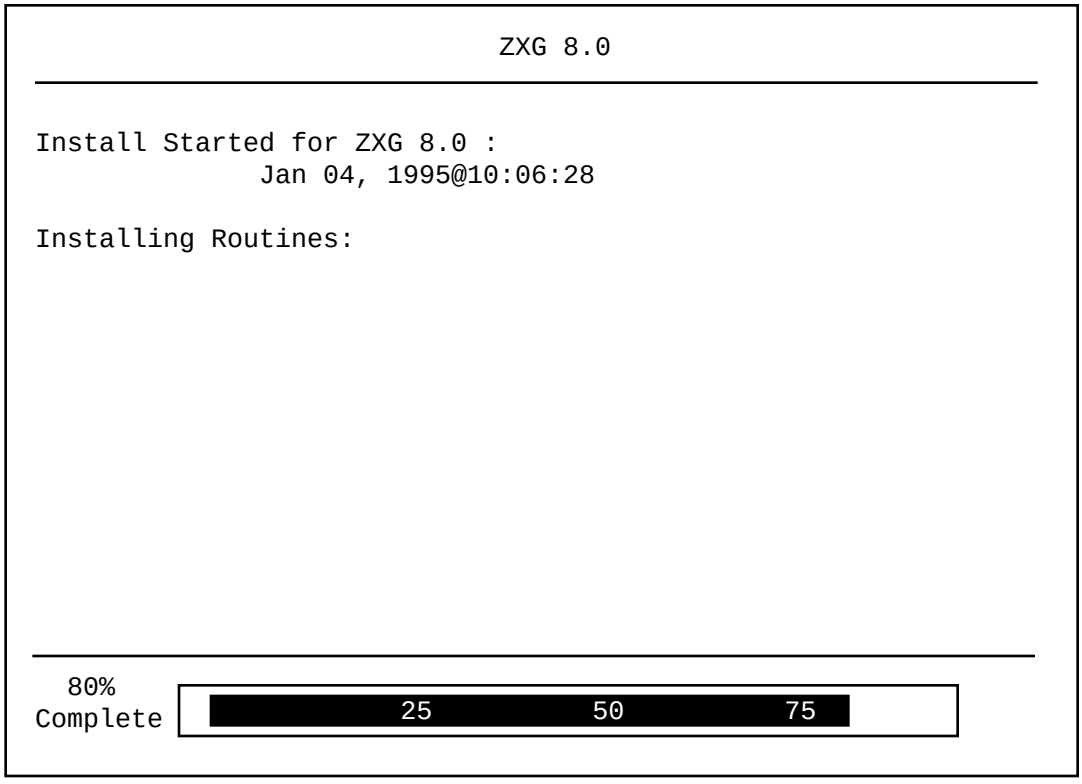


Installation Progress

If the device selected for output is a VT100-compatible (or higher) terminal, KIDS displays the installation output in a virtual window on the terminal. Below the virtual window, a progress bar graphically illustrates the percentage complete that the current part of the installation has reached. KIDS is able to report progress for the installation of files and for all components (print templates, forms, help frames, routines, options, etc.). KIDS does not show progress for installing data, nor for pre- and post-install tasks.

On all other devices, progress is reported using dots.

■ Installation Progress



Once the Installation Finishes

When the installation runs, its output is sent to the device you specified when you answered the installation questions. If, for example, you queued the installation to a printer, the output is sent to the printer.

You can find out whether an installation finished by looking up the entry in the INSTALL file for that installation (use the Install File Print option). You should check whether an installation completed successfully or not. If the install completed successfully, the STATUS field in the INSTALL file entry will be set to "Install Completed." If the install errored out, the STATUS field in the INSTALL file entry will still be set to "Install Started." If it errored out, you need to find out what went wrong, and restart the installation (see below).

If you disabled scheduled options, options, and protocols, KIDS should have re-enabled those (unless the install errored out).

You should refer to the instructions that came with the package you installed to see what post-installation tasks, if any, you should perform.

Restarting an Aborted Installation

A new feature of KIDS is the ability to restart an aborted installation. KIDS uses a checkpoint system to keep track of how many phases of an installation it completed. When an installation aborts for some reason, you can restart the installation (using the RESTART INSTALL OF PACKAGE(S) option). KIDS does not automatically re-run the entire installation from the beginning; instead, it re-runs the installation only from the last completed checkpoint.

As well as some standard checkpoints built into KIDS (such as completion of pre-install, completion of each component type, and completion of post-install), KIDS lets developers create checkpoints for use within their pre- and post-install routines. So depending on how the developer has designed a pre- or post-install, it is possible that, when re-started, the pre- or post-install doesn't have to be re-run in its entirety either (if the error occurred there). Instead, KIDS only re-runs the pre- or post-install from the last completed developer checkpoint (if any) within the pre- or post-install.

Before restarting an installation, you should try to determine what caused the installation to abort. If an error occurred, any error messages will be in the INSTALL file entry, in the MESSAGES word processing field. Once you've fixed the problem, you can use the RESTART INSTALL OF PACKAGE(S) option to continue with the installation.

Moving Routines to Other CPUs

On MSM systems only, during the main installation, you are asked if you want to move routines to other CPUs. You are not asked this question on DSM for OpenVMS systems, which are typically configured with cluster-mounted volume sets, where routine updating only needs to occur once for all CPUs in the configuration.

On an MSM system, if you say that you would like to move routines to other CPUs, you will be asked for the names of the other systems to update. Routines in the transport global will be moved to the CPUs you specify, along with any compiled templates and compiled cross-references.

If Task Manager is running, routines will be moved to the other CPUs automatically. KIDS will display a message when each CPU is updated. If Task Manager is not running, however, KIDS will tell you so, and ask you to use the ^XPDCPU direct-mode utility to update other CPUs.

> D ^XPDCPU

On MSM systems only, use this direct-mode utility to manually update CPUs (other than the one where the main installation is running) with new routines.

KIDS will ask you to use this utility if you run an installation when Task Manager is not running. The disadvantage to not having Task Manager running is that you must start ^XPDCPU running on each other CPU before the main installation completes. If you are not able to start ^XPDCPU on all other CPUs before the installation completes, you will need to perform a manual update for any CPUs that you didn't get to (see Manually Moving Routines to Another CPU below).

Manually Moving Routines to Another CPU

In some situations you may need to manually update routines on a CPU (other than the CPU on which the main installation occurred). Situations where you may need to do this include:

- **you answered NO to the "move routines to other CPUs" question, or, if you answered YES, you didn't list every CPU that needed updating**
- **an error occurred during the updating of a CPU**
- **the main installation finished before you had a chance to run ^XPDCPU on all CPUs**

In cases such as these, you can perform a manual update using two direct-mode utilities, MOVE^XPDCPU and INSTALL^XPDCPU, as described below.

1. **Run the utility MOVE^XPDCPU on the system where the main installation completed.**
2. **On each CPU other than the one that ran the main installation, run the utility INSTALL^XPDCPU.**

> D MOVE^XPDCPU

Use this direct mode utility on MSM systems only. This utility should be used when a KIDS installation completes, but routines on a particular CPU were not updated. You must run MOVE^XPDCPU on the CPU where the main installation completed. It copies all the routines that need updating into ^XTMP, once you choose the INSTALL file entry for the package in question. Then, you can use the INSTALL^XPDCPU utility on the CPUs that need routines updated.

> D INSTALL^XPDCPU

Use this direct mode utility on MSM systems only. Before using INSTALL^XPDCPU, you must use the MOVE^XPDCPU utility (described above) to copy routines that need updating to the ^XTMP global. Once you have done this, on each CPU that didn't have routines updated during the main installation, you can run INSTALL^XPDCPU. It loads routines from ^XTMP (where they're placed by MOVE^XPDCPU) onto the current CPU.

Installing Global Distributions

The second type of distribution supported by KIDS is called a global distribution. This type of distribution, unlike standard distributions, is used to export one item only: globals.

You still use the Load a Distribution option to install global distributions. Unlike loading a standard distribution, however, KIDS installs global distributions immediately from the Load a Distribution option. Also, there is no queuing of the installation.

A global distribution can only contain one transport global, and the transport global can only export globals. You know that the distribution you're installing is a global distribution rather than a standard distribution, because when you load it with the Load a Distribution option, KIDS will tell you:

```
This is a Global Distribution. It contains Global(s) that will
update your system at this time. The following Global(s) will be
installed:
```

The Load a Distribution option lists each global that will be installed from the distribution. Each global in the list is marked OVERWRITE or REPLACE. OVERWRITE means load in the global without purging the site's version of the global beforehand. REPLACE means the site's version of the global is purged first, and then the global is loaded in.

You are given two chances to abort the installation of the global distribution. if you answer YES to both questions, the globals in the global distribution are installed immediately.

Purging the BUILD and INSTALL Files

Each KIDS installation adds one entry to the BUILD and INSTALL files for every transport global installed from the distribution.

For information about purging these files, please see the discussion of the Purge Build or Install Files option, in the KIDS Utilities chapter.

Installation of a Global Distribution

```
Select Installation Option: LOAD a Distribution <RET>
Enter a Host File: [DMANAGER]XGGLOBAL.DAT <RET>

KIDS Distribution save on Jan 26, 1995@12:58:25
Comment: GLOBAL PACKAGE

This Distribution contains the following Transport global(s):
     GLOBAL PACKAGE 1.0

This is a Global Distribution. It contains Global(s) that will
update your system at this time. The following Global(s) will be
installed:
^XGRON(1)     Overwrite
^XGRON("PX")  Replace
^XGRON("TX")  Overwrite

If you continue with the Load, the Global(s) will be
Installed at this time.

Want to Continue with Load? YES// <RET>
Loading Distribution...


Globals will now be installed, OK? YES// <RET>


Installing Globals...
               Jan 26, 1995@13:04:16



 GLOBAL PACKAGE 1.0 Installed.
               Jan 26, 1995@13:04:17


Select Installation Option:
```

Alpha/Beta Tracking

```
Operations Management ...                                    [XUSITEMGR]
  Alpha/Beta Test Option Usage Menu ...                      [XQAB MENU]
    Actual Usage of Alpha/Beta Test Options[XQAB ACTUAL OPTION USAGE]
    Low Usage Alpha/Beta Test Options      [XQAB LIST LOW USAGE OPTS]
    Print Alpha/Beta Errors (Date/Site/Num/Rou/Err)
                                    [XQAB ERR DATE/SITE/NUM/ROU/ERR]
    Send Alpha/Beta Usage to Developers              [XQAB AUTO SEND]
```

Kernel provides a mechanism for tracking of installation and option usage during the alpha and beta testing of new application packages. This tool is primarily intended for application developers to use in monitoring the testing process.

Alpha/Beta tracking provides the following services to developers:

- **Notification when a new package version is installed.**
- **Periodic option usage reports.**
- **Periodic listings of errors in the package's namespace.**

The tracking of option usage is transparent to users. If the option counter is turned on, it records the number of times an option is invoked within the menu system when entered in the usual way via ^XUS. Options are not counted when navigated past in the course of menu jumping. Also, the counter is not set when entering the menu system with the programmers ^XUP utility.

There are a number of options available to sites to monitor the progress of testing.

Usage Reports for Alpha/Beta Test Options

During the testing of a package that is making use of the option counter, IRM may review the tallies with the Actual Usage of Alpha/Beta Test Options option. ADPACs may also be interested in being able to generate this information. The following example shows a printout of the actual usage of options within the XU namespace.

A similar report can be obtained of low usage options since the current version of the tracked package was installed, using the Low Usage of Alpha/Beta Test Options option.

■ Option Usage Report

```
    OPTION USAGE SINCE 08-05-92


XUSERINQ                  I        44   User Inquiry
XUUSERDISP                R        49   Display User Characteristics
XUFILEACCESS              M        50   File Access Management
XUSERBLK                  R        51   Grant Access by Profile
XUTIME                    A        53   Time
XUHALT                    A        71   Halt
XUMAINT                   M        83   Menu Management
XUSITEMGR                 M        86   Operations Management
XUSEREDITSELF             R        87   Edit User Characteristics
XUSERTOOLS                M       129   User's Toolbox
XUSEREDIT                 A       175   Edit an Existing User
XUPROG                    M       191   Programmer Options
XUSER                     M       265   User Edit
XUPROGMODE                R       268   Programmer mode
```

Sending Alpha/Beta Usage to Developers

At any time during testing, IRM may send an interim summary message back to the developers, with the Send Alpha/Beta Usage to Developers option. It may be convenient to schedule this task to run, perhaps on a weekly basis. The developer may ask you to schedule to run at a specified frequency. The usage reports are sent to the mail group and domain specified by the national developer when they exported the package.

Error Tracking

As well as tracking option usage and installations, Kernel also lets developers track errors that occur in the namespace of the tracked package. To report these errors to developers, the site should schedule the [XQAB ERROR LOG XMIT] option. This option cannot be run directly; it is on the [ZTMQUEUABLE OPTIONS] menu which is not on any Kernel menu tree. This option collects error information and sends it to a server at the development domain. The developer may ask you to schedule this option to run at a specified frequency, usually nightly.

The Print Alpha/Beta Errors (Date/Site/Num/Rou/Err) option is used at the development domain, to print error information collected from sites. It does not report meaningful information when used at a site.

Terminating Nationally Initiated Alpha/Beta Tracking

Information stored during tracking is purged each time a new version of the package is installed. A final summary report of option usage is prepared and sent to the developer's mail group just before the purge. If the new version is another test version, tracking is re-initiated with a clean slate. If the new version is the nationally released verified version, tracking ceases.

Tracked information is stored in subfiles of the KERNEL SYSTEM PARAMETERS file (#4.3), the ^XMB global. The ALPHA/BETA TEST PACKAGE multiple stores the list of package namespaces. The ALPHA/BETA TEST OPTION multiple stores pointers to entries in the OPTION file (#19). Both subfiles are purged when the verified package is installed.

Initiating and Terminating Tracking of Local Option Usage

Tracking local to a site can be initiated by making entries in the two relevant KERNEL SYSTEM PARAMETER file multiples (ALPHA/BETA TEST OPTION and ALPHA/BETA TEST PACKAGE). If there are any entries in these multiples, the menu system's XQABTST variable is set and the options are tracked.

In the case of local tracking, it is IRM's responsibility to terminate the audit and purge the data when appropriate. There is no Kernel option to purge locally collected option counts. To stop tracking, IRM should remove any corresponding entries from the two relevant KERNEL SYSTEM PARAMETER file multiples (ALPHA/BETA TEST OPTION and ALPHA/BETA TEST PACKAGE):

```
Select Kernel Management Menu Option: EN <RET> ter/Edit Kernel Site
Parameters

Note: the TaskMan site parameters have been moved out of this file.
Use the Edit TaskMan Parameters option to edit those values.

DEFAULT # OF ATTEMPTS: 3// ^ALPHA BETA TEST PACKAGE<RET>
Select ALPHA/BETA TEST PACKAGE: ZZLOCAL// @ <RET>
   SURE YOU WANT TO DELETE THE ENTIRE ALPHA,BETA TEST PACKAGE? Y
<RET>
Select ALPHA/BETA TEST PACKAGE: <RET>
Select ALPHA,BETA TEST OPTION: ZZSAMPLE// @ <RET>
   SURE YOU WANT TO DELETE THE ENTIRE ALPHA,BETA TEST OPTION? Y <RET>
```

Chapter 27 KIDS Utilities

KIDS provides the following utility options:

```
Kernel Installation and Distribution System...          [XPD MAIN]
  Utilities...                                       [XPD UTILITY]
    Build File Print                               [XPD PRINT BUILD]
    Install File Print                     [XPD PRINT INSTALL FILE]
    Convert Loaded Package for Redistribution[XPD CONVERT PACKAGE]
    Purge Build or Install Files                    [XPD PURGE FILE]
    Update Routine File                          [XPD ROUTINE UPDATE]
    Verify a Build                                 [XPD VERIFY BUILD]
    Verify Package Integrity                    [XPD VERIFY INTEGRITY]
```

These utilities can be used both by developers and by sites who install packages created by KIDS.

Build File Print

The Build File Print option prints out the build entry for a package. It lists the complete definition of the package, including all files, components, install questions, and the environment, pre-install, and post-install routines.

Install File Print

The INSTALL FILE PRINT option prints out the results of an installation, as stored in the INSTALL file. Use this option to check on the status of an installation in progress or to print out the results of a completed installation.

■ Sample Print from BUILD File

```
PACKAGE: ZXG DEMO 1.0                                                PAGE 1
-------------------------------------------------------------------------------
NATIONAL PACKAGE:
DESCRIPTION:
Package containing demonstration of ZXG* functions.

ENVIRONMENT CHECK : ZXGENV
PRE-INIT ROUTINE : ZXGPRE
POST-INIT ROUTINE: ZXGPOS


                                     UP    SEND  DATA               USER
                                     DATE  SEC.  COMES  SITE   RSLV  OVER
FILE #      NAME                     DD    CODE  W/FILE DATA   PTS   RIDE
-------------------------------------------------------------------------------

662105      ZXG DEMO                 YES   YES   NO


PRINT TEMPLATE:
   ZXG PRINT    FILE #662105                SEND TO SITE

ROUTINE:
   ZXGC00                                   SEND TO SITE
   ZXGC01                                   SEND TO SITE
   ZXGC02                                   SEND TO SITE
   ZXGC03                                   SEND TO SITE
   ZXGC04                                   SEND TO SITE
   ZXGC05                                   SEND TO SITE
   ZXGC06                                   SEND TO SITE
   ZXGC07                                   SEND TO SITE
   ZXGC08                                   SEND TO SITE

OPTION:
   ZXG TEST                                 SEND TO SITE

INSTALL QUESTIONS:
```

Sample Print from INSTALL File

```
PACKAGE: ZXG DEMO 1.0                                                PAGE 1
                                              COMPLETED          ELASPED
------------------------------------------------------------------------------
STATUS: Install Completed                DATE LOADED: FEB 07, 1995@07:51:59
NATIONAL PACKAGE:

INSTALL STARTED: FEB 07, 1995@07:52:14       07:52:23             0:00:09

ROUTINES:                                    07:52:15             0:00:01

PRE-INIT CHECK POINTS:
XPD PREINSTALL STARTED                       07:52:15
XPD PREINSTALL COMPLETED                     07:52:15

FILES:
ZXG DEMO                                     07:52:16             0:00:01

PRINT TEMPLATE                               07:52:17             0:00:03
OPTION                                       07:52:21             0:00:02

POST-INIT CHECK POINTS:
XPD POSTINSTALL STARTED                      07:52:21
XPD POSTINSTALL COMPLETED                    07:52:21

INSTALL QUESTION PROMPT                                             ANSWER

XPZ1   Want to DISABLE Scheduled Options, Options and Protocols     NO
MESSAGES:

 Install Started for ZXG DEMO 1.0 :
               Feb 07, 1995@07:52:14

 Installing Routines:
               Feb 07, 1995@07:52:15

 Running Pre-Install Routine: ^ZXGPRE

 Installing Data Dictionaries:
               Feb 07, 1995@07:52:16

 Installing PACKAGE COMPONENTS:

 Installing PRINT TEMPLATE

 Installing OPTION
               Feb 07, 1995@07:52:21

 Running Post-Install Routine: ^ZXGPOS

 Updating Routine file...

 Updating KIDS files...

 ZXG DEMO 1.0 Installed.
               Feb 07, 1995@07:52:23
```

Convert Loaded Package for Redistribution

Use this option to add packages to an existing distribution.

A KIDS distribution can transport one or more packages. What if you want to add additional packages to an existing distribution? For example, suppose you have a distribution for a package. Further suppose that patches are transported as individual KIDS packages, and you want to add all existing patches to the package's distribution? The Convert Loaded Package for Redistribution option lets you do this.

In the example used below, distributions for a package (ZXG 1.0) and a patch (ZXG*1.0*1) are both loaded. The Convert Loaded Package for Redistribution option is used to build a new distribution combining both original distributions.

Follow these steps to create a new distribution from existing distributions:

1. **Load the original distributions (there is no need to install them, however).**

   In this example, we would load the distributions for ZXG 1.0 and ZXG*1.0*1 (but we wouldn't install them).

2. **Use the Convert Loaded Package for Redistribution option. It lets you choose loaded transport globals, and transfers them into a format ready for export. Also, it creates build entries for each package contained in the distributions. This allows you to create a new distribution containing the transport globals from the existing distributions.**

   In this example, we would first convert the loaded distribution ZXG 1.0 into a form ready to re-distribute:

```
Select Utilities Option: Convert <RET>Loaded Package for Redistribution
Select INSTALL NAME: ZXG 1.0 <RET>          Loaded from Distribution

This distribution was loaded on Feb 28,1995@08:15:05 with header of

The tape consisted of the following Install(s):
ZXG 1.0

Want to continue with the conversion of the package(s)? NO//YES <RET>
  ** DONE **

Select Utilities Option:
```

   Then we would convert the patch distribution, ZXG*1.0*1, into a form ready to re-distribute:

```
Select Utilities Option: Convert <RET>Loaded Package for Redistribution
Select INSTALL NAME: ZXG*1.0*1 <RET>          Loaded from Distribution

This distribution was loaded on Feb 28,1995@08:15:35 with header of

The tape consisted of the following Install(s):
ZXG*1.0*1

Want to continue with the conversion of the package(s)? NO//YES <RET>
  ** DONE **
```

3. Create the new distribution with the Transport a Distribution option.

Select each build from the original distributions that you want to be part of the new distribution. For each build that you select, you should be told that the transport global already exists and be asked if you want to use this transport global. Answer YES in each case to use the current transport global.

Once you have selected all of the builds for the new distribution, go ahead and create the new distribution.

In the example below, we create a new distribution containing both ZXG 1.0 (the original package) and ZXG*1.0*1 (an added package):

```
Select Edits and Distribution Option: Transport <RET> a Distribution

Enter the Package Names to be transported. The order in which they are
entered will be the order in which they are installed.

First Package Name: ZXG 1.0 <RET>     **Transport Global exists**
    Use this Transport Global? YES <RET>
Another Package Name: ZXG*1.0*1 <RET>     **Transport Global exists**
    Use this Transport Global? YES <RET>
Another Package Name: <RET>

Order
  1    ZXG 1.0    **will use current Transport Global**
  2.   ZXG*1.0*1    **will use current Transport Global**

OK to continue? NO//YES <RET>

Enter a Host File: ZXG1.KID <RET>
Header Comment: PATCHED DISTRIBUTION ZXG 1.0

     ZXG 1.0...
     ZXG*1.0*1...

Package Transported Successfully
```

Note: Changing a distribution's build entries before redistributing is not recommended.

Purging Build and Install Files

Each KIDS installation adds one entry to the BUILD and INSTALL files for every transport global installed from the distribution. You can use the Purge Build or Install Files option to purge entries in these files.

The first question the option asks is which file to purge, the BUILD or INSTALL file. Choose one of these files.

The next question asked is the number of versions to retain.

Versions to Retain

When you choose to retain some number entries for a package, the option must decide which entries are most recent. The Purge Install or Build Files option uses numeric order based on package version number to decide which entries are the most recent. When there are multiple entries for the same version number (for example, alpha or beta installs took place), the following order of precedence is used:

1. **Released Version is the most recent (version number contains no letters, e.g., 8.0)**
2. **Beta Test Version (version number contains V, e.g., 8.0V10)**
3. **Alpha Test Version (version number contains T, e.g., 8.0T10)**

Selecting Package Names for Purging

After versions to retain, the next prompt is Package Name. You can enter a partial or full package name. You will continue to be prompted for additional package names until you are told to press return at the Package Name prompt.

Packages: To select package entries for purging, at the Package Name prompt, enter a partial or full package name. You can optionally enter partial or full version numbers. The list of candidates for purging will contain all entries (excluding patch entries) whose first characters match all characters in the package name that you specify. If you enter "ALL", all packages (but not patches) will be selected for purging.

Patches: Patches are a special case. To select patch entries for purging, you must enter the full namespace of the patch, the full version number, and an asterisk. You can optionally add a partial or full patch number after the asterisk. The list of candidates for purging will contain all entries whose first characters match all characters in the string you specify.

■ Purging Build File Entries for a Package

```
Select Utilities Option: Purge or Install Files <RET>

      Select one of the following:

           B         Build
           I         Install

Purge from what file: B <RET>
Versions to Retain:  (0-100): 1//0 <RET>
Package Name: ALL// ZXG <RET>
Another Package Name: <RET> ...

Package(s) in Build file, Don't retain any versions          Page 1
-------------------------------------------------------------------
ZXG 1.0
ZXG 2.0
ZXG 3.0

OK to DELETE these entries? NO// YES <RET>

Select Utilities Option:
```

Purging Selected Entries

Based on the package name you enter and the number of entries you ask to retain, the option lists the packages it finds to purge. If you answer YES to the OK to DELETE these entries prompt, the option purges the listed entries.

Reasons to Retain BUILD and INSTALL File Entries

BUILD file: Entries in the BUILD file are created by the package developers and identify every component in the package. BUILD file entries also contain the checksums for a package's components. You may want to retain the build entry for the most recent versions of installed packages, so that you can verify the checksums of the loaded package against its original checksums.

INSTALL file: Each entry in the INSTALL file contains a record of the installation for a given package. This information is useful as a record of each installation. Also, for MSM systems, if you need to manually move routines from the installation CPU to other CPUs at some point after the installation takes place, the routines that do this (MOVE^XPDCPU and INSTALL^XPDCPU) depend on the INSTALL file entry for the list of routines that need to be moved.

Update Routine File

The Update Routine File option updates the ROUTINE file to match the routine set stored on the current system.

Ideally, the ROUTINE file would contain an entry for every routine on the current system. However, the ROUTINE file does not get updated automatically when routines are added to or deleted from the system. But KIDS needs the ROUTINE file so that it can store the list of routines in a package as pointers to the ROUTINE file (rather than relying on namespace alone).

Developers should use this option to update the ROUTINE file before editing the routine component in a build entry, to ensure that all the routines they want to include in a package can be selected by the routines' matching entries in the ROUTINE file.

If you answer YES to the question "Want me to clean up the Routine file before updating?", the option goes through the ROUTINE file and deletes any entries across all namespaces that have no matches with an actual routine on the current system.

Then, Update Routine File re-populates the ROUTINE file with all routines currently on the system for the namespaces you enter (you can exclude parts of a namespace if you want, as well).

■ Updating Routine File

```
Select Utilities Option: Update Routine File <RET>


Routine Namespace: XU <RET>
Routine Namespace: -XUI <RET>
Routine Namespace: <RET>

NAMESPACE  INCLUDE                EXCLUDE
           -------                -------
           XU                     XUI

OK to continue? YES// <RET>

Want me to clean up the Routine File before updating? YES// <RET>
...SORRY, THIS MAY TAKE A FEW MOMENTS...    ...Done.
```

Verify a Build

The Verify a Build option checks whether a build entry's listed components actually exist on the current system. This is useful for developers who are preparing to create a transport global: they can check that there are actual components on the system matching the components requested in the build entry, in advance of trying to create a transport global.

For any component in the build entry that doesn't actually exist on the system, the option outputs a one-line message identifying the missing component, with the appellation **NOT FOUND.**

Developers should use VERIFY A BUILD before creating transport globals from build entries. If there are missing components, the developer should either create the missing component or remove the component from the build entry, before creating a transport global.

■ Verifying a Build

```
Select Utilities Option: Verify a Build <RET>
Select BUILD NAME:    ZXG DEMO 1.0 <RET>
   ZXGTMP  in ROUTINE File   **NOT FOUND**

  **DONE**
```

Verify Package Integrity

You can use the Verify Package Integrity option to compare checksums of package components on the system against the checksums of the components when they were originally transported. Any discrepancies are reported. Currently, routines are the only components that are checked, but checksums will be extended to other package components in the future.

The checksums of components for the currently installed package are verified against checksums stored in the BUILD file entry for the package. If the most recent version of the BUILD file entry for a package has been purged, the Verify Package Integrity option will no longer be able to verify checksums for the loaded package. Because of this, in most cases you should not purge the most recent build entry for a package.

Chapter 28 KIDS Programmer Tools: Creating Builds

KIDS introduces significant revisions to the process of exporting packages over the previous export mechanism, DIFROM. For an introduction to KIDS, please see the KIDS System Management: Installations chapter.

A functional listing of the KIDS options supporting package export is:

Task	Option Name
Create Build Entry	• **Create a Build using Namespace** • **Copy Build to Build** • **Edit a Build**
Create a Distribution	• **Transport a Distribution**

The menu path for these options is:

```
Kernel Installation and Distribution System...          [XPD MAIN]
  Edits and Distribution ...                [XPD DISTRIBUTION MENU]
    Create a Build Using Namespace            [XPD BUILD NAMESPACE]
    Copy Build to Build                            [XPD COPY BUILD]
    Edit a Build                                   [XPD EDIT BUILD]
    Transport a Distribution                [XPD TRANSPORT PACKAGE]
```

This chapter covers each of these tasks, describing how to accomplish them using KIDS options.

Build Entries

KIDS stores the definition of a package in a new file, called the BUILD file. Individual entries in the BUILD file are called build entries, or builds for short. To export a package, you must first define a build entry for it in the BUILD file.

Unlike DIFROM, where you re-used the same PACKAGE file entry each time you exported a new version of a package, with KIDS you create a new BUILD file entry each time you export a package version. One advantage of having one BUILD entry per package version is that you have a complete history of each version of your package, which makes it easier to compare previous versions of a package with the current version.

The following KIDS options, described below, support creating and maintaining build entries:

- **Create a Build Using Namespace**
- **Copy Build to Build**
- **Edit a Build**

Create a Build Using Namespace

You can quickly create a build entry and populate its components by namespace. The Create a Build Using Namespace option searches for all components in the current database matching a given list of namespaces (you can exclude by namespace also). The option searches for components of every type that match the namespace(s) and populates the build entry with all matches it finds on the system. You can then use Edit a Build to fine-tune the build entry.

As well as creating a new build entry, you can use this option to populate an existing build entry by namespace. In this case, you are asked if you want to purge the existing data. If you answer YES, the option purges the build components in the entry, and then populates the build components by namespace. If you answer NO, the option merges all components matching the selected namespaces into the existing build entry; it removes nothing already in the current build entry.

■ Kernel V. 8.0 Component Types

Print Template	**Function**	**Routine**
Sort Template	**Dialog**	**Option**
Input Template	**Bulletin**	**Security Key**
Form	**Help Frame**	**Protocol**

■ Populating a Build Entry by Namespace

```
Select Edits and Distribution Option:  Create a Build Using
Namespace <RET>

Select BUILD NAME: ZXGY 1.0 <RET>
  Are you adding 'ZXGY 1.0' as a new BUILD (the 14th)? YES <RET>
   BUILD PACKAGE FILE LINK: <RET>


Namespace: ZXG <RET>
Namespace: -ZXGI <RET>
Namespace: <RET>

NAMESPACE  INCLUDE                 EXCLUDE
           -------                 -------
           ZXG                     ZXGI

OK to continue? YES// <RET>
...SORRY, LET ME THINK ABOUT THAT A MOMENT...

    ...Done.
```

■ Copying a Build Entry

```
Select Edits and Distribution Option: COPY Build to Build <RET>

Copy FROM what Package: ZXG TEST 1.0<RET>
Copy TO what Package: ZXG TEST 1.1 <RET>
  ARE YOU ADDING 'ZXG TEST 1.1' AS A NEW BUILD (THE 5TH)? Y <RET>
(YES)
   BUILD PACKAGE FILE LINK: <RET>

OK to continue? YES// <RET>
...HMMM, LET ME PUT YOU ON 'HOLD' FOR A SECOND...    ...Done.
```

Copy Build to Build

You can create a new build entry based on a previous entry using the Copy Build to Build option. Note that with KIDS, you must create a new build entry for each new version of a package. This option gives you a way to quickly copy a previous build entry to a new entry. You can then use the Edit a Build to fine-tune the copied build entry.

If you choose an existing entry to copy into, the option purges the existing entry first before copying into it.

Edit a Build

Using the Edit a Build option, you can create new build entries and edit all parts of existing build entries. Edit a Build is a VA FileMan ScreenMan-driven option. There are four main screens in the Edit a Build. The following sections describe in detail each part of a build entry and how you can edit each part.

KIDS Build Checklists

KIDS Build Checklists are provided in an appendix. They are designed in conjunction with the Edit a Build option to help you plan your build entries.

■ Functional Layout, Edit a Build

Screen	Build Section	Build Sub-Section
Screen 1	Build Name Date Distributed Description Environment Check Routine Pre-Install Routine Post-Install Routine	
Screen 2	Files and Data	Partial DD Definition Send Data Definition
Screen 3	Build Components	Print Template Sort Template Input Template Form Function Dialog Bulletin Help Frame Routine Option Security Key Protocol
Screen 4	Install Questions Package File Link Package Tracking	

Edit a Build: Name & Version, Build Information

When you invoke the Edit a Build option, KIDS loads a four-page ScreenMan form. The first screen of the form lets you edit the following package settings:

- **Name**
- **Date Distributed**
- **Description**
- **Environment Check Routine**
- **Pre-Install Routine**
- **Post-Install Routine**

Build Name

The name of a build entry is where KIDS stores both the package's name and version number. The build name must be a package name, followed by a space and then followed by a version number. This means that every version of a package requires a separate entry in the BUILD file. One way that this is an advantage is that you have a record of the contents of every version of a package that you export.

■ Screen 1 of Edit a Build

```
                          Edit a Build                          PAGE 1 OF 4
Name: ZXG TEST 1.0
------------------------------------------------------------------------------


                     Name: ZXG DEMO 1.0

          Date Distributed: AUG 29,1994

               Description:

Environment Check Routine: ZXGENV

      Pre-Install Routine: ZXGPRE

     Post-Install Routine: ZXGPOS

_______________________________________________________________________________




COMMAND:                                  Press <PF1>H for help      Insert
```

Edit a Build: Files

The second screen of Edit a Build is where you enter all the files to export with your package. For each file, you can choose whether or not to send data with the file definition.

Data Dictionary Update

The installing site is no longer asked whether they want to override data dictionary updates: data dictionary updates are determined entirely by how you, the developer, export the file. There are two settings in KIDS you can use to determine whether KIDS should update a file's data dictionary at the installing site.

If you answer YES to Update the Data Dictionary, the data dictionary will be updated at the installing site. If you answer NO, the only time the data dictionary is updated is if the file does not exist on the installing system.

You can enter M code in the Screen to Determine DD Update field. The code should set the value of $T. If $T is true, KIDS installs the data dictionary; if $T=0, KIDS does not. The screen is only executed if the data dictionary already exists on the installing system, however; if the data dictionary doesn't already exist, the file is installed unconditionally (the screen is not executed). You can use the code in this field, for example, to examine the target environment to determine whether to update a data dictionary (providing the data dictionary already exists).

Sending Security Codes

With KIDS, you can specify on a file-by-file basis whether to send security codes. For each file, you can set Send Security Code to either YES or NO.

If you answer YES to send security codes, KIDS sends the security codes of the files on the development system. KIDS only updates security codes at the installing site on new files (i.e., files that don't already exist), however. Security codes for a file are not updated at the installing site if the file already exists.

Screen 2 of Edit a Build: Selecting Files

```
                          Edit a Build                          PAGE 2 OF 4
Name: ZXG DEMO 1.0
--------------------------------------------------------------------------------
                          File List (Name or Number)

      NEW PERSON

_______________________________________________________________________________



COMMAND:                                    Press <PF1>H for help      Insert
```

Data Dictionary and Data Settings

```
                          Edit a Build                          PAGE 2 OF 4
Name: ZXG DEMO 1.0
--------------------------------------------------------------------------------
                          File List (Name or Number)
                        DD Export Options
                    File: NEW PERSON

   Send Full or Partial DD: PARTIAL

Update the Data Dictionary: YES        Send Security Code: NO

Screen to Determine DD Update

      Data Comes With File: YES

_______________________________________________________________________________



COMMAND:                                    Press <PF1>H for help      Insert
```

Sending Full or Partial Data Dictionaries

KIDS supports sending out full data dictionaries (the entire file definition), and partial data dictionaries (specified fields in a file).

Full DD (All Fields)

To send the entire data dictionary, answer FULL at the Send Full or Partial DD prompt. In this case, all field definitions are exported. If you are sending data, you must export the FULL data dictionary.

Partial DD (Some Fields)

If you answer PARTIAL at the "Send Full or Partial DD" prompt, KIDS lets you choose what data dictionary level(s) to export. A data dictionary level is either the file number (top level of the file) or a sub-data dictionary number of a multiple (also known as a subfile). You can export any subfile, no matter how deep (every subfile's data dictionary number will be selectable).

For each data dictionary number (level) you choose to export, you can select which fields to export at that data dictionary level:

- **If you don't specify any fields, all fields are sent. This includes any multiple fields (subfiles) at the selected data dictionary level and all descendant subfiles.**

- **If you do specify fields, only the specified fields are sent. When you specify individual fields within a data dictionary level, however, you cannot choose any multiples at that data dictionary level.**

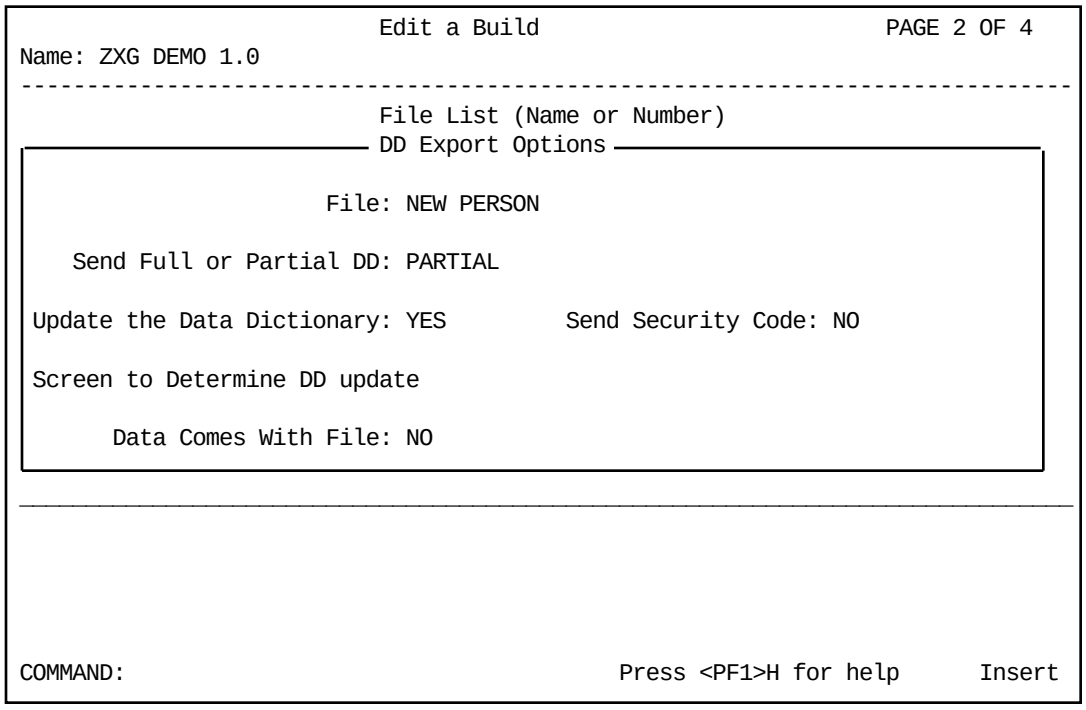
Unlike DIFROM, KIDS does not require sending the .01 field of the file if you send a partial data dictionary. Also, KIDS does not require sending out a multiple's entire data dictionary when you export a field within a multiple; you can choose to export a specific field within a multiple, at any depth.

When you export the .01 field of a multiple (by itself or by sending the entire multiple), KIDS also sends the "parent field" of the multiple (that is, the field at the next higher level of the data dictionary holding the multiple).

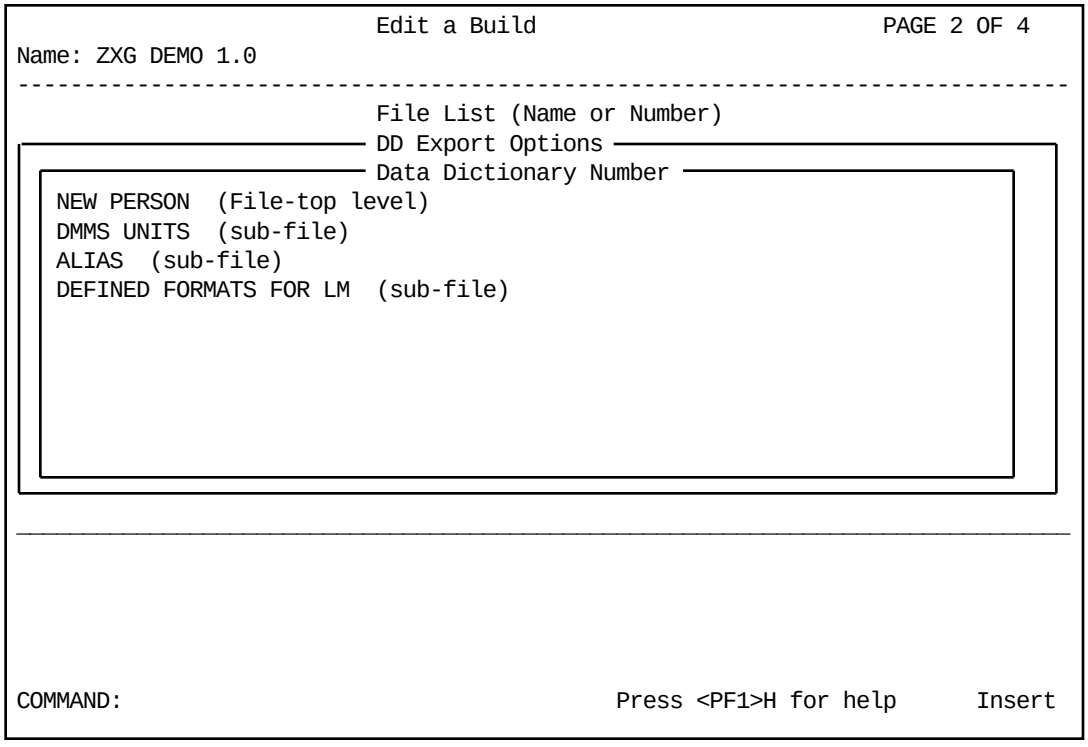
Whenever you export a multiple, all "parents" of the multiple all the way up to the .01 field of the file must exist at the installing site, or else you must export all "parents" (higher data dictionary levels) yourself. Otherwise, the multiple will not be installed.

Note that certain attributes (identifiers, "ID" nodes, etc.) are considered file attributes (as opposed to field attributes), and so are sent only when you send a full DD. They aren't sent with a partial DD.

Data Dictionary Settings Screen



Partial DD: Choosing DD Levels (Top Level and Sub-File) to Send



Choosing What Data to Send with a File

When you send data, you can send all of the data in a file. But KIDS also lets you send a subset of a file's data to installing sites.

In the Screen to Select Data field, you can enter M code to screen data. The M code should set $T; if $T is set to 1, the entry is sent, and if $T is set to 0, the entry is not sent. At the moment your code for the screen is executed, the local variable Y is set to the internal entry number of the entry being screened, and the M naked indicator is set to the global level @fileroot@(Y,0). Therefore, you can use the values of Y and the naked indicator in your screen.

In the Data List field, you can select a search template. The contents of the template will be the entries that are exported.

If you choose both a screen and a search template, the screen is applied to the entries stored in the search template.

■ Settings for Sending Data

```
                          Edit a Build                          PAGE 2 OF 4
Name: ZXG DEMO 1.0
--------------------------------------------------------------------------------
                          File List (Name or Number)
                        DD Export Options
                      Data Export Options

     Site's Data: OVERWRITE

 Resolve Pointers: YES            May User Override Data Update: YES

       Data List:

 Screen to Select Data


--------------------------------------------------------------------------------


COMMAND:                                  Press <PF1>H for help      Insert
```

Determining How Data is Installed at the Receiving Site

When you send data with a file, KIDS gives you several options about how the data is sent. There are four ways KIDS can install file entries at the receiving site:

ADD ONLY IF NEW FILE	Installs data at the installing site only if this file is new to the site or if there is no data in this file at the site.
MERGE	If no matching entry is found, the incoming entry is added. When the incoming entry matches an existing entry on the system, site fields that are non-null are preserved. Only null fields in a matching site entry are overwritten by incoming values.
OVERWRITE	If no matching entry is found, the incoming entry is added. When the incoming entry matches an existing entry on the system, site fields that are non-null are overwritten by incoming data. Values in the site's fields are preserved when the incoming field value is null, however.
REPLACE	If no matching entry is found, the incoming entry is added. When the incoming entry matches an existing entry at the top level of a file, all fields in the existing entry that are fields in the incoming data dictionary are purged; then field values for the new entry are brought in. Values in fields that aren't part of the incoming data dictionary are preserved. With multiples, if the .01 field of an incoming multiple matches the .01 field of an existing multiple, the existing multiple entry is completely purged, and the data from the incoming multiple replaces the current multiple entirely; values for fields in the existing multiple that aren't in the incoming data dictionary are not restored.

You can specify different settings for separate files; within a file, however, all data must be installed in one of these four ways.

You can give the installing site the choice of overriding the data update. If you set May User Override Data Update to YES, the installing site has the choice of whether to bring in data that has been sent with this file. They are not given the choice of how to install data, however (add only if new file vs. merge vs. overwrite vs. replace). If you set this field to NO, the installing site cannot override bringing in data.

How KIDS Matches Incoming Entries with Existing Entries

When KIDS installs VA FileMan data, it treats incoming entries differently depending on whether the entry is a new entry for the file or the incoming entry matches an existing entry in the file.

KIDS decides if an incoming entry is new or matches an existing entry by checking, in order:

1. **The B index of the file or multiple, or the .01 field if there is no B index.**
2. **The internal entry number (ien) of the entry (if applicable).**
3. **The identifiers of the entry (if applicable).**

First, KIDS makes a tentative match based on the B index. If there is no B index, KIDS goes through the .01 field entries of the file one-by-one looking for a match. Note: the B cross-reference holds the name as a subscript. The maximum length of subscripts is defined for each operating system and is stored in the MUMPS OPERATING SYSTEM file. KIDS uses this length [for example, 63 (default) or 99] as the limit of characters to compare.

If a match (either by the B cross-reference or by the first piece of the zero node) is not found, the incoming entry is considered new and is added to the file. If a match or matches are found, two additional checks are made to determine whether any of the existing entries are a match.

KIDS next checks whether the iens of any tentatively matched entries are related. If the file has a defined .001 field, the ien is a meaningful attribute of an entry. In this case, the iens must match. If the input transform of the .01 field contains DINUM, it operates the same way as a .001 field. If the ien is meaningful, and no match is found, the incoming entry is considered new and is added to the file.

If the possibility of a match remains after checking iens, KIDS performs a final check based on identifiers.

A well-designed file uses one or more identifiers to act as key fields, so that each entry is unique with respect to name and identifiers. If identifiers exist on either the target file or the incoming data dictionary, KIDS checks the values of all such identifier fields. The value of each identifier field must be the same for the existing entry and the incoming entry to be considered a match. Only the internal value of the identifier field is checked (so if an identifier is a pointer field, problems could result). Only identifiers that have valid field numbers are used in this process.

If there is still more than one matching entry after checking .01 fields, iens, and identifiers, the lowest numbered entry in the site's file is considered a match for the incoming entry for the file. On the other hand, if no match is

found after checking .01 fields, iens, and identifiers, the entry is considered new and is added to the file.

Limited Resolution of Pointers

A new feature of data export provided by KIDS is resolving pointers. For each file exported with data, you can choose whether to perform pointer resolution on that file's pointer fields (with the exception of .01 fields, identifier fields, and pointer fields pointing to other pointer fields).

KIDS does not resolve pointers for .01 fields and identifier fields in files or subfiles, nor fields that point to other pointer fields. KIDS can resolve pointers, however, for all other pointer fields in a given file or subfile.

When you don't resolve pointers, and the file being installed has pointer fields, data entries for that file are installed with whatever numerical pointer values are in the pointer fields. In which case, there is a good chance that the pointer fields no longer point to the intended entries in the pointed to file.

Resolution of pointers remedies this by exporting the free text value of the pointed-to entry. When KIDS has finished installing all files and data entries at the installing site, it begins the process of resolving pointers (if any files are set to have pointers resolved).

For each field in an entry that is a pointer field, KIDS does a lookup in the pointed to file for the free text value of the original pointed-to entry. If it finds an exact and unique match, it resolves the original pointer by storing the ien of the new matching entry in the pointer field. If it can't find an exact match, either because there are no matching entries or there are multiple matching entries, then the pointer field is left blank, and KIDS displays an error message.

Resolution of pointers works with pointed-to entries that are themselves variable pointers. In these cases, it stores which file the pointed-to entry was pointing to, and then resolves the pointer in the appropriate target file only.

Once all pointers are resolved, KIDS re-indexes each file. Each time KIDS finishes resolving pointer fields in a given file, it re-indexes that file.

Re-Indexing Files

Once all new data has been added to all files, KIDS re-indexes the files. If any of the files have compiled cross-references, the compiled cross reference routines are rebuilt. Then, if any data was sent for a file, KIDS re-indexes ALL cross references for ALL the records in the file. Only the SET logic is executed.

Data Dictionary Cleanup

If you change the definition of a field or remove a cross-reference, you must delete the field or cross reference or otherwise clean it up on the target account during the Pre-install routine. You must completely purge the target site's data dictionary of the old field definition, even if you are re-using the same node and piece for a new field. This cleanup ensures that the data dictionary will not end up with an inconsistent structure after the installation.

You no longer need to clean up word processing fields in the data dictionary, however. Before KIDS, updated data dictionary field attributes stored in word processing fields (e.g., field description or technical description) did not completely overwrite a pre-existing attribute when installed. If the incoming value had fewer lines than the pre-existing one, the install of the data dictionary did not delete the surplus lines automatically; this deletion had to be done in the pre-install. KIDS, on the other hand, completely replaces the values of word processing fields in data dictionaries.

Edit a Build: Components

In the third screen in the Edit a Build option, you can select the components of a package to include in the build.

KIDS lets you enter an explicit list of components for each component type. You are not restricted by namespace. You can select items for each type of component simply by choosing them. Items can also be selected with the * wildcard and the – exclusion sign.

With most component types, the permissible installation actions are SEND TO SITE and DELETE AT SITE. Some component types, however, have additional installation actions available; the special cases are discussed on the following pages.

■ Kernel V. 8.0 Component Types

Print Template	**Function**	**Routine**
Sort Template	**Dialog**	**Option**
Input Template	**Bulletin**	**Security Key**
Form	**Help Frame**	**Protocol**

■ Screen 3 of Edit a Build: Components

```
                          Edit a Build                          PAGE 3 OF 4
Name: ZXG 1.0
------------------------------------------------------------------------------
                          Build Components

PRINT TEMPLATE             (1)
SORT TEMPLATE              (0)
INPUT TEMPLATE             (0)
FORM                       (0)
FUNCTION                   (0)
DIALOG                     (0)
BULLETIN                   (0)
HELP FRAME                 (0)
ROUTINE                    (4)
OPTION                     (1)
SECURITY KEY               (0)
PROTOCOL                   (0)
_____________________________________________________________________________




COMMAND:                                  Press <PF1>H for help    Insert
```

Edit a Build: Options and Protocols

Menus and Protocols are similar to other component types, except for menus and protocols, which have more than the standard SEND TO SITE and DELETE AT SITE installation actions.

Note that beginning with Kernel V. 8.0, you can no longer send out an option with an attached scheduling frequency. Scheduling of options has been moved out of the OPTION file and into the new OPTION SCHEDULING file. One advantage to this is that a developer's scheduling settings will no longer overwrite a site's scheduling settings.

To indicate to the site that an option should be scheduled regularly, you should fill in the SCHEDULING RECOMMENDED field for the option. You can enter Yes, No, or Startup. This indicates to the site whether they should regularly schedule the option or not. You should list the actual frequency you recommend in the option's description. The site can then use the TaskMan option Print Recommended for Queuing Options to list all options that developers have recommended scheduling for.

■ Option and Protocol Installation Actions

Installation Action	Description
SEND TO SITE	Menu or protocol is installed at the site; any existing version already at the site is completely purged beforehand.
DELETE AT SITE	Menu or protocol is deleted at site.
USE AS LINK FOR MENU ITEMS	Designates a menu or protocol to be used as a link. The menu or protocol is not exported to the site; instead, its name is sent so that any item you link to it as a menu item or protocol (and send) becomes a sub-item on the corresponding menu or protocol at the site.
MERGE MENU ITEMS	All fields in the menu or protocol except for items are purged and replaced by the incoming values for those fields. Any items at the site that don't match incoming items are left as is. Any items that do match incoming items are completely replaced by the incoming items. The advantage with this action is that it preserves locally added items at the site. The disadvantage is that if you have removed items, the removed items are not purged at the site.

Edit a Build: Routines

Routine selection is done based on pointers to entries in the ROUTINE file, but this file is not automatically updated when programs are saved and deleted on an M system. So before adding routines to a build entry, you should run KIDS' Update Routine File option. Be sure to update all the routines and routine namespaces that you'll need to select for your build.

When selecting routines for the build, you can select individual routines by typing in their individual names. You can select a namespace group of routines by using the * wildcard. For example, to include all routines in the namespace XQ, type in XQ*. You can exclude routines by inserting the - exclusion sign before either a single name or a wild-carded namespace. For example, to exclude all routines in the XQI namespace, type -XQI*.

For each routine, you can choose one of two actions: SEND TO SITE and DELETE AT SITE. The default action is SEND TO SITE. If you choose DELETE AT SITE, the routine will be deleted at the installing site.

Installers of KIDS packages have a choice to update routines across multiple CPUs. If they choose to do this, routines will be installed (or deleted) across all CPUs the site selects. Sites cannot automatically install routines in the site's manager accounts, however; you still must instruct the site to manually install any routine that goes in the manager's account.

■ Choosing Routines

```
                          Edit a Build                          PAGE 3 OF 4
NAME: XU 99.0
------------------------------------------------------------------------------
                          BUILD COMPONENTS
                          ROUTINE
 +XQSRV4                                        SEND TO SITE
  XQSTCK                                        DELETE AT SITE
  XQT                                           SEND TO SITE
  XQT1                                          SEND TO SITE
  XQT2                                          SEND TO SITE
  XQT3                                          SEND TO SITE
  XQT4                                          SEND TO SITE
  XQTOC                                         SEND TO SITE
  XQUSR                                         SEND TO SITE
_____________________________________________________________________________


COMMAND:                                Press <PF1>H for help      Insert
```

Edit a Build: Dialog Entries (DIALOG File)

VA FileMan, starting with V. 21.0, supports the capability for other packages to store their dialog in the VA FileMan DIALOG file. Some advantages to using the DIALOG file for user interaction include:

- **Separating user interaction from other program functionality. This is a helpful step for creating GUI interfaces.**
- **Reusing dialog. When dialog is stored in the DIALOG file, it can be re-used.**
- **Easily generating package error lists. If error lists are stored in DIALOG file, there is a single point of access to print a complete list of errors.**
- **Implementing alternate language interfaces. Multiple language versions of a dialog can be exported; also, entries for one language's set of dialogs can be swapped with entries for another language's set of dialogs.**

KIDS allows you to export entries your package maintains in the DIALOG file. Simply select which DIALOG entries you want to include in your package, as you would for any other package component, and choose an installation action for each item (the default is SEND TO SITE, the other permissible choice is DELETE AT SITE).

For more information on using the DIALOG file, please see the *VA FileMan Programmer Manual* for V. 21.0 and later.

Edit a Build: Forms

You do not need to select which blocks to send when you send VA FileMan ScreenMan forms. You only need to select the form: KIDS sends all blocks associated with a form once you've chosen the form.

Edit a Build: Templates

When you select templates (either print, sort, or input templates), KIDS appends the file number to the name of the template. This ensures that a unique entry exists for each template (since two templates of the same name could exist for two different files).

■ Selecting Templates

```
                          Edit a Build                          PAGE 3 OF 4
Name: KERNEL 8.0T14
--------------------------------------------------------------------------------
                          Build Components
                          PRINT TEMPLATE

+XUSER LIST    FILE #200                          SEND TO SITE
 XUSERINQ    FILE #200                          SEND TO SITE
 XUSERVER DISPLAY    FILE #19.081               SEND TO SITE
 XUSERVER HEADER    FILE #19.081                SEND TO SITE
 XUUFAA    FILE #3.05                           SEND TO SITE
 XUUFAAH    FILE #3.05                          SEND TO SITE
 XUUSEROPTH    FILE #19.081                     SEND TO SITE
 XUUSEROPTP    FILE #19.081                     SEND TO SITE


COMMAND:                                  Press <PF1>H for help      Insert
```

Transporting a Distribution

Once you have created a build entry and added all of the files and components you want to export, you are ready to export your package. KIDS uses a transport global as the mechanism to move data. INIT routines are no longer the transport mechanism (which removes the old restrictions on the amount of data you can export). Transport globals can then be written to distributions, which are HFS files. Use the TRANSPORT option to generate transport globals and create distributions.

Depending on how you answer the questions in this option, the transport globals this option generates can be stored:

- **In a distribution, which is then ready to export as a host file.**
- **In a PackMan message (to be sent over the network).**
- **In the ^XTMP global on your local system.**

If you enter a host file name at the "Enter a Host File" prompt, the option creates transport globals and puts them in the distribution (HFS file) that you specify.

■ Creating a Distribution

```
Select Edits and Distribution Option: Transport a Distribution <RET>

Enter the Package Names to be transported. The order in which
they are entered will be the order in which they are installed.


First Package Name: ZXG DEMO 1.0 <RET>
Another Package Name: ZXG TEST 1.0 <RET>
Another Package Name: <RET>

ORDER   PACKAGE
  1    ZXG DEMO 1.0
  2    ZXG TEST 1.0


OK to continue? NO// YES <RET>

Enter a Host File: ZXG_EXPT.DAT <RET>
Header Comment: export of ZXG package <RET>


     ZXG DEMO 1.0...
     ZXG TEST 1.0...

Package Transported Successfully

Select Edits and Distribution Option:
```

If you don't enter a host file name at this prompt, KIDS asks if you would like this sent via the network. If you answer YES, you are sent into the standard PackMan message creation dialogue. You can only send one transport global per PackMan message, however.

If you don't enter a host file name, and don't want the transport globals sent via network (answer NO to this question), then KIDS creates the transport globals and stores them in your local ^XTMP global, but does not write them to a distribution file.

If you've previously created a transport global for this package in the ^XTMP global and it still exists, KIDS asks you if you want to use what was already generated or if you want to re-generate the transport globals instead.

■ Sending via Network (PackMan Message)

```
Enter a Host File: <RET>
Would you like this sent via the network? NO// YES <RET>


     ZXG DEMO 1.0...
Enter the scramble hint: <RET>
Enter scramble password: <RET>
Subject: ZXG DEMO 1.0 package <RET>
Please enter description of Packman Message

  1> <RET>
EDIT Option:
```

Splitting a Distribution Across Diskettes

You can also split a distribution across diskettes. You are only offered this choice if the M implementation you are using is MSM, and the host file name you enter for the distribution starts with "A:" or "B:". If this is the case, you will be asked, by the Transport a Distribution option:

```
Size of Diskette (1K blocks):
```

You can enter a number from 0 to 1440, where 0 means unlimited size. KIDS will then split the distribution into pieces, with each piece containing the number of 1K blocks you specify (if you choose a number other than 0). KIDS will ask, between pieces:

```
Insert the next diskette and Press the return key:
```

KIDS uses the same filename for each diskette, so be sure to use a separate diskette for each piece of the distribution file. You need to label the diskettes sequentially, e.g., #1, #2, etc. The installing site will be asked to insert the diskettes by sequence number when they install the distribution.

When to Transport More than One Transport Global in a Distribution

If several packages are unrelated, they should be sent as separate distributions. This gives the installing site optimum flexibility to decide when to do each installation.

If a group of packages is to be installed together, however, and if there are dependencies between the packages, sending the packages together in one distribution can give you more control over how the group of packages is installed. If in some cases only packages A and B should be installed, and in other situations only packages A and C should be installed, and you can do the determination yourself (in each package's environment check routine), sending the group of packages in a single distribution lets you control which packages in the distribution actually are installed.

When you are using PackMan messages to send your package (rather than using a distribution), you are limited to sending only one transport global per PackMan message.

Exporting Globals with KIDS

The version of KIDS in Kernel V. 8.0 supports the installation of global distributions (distributions that export globals). This version of KIDS does not support, however, the creation of global distributions by developers. The functionality to do this will be released either as a patch or with a future version of Kernel. For more information on global distributions, see the KIDS System Management: Installations chapter.

Creating Transport Globals that Install Efficiently

There are some choices you can make when designing your build entries, to make your transport globals install efficiently at the receiving site. In particular, you can improve the efficiency of exporting data entries using KIDS:

- **When exporting data, you can use the ADD IF NEW option to only add entries if the file didn't exist prior to the installation. Data is only added if the file is created by the installation. You can use this option to avoid re-exporting data for static files.**
- **When exporting data, send only the data you need to (KIDS no longer forces you to send all data in a file when you only need to send some of the data). You can select a subset of data to send by using a screen, a search template, or both a screen and a search template.**
- **When exporting data, resolve pointers only if necessary, because resolving pointers adds significant overhead to the process of loading data entries.**

Chapter 29 KIDS Programmer Tools: Advanced Build Techniques

The previous chapter introduced KIDS from the developer's perspective, describing the basics of how to create build entries and how to transport distributions. This chapter describes advanced build techniques that developers can use when creating builds. The following subject are covered:

- **Environment Check Routine**
- **Pre- and Post-Install Routines: Special Features**
- **Obtaining Package Name and Version Information**
- **How to Ask Installation Questions**
- **Using Checkpoints (Pre- and Post-Install Routines)**
- **Package File Link**
- **Track Package Nationally**
- **Alpha/Beta Tracking**
- **Callable Entry Point Summary**

Environment Check Routine

KIDS, like DIFROM, lets you specify an environment check routine. Typically, the environment check routine looks at the installing system and determines whether it's appropriate to install the package, based on conditions on the installing site's current system or environment.

You don't have to specify an environment check at all in order for your package to be installed. If, however, you have some special checks that you want to make to decide whether it's appropriate to go ahead with the installation, the environment check routine is the place to do it.

KIDS lets you specify the name of the environment check routine in screen one of EDIT A BUILD.

Self-Contained Routine

The environment check routine itself must be a single, self-contained routine. The reason is that it is the only routine from your build that will be loaded on the installing site's system at the time it is executed by KIDS. Based on what you find out about the installing system during the environment check, you can tell KIDS to continue installing the package, abort installing the package, or abort installing all packages (transport globals) in the distribution.

Although output during the pre-install and post-install should be done with the MES^XPDUTL and BMES^XPDUTL entry points, during the environment check routine you should use direct reads and writes.

Environment Check is Run Twice

KIDS runs the environment check routine twice. It runs the environment check routine first when the installer loads the transport global from the distribution (with the Load a Distribution option).

KIDS runs the environment check a second time when the user runs the Install Package(s) option to install the packages in the loaded distribution.

The KIDS key variable XPDENV (see below) indicates which phase (load or install) the environment check is running in.

Key Variables during Environment Check

XPDNM The KIDS key variable XPDNM is available during the environment check, as well as during the pre- and post-install phases of a KIDS installation. XPDNM is set to the name of the transport global currently being installed. It is in the format of the .01 field of the package's BUILD file entry, which is package name, concatenated with a space, concatenated with version number.

XPDENV The KIDS key variable XPDENV is available during the environment check only. It can have the following values:

- **0** The environment check is being run by the KIDS Load a Distribution option.
- **1** The environment check is being run by the KIDS Install Package(s) option.

You can use XPDENV if, for example, there is a check that is valid to perform at install time, but not at load time.

DIFROM For the purpose of backward compatibility, the variable DIFROM is available during the environment check, as well as during the pre- and post-install phases of a KIDS installation. DIFROM is set to the version number of the incoming package.

Package Version vs. Installing Version

KIDS provides several functions that you can use during the environment check to compare version numbers of the current package at the site to the incoming transport global. See the Obtaining Package Name and Version Information section in this chapter for more on this subject.

Telling KIDS to Skip Installing or Delete a Routine

During the environment check, you can tell KIDS to skip installing any routine and also change a routine's installation status to delete at site.

For example, suppose you have one version of a routine for MSM sites, one version for DSM for OpenVMS sites, and one version for DataTree sites. Based on the type of system your environment check finds, you can use the $$RTNUP^XPDUTL function to tell KIDS which routines to skip installing.

Aborting Installations During the Environment Check

In the environment check you can decide whether an installation should continue or stop, or whether the installation of all transport globals in the distribution should be aborted.

When you abort the installation of a transport global by setting XPDQUIT or XPDABORT, KIDS outputs a message to the effect that a particular transport global in the installation is being aborted. You should also issue your own message when aborting an installation, however, to give the site some diagnostic information as to why you've chosen to abort the install.

The following table lists ways you can ask KIDS to continue or abort an installation, based on the conclusions of your environment check routine:

Desired Action, Based on Environment Check Conclusions	How to Tell KIDS to Take This Action (during Environment Check)
OK to install this transport global.	(Take no action)
Don't install this transport global and kill it from ^XTMP.	S XPDQUIT =1
Don't install this transport global but leave it in ^XTMP.	S XPDQUIT=2
Abort another transport global named pkg_name in distribution and kill it from ^XTMP.	S XPDQUIT(pkg_name)=1
Abort another transport global named pkg_name in distribution but leave it in ^XTMP.	S XPDQUIT(pkg_name)=2
Abort all transport globals in distribution and kill them from ^XTMP.	S XPDABORT=1
Abort all transport globals in distribution but leave them in ^XTMP.	S XPDABORT=2

Controlling the Disable Options/Protocols Question

By default, KIDS asks the following question during KIDS installations:

```
Want to DISABLE Scheduled Options, Options, and Protocols? YES//
```

You can control the way this question is asked by defining the array XPDDIQ("XPZ1") during the environment check. The environment check runs twice (once during load and once during installation), but setting this array only has an effect during the installation. Therefore, you may want to define the array only when XPDENV=1. You can use this array as follows (each node is optional):

XPDDIQ("XPZ1")	**Set to 0 to force answer to NO or set to 1 to force answer to YES. When XPDDIQ("XPZ1") is set, the site is not asked the question.**
XPDDIQ("XPZ1","A")	**Replace default question prompt with value of this node.**
XPDDIQ("XPZ1","B")	**Set this node to new default answer, in external form ("YES" or "NO").**

Controlling the Move Routines to Other CPUs Question

By default, KIDS asks the following question during KIDS installations:

```
Want to MOVE routines to other CPUs? NO//
```

You can control the way this question is asked by defining the array XPDDIQ("XPZ2") during the environment check. The environment check runs twice (once during load and once during installation), but setting this array only has an effect during the installation. Therefore, you may want to define the array only when XPDENV=1. You can use this array as follows (each node is optional):

XPDDIQ("XPZ2")	**Set to 0 to force answer to NO, or set to 1 to force answer to YES. When this node is set, the question will not be asked.**
XPDDIQ("XPZ2","A")	**Replace default question prompt with value of this node.**
XPDDIQ("XPZ2","B")	**Set node to new default answer in external form ("YES" or "NO").**

Sample Environment Check Routine

```
ZZRON1   ;SFISC/RWF - CHECK TO SEE IF OK TO LOAD ; 8 Sep 94 10:39
      ;;8.0T13;KERNEL;;Aug 01, 1994
      N Y
      I $S($D(DUZ)[0:1,$D(DUZ(0))[0:1,'DUZ:1,1:0) W !!,*7,">> DUZ and
DUZ(0) must be defined as an active user to initialize." S XPDQUIT=2
      I $D(^DD(200,0))[0,XPDNM'["VIRGIN INSTALL" W !!,"You need to
install the KERNEL - VIRGIN INSTALL 8.0 package, instead of this
package!!" G ABRT
      ;check for Toolkit 7.3
      I $$VERSION^XPDUTL("XT")<7.3 W !!,"You need Toolkit 7.3
installed!" G ABRT
      ;
      W !,"I'm checking to see if it is OK to install KERNEL
v",$P($T(+2),";",3)," in this account.",!
      W !!,"Checking the %ZOSV routine" D GETENV^%ZOSV
      I $P(Y,"^",4)="" W !,"The %ZOSV routine isn't
current.",!,"Check the second line of the routine, or your routine
map table." S XPDQUIT=2
      ;must have Kernel 7.1
      S Y=$$VERSION^XPDUTL("XU") G:Y<7.1 OLD
      ;Test Access to % globals, only check during install
      D:$G(XPDENV) GBLOK
      I '$G(XPDQUIT) W !!,"Everything looks OK, Lets continue.",!
      Q
      ;
OLD   W !!,*7,"It looks like you currently have version ",Y," of
KERNEL installed."
      W !,*7,"You must first install KERNEL v7.1 before this version
can be installed.",!
      ;abort install, delete transport global
ABRT  S XPDQUIT=1
      Q
      ;
GBLOK ;Check to see if we have write access to needed globals.
      W !,"Now to check protection on GLOBALS.",!,"If you get an
ERROR, you need to add Write access to that global.",!
      F Y="^%ZIS","^%ZISL","^%ZTER","^%ZRTL","^%ZUA" W !,"Checking
",Y S @(Y_"=$G("_Y_")")
      Q
```

Callable Entry Points

• $$RTNUP^XPDUTL: Update Routine Action

Usage `S Y=$$RTNUP^XPDUTL(routine,action)`

Input	**routine:**	**Routine name.**
	action:	**1 to delete at site;** **2 to skip installing at site.**
Output	**return value:**	**1 if routine found in routine installation list, 0 if routine not found in routine installation list.**

Description

For use during KIDS installations, during the environment check only. Use this function to update the installation action for a routine.

Pre- and Post-Install Routines: Special Features

KIDS, like DIFROM, lets you specify pre-install and post-install routines. Typically, the pre- and post-install routines are used to perform pre-install and post-install conversions. This section describes how to use pre- and post-install routines with KIDS installations.

Pre- and post-routines are optional; you don't need to specify them at all in order for your package to be installed. If, however, you have some special actions you want to take, either before or after your installation, the pre- and post-install routines are the places to do it.

KIDS lets you specify the names for pre- and post-install routines in screen one of EDIT A BUILD.

Don't set up variables during the pre-install for use during the installation or the post-install, because these variables will be lost if the installation aborts midway through and then is restarted by the site using the restart option.

You can reference any routine exported in your build, since all routines with a SEND TO SITE action are installed by the time the pre- and post-install routines run.

Aborting an Installation during the Pre-Install Routine

You can abort an installation during the pre-install routine by setting the variable XPDABORT to 1 and quitting. Note: this is exactly as if the installing site hit <CTRL>C, in the sense that no cleanup is done; options are left disabled. KIDS prints one message to the effect that the install aborted in the pre-install program. If you abort an installation in this manner, you need to tell the site what to do to either re-start the installation or clean up the system from the state it was left in.

Setting a File's Package Revision Data Node (Post-Install)

A new Package Revision Data node can now be updated during the post-install. This node is located in ^DD(filenumber,0,"VRRV"). It is defined by the developer who distributes the package and may contain patch or revision information regarding the file. $$GET1^DID can be used to retrieve the content of the node and PRD^DILFD is used to update the node. See the *VA FileMan Programmer Manual* for more information.

Key Variables during Pre- and Post-Install Routines

XPDNM The KIDS key variable XPDNM is available during the pre- and post-install (as well as environment check) phases of a KIDS installation. XPDNM is set to the name of the build currently being installed. It is in the format of the .01 field of the package's BUILD file entry, which is package name, concatenated with a space, concatenated with version number.

DIFROM For the purpose of backward compatibility, the variable DIFROM is available during the pre- and post-install (as well as environment check) phases of a KIDS installation. DIFROM is set to the version number of the incoming package.

ZTQUEUED If the variable ZTQUEUED is present, you know that you're running as a queued installation. If ZTQUEUED is not present, you know that the installer chose to run the installation directly instead of queuing it.

Be Sure to NEW the DIFROM Variable when Calling MailMan

You are free to use the MailMan API to send mail messages during pre- and post-install routines (provided MailMan exists on the target system). Make sure that you NEW the DIFROM variable before calling any of the MailMan entry points, however. MailMan entry points may terminate prematurely if the DIFROM variable is present because the DIFROM variable has a special meaning within MailMan.

Callable Entry Points

For all output during pre- and post-installs, use the MES^XPDUTL and BMES^XPDUTL entry points. These functions write output to both the INSTALL file and the output device.

• BMES^XPDUTL: Output Message with Blank Line

Usage `D BMES^XPDUTL(msg)`

Input **msg:** **String to output.**

Output **none**

Description

For use during KIDS installations. Use this function to output a string to the installation device. Message is also recorded in INSTALL file entry for the installation. Similar to MES^XPDUTL, except that it outputs a blank line before it outputs the message, and it doesn't take arrays.

• MES^XPDUTL: Output a Message

Usage `D MES^XPDUTL([.]msg)`

Input **msg:** **Message to output, either in a variable, or passed by reference as an array of strings.**

Output **none**

Description

For use during KIDS installations. Use this function to output a message to the installation device. Message is also recorded in INSTALL file entry for the installation.

Obtaining Package Name and Version Information

• $$PKG^XPDUTL: Parse Package from Build Name

Usage `S Y=$$PKG^XPDUTL(buildname)`

Input **buildname:** **Name of build (.01 field of BUILD file).**

Output **return value:** **Package name.**

Description

Use this function to parse the name of a package from a package's build name. You can obtain the name of the build KIDS is installing from the KIDS key variable XPDNM which is defined throughout a KIDS installation.

• $$VER^XPDUTL: Parse Version from Build Name

Usage `S Y=$$VER^XPDUTL(buildname)`

Input **buildname:** **Name of build (.01 field of BUILD file).**

Output **return value:** **Version of build identified in buildname parameter; null if no match in the BUILD file.**

Description

Use this function to parse the version of a package from a package's build name. You can obtain the name of the build KIDS is installing from the KIDS key variable XPDNM, which is defined throughout a KIDS installation.

• $$VERSION^XPDUTL: PACKAGE File Current Version

Usage `S Y=$$VERSION^XPDUTL(package_id)`

Input **package_id:** **Package's name or namespace, from its entry in the PACKAGE file.**

Output **return value:** **Current version of package at site, according to package's entry in site's PACKAGE file. Returns null if the package is not matched.**

Description

Use this function to obtain the current version of a site's package.

How to Ask Installation Questions

You don't have to ask any installation questions at all in order for your package to be installed. If, however, you have some special actions that you can take in your pre-install and post-install processes, and these special actions depend on information you need to get from your installer, then you need a way to ask these questions.

Screen four of EDIT A BUILD option is where you can set up the install questions for a build.

To ask questions, you need to supply KIDS with the proper DIR input values for each question. Then, KIDS uses the DIR utility to ask installation questions when performing installations. The DIR input values you can supply for each question are:

DIR(0)	**question format**
DIR(A)	**question prompt**
DIR(A,#)	**additional message before question prompt**
DIR(B)	**default answer**
DIR(?)	**simple help string**
DIR(?,#)	**additional simple help**
DIR(??)	**help frame**

For information on the purpose of these variables, permissible values for them, and which are required versus which are optional, please refer to the *VA FileMan Programmer'Manual*.

Question Subscripts

For each question you want to ask, the .01 field of the question (as stored by KIDS) is a subscript. The subscript must be in one of two forms:

Pre-Install Questions	**PRExxx**
Post-Install Questions	**POSxxx**

"xxx" in the subscript can be any string up to 27 characters in length. KIDS asks questions whose subscript starts with PRE during the pre-install and questions whose subscript starts with POS during the post-install.

The order in which questions are asked during either the pre- or post- installs is the same as the sorting order of the subscript itself. KIDS asks questions with the lowest sorting subscript first and proceeds to the highest sorting subscript.

M Code in Questions

Besides specifying the DIR input variables, you can specify a line of M code which is executed after the DIR input variables have been set up but prior to the DIR call. The purpose of this line of M code is so that you can modify the DIR parameters, if necessary, before ^DIR is actually called.

The M code must be standalone, however; it cannot depend on any routine in the package (other than the environment check routine) since no other exported routines besides the environment check routine will be loaded on the installing system.

Skipping Installation Questions

If you want to prevent a question from being asked, you should kill the DIR variable in the line of M code for that question (execute K DIR).

Accessing Questions and Answers

Once the questions have been asked, the results of the questions are available (during pre-install and post-install only) in the following locations:

(Pre-Install Questions)

```
XPDQUES(PRExxx)=internal form of answer
XPDQUES(PRExxx,"A")=prompt
XPDQUES(PRExxx,"B")=external form of answer
```

(Post-Install Questions)

```
XPDQUES(POSxxx)=internal form of answer
XPDQUES(POSxxx,"A")=prompt
XPDQUES(POSxxx,"B")=external form of answer
```

The results of the questions for the pre-install can only be accessed (in XPDQUES) during the pre-install, and the results of the questions for the post-install can only be accessed (in XPDQUES) during the post-install. At all other times, XPDQUES is undefined for pre- and post-install questions.

■ Sample Pre-Install Question (Setting Up)

```
                          Edit a Build                          PAGE 4 OF 4
                          Install Questions
    Name: PRE1

  DIR(0): YA^^

  DIR(A): Do you want to run the pre-install conversion?
DIR(A,#):

  DIR(B): YES

  DIR(?): Answer YES to run the pre-install conversion, NO to skip it...
DIR(?,#):
 DIR(??):

  M Code:


COMMAND:                                    Press <PF1>H for help      Insert
```

■ Appearance of Question During Installation:

```
Do you want to run the pre-install conversion? YES// ? <RET>

Answer YES to run the pre-install conversion, NO to skip it...

Do you want to run the pre-install conversion? YES//
```

Where Questions Are Asked During Installations

KIDS asks the pre- and post-install questions when a site initiates an installation of the package. The order of the questions is:

1. **KIDS runs environment check routine, if any.**
2. **KIDS asks pre-Install questions.**
3. **KIDS asks generic KIDS installation questions.**
4. **KIDS asks post-Install questions.**
5. **KIDS asks site to queue the installation or run it directly.**

Using Checkpoints (Pre- and Post-Install Routines)

A new feature of KIDS allows the installing site to restart installations that have aborted. This means that your pre-install and post-install routines must be "restart-aware:" that is, they must be able to run correctly whether it's the first time they're executed or whether it is the nth time through.

KIDS maintains a set of internal checkpoints during an installation. For each phase of the installation (for example, completion of each package component), it uses a checkpoint to record whether that phase of the installation has completed yet. If an installation errors out, checkpointing allows the installation to be restarted, not from the very beginning, but instead only from the last completed checkpoint onward.

In your pre- and post-install routines, you can use your own checkpoints. If there's an error during the pre- or post-install, and you use checkpoints, when the sites restart the installation, it will resume from the last completed checkpoint rather than running through the entire pre- or post-install again.

Another advantage of using checkpoints is that you can record timing information for each phase of your pre- and post-install routines, which allows you to evaluate the efficiency of each phase you define.

There are two distinct types of checkpoints you can create during pre- and post-install routines: checkpoints with call backs and check points without call backs.

Checkpoints With Call Backs

The preferred method of using checkpoints is to use checkpoints with call backs. When you create a checkpoint with a call back, you give the checkpoint an entry point (the call back routine). That is all you have to do during your pre- or post-install routine: create a checkpoint with a call back. You do not have to execute the call back. At the completion of the pre- or post-install routine, KIDS manages the created checkpoints by calling, running, and completing the checkpoint and its call back routine.

The reason to let KIDS execute checkpoints (by creating checkpoints with call backs) is to ensure that the pre-install or post-install runs in the same way whether it is the first installation pass, or if the installation aborted and has been restarted. If the installation has restarted, KIDS skips any checkpoints in the pre-install or post-install that have completed, and only executes the call backs of checkpoints that have not yet completed (and completes them).

In this scenario (checkpoints with call back routines), your pre-install and post-install routine should consist only of calls to $$NEWCP^XPDUTL to create checkpoints (with call backs). Once you create all of the checkpoints

for each discrete pre- or post-install task you need to accomplish, your pre-install or post-install should quit.

Once your pre- or post-install routine finishes, KIDS executes each created checkpoint (that has a call back) in the order you created them. If it is the first time through, each checkpoint is executed. If the installation has been restarted, KIDS skips any completed checkpoints, and only executes checkpoints that have not completed.

A summary of the KIDS checkpoint functions that apply when using checkpoints with call backs is:

$$NEWCP^XPDUTL	**Create checkpoint (use during pre- or post-install routine only.)**
$$CURCP^XPDUTL	**Retrieve current check point name (use during pre- or post-install routine). Useful when using the same tag^routine for multiple call backs; this is how you determine which call back you're in.**
$$PARCP^XPDUTL	**Retrieve checkpoint parameter (use within call back routine.)**
$$UPCP^XPDUTL	**Update checkpoint parameter (use within call back routine.)**

Checkpoint Parameter Node

You can store how far you have progressed with a task you're performing in the call back by using a checkpoint parameter node. The $$UPCP^XPDUTL entry point updates the value of a checkpoint's parameter node; the $$PARCP^XPDUTL function retrieves the value of a checkpoint's parameter node.

Being able to update and retrieve a parameter within a checkpoint can be quite useful. For example, if you're converting each entry in a file, as you progress through the file you can update the checkpoint's parameter node with the internal entry number (ien) of each entry as you convert it. Then if the conversion errors out and has to be re-started, you can write your checkpoint call back in such a way that it always retrieves the last completed ien stored in the checkpoint's parameter node. Then, it can process entries in the file starting from the last completed ien, rather than the first entry in the file. This is one example of how you can save the site time and avoid re-processing.

Using Checkpoints with Call Backs: Combined Pre- and Post-Install Routine

The pre-install entry point in this example is PRE^ZZRON2; the post-install entry point is POST^ZZRON2.

```
ZZRON2   ;RON TEST 1.0 PRE AND POST INSTALL
      ;;1.0
      ;build checkpoints for PRE
PRE   N %
      S %=$$NEWCP^XPDUTL("ZZRON1","PRE1^ZZRON2","C-")
      Q
PRE1  ;check terminal type file
      N DA,UPDATE,NAME
      ;quit if answer NO to question 1
      Q:'XPDQUES("PRE1")
      S UPDATE=XPDQUES("PRE2")
      ;write message to user about task
      D BMES^XPDUTL("Checking Terminal Type File")
      ;get parameter value to inialize NAME
      S NAME=$$PARCP^XPDUTL("ZZRON1")
      F  S NAME=$O(^%ZIS(2,"B",NAME)) Q:$E(NAME,1,2)'="C-"  D
      .S DA=+$O(^%ZIS(2,"B",NAME,0))
      .I DA,$D(^%ZIS(2,DA,1)),$P(^(1),U,5)]"" D MES^XPDUTL(NAME_"
still has data in field 5") S:UPDATE $P(^%ZIS(2,DA,1),U,5)=""
      .;update parameter NAME
      .S %=$$UPCP^XPDUTL("ZZRON1",NAME)
      Q
      ;build checkpoints for POST
POST  N %
      S %=$$NEWCP^XPDUTL("ZZRON1","POST1^ZZRON2")
      S %=$$NEWCP^XPDUTL("ZZRON2")
      Q
POST1 ;check version multiple
      N DA,VER,%
      ;quit if answer NO to question 1
      Q:'XPDQUES("POST1")
      ;write message to user about task
      D BMES^XPDUTL("Checking Package File")
      ;get parameter value to inialize DA
      S DA=+$$PARCP^XPDUTL("ZZRON1")
      F  S DA=$O(^DIC(9.4,DA)) Q:'DA  D
      .S VER=+$$PARCP^XPDUTL("ZZRON2")
      .F  S VER=$O(^DIC(9.4,DA,22,VER)) Q:'VER  D
      ..;here is where we could do something
      ..;update parameter VER
      ..S %=$$UPCP^XPDUTL("ZZRON2",VER)
      .;update parameter DA
      .S %=$$UPCP^XPDUTL("ZZRON1",DA),%=$$UPCP^XPDUTL("ZZRON2",VER)
      Q
```

Checkpoints Without Call Backs (Data Storage)

KIDS ignores checkpoints that don't have call back routines specified. The ability to create checkpoints without a call back routine is provided mainly as a facility for programmers to store information during the pre- or post-install routine. The parameter node of the checkpoint serves as the data storage mechanism. It is not safe to store important information in local variables during pre- or post-install routines: because installations can now be re-started in the middle, variables defined prior to the restart may no longer be defined after a restart.

An alternative use lets you expand the scope of checkpoints without call backs beyond simply storing data. If you want to manage your own checkpoints instead of letting KIDS manage them, you can create checkpoints without call backs, but use them to divide your pre- and post-install routine into phases. Rather than having KIDS execute and complete them (as happens when the checkpoint has a call back routine), you would then be responsible for executing and completing the checkpoints. In this style of coding a pre- or a post-install routine, you would:

1. **Check if each checkpoint exists ($$VERCP^XPDUTL); if it doesn't exist, create it ($$NEWCP^XPDUTL).**

2. **Retrieve the current checkpoint parameter as the starting point if you want to ($$PARCP^XPDUTL); do the work for the checkpoint; update the parameter node if you want to ($$UPCP^XPDUTL).**

3. **Complete the checkpoint when the work is finished ($$COMCP^XPDUTL).**

4. **Proceed to the next checkpoint.**

You have to do more work this way than if you let KIDS manage the checkpoints (by creating the checkpoints with call back routines).

A summary of the KIDS checkpoint functions that apply when using checkpoints without call backs is:

$$NEWCP^XPDUTL	**Create checkpoint (use during pre- or post-install routine.)**
$$PARCP^XPDUTL	**Retrieve checkpoint parameter (use during pre- or post-install routine.)**
$$UPCP^XPDUTL	**Update checkpoint parameter (use during pre- or post-install routine.)**

$$VERCP^XPDUTL	**Verify if checkpoint exists and if it has completed (use during pre- or post-install routine.)**
$$COMCP^XPDUTL	**Complete checkpoint (use during pre- or post-install routine.)**

Callable Entry Points (Checkpoints)

• $$COMCP^XPDUTL: Complete Checkpoint

Usage `S Y=$$COMCP^XPDUTL(name)`

Input	**name:**	**Checkpoint name.**
Output	**return value:**	**1 if successfully complete checkpoint, 0 if error completing checkpoint.**

Description

For use during KIDS installations. Use this function to complete a checkpoint, in pre- or post-install routines. Use this only to complete checkpoints that don't have call back routines; if the checkpoint has a call back routine, KIDS itself completes the checkpoint. You can only complete checkpoints that are for the same installation phase (pre-install or post-install) that you are currently in.

Use this entry point only for checkpoints with no call back. KIDS completes checkpoints that have a call back.

• $$CURCP^XPDUTL: Get Current Checkpoint Name/IEN

Usage `S Y=$$CURCP^XPDUTL(format)`

Input	**format**	**Pass as 0 to return checkpoint name; pass as 1 to return checkpoint internal entry number (ien).**
Output	**return value:**	**Current checkpoint name. Returns null string if not currently in a checkpoint call back.**

Description

For use during KIDS installations. Use this function to return the name of the current checkpoint. It can be useful if, for example, you use the same

tag^routine entry point for more than one call back. Using this function, you can determine which call back you are in.

Use this entry point only for checkpoints with a call back. It will return the null string if you call it when working with a checkpoint with no call back (in which case, you would really be in either the pre- or post-install routine).

• $$NEWCP^XPDUTL: Create Checkpoint

Usage `S Y=$$NEWCP^XPDUTL(name,[callback],[par_value])`

Input	**name:**	**Checkpoint name.**
	callback:	**[optional] Call back (^routine or tag^routine reference).**
	par_value:	**[optional] Value to set checkpoint parameter to.**
Output	**return value:**	**Internal entry number of created checkpoint if created or if checkpoint already exists, or 0 if error occurred while creating checkpoint.**

Description

For use during KIDS installations. Use this function to create a checkpoint, in pre- or post-install routines. The checkpoint is stored in the INSTALL file.

Pre-and post-install checkpoints are stored separately, so you can use the same name for a pre- and post-install checkpoint if you wish. Checkpoints created with this function from the pre-install routine are pre-install checkpoints; checkpoints created during the post-install routine are post-install checkpoints.

You can use $$NEWCP^XPDUTL to create a checkpoint with or without a call back. You can also store a value for the parameter node, if you wish.

Checkpoints created with call backs have that call back automatically executed by KIDS during the appropriate phase of the installation. If the checkpoint is created during the pre-install routine, KIDS executes the call back as soon as the pre-install routine completes. If the call back is created during the post-install, KIDS executes the call back as soon as the post-install routine completes. If multiple checkpoints are created during the pre- or post-install routine, KIDS executes the call backs (and completes the checkpoints) in the order the corresponding checkpoints were created. Checkpoints created without a call back cannot be executed by KIDS; instead, they provide a way for developers to store and retrieve information during the pre-install and post-install phases. Rather than storing information in a local

or global variable, you can store information in a checkpoint parameter node and retrieve it (even if an installation is re-started).

If the checkpoint you are trying to create already exists, the original parameter and call back will not be overwritten.

• $$PARCP^XPDUTL: Get Checkpoint Parameter

Usage `S Y=$$PARCP^XPDUTL(name[,pre])`

Input	**name:**	**Checkpoint name.**
	pre:	**[optional] To retrieve a parameter from a pre-install checkpoint while in the post-install, set this parameter to "PRE".**
Output	**return value:**	**Current parameter node for checkpoint named in name parameter.**

Description

For use during KIDS installations. Retrieves the current value of a checkpoint's stored parameter. The parameter is stored in the INSTALL file.

Use this entry point for checkpoints both with and without call backs.

Use the optional second parameter to retrieve a pre-install checkpoint's parameter during a post-install.

• $$UPCP^XPDUTL: Update Checkpoint

Usage `S Y=$$UPCP^XPDUTL(name[,par_value])`

Input	**name:**	**Checkpoint name.**
	par_value:	**[optional] Value to set checkpoint parameter to.**
Output	**return value:**	**Internal entry number of updated checkpoint if successful or 0 if error updating checkpoint.**

Description

For use during KIDS installations. Use this function to update the parameter node of an existing checkpoint, in pre- or post-install routines. The parameter node is stored in the INSTALL file.

Use this entry point for checkpoints both with and without call backs.

During the pre-install, you can only update pre-install checkpoints; during the post-install, you can only update post-install checkpoints.

• $$VERCP^XPDUTL: Verify Checkpoint

Usage `S Y=$$VERCP^XPDUTL(name)`

Input	**name:**	**Checkpoint name.**
Output	**return value:**	**1 if checkpoint has completed, 0 if checkpoint has not completed but exists, -1 if checkpoint doesn't exist.**

Description

For use during KIDS installations. Use this function to check whether a given checkpoint exists and, if it exists, whether it has completed or not.

Use this entry point only for checkpoints with no call back.

During the pre-install, you can only verify pre-install checkpoints; during the post-install, you can only verify post-install checkpoints.

PACKAGE FILE LINK

In the fourth screen of the EDIT A BUILD option, you can link your build to an entry in the national PACKAGE file. Use this link if you want to update the site's PACKAGE file when the package you're creating is installed or if you want to use Kernel's Alpha/Beta Testing module. You can only link to a PACKAGE file entry that is the same name (minus the version number) as the build you're creating.

If you specify a PACKAGE file entry in the PACKAGE FILE LINK field, and the installing site does not have a matching entry in their PACKAGE file, KIDS creates a new entry in the installing site's PACKAGE file.

When you link to an entry in the PACKAGE file, your installation automatically updates the VERSION multiple in the installing site's corresponding PACKAGE file entry. KIDS makes a new entry in the VERSION multiple for the version of the package you are installing. KIDS fills in the following fields in the new VERSION entry:

- **VERSION**
- **DATE DISTRIBUTED**
- **DATE INSTALLED AT THIS SITE**
- **INSTALLED BY**
- **DESCRIPTION OF ENHANCEMENTS**

In addition, you can choose to update the following fields at the top level of the national package file:

PRIMARY HELP FRAME	**Select the primary help frame for the package.**
AFFECTS RECORD MERGE	**(multiple) Select files that, if merged, affect this package.**
ALPHA/BETA TESTING	**YES means that this package is currently in alpha or beta test and that you want to track option usage and errors relating to this package at the sites.** **NO means that you want to discontinue tracking of alpha or beta testing at the sites.**

Beyond these fields, KIDS does not support maintaining any other information in the PACKAGE file.

Track Package Nationally

The fourth screen of the EDIT A BUILD option also lets you choose whether to send a message to the National Package File on FORUM, each time the package is installed at a site. If you enter YES in the TRACK PACKAGE NATIONALLY field, KIDS sends a message to FORUM when a site installs the package, provided the following conditions are met:

- **The PACKAGE FILE LINK field in the build entry points to an entry in the PACKAGE file.**
- **The package is installed at a site that is a primary VA domain.**
- **The package is installed in a production UCI.**

Answering NO to TRACK PACKAGE NATIONALLY (or leaving it blank) means that KIDS does not send a message to FORUM.

Alpha/Beta Tracking

Alpha/Beta tracking provides the following services to developers:

- **Notification when a new package version is installed.**
- **Periodic option usage reports.**
- **Periodic listings of errors in the package's namespace.**

This section describes how to set up alpha/beta tracking for packages you export.

Setting up Alpha/Beta Tracking in the Build Entry

The first step to setting up alpha/beta tracking occurs when you are creating your build entry. In the PACKAGE FILE LINK section of the build entry, you can turn on alpha/beta tracking by answering YES to Alpha/Beta Testing. KIDS places you in a form that lets you edit the following alpha/beta testing options:

Installation Message	**Answering YES means that an installation message will be sent when this package is installed at a site. The message will be sent to the mail group specified in the ADDRESS FOR USAGE REPORTING field.**
Address for Usage Reporting	**Should be set to the MailMan address of a mail group at the developer's domain. This MailMan address is where installation and option usage messages are sent by the Alpha/Beta Tracking module. Also, the domain specified in the address is where server requests are sent from the sites to report errors.**
Select Package Namespace or Prefix	**This field is where you identify the alpha/beta package namespaces to track.**

There is additional setup required to enable alpha/beta tracking, however. The remaining tasks are:

- **The server option must be set up correctly at the development domain.**
- **Errors Logged in the Alpha/Beta Test (Queued) option should be scheduled to run at sites to gather errors and report these to the development server.**
- **Send Alpha/Beta Usage to Developers option should be scheduled at the sites to send mail messages containing option usage.**

The Site Option to Report Errors

```
ZTMQUEUABLE OPTIONS                              [ZTMQUEUABLE OPTIONS]
   Errors Logged in Alpha/Beta Test (QUEUED)     [XQAB ERROR LOG XMIT]
```

The Errors Logged in Alpha/Beta Test (QUEUED) option identifies any errors associated with an application package which is in either alpha or beta test. You should instruct your test sites to schedule it as a task to run daily, after midnight.

The identified errors are combined in a mail message which includes the type of error, routine involved, date (usually the previous day), the option which was being used at the time of the error, and the number of times the error was logged. The volume and UCI are included so that stations with error logs being maintained on different CPUs can run the task on each different system.

Collecting Error Reports (Developer's Domain)

In order to track errors at test sites, make sure that the server option [XQAB ERROR LOG SERVER] resides at your development site (which should be the domain specified in the ADDRESS FOR USAGE REPORTING field in the build entry).

This option processes server requests from the test sites, from the [XQAB ERROR LOG XMIT] option. The server stores the data from the requests into the XQAB ERRORS LOGGED file.

You can use the Print Alpha/Beta Errors (Date/Site/Num/Rou/Err) option to print out the collected errors.

```
Operations Management ...                                  [XUSITEMGR]
  Alpha/Beta Test Option Usage Menu ...                    [XQAB MENU]
    Print Alpha/Beta Errors (Date/Site/Num/Rou/Err)
                                    [XQAB ERR DATE/SITE/NUM/ROU/ERR]
```

Send Alpha/Beta Usage to Developers Option

To receive option usage reports, you should instruct the sites to schedule this option to run at whatever frequency you want to receive option usage reports. The option can also be run manually by the sites to send option usage information. Mail messages are sent to the mail group specified in the build entry for Address for Usage Reporting. Make sure that this mail group exists at your development domain!

Turning Off Alpha/Beta Tracking

Alpha/Beta Tracking, once initiated for a package, should be turned off when the final version of the package is released. You can turn off alpha/beta tracking by answering NO to the ALPHA/BETA TESTING field in the build. When the sites install the build, tracking is shut off.

To manually shut down tracking at an individual site, they can use the Enter/Edit Kernel Site Parameters option to remove the desired entries from the ALPHA/BETA TEST PACKAGE multiple in the KERNEL SYSTEM PARAMETERS file.

KIDS Callable Entry Point Summary

KIDS Environment Check Functions

Entry Point	Description
$$RTNUP^XPDUTL(routine,action)	Update routine installation action (install, delete, or skip).

KIDS Installation Output Functions

Entry Point	Description
BMES^XPDUTL(msg)	Output a string during install, preceded by blank line.
MES^XPDUTL([.]msg)	Output a string during install.

KIDS Package Name and Version Functions

Entry Point	Description
$$PKG^XPDUTL(buildname)	Parse package name from build name.
$$VER^XPDUTL(buildname)	Parse package version from build name.
$$VERSION^XPDUTL(package_id)	Return site's currently installed package version from PACKAGE file.

KIDS Check Point Functions

Entry Point	Description
$$COMCP^XPDUTL(name)	Complete a checkpoint.
$$CURCP^XPDUTL(format)	Get current checkpoint name or internal entry number.
$$NEWCP^XPDUTL(name,[callback], [par_value])	Create a checkpoint.
$$PARCP^XPDUTL(name[,pre])	Get checkpoint parameter node.
$$UPCP^XPDUTL(name[,par_value])	Update a checkpoint.
$$VERCP^XPDUTL(name)	Verify a checkpoint.

Part 6: Other Tools

Chapter 30 Operating System Interface

Kernel provides several utilities to work with the underlying operating system. In addition, Kernel's ^%ZOSF global holds operating system-dependent logic so that application programs may be written independently of any specific operating system. Each CPU or node in a system should have its own copy of the ^%ZOSF global; the ^%ZOSF global should not be translated.

System Management

>D ^%ZTBKC: Global Block Count

You can count the data blocks in a global using the direct mode utility ^%ZTBKC. An entire global or a subscripted section can be measured (e.g., ^DIC or ^DIC(9.2)). There is a corresponding option that can be used from the Programmer Options menu, called Global Block Count.

>D ^ZTMGRSET: Update ^%ZOSF Nodes

This direct mode utility is only available from the manager's account. It is ordinarily run during Kernel installations to initialize Kernel in the manager's account. It can be used at a later time, however, to update an account's ^%ZOSF nodes with new UCI and volume set information. The ^%ZOSF nodes that ^ZTMGRSET updates are:

^%ZOSF("MGR")
^%ZOSF("PROD")
^%ZOSF("VOL")

An example of a use for re-running ^ZTMGRSET would be when creating a new print, compute, file, or shadow server by copying an existing server's account. Although Kernel is already set up in the copied account, the new server's UCI and volume set ^%ZOSF nodes would need to be updated from their old values to the values needed for the new server. Re-running ^ZTMGRSET allows these values to be updated.

Programmer Tools

Callable Entry Points

• GETENV^%ZOSV

Usage D GETENV^%ZOSV

Input **none**

Output **Y** **String in format: "UCI^VOL/DIR^NODE^BOX LOOKUP"**

Description

Use GETENV^%ZOSV to return information about the current system.

• $$LGR^%ZOSV

Usage S X=$$LGR^%ZOSV

Input **none**

Output **return value:** **String set to the last full global reference.**

Description

Use $$LGR^%ZOSV to return the last global reference.

• $$VERSION^%ZOSV

Usage S X=$$VERSION^%ZOSV[(flag)]

Input **flag** **[optional] If you pass a value of 1, the operating system name is returned instead of the version number. Note that the name is as defined by the vendor and doesn't necessarily correspond with the OS name stored in ^%ZOSF("OS").**

Output **return value:** **Operating system version number or name, depending on optional first parameter.**

Description

Use $$VERSION^%ZOSV to return the operating system version number or name.

• ^%ZOSF Global

The ^%ZOSF global holds operating system-dependent logic so that application programs may be written independently of any specific operating system.

Most of the nodes contain logic that must be executed to return a value, for example X ^%ZOSF("SS"). Those prefaced with one asterisk in the following list, however, are reference values. For example, to write the operating system, use W ^%ZOSF("OS"). The nodes prefaced with two asterisks below should be used with the DO command, as in the following:

```
D @^%ZOSF("ERRTN").
```

****** indicates those nodes that hold reference values.***
******* indicates those nodes that are invoked with a DO statement (D).***

ACTJ — **Return in Y the number of active jobs on the system.**

AVJ — **Return in Y the number of jobs that can be started. The number of available jobs is the maximum number less the number of active jobs.**

BRK — **Allow the user to break the running of a routine.**

DEL — **Delete the routine named in X from the UCI.**

EOFF — **Turn off echo to the $I device.**

EON — **Turn on echo to the $I device.**

EOT — **Return Y=1 if magtape end-of-tape mark is detected.**

****ERRTN** — **This node is set to the name of the routine that should be used to record errors. For most systems this will be the KERNEL error recording routine (%ZTER):**

```
D @^%ZOSF("ERRTN")
```

To initially set the trap,

```
S X=^%ZOSF("ERRTN"),@^%ZOSF("TRAP")
```

ETRP — **Obsolete.**

GD — **Display the global directory.**

JOBPARAM — **When passed the job in X, returns the UCI for that job in Y. It is used to determine whether the job is valid on the system.**

LABOFF **Turn off echo to the IO device.**

LOAD **Load routine X into @(DIE_"XCNP,0)".**

LPC **Returns in Y the longitudinal parity check of the string in X.**

MAGTAPE **Sets the %MT local variable to hold magtape functions. Issue the backspace command as follows:**

```
W @%MT("BS").
```

The full list of functions are:

"BS"	**back space**
"FS"	**forward space**
"WTM"	**write tape mark**
"WB"	**write block**
"REW"	**rewind**
"RB"	**read block**
"REL"	**read label**
"WHL"	**write HDR label**
"WEL"	**write EOF label.**

MAXSIZ **For M/SQL-VAX only: sets the partition size to X.**

***MGR** **Holds the name of the MGR account (UCI, Volume Set).**

MTBOT **Returns Y=1 if the magtape is at BOT.**

MTERR **Returns Y=1 if a magtape error is detected.**

MTONLINE **Returns Y=1 if the magtape is on-line.**

MTWPROT **Returns Y=1 if the magtape is Write Protected.**

NBRK **Don't allow the user to break a routine.**

NO-PASSALL **Sets device $I to interpret tabs, carriage returns, line feeds, or control characters (normal text mode).**

NO-TYPE-AHEAD **Turn off the type-ahead for the device $I.**

***OS** **In the first "^" piece, holds the type of MUMPS, like MSM, VAX DSM, M/SQL-VAX.**

PASSALL **Sets device $I to pass all codes, allow tabs, carriage returns, and other control characters to be passed (binary transfer).**

PRIINQ Returns Y with the current priority of the job.

PRIORITY Sets the priority of the job to X (1 is low, 10 is high).

***PROD** Holds the name of the Production account (UCI, Volume Set).

PROGMODE Returns Y=1 if the user is in Programmer Mode.

RD Displays the routine directory.

RESJOB References the operating system routine for restoring a job.

RM Sets the $I width to X characters. If X=0, then the line in set to no wrap.

RSEL Returns the user's selection of routines as follows:

```
^UTILITY($J,"routine name").
```

RSUM Passed a routine name in X, it returns the checksum in Y. Used by XTSUMBLD. The second line and comments are not included the total.

SAVE Saves the code in @(DIE_"XCN,0)") as routine X.

SIZE Returns Y=size (in bytes) of the current routine.

SS Displays the system status.

TEST Returns $T=1 if routine X exists.

TMK Returns Y=1 if a tape mark was detected on the last read.

TRAP To set the error trap:

```
S X="error routine",@^%ZOSF("TRAP")
```

TRMOFF Resets terminators to normal.

TRMON Turns on all controls as terminators.

TRMRD Returns in Y what terminated the last read.

TYPE-AHEAD Allow type-ahead for the device $I.

UCI Returns Y with the current account (UCI, Volume Set).

UCICHECK	**Returns Y'="" if X is a valid UCI name.**
UPPERCASE	**Converts lowercase to uppercase. Setting X="User Name" returns Y="USER NAME". Applications may gain efficiency by executing this node rather than performing checks within the application program.**
***VOL**	**Contains the current Volume Set (CPU) name.**
XY	**Sets $X=DX and $Y=DY (may not work on all systems).**
ZD	**Given X in $H format, returns the printable form of X in Y.**

Chapter 31 XGF Function Library

The XGF Function Library supports programmers designing text-based applications. The functions in this library support cursor positioning, overlapping text windows, video attribute control, and keyboard escape processing, all in a text-mode environment.

If you intend to make simple interface enhancements for an existing text-mode application, then you may find the XGF Function Library useful. The XGF Function Library provides the following functionality:

- **Text-mode overlapping windows.**
- **Text-mode cursor positioning by screen coordinate.**
- **Text-mode video attribute control (bold, blink, etc.).**
- **Keyboard reader using M escape processing (thereby making use of keystrokes like <UP-ARROW>, <DOWN-ARROW>, <PREV>, <NEXT>, etc.).**

The XGF Function Library may not be appropriate if you need:

- **A full GUI front end for your application.**
- **Support for non-ANSI VT-compatible display devices.**

System Management

To use the XGF Function Library, your system must use an M implementation that complies with the proposed 1995 ANSI M standard. At a minimum, the M implementation must support the following features to use the XGF Function Library:

FEATURE	EXAMPLE
SET into $EXTRACT	`S X="this is a string",$E(X,1,4)="that"`
Reverse $ORDER	`S X=$O(^TMP(""),-1)`
Two argument $GET	`K Y S X=$G(Y,"DEFAULT")`
Skipping parameters	`D TAG^ROUTINE(,P2,,P4)`
$NAME	`W $NA(^TMP($J))`
Set $X and $Y	`S $X=10`

This XGF Function Library supports terminals that are ANSI-compatible and at least VT100-compatible. As a result, this software does not support QUME QVT102/QVT102A terminals.

Programmer Tools

> D ^XGFDEMO: Demo Program

To run an interactive demonstration showing the capabilities provided by the XGF Function Library, you can run the XGF demo program. From the programmer prompt, type the following:

```
>D ^XGFDEMO <RET>
```

■ Functional Division of XGF Function Library

Cursor/Text Output:	**IOXY^XGF, SAY^XGF, SAYU^XGF.**
Keyboard Reader:	**$$READ^XGF.**
Setup/Cleanup:	**CLEAN^XGF, INITKB^XGF, PREP^XGF, RESETKB^XGF.**
Text Window:	**CLEAR^XGF, FRAME^XGF, RESTORE^XGF, SAVE^XGF, WIN^XGF.**
Video Attribute:	**CHGA^XGF, SETA^XGF.**

Callable Entry Points

• CHGA^XGF

Usage `D CHGA^XGF(atr_codes)`

Input

atr_codes:	**B1**	**Blink on**
	B0	**Blink off**
	I1	**Intensity high**
	I0	**Intensity normal**
	R1	**Reverse video on**
	R0	**Reverse video off**
	G1	**Graphics on**
	G0	**Graphics off**
	U1	**Underline on**
	U0	**Underline off**
	E1	**Turn all off**

Output

XGCURATR:	**This variable always holds the current screen attribute coded as a single character, and is updated when you call CHGA^XGF.**
$X,$Y:	**Left unchanged.**

Description

Changes individual video attributes for subsequent screen writes.

Use this entry point to change individual video attributes for subsequent output. This entry point is different from SETA^XGF in that individual video attributes can be set without affecting all video attributes at once.

A call to PREP^XGF must be made at some point prior to calling CHGA^XGF.

The attribute codes are not case sensitive. You can append them if you want to set more than one attribute. If you include more than one attribute, their order is not important.

B0 and B1 turn off and on the blink attribute; I0 and I1 turn off and on the intensity attribute; R0 and R1 turn off and on the reverse attribute; U0 and U1 turn off and on the underline attribute. E1 turns off all attributes. G0 and G1 turn off and on recognition of an alternate graphics character set so that you can use special graphic characters, in particular those set up by Kernel's GSET^%ZISS entry point. To use graphics characters, be sure you turn on graphics first (with G1) and turn graphics off afterwards (with G0).

The change in attribute remains in effect until another CHGA^XGF, PREP^XGF or SETA^XGF CALL is made. If you want only a temporary change in attribute, SAY^XGF may be a better function to use.

For example, to clear the screen in blinking, reverse video and high intensity, do:

```
D CHGA^XGF("R1B1I1"),CLEAR^XGF(0,0,23,79)
```

To print Hello World, do:

```
D CHGA^XGF("I1"),SAY^XGF(,,"Hello ")
D CHGA^XGF("U1"),SAY^XGF(,,"World")
```

To draw the bottom of a small box, do:

```
D CHGA^XGF("G1")
D SAY^XGF(,,IOBLC_IOHL_IOHL_IOBRC)
D CHGA^XGF("G0")
```

See also: SETA^XGF.

• CLEAN^XGF

Usage `D CLEAN^XGF`

Input **none**

Output **none**

Description

Use CLEAN^XGF to exit the XGF screen and keyboard environments. It removes XGF screen and keyboard variables and tables, turns all video attributes off, turns echo on, turns the cursor on, and sets the keypad to numeric mode.

In addition, CLEAN^XGF does everything that the RESETKB^XGF entry point does to exit the XGF keyboard environment, including turning terminators and escape processing off. Subsequent reads are processed normally. If you call CLEAN^XGF, a separate call to RESETKB^XGF is not necessary.

See also: PREP^XGF.

• CLEAR^XGF

Usage `D CLEAR^XGF(top,left,bottom,right)`

Input

top:	**Top screen coordinate for box.**
left:	**Left screen coordinate for box.**
bottom:	**Bottom screen coordinate for box.**
right:	**Right screen coordinate for box.**

Output

$X and $Y:	**Set to the right and bottom specified as parameters.**

Description

Clears a rectangular region of the screen.

This entry point is useful to clear a portion of the screen. The CLEAR function works by printing spaces using the current screen attribute in the specified region. If the screen attribute is changed and then the CLEAR function used, the rectangular region is cleared in the new attribute.

A call to PREP^XGF must be made at some point prior to calling CLEAR^XGF.

Acceptable values for the top and bottom parameters range from 0 to IOSL-1. Acceptable values for the left and right parameters range from 0 to IOM-1.

For example, to clear the entire screen, do:

```
D CLEAR^XGF(0,0,23,79)
```

To clear a rectangular region in the center of the screen, do:

```
D CLEAR^XGF(5,20,15,60)
```

See also: RESTORE^XGF, SAVE^XGF, WIN^XGF.

• FRAME^XGF

Usage `D FRAME^XGF(top,left,bottom,right)`

Input

top:	**Top screen coordinate for box.**
left:	**Left screen coordinate for box.**
bottom:	**Bottom screen coordinate for box.**
right:	**Right screen coordinate for box.**

Output

$X and $Y:	**Set to the right and bottom specified as parameters.**

Description

Draws a box frame on the screen.

Use this entry point to display boxes on the screen. The FRAME function does not clear or otherwise change the region that it encompasses. If you need to open an empty framed window you should use WIN^XGF entry point instead.

A call to PREP^XGF must be made at some point prior to calling FRAME^XGF.

Acceptable values for the top and bottom parameters range from 0 to IOSL-1. Acceptable values for the left and right parameters range from 0 to IOM-1.

For example, to draw a box in the center of the screen, do:

```
D FRAME^XGF(5,20,15,60)
```

See also: RESTORE^XGF, WIN^XGF.

• INITKB^XGF

Usage `D INITKB^XGF([term_str])`

Input

term_str: **[optional] String of characters that should terminate the read.**

This parameter can be one of two forms: A single "*" character turns on all terminators. The other possibility is to pass the string of terminating characters, e.g., $C(9,13,127). If this parameter is not passed, or if it is an empty string, the terminators are not turned on.

Output **none**

Description

Sets up the XGF keyboard environment only. You should call INITKB^XGF once, before you start making calls to the $$READ^XGF function. This entry point turns on escape processing and any terminators that are passed.

Use this entry point only if you are using XGF's Keyboard Reader independently from XGF's screen functions. Otherwise, a call to PREP^XGF does everything to set up keyboard processing that INITKB^XGF does, and a separate call to INITKB^XGF is not necessary.

Unlike PREP^XGF, INITKB^XGF does not set the keypad to application mode.

INITKB *does not call* %ZISS. Thus, documented Kernel variables such as IOKPAM and IOKPNM are not available for use without a separate call to ENS^%ZISS.

See also: RESETKB^XGF.

• IOXY^XGF

Usage `D IOXY^XGF(row,col)`

Input	**row:**	**Row position to move cursor to.**
	col:	**Column position to move cursor to.**
Output	**$X, $Y:**	**Set to the row and column specified as parameters.**

Description

Positions cursor on the screen at a screen coordinate. This entry point is similar to Kernel's X IOXY function. The row parameter must be between 0 and IOSL-1; the column parameter must be between 0 and IOM- 1.

A call to PREP^XGF must be made at some point prior to calling IOXY^XGF.

You can specify row and column parameters relative to the current $X and $Y by specifying "+" or "-" to increment or decrement $X or $Y by 1. You can increment or decrement by more than one if you add a number as well, such as "-5" or "+10". Note that you must use quotes to pass a "+" or "-". Otherwise, to specify exact locations for row and column, pass numbers.

For example, to position the cursor at row 12, column 39, do:

```
D IOXY^XGF(12,39)
```

See also: SAY^XGF, SAYU^XGF.

• PREP^XGF

Usage `D PREP^XGF`

Input **none**

Output **XGCURATR:** **One-character variable containing state of current video attribute.**

Also, Kernel's GSET^%ZISS entry point is called, so all output variables for screen graphics from GSET^%ZISS are defined.

Description

Sets up the XGF screen and keyboard environments.

Before using any XGF screen functions, you must call the PREP^XGF entry point. PREP^XGF sets up screen control variables and tables. It also turns off all video attributes, turns echo off, turns the cursor off, sets the keypad to application mode, and clears the screen.

In addition, PREP^XGF does everything that INITKB^XGF does to set up the XGF keyboard environment, including turning escape processing and terminators on. If you call PREP^XGF, a call to INITKB^XGF would be redundant.

See also: CLEAN^XGF.

• $$READ^XGF

Usage `S ZYXSTR=$$READ^XGF([no_of_char][,timeout])`

Input

no_of_char: **[optional] Maximum # of characters to read.**

timeout: **[optional] Maximum duration of read, in seconds.**

Output

return value: **The string read from the user.**

XGRT: **Set to the mnemonic of the key that terminated the read; see list below or the table in routine XGKB for list of possible values.**

DTOUT: **If defined, signifies that the read timed out.**

Description

$$READ^XGF provides a way to perform reads using escape processing. Reads, when escape processing is turned on, are terminated by <UP-ARROW>, <DOWN-ARROW>, <PREV>, <NEXT>, <TAB>, and other special keystrokes.

$$READ^XGF is a low-level reader compared to the VA FileMan reader. In some respects it is as simple as using the M read command. This read function incorporates escape processing which puts the burden on the operating system to read the arrow, function, and all other keys.

A call to INITKB^XGF or PREP^XGF must be made at some point prior to calling $$READ^XGF.

If the number of characters you request with the first parameter is not entered, the read does not terminate until some terminating character is pressed (or the timeout period is reached).

If you don't pass the timeout parameter, DTIME is used for the timeout period. If the read times out, ^ is returned and DTOUT is left defined.

The list of mnemonics for keys that can terminate reads is:

Key type	Mnemonic
Control	^A, ^B, ^C, ^D, ^E, ^F, ^G, ^H, ^J, ^K, ^L, ^N, ^O, ^P, ^Q, ^R, ^S, ^T, ^U, ^V, ^W, ^X, ^Y, ^Z, ^\, ^], ^6, ^_
Cursor	UP, DOWN, RIGHT, LEFT, PREV, NEXT
Editing	FIND, INSERT, REMOVE, SELECT
Function	F6 to F14, HELP, DO, F17 to F20
Keyboard	TAB, CR
Keypad	KP0 to KP9, KP-, KP+, KP., KPENTER
PF	PF1, PF2, PF3, PF4

For example, to read a name (with a maximum length of 30) from input and display that name on the screen, do:

```
D INITKB^XGF("*")
W "Name: " S NM=$$READ^XGF(30)
D SAY^XGF(10,20,"Hello "_NM)
```

To accept only <Up-Arrow> or <Down-Arrow> keys to exit a routine, do:

```
;Only accept UP or DOWN arrow keys
F  S %=$$READ^XGF(1) Q:XGRT="UP"!(XGRT="DOWN")
```

NOTE: When you set up the XGF keyboard environment using INITKB^XGF rather than PREP^XGF, the keypad is not automatically set to application mode. For reads to be terminated by the keypad keys (<KP0> to <KP9>, <KPENTER>, <KP+>, <KP->, and <KP.>), the keypad must be in application mode. You can put the keypad in application mode by using an M write statement (W IOKPAM to set application mode, IOKPNM to set numeric mode). Take care to preserve the value of $X when using a direct M write, so that relative positioning in XGF cursor/text output calls is not thrown off:

```
S X=$X W IOKPAM S $X=X
```

• RESETKB^XGF

Usage D RESETKB^XGF

Input **none**

Output **none**

Description

Exits the XGF keyboard environment. You should use the RESETKB^XGF call once you finish making calls to the $$READ^XGF function. The RESETKB^XGF entry point turns terminators and escape processing off and removes any XGF keyboard environment variables. Subsequent reads are processed normally.

Use this entry point only if you are using XGF's Keyboard Reader independently from XGF's screen functions. Otherwise, a call to CLEAN^XGF does everything to clean up keyboard processing that RESETKB^XGF does, and a separate call to RESETKB^XGF is not necessary.

Unlike CLEAN^XGF, RESETKB^XGF does not set the keypad to numeric mode.

See also: INITKB^XGF.

• RESTORE^XGF

Usage `D RESTORE^XGF(save_root)`

Input **save_root:** **Global/local array node, closed root form.**

Output **$X and $Y:** **Set to the bottom right coordinate of the restored window.**

Description

Use RESTORE^XGF to restore a previously saved screen region. You can save screen regions using the WIN^XGF and SAVE^XGF entry points. RESTORE^XGF restores the saved screen region in the same screen position as the screen region was saved from.

A call to PREP^XGF must be made at some point prior to calling RESTORE^XGF.

Specify the array node under which to save the overlaid screen region in closed root and fully resolved form: that is, closed right parenthesis and with variable references such as $J fully resolved. Using M $NAME function is a quick way to pass fully resolved node specifications.

For example, to restore the screen contents saved to the local array SELECT to their original position, do:

```
D RESTORE^XGF("SELECT")
```

See also: CLEAR^XGF, SAVE^XGF, WIN^XGF.

• SAVE^XGF

Usage `D SAVE^XGF(top,left,bottom,right,save_root)`

Input **top:** **Top screen coordinate for box.**

left: **Left screen coordinate for box.**

bottom: **Bottom screen coordinate for box.**

right: **Right screen coordinate for box.**

save_root: **Global/local array node, closed root form.**

Output **$X and $Y:** **Left unchanged.**

Description

Use this entry point to save a screen region. In order to save and restore screen regions, you must do all screen output using calls in the XGF Function Library output. If you instead use the M write command for output, the screen contents cannot be saved and restored. Also, a call to PREP^XGF must be made at some point prior to calling SAVE^XGF.

Specify the array node under which to save the overlaid screen region in closed root and fully resolved form: that is, closed right parenthesis and with variable references such as $J fully resolved. Using M $NAME function is a quick way to pass fully resolved node specifications.

For example, to save the screen contents between rows 5 and 15 and columns 20 and 60 in the SELECT local array, do:

```
D SAVE^XGF(5,20,15,60,"SELECT")
```

See also: CLEAR^XGF, RESTORE^XGF, WIN^XGF.

• SAY^XGF

Usage `D SAY^XGF([row],[col],str[,atr])`

Input	**row:**	**[optional] Row position to start write.**
	col:	**String to write.**
	str:	**[optional] Video attribute to write string with.**
	atr:	**See CHGA^XGF for description of atr codes.**
Output	**$X,$Y:**	**Set to position of the last character output.**

Description

Outputs a string to the screen (with optional positioning and attribute control).

Use this entry point rather than the M write command to output strings to the screen. The row and column parameters specify where to print the string. If omitted, the current row and column positions are used. If specified, the row must be between 0 and IOSL-1, and the column must be between 0 and IOM-1.

A call to PREP^XGF must be made at some point prior to calling SAY^XGF.

You can specify row and column parameters relative to the current $X and $Y by specifying "+" or "-" to increment or decrement $X or $Y by 1. You can increment or decrement by more than 1 if you add a number as well, such as "-5" or "+10". Note that you must use quotes to pass a "+" or "-". Otherwise, to specify exact locations for row and column, pass numbers.

Without the fourth argument for video attribute, SAY^XGF displays the string using the current video attribute. With the fourth argument, SAY^XGF displays the string using the attributes you specify. SAY^XGF changes the video attribute only for the output of the string; upon termination of the function, it restores video attributes to their state prior to the function call. For discussion of valid video attribute codes for the video attribute parameter, see the SETA^XGF function under the Video Attribute Calls listing.

For example, to print "Hello, World" in the center of the screen, in the current video attribute, do:

```
D SAY^XGF(11,35,"Hello World")
```

To print "ERROR!" at (row,col) position ($X+1,$Y+5), in reverse and bold video attributes, do:

```
D SAY^XGF("+","+5",0,"ERROR!","R1B1")
```

To print "..." at the current cursor position, in the current video attribute, do:

```
D SAY^XGF(,,"...")
```

See also: IOXY^XGF, SAYU^XGF.

• SAYU^XGF

Usage `D SAYU^XGF([row],[col],str[,atr])`

Input		
	row:	**[optional] Row position to start write.**
	col:	**[optional] Column position to start write.**
	str:	**String to write ("&" underlines next character).**
	atr:	**[optional] Video attribute to write string with (see CHGA^XGF for description of atr codes).**
Output	**$X,$Y:**	**Set to the position of the last character output.**

Description

Outputs a string to the screen (with optional position and attribute control), including the ability to underline an individual character.

This entry point is similar to SAY^XGF. The difference is that the first ampersand ("&") character has a special meaning in the output string; it acts as a flag to indicate that the next character should be underlined. You are only allowed one underlined character per call. Typically you would use SAYU^XGF when writing a menu option's text, in order to underline that option's speed key.

A call to PREP^XGF must be made at some point prior to calling SAYU^XGF.

You can specify row and column parameters relative to the current $X and $Y by specifying "+" or "-" to increment or decrement $X or $Y by 1. You can increment or decrement by more than 1 if you add a number as well, such as "-5" or "+10". Note that you must use quotes to pass a "+" or "-". Otherwise, to specify exact locations for row and column, pass numbers.

If the first ampersand is followed by another ampersand, this initial "&&" is interpreted and displayed as one ampersand character, "&", and you still have the opportunity to use a single ampersand as an underlining flag.

For example, to print Save at row 5, column 10, do:

```
D SAYU^XGF(5,10,"&Save")
```

See also: IOXY^XGF, SAY^XGF.

• SETA^XGF

Usage `D SETA^XGF(atr_code)`

Input	**atr_code:**	**Single character containing the states of all video attributes as the bit values. This argument itself should be derived from a previous call to PREP^XGF, CHGA^XGF, or SETA^XGF.**
Output	**XGCURATR:**	**This variable always holds the current screen attribute coded as a single character, and is updated when you call SETA^XGF.**
	$X,$Y:	**Left unchanged.**

Description

SETA^XGF sets all video attribute simultaneously, for subsequent screen output. This entry point is different from CHGA^XGF in that it takes a different form of the attribute argument, and, unlike CHGA^XGF, sets all attributes. The change in attribute remains in effect until you make another CHGA^XGF, CLEAN^XGF or SETA^XGF call. If you want only a temporary change in attribute, SAY^XGF may be a better function to use.

A call to PREP^XGF must be made at some point prior to calling SETA^XGF.

The value of the attribute parameter uses one bit for the value of each video attribute. The format of the bits is not documented. The current setting of all video attributes is accessible via the variable XGCURATR, however. Rather than trying to use SETA^XGF to control an individual video attribute's setting, you should use it mainly to restore the screen attributes based on a previously saved value of XGCURATR.

For example, to save the initial screen attribute settings to variable SAVEATR, do a function called SOME^THING, and then reset all the video attributes to their initial state, do:

```
D PREP^XGF S SAVEATR=XGCURATR
D SOME^THING
D SETA^XGF(SAVEATR)
```

See also: CHGA^XGF.

• WIN^XGF

Usage `D WIN^XGF(top,left,bottom,right[,save_root])`

Input

top:	**Top screen coordinate for box.**
left:	**Left screen coordinate for box.**
bottom:	**Bottom screen coordinate for box.**
right:	**Right screen coordinate for box.**
save_root:	**[optional] Global/local array node, closed root form.**

Output

save_root:	**If you specify a node as a fifth parameter for save_root, WIN^XGF saves the screen region you overlay in an array at that node.**
$X and $Y:	**Set to the right and bottom coordinates you specify as parameters.**

Description

Use this entry point to open a text window on the screen and optionally remember what it overlays. If the save_root parameter is not passed, you cannot restore the screen behind the window.

In order to save the screen region that the window overlays it is absolutely necessary that screen output is done using only the functions in the XGF Function library. If you use the M write command for output, the screen contents cannot be saved.

A call to PREP^XGF must be made at some point prior to calling WIN^XGF.

Specify the array node under which to save the overlaid screen region in closed root and fully resolved form: that is, closed right parenthesis and with variable references such as $J fully resolved. Using M $NAME function is a quick way to pass fully resolved node specifications.

To restore screens you save with the WIN^XGF function, use the RESTORE^XGF entry point.

For example, to draw an empty box in the center of the screen (and save the underlying screen region under array SELECT), do:

```
D WIN^XGF(5,20,15,60,"SELECT")
```

To save the same window to a global array (to illustrate the use of $NAME to specify a fully resolved root), do:

```
D WIN^XGF(5,20,15,60,$NA(^TMP($J)))
```

See also: CLEAR^XGF, FRAME^XGF, RESTORE^XGF, SAVE^XGF.

Chapter 32 XLF Function Library

The XLF Function Library provides functions supporting date, hyperbolic trigonometric, mathematical, measurement, and string computations, and provides some utility functions as well.

Date Functions—XLFDT

Entry Point **Function**

$$DOW^XLFDT(x[,y]) Day of Week

Return the corresponding day of the week from a date in VA FileMan format.

arguments:
```
x   VA FileMan date.
y   [optional] 1 to return a day-of-week number.
```

examples:
```
W $$DOW^XLFDT(2901231.111523)
     --> Monday
W $$DOW^XLFDT(2901231.111523,1)
     --> 1
```

$$DT^XLFDT() **Return current date in VA FileMan format.**

example:
```
W $$DT^XLFDT                          --> 2921009
```

$$FMADD^XLFDT(x,d,h,m,s) VA FileMan Add

Return result of adding days, hours, minutes, and seconds to a date in VA FileMan format.

arguments:
```
x   VA FileMan date (in quotes).
d   Days.
h   Hours.
m   Minutes.
s   Seconds.
```

example:
```
W $$FMADD^XLFDT("2901231.01",2,2,20,15)
     --> 2910102.032015
```

Date Functions—XLFDT (continued):

Entry Point **Function**

$$FMDIFF(x1,x2[,x3]) **VA FileMan Difference**

Return the difference between two VA FileMan format dates.

arguments:
```
x1  VA FileMan date.
x2  VA FileMan date, to subtract from the x1 date.
x3  [optional] If null ('$D(x3), return the
    difference in days.
    If x3=1, return the difference in days.
    If x3=2, return the difference in seconds.
    If x3=3, return the difference in DD HH:MM:SS format.
```

examples:
```
W $$FMDIFF^XLFDT(2901229,2901231.111523,1)
--> -2
The first date is 2 days less than the second date.

W $$FMDIFF^XLFDT(2901231.111523,2901229.173404,2)
--> 150079
The first date is 150079 seconds greater than the
second date.

W $$FMDIFF^XLFDT(2901231.024703,2901230.012301,3)
--> 1 1:24:2
```

$$FMTE(x[,y]) **VA FileMan to External**

Return conversion to an external format of a date in VA FileMan format.

arguments:
```
x   VA FileMan date
y   [optional] Affects output as follows:

    If null ('$D(y)) return the written-out format.
    If '$D(y) then return standard VA FileMan format.
    If +y = 1 then return standard VA FileMan format.
    If +y = 2 then return MM/DD/YY@HH:MM:SS format.
    If +y = 3 then return DD/MM/YY@HH:MM:SS format.
    If +y = 4 then return YY/MM/DD@HH:MM:SS format.
    If y contains a "D" then Date only.
    If y contains a "F" then output with leading blanks
    If y contains a "P" then output ' HH:MM:SS am/pm'.
    If y contains a "S" then force seconds in the output.
```

Date Functions—XLFDT (continued):

examples:
```
W $$FMTE^XLFDT(2940629.105744,1)
--> Jun 29, 1994@10:57:44
W $$FMTE^XLFDT(2940629.105744,2)
--> 6/29/94@10:57:44
W $$FMTE^XLFDT(2940629.1057,"2S")
--> 6/29/94@10:57:00
W $$FMTE^XLFDT(2940629.1057,"2SF")
--> 6/29/94@10:57:00
W $$FMTE^XLFDT(2940629.1057,"3F")
--> 29/ 6/94@10:57
W $$FMTE^XLFDT(2940629.1057,"4D")
--> 94/6/29
W $$FMTE^XLFDT(2940629.1057,"1P")
--> Jun 29, 1994 10:57 am
W $$FMTE^XLFDT(2940629.1057,"2P")
--> 6/29/94 10:57 am
```

To output a really short date/time try:

```
W $TR($$FMTE^XLFDT(2940629.1057,"4F")," /","0")
--> 940629@10:57
(Convert space to zero and remove slash.)
```

$$FMTH(x[,y]) **VA FileMan to $H**

Return conversion to $H format of a date in VA FileMan format.

arguments:
```
x   VA FileMan date.
y   [optional] 1 to return the date portion only (no
    seconds).
```

examples:
```
W $$FMTH^XLFDT(2901231.111523)
--> 54786,40523
W $$FMTH^XLFDT(2901231.111523,1)
--> 54786
```

$$HADD(x,d,h,m,s) **$H Add**

Return result of adding days, hours, minutes, and seconds to a date in $H format.

arguments:
```
x   $H date (in quotes)
d   Days.
h   Hours.
m   Minutes.
s   Seconds.
```

examples:
```
W $$HADD^XLFDT("54786,3600",2,2,20,15)
--> 54788,12015
```

Date Functions—XLFDT (continued):

Entry Point	Function
$$HDIFF(x1,x2,x3)	**$H Differences**

Return the difference between two $H formatted dates.

arguments:
```
x1   $H date (in quotes)
x2   $H date (in quotes), to subtract from the x1
     date.
x3   See below, FMDIFF(x1,x2,x3) for alternative
     values.
```

examples:
```
W $$HDIFF^XLFDT("54789,40523","54786,25983",1)
--> 3
W $$HDIFF^XLFDT("54789,40523","54786,25983",2)
--> 273740
W $$HDIFF^XLFDT("54789,40523","54786,25983",3)
--> 3 4:02:20
```

$$HTE(x,y) — **$H to External**

Return conversion to an external format of a date in $H format.

arguments:
```
x    $H date (in quotes).
y    See below, FMTE(x,y), for alternate values.
```

examples:
```
W $$HTE^XLFDT("54786,40523")
--> Dec 31, 1990@11:15:23
W $$HTE^XLFDT("54786,40523",2)
--> 2/31/90@11:15:23
```

$$HTFM(x[,y]) — **$H to VA FileMan**

Return conversion to VA FileMan format of a date in $H format.

arguments:
```
x    $H date (in quotes).
y    [optional] 1 to return the date portion only (no
     seconds).
```

examples:
```
W $$HTFM^XLFDT("54786,40523")
--> 2901231.111523
W $$HTFM^XLFDT("54786,40523",1)
--> 2901231
```

Date Functions—XLFDT (continued):

Entry Point **Function**

$$NOW **Return current date/time in VA FileMan format.**

example:
```
W $$NOW^XLFDT
--> 2921009.08425
```

$$SCH(schedule_code,base_date[,force_future])

Return next run-time based on Schedule code.

arguments: schedule_code Interval to add to base_date, as follows:

- **nS** Add n seconds to base_date.
- **nH** Add n hours to base_date.
- **nD** Add n days to base_date.
- **nM** Add n months to base_date.
- **$H;$H;$H** List of $H dates.
- **daycode[@time]** Day of week (see daycode list below).
- **D[@time]** Every weekday.
- **E[@time]** Every weekend day (Saturday,Sunday).
- **nM(entry[,entry[,...]])** Every n months, at each entry in the paramter list (for every n months only), entries can be:
  - dd[@time] Day of month i.e.: 15
  - ndaycode[@time] Nth day of week in month e.g., 1W, 3W.
  - L Last day of month.
  - Ldaycode Last specific DAY in month, e.g., LM,LT,LW...

daycodes (used in schedule codes above)

M Monday	F Friday
T Tuesday	S Saturday
W Wednesday	U Sunday
R Thursday	

base_date VA FileMan date to add interval to.

force_future [optional] if passed with value of 1, forces returned date to be in future, by repeatedly adding interval to base_date until a future date is produced. Otherwise, interval is added once.

examples:
```
W $$SCH^XLFDT("1M(15@2PM,L@6PM)",2931003)
--> 2931015.14
W $$SCH^XLFDT("1M(15@2PM,L@6PM)",2931028)
--> 2931031.18
W $$SCH^XLFDT("3M(L@6PM)",2930931)
--> 2931231.18
```

Hyperbolic Trigonometric Functions—XLFHYPER

The following hyperbolic trigonometric functions provide an additional set of mathematical operations beyond the math functions in XLFMTH.

example:
```
W $$SINH^XLFHYPER(.707)
-->   .767388542
```

Entry Point	Function

NOTE: The optional second parameter, in [ ] brackets, denotes the precision for the function. Precision means the detail of the result, in terms of number of digits.

$$ACOSH(x[,12])	**Hyperbolic arc-cosine.**
$$ACOTH(x[,12])	**Hyperbolic arc-cotangent.**
$$ACSCH(x[,12])	**Hyperbolic arc-cosecant.**
$$ASECH(x[,12])	**Hyperbolic arc-secant.**
$$ASINH(x[,12])	**Hyperbolic arc-sine.**
$$ATANH(x[,12])	**Hyperbolic arc-tangent.**
$$COSH(x[,12])	**Hyperbolic cosine.**
$$COTH(x[,12])	**Hyperbolic cotangent.**
$$CSCH(x[,12])	**Hyperbolic cosecant.**
$$SECH(x[,12])	**Hyperbolic secant.**
$$SINH(x[,12])	**Hyperbolic sine.**
$$TANH(x[,12])	**Hyperbolic tangent.**

Measurement Functions—XLFMSMT

This routine contains entry points to allow conversion between U.S. (English) and Metric units.

Entry Point	Function
$$BSA(ht,wt)	**Body Surface Area Measurement**

Returns body surface area.

arguments:

```
ht    Height in centimeters.
wt    Weight in kilograms.
```

examples:

```
W $$BSA^XLFMSMT(175,86)
--> 1.63

W $$BSA^XLFMSMT($$LENGTH^XLFMSMT(69,"IN","CM"),
 $$WEIGHT^XLFMSMT(180,"LB","KG"))
--> 1.57
```

$$LENGTH(val,from,to) **Length Measurement**

Returns conversion between metric length and U.S. length (either direction). Returns equivalent value with units.

arguments:

```
val     A positive numeric value.
from    Unit of measure of val.
to      Unit of measure to convert val to.
```

example:

```
W $$LENGTH^XLFMSMT(12,"IN","CM")
--> 30.480 CM
```

Valid units in either uppercase or lowercase are:

```
km   = kilometers       mi  =  miles
m    = meters           yd  =  yards
cm   = centimeters      ft  =  feet
mm   = millimeters      in  =  inches
```

Measurement Functions—XLFMSMT (continued):

Entry Point — **Function**

$$TEMP(val,from,to) — **Temperature Measurement**

Converts metric temperature to U.S. temperature. Returns equivalent value with units.

arguments:

```
val     A positive numeric value.
from    Unit of measure of val.
to      Unit of measure to convert val to.
```

example:

```
W $$TEMP^XLFMSMT(72,"F","C")
--> 22.222 C
```

Valid units in either uppercase or lowercase are:

```
f  =    Fahrenheit        c  =  Celsius
```

$$VOLUME(val,from,to) — **Weight Measurement**

Converts metric volume to U.S. volume and vice versa. Converts milliliters to cubic inches or quarts or ounces. Returns equivalent value with units.

arguments:

```
val     A positive numeric value.
from    Unit of measure of val.
to      Unit of measure to convert val to.
```

example:

```
W $$VOLUME^XLFMSMT(12,"CF","ML")
--> 339800.832 ML
```

Valid units in either uppercase or lowercase are:

```
kl   =   kiloliter       cf   =   cubic feet
hl   =   hectoliter      ci   =   cubic inch
dal  =   dekaliter       gal  =   gallon
l    =   liters          qt   =   quart
dl   =   deciliter       pt   =   pint
cl   =   centiliter      c    =   cup
ml   =   milliliter      oz   =   ounce
```

Measurement Functions—XLFMSMT (continued):

Entry Point	Function
$$WEIGHT(val,from,to)	**Weight Measurement**

Returns conversions between metric weights to approximate U.S. weights (in either direction). Returns equivalent value with units.

arguments:

```
val     A positive numeric value.
from    Unit of measure of val.
to      Unit of measure to convert val to.
```

example:

```
W $$WEIGHT^XLFMSMT(12,"LB","G")
--> 5448 G
```

Valid units in either uppercase or lowercase are:

```
t   =  metric tons      tn =  tons
kg  =  kilograms        lb =  pounds
g   =  grams            oz =  ounces
mg  =  milligram        gr =  grain
```

Math Functions—XLFMTH

These calls are provided as an enhancement to what is offered in standard M. In addition, extended math functions provide mathematical operations with adjustable and higher precision. Additional trigonometric functions are available. Angles can be specified either in decimal format or in degrees:minutes:seconds.

Entry Point	Function

NOTE: Each optional parameter, in [ ] brackets, denotes the precision for the function. Precision means the detail of the result, in terms of number of digits.

$$ABS(x) Returns the **absolute value** of the number in x.

example:
```
W $$ABS^XLFMTH(-42.45)
--> 42.45
```

$$ACOS(x[,10]) Returns **Arc-cosine**, with **radians** output.

example:
```
W $$ACOS^XLFMTH(.5)
-->   .636772483
```

$$ACOSDEG(x[,10]) Returns **Arc-cosine**, with **degrees** output.

example:
```
W $$ACOSDEG^XLFMTH(.5)
--> 36.48437577
```

$$ACOT(x[,10]) Returns **Arc-cotangent**, with **radians** output.

example:
```
W $$ACOT^XLFMTH(.5)
-->  1.107148718
```

$$ACOTDEG(x[,10]) Returns **Arc-cotangent**, with **degrees** output.

example:
```
W $$ACOTDEG^XLFMTH(.5)
--> 63.43494882
```

$$ACSC(x[,10]) Returns **Arc-cosecant**, with **radians** output.

example:
```
W $$ACSC^XLFMTH(.5)
-->  1.193008186
```

Math Functions—XLFMTH (continued):

Entry Point	Function

NOTE: Each optional parameter, in [ ] brackets, denotes the precision for the function. Precision means the detail of the result, in terms of number of digits.

$$ACSCDEG(x[,10]) Returns **Arc-cosecant**, with **degrees** output.

example:
```
W $$ACSCDEG^XLFMTH(.5)
--> 68.35433397
```

$$ASEC(x[,10]) Returns **Arc-secant**, with **radians** output.

example:
```
W $$ASEC^XLFMTH(.5)
-->  1.111535102
```

$$ASECDEG(x[,10]) Returns **Arc-secant**, with **degrees** output.

example:
```
W $$ASECDEG^XLFMTH(.5)
--> 63.68627011
```

$$ASIN(x[,10]) Returns **Arc-sine**, with **radians** output.

example:
```
W $$ASIN^XLFMTH(.5)
-->   .523598776
```

$$ASINDEG(x[,10]) Returns **Arc-sine**, with **degrees** output.

example:
```
W $$ASINDEG^XLFMTH(.5)
--> 30
```

$$ATAN(x[,10]) Returns **Arc-tangent**, with **radians** output.

example:
```
W $$ATAN^XLFMTH(.5)
-->   .463647609
```

$$ATANDEG(x[,10]) Returns **Arc-tangent**, with **degrees** output.

example:
```
W $$ATANDEG^XLFMTH(.5)
--> 26.56505118
```

Math Functions—XLFMTH (continued):

Entry Point	Function

NOTE: Each optional parameter, in [ ] brackets, denotes the precision for the function. Precision means the detail of the result, in terms of number of digits.

$$COS(x[,10]) Returns **Cosine**, with **radians** input.

example:
```
W $$COS^XLFMTH(1.5)
-->   .070737202
```

$$COSDEG(x[,10]) Returns **Cosine**, with **degrees** input.

example:
```
W $$COSDEG^XLFMTH(45)
-->   .707106781
```

$$COT(x[,10]) Returns **Cotangent**, with **radians** input.

example:
```
W $$COT^XLFMTH(1.5)
-->   .070914844
```

$$COTDEG(x,[10]) Returns **Cotangent**, with **degrees** input.

example:
```
W $$COTDEG^XLFMTH(45)
-->  1
```

$$CSC(x[,10]) Returns **Cosecant**, with **radians** input.

example:
```
W $$CSC^XLFMTH(1.5)
-->  1.002511304
```

$$CSCDEG(x[,10]) Returns **Cosecant**, with **degrees** input.

example:
```
W $$CSCDEG^XLFMTH(45)
-->  1.414213562
```

$$DECDMS(x[,5]) Returns conversion to **Degrees:minutes:seconds** from **Decimal.**

example:
```
W $$DECDMS^XLFMTH(30.7)
--> 30:42:0
```

Math Functions—XLFMTH (continued):

Entry Point	Function

NOTE: Each optional parameter, in [ ] brackets, denotes the precision for the function. Precision means the detail of the result, in terms of number of digits.

$$DMSDEC(x[,12]) Returns conversion to **Decimal** from **Degrees:minutes:seconds.**

example:
```
W $$DMSDEC^XLFMTH("30:42:0")
--> 30.7
```

$$DTR(x[,12]) Returns conversion to **radians** from **degrees**.

example:
```
W $$DTR^XLFMTH(45)
-->   .7853981634
```

$$E([12]) Returns **e**.

example:
```
W $$E^XLFMTH(12)
-->  2.71828182846
```

$$EXP(x[,11]) Return **e** to the x power (**exponent)**.

example:
```
W $$EXP^XLFMTH(1.532)
-->  4.6274224185
```

$$LN(x[,11]) Returns the **natural log** of x (base e).

example:
```
W $$LN^XLFMTH(4.627426)
-->  1.532000774
```

$$LOG(x[,11]) Returns the **logarithm** (base 10) of x.

example:
```
W $$LOG^XLFMTH(3.1415)
-->   .4971370641
```

$$MAX(x,y) Returns the **maximum** value comparing the number in x with the number in y.

example:
```
W $$MAX^XLFMTH(53,24)
--> 53
```

Math Functions—XLFMTH (continued):

Entry Point	Function

NOTE: Each optional parameter, in [ ] brackets, denotes the precision for the function. Precision means the detail of the result, in terms of number of digits.

$$MIN(x,y) Returns the **minimum** value comparing the number in x with the number in y.

example:
```
W $$MIN^XLFMTH(53,24)
--> 24
```

$$PI([12]) Returns **pi**.

example:
```
W $$PI^XLFMTH(12)
-->  3.14159265359
```

$$PWR(x,y[,11]) Returns x to the y **power**. This function makes use of LN and EXP.

example:
```
W $$PWR^XLFMTH(3.2,1.5)
-->  3.6600922278
```

$$RTD(x[,12]) Returns conversion to **degrees** from **radians**.

example:
```
W $$RTD^XLFMTH(1.5)
--> 85.9436692696
```

$$SEC(x[,10]) Returns **Secant**, with **radians** input.

example:
```
W $$SEC^XLFMTH(1.5)
--> 14.1368329
```

$$SECDEG(x[,10]) Returns **Secant**, with **degrees** input.

example:
```
W $$SECDEG^XLFMTH(45)
-->  1.414213562
```

$$SIN(x[,10]) Returns **Sine**, with **radians** input.

example:
```
W $$SIN^XLFMTH(.7853982)
-->   .707106807
```

Math Functions—XLFMTH (continued):

Entry Point	Function

NOTE: Each optional parameter, in [ ] brackets, denotes the precision for the function. Precision means the detail of the result, in terms of number of digits.

$$SINDEG(x[,10]) Returns **Sine**, with **degrees** input.

example:
```
W $$SINDEG^XLFMTH(45)
-->   .707106781
```

$$SQRT(x[,12]) Returns the **square root** of x.

example:
```
W $$SQRT^XLFMTH(153)
--> 12.3693168769
```

$$TAN(x[,10]) Returns the **tangent** of x (tan x = sin X/cos X) , with **radians** input.

example:
```
W $$TAN^XLFMTH(.7853982)
-->  1.000000073
```

$$TANDEG(x[,10]) Returns **Tangent**, with **degrees** input.

example:
```
W $$TANDEG^XLFMTH(45)
-->  1
```

String Functions—XLFSTR

These functions are provided to help process strings.

Entry Point	Function

$$CJ(s,i[,p]) Return a **Center Justified** character string.

arguments:
```
s   Character string.
i   Field size.
p   [optional] Pad character.
```

example:
```
W "[",$$CJ^XLFSTR("SUE",10),"]"
--> [  "SUE"   ]
W "[",$$CJ^XLFSTR("SUE",10,"-"),"]"
--> [---SUE----]
```

$$INVERT(x) Return an **Inverted** string.

Inverts the order of characters in a string.

arguments:
```
x   Character string.
```

example:
```
W $$INVERT^XLFSTR("ABC")
-->  "CBA"
```

$$LJ(s,i[,p]) Return a **Left Justified** character string.

arguments:
```
s   Character string.
i   Field size.
p   [optional] Pad character.
```

example:
```
W "[",$$LJ^XLFSTR("DON",10),"]"
--> [DON       ]
W "[",$$LJ^XLFSTR("DON",10,"-"),"]"
--> [DON-------]
```

$$LOW(x) **Lowercase**

Return a string converted to all lowercase letters.

arguments:
```
x   Character string.
```

example:
```
W $$LOW^XLFSTR("JUSTICE")
-->  "justice"
```

String Functions—XLFSTR (continued):

Entry Point	Function
$$REPEAT(x,y)	**Repeat a String**

Return a string that repeats the value of x for y number of times.

arguments:
```
x   Character string.
y   Number of times to repeat the string in x.
```

example:
```
W $$REPEAT^XLFSTR("-",10)
 --> "----------"
```

$$REPLACE(in,.spec) Replace Strings

Uses a multi-character $TRanslate to return a string with the specified string replaced.

arguments:
```
in    Input string.
spec  An array passed by reference.
```

example:
```
SET spec("aa")="a",spec("pqr")="alabama"
W $$REPLACE^XLFSTR("aaaaaaapqraaaaaaa",.spec)
--> "aaaaalabamaaaaa"

SET spec("F")="File",spec("M")="Man"
W $$REPLACE^XLFSTR("FM",.spec)
--> "FileMan"
```

$$RJ(s,i[,p]) Return a **Right Justified** character string.

arguments:
```
s     Character string.
i     Field size.
p     [optional] Pad character.
```

example:
```
W "[",$$RJ^XLFSTR("SAM",10),"]"
--> [       SAM]
W "[",$$RJ^XLFSTR("SAM",10,"-"),"]"
--> [-------SAM]
```

String Functions—XLFSTR (continued):

Entry Point	Function
$$STRIP(x,y)	**Strip a String**

Return a string stripped of all instances of a character.

arguments:
```
x   Character string.
y   The character to strip out.
```

example:
```
W $$STRIP^XLFSTR("hello","e")
--> "hllo"
```

$$UP(x) **Uppercase**

Return a string converted to all uppercase letters.

arguments:
```
x   Character string.
```

example:
```
W $$UP^XLFSTR("freedom")
--> "FREEDOM"
```

Utility Functions—XLFUTL

These functions are provided to help with a variety of tasks.

Entry Point	Function
$$BASE(n,from,to)	**Converts a number from one base to another. The base must be between 2 and 16, both from and to.**
arguments:	n Number to convert. from Base of number being converted. to Base to convert number to.
example:	W $$BASE^XLFUTL(1111,2,16) **-->** F
$$CCD(number)	**Returns a number appended with a computed check digit. To check if original number corresponds with appended check digit, use VCD^XLFUTL function.**
arguments:	number Number to compute check digit for.
example:	W $$CCD^XLFUTL(99889) **-->** 998898
$$CNV(n,base)	**Converts a number from base 10 to another base (which must be between 2 and 16).**
arguments:	n Number to convert. base Base to convert number to.
example:	W $$CNV^XLFUTL(15,2) **-->** 1111
$$DEC(n,base)	**Converts a number from a specified base (must be between 2 and 16) to base 10.**
arguments:	n Number to convert. base Base of number being converted.
example:	W $$DEC^XLFUTL("FF",16) **-->** 255
$$VCD(number)	**Verifies integrity of number with appended check digit (check digit must be appended by the CCD^XLFUTL function).**
arguments:	number Number to verify, including appended check digit. Returns 1 if number corresponds to check digit, 0 if number does not.
example:	W $$VCD^XLFUTL(998898),", ",$$VCD^XLFUTL(998878) **-->** 1, 0

Other Functions

• ^XUWORKDY: Workday Calculation

NOTE: The XUWORKDY routine is maintained for code that might still use it.

Usage D ^XUWORKDY

Input	**X**	**Starting date in the YYYMMDD format (e.g., 2850420).**
	X1	**Ending date in the YYYMMDD format (e.g., 2850707).**
Output	**X**	**The number of workdays in the interval.**

Description

To use the ^XUWORKDY routine, you must make sure, on the system that it is being used on, that Kernel's HOLIDAY file is populated with each year's holidays for the workday calculation to work correctly. If it is not populated, you need to populate it yourself (Kernel distributes this file without data). Only enter holidays that fall on weekdays, however.

You can call the ^XUWORKDY routine to calculate the number of workdays between two dates (X, X1) and return the answer in X. If either date is imprecisely specified, or if the HOLIDAY global is empty, then ^XUWORKDY returns X as the null string. Otherwise, the returned value is positive if X<X1 and negative if X>X1.

The first FOR loop in ^XUWORKDY checks the HOLIDAY global and sets %H equal to the number of holidays between the two dates. It is assumed that the HOLIDAY global contains only weekday holidays.

The second FOR loop (F %J=%J:1 ...) steps forward from the earliest date and stops at the first Sunday or at the ending date (whichever comes first) counting the number of workdays.

The third FOR loop (F %K=%K:-1 ...) steps backward from the latest date and stops at the first Sunday or at the beginning date (whichever comes first), counting the workdays.

Then %I is set equal to the number of days between the two Sundays.

Finally, X is set equal to the total counted days minus the number of weekend days between the two Sundays (-(%I\7*2)).

• $$EN^XUA4A71: Convert String to Soundex

Usage `S X=$$EN^XUA4A71(string)`

Input	**string:**	**String to convert into soundex form.**
Output	**return value:**	**Soundex version of the string.**

Description

Use this function to convert a string into a numeric representation of the string, using soundex methods. Soundex represents the phonetic properties of a string; its chief feature is that it assigns similar strings the same soundex representation.

Appendices

Appendix A KIDS Build Checklists

KIDS Build Checklist (Page 1 of 6)

Package: _____

Files to Send:

	DD Issues				**Data (only if sending a full DD):**			
Files	**Send full or partial DD? (F/P)**	**Update DD? (Y/N)**	**Send security code? (Y/N)**	**Screen to determine DD update?**	**Site data (Over-write, Merge, Replace, Add)?**	**Resolve Pointers ?(Y/N)**	**User over-ride data update? (Y/N)**	**Use a template or screening logic to specify data subset?**

KIDS Build Checklist (Page 2 of 6)

Build Components to Send:

Build Component Type	Names	Action (Send, Delete, Merge, Link):
Print templates		
Sort templates		
Input templates		
Forms		
Functions		
Dialogs		
Bulletins		
Help frames		
Routines		
Options		
Security keys		
Protocols		
Window Objects		

KIDS Build Checklist (Page 3 of 6)

Package File Link (optional):

Will you be updating the Package File? ____Yes _____No
If yes, please answer the following questions:

Package File Entry Name	
Will this package be tracked nationally?	**___Yes ___ No**
Alpha/beta testing?	**___Yes ___ No**
Describe the enhancements in this version:	

Environment Check:

Information you need to check for	M code to perform check

KIDS Build Checklist (Page 4 of 6)

Installation Questions:

The question subscript must begin with "PRE" or "POS". "PRE" indicates a question for the pre-install; "POS" indicates a question for the post-install.

Question Subscript	Question Type: DIR(0)	Question Prompt: DIR("A")	Question Default: DIR("B")	M code

KIDS Build Checklist (Page 5 of 6)

Pre-Installation Worksheet:

Task	Checkpoint Call Back (Tag^Routine Reference)	Parameter Used by Checkpoint	Checkpoint Name

KIDS Build Checklist (Page 6 of 6)

Post-Installation Worksheet:

Task	Checkpoint Call Back (Tag^Routine Reference)	Parameter Used by Checkpoint	Checkpoint Name

Glossary

Access Code	**A password used along with the verify code to provide secure user access. It is used by the Kernel's Sign-on/Security system to identify the user.**
ADPAC	**Automated Data Processing (ADP) Application Coordinator (see Application Coordinator, below).**
Alerts	**Brief on-line notices that are issued to users as they complete a cycle through the menu system. Alerts are designed to provide interactive notification of pending computing activities, such as the need to reorder supplies or review a patient's clinical test results. Along with the alert message is an indication that the View Alerts common option should be chosen to take further action.**
ANSI	**American National Standards Institute.**
ANSI M	**An implementation of the M computer language that conforms to ANSI standards.**
Application Coordinator	**Designated individuals responsible for user-level management and maintenance of an application package such as IFCAP or Lab. Also abbreviated as ADPAC (ADP Application Coordinator).**
Application Package	**In DHCP, software and documentation that support the automation of a service, such as Laboratory or Pharmacy within VA medical centers (see Package).**
Application Programmer	**The person who writes code for application packages. The Kernel provides tools to facilitate package development.**
Application Programming Interface (API)	**Programmer calls provided by the Kernel for use by application programmers. APIs allow programmers to carry out standard computing activities without needing to duplicate Kernel utilities in their own packages. APIs also further DBA goals of system integration by channeling activities, such as adding new users, through a limited number of callable entry points.**
Array	**An arrangement of elements in one or more dimensions. A MUMPS array is a set of nodes referenced by subscripts that share the same variable name.**

ASCII	**American Standard Code for Information Interchange. A series of 128 characters, including upper and lower case alpha characters, numbers, punctuation, special symbols, and control characters.**
Audit Access	**A user's authorization to mark the information stored in a computer file to be audited.**
Auditing	**Monitoring computer usage such as changes to the database and other user activity. Audit data can be logged in a number of VA FileMan and Kernel files.**
Auto-menu	**An indication to Menu Manager that the current user's menu items should be displayed automatically. When auto-menu is not in effect, the user must enter a question mark at the menu's select prompt to see the list of menu items.**
Backup	**The process of creating duplicate data files and program copies or both as a reserve in case the original is lost or damaged.**
Bug	**An error in a program. Bugs may be caused by syntax errors, logic errors, or a combination of both.**
Bulletins	**Electronic mail messages that are automatically delivered by MailMan under certain conditions. For example, a bulletin can be set up to fire when database changes occur, such as adding a record to the file of users. Bulletins are fired by bulletin-type cross references.**
Callable Entry Point	**An authorized programmer call that may be used in any DHCP application package. The DBA maintains the list of DBIC-approved entry points.**
Capacity Management	**The process of assessing a system's capacity and evaluating its efficiency relative to workload in an attempt to optimize system performance. The Kernel provides several utilities.**
Caret	**A symbol expressed as ^ (caret). In many M systems, a caret is used as an exiting tool from an option. Also known as the up-arrow symbol.**
Checksum	**A numeric value that is the result of a mathematical computation involving the characters of a routine or file.**

Cipher	**A system that arbitrarily represents each character as one or more other characters. See also encryption.**
Command	**A combination of characters that instruct the computer to perform a specific operation.**
Common Menu	**Options that are available to all users. Entering two question marks at the menu's select prompt will display any secondary menu options available to the signed-on user along with the common options available to all users.**
Compiled Menu System (^XUTL global)	**Job-specific information that is kept on each CPU so that it is readily available during the user's session. It is stored in the ^XUTL global, which is maintained by the menu system to hold commonly referenced information. The user's place within the menu trees is stored, for example, to enable navigation via menu jumping.**
Computed Field	**This field takes data from other fields and performs a predetermined mathematical function (e.g., adding two columns together). You will not, however, see the results of the mathematical function on the screen. Only when you are printing or displaying information on the screen will you see the results for this type of field.**
Control Key	**The Control Key (Ctrl on the keyboard) performs a specific function in conjunction with another key. On some systems, for example, Ctrl-S causes printing on the terminal screen to stop, while Ctrl-Q restarts printing on the terminal screen.**
CORE	**The fundamental clinical application packages of the DHCP.**
CPU	**Central Processing Unit. Those parts of computer hardware that carry out arithmetic and logic operations, control the sequence of operations performed, and contain the stored program of instructions.**
Cross Reference	**An indexing method whereby files can include pre-sorted lists of entries as part of the stored database. Cross references (x-refs) facilitate look-up and reporting.**
CRT	**An acronym for cathode ray tube, the basis of the television screen and the standard microcomputer display screen. See also Terminal, Monitor, VDT.**

Data Attribute	**A characteristic of a unit of data such as length, value, or method of representation. VA FileMan field definitions specify data attributes.**
Data Dictionary	**Definition of the structure of a VA FileMan file, its attribute fields, and its relationships with other files.**
Data Dictionary Access	**A DHCP user's authorization to write/update/edit the data format for a computer file. Also known as DD Access.**
Database	**A set of data, consisting of at least one file, that is sufficient for a given purpose. The DHCP database is composed of a number of VA FileMan files.**
DBA	**Database Administrator. In DHCP, the person who monitors namespacing conventions and other procedures that enable various DHCP packages to coexist within an integrated database system.**
DBIA	**Database Integration Agreement. The DBA maintains a list of DBIAs or mutual agreements between package developers allowing the use of internal entry points or other package-specific features that are not available to the general programming public.**
DBIC	**Database Integration Committee. Within the purview of the DBA, the committee maintains a list of DBIC-approved callable entry points and publishes the list on FORUM for reference by application programmers and verifiers.**
Debug	**To correct logic errors or syntax errors or both in a computer program. To remove errors from a program.**
Default Response	**A response considered the most probable answer to the prompt. In DHCP, a default response is identified by double slash marks (//) immediately following it. This allows you the option of accepting the default answer or entering your own answer. To accept the default you simply press the enter (or return) key. To change the default answer, type in your response.**

Delete	**A key on your keyboard that allows you to delete characters. In DHCP, the @ sign (uppercase of the 2 key) may also be used to delete an entire response in a field. The computer will ask "Are you sure you want to delete this entry?" to insure you do not delete an entry by mistake.**
Delete Access	**A user's authorization to remove information stored in a computer file.**
Device	**Terminals, printers, modems and other types of peripheral equipment associated with a computer. An operating system file like the ones found in the VAX computer system may also be considered a device for input/output.**
Device Handler	**The Kernel module that provides a mechanism for accessing peripherals and using them in controlled ways (e.g., user access to printers or other output devices).**
DHCP	**The Decentralized Hospital Computer Program of the Veterans Health Administration (VHA), Department of Veterans Affairs (VA). DHCP application packages, developed within VA, are used to support clinical and administrative functions at VA medical centers nationwide.**
DIFROM	**VA FileMan utility that gathers all package components and changes them into routines (namespaceI* routines) so that they can be exported and installed in another VA FileMan environment.**
Direct Mode Utility	**A programmer call that is made when working in direct programmer mode. A direct mode utility is entered at the MUMPS prompt (e.g., >D ^XUP). Calls that are documented as direct mode utilities *cannot* be used in application package code.**
Double Quote (")	**A symbol used in front of a Common option's menu text or synonym to select it from the Common menu. For example, the five character string "TBOX selects the User's Toolbox Common option.**

DR String	**The set of characters used to define the variable DR when calling VA FileMan. Since a series of parameters may be included within quotes as a literal string, the variable's definition is often called the DR string. To define the fields within an edit sequence, for example, the programmer may specify the fields using a DR string rather than an input template.**
DUZ	**A local variable holding the user number that identifies the signed-on user.**
DUZ(0)	**A local variable that holds the File Manager Access Code of the signed-on user.**
Electronic Signature Code	**A secret password that some users may need in order to sign documents via the computer.**
Encryption	**Scrambling data or messages with a cipher or code so that they are unreadable without a secret key. In some cases encryption algorithms are one directional, that is, they only encode and the resulting data cannot be unscrambled (e.g., access/verify codes).**
Entry	**A VA FileMan record. It is uniquely identified by an internal entry number (the .001 field) in a file.**
Error Trap	**A mechanism to capture system errors and record facts about the computing context such as the local symbol table, last global reference, and routine in use. Operating systems provide tools such as the %ER utility. The Kernel provides a generic error trapping mechanism with use of the ^%ZTER global and ^XTER* routines. Errors can be trapped and, when possible, the user is returned to the menu system.**
Field	**A field is similar to blanks on forms. It is preceded by words that tell you what information goes in that particular field. The blank, marked by the cursor on your terminal screen, is where you enter the information. A reserved area in a record used for storage of specific information.**
File	**A set of related records treated as a unit. VA FileMan files maintain a count of the number of entries or records.**

File Access Security system — Formerly known as Part 3 of the Kernel Inits. If the File Access Security conversion has been run, file-level security for VA FileMan files is controlled by Kernel's File Access Security system, not by VA FileMan access codes.

File Manager — See VA FileMan.

Forced Queuing — A device attribute indicating that the device can only accept queued tasks. If a job is sent for foreground processing, the device will reject it and prompt the user to queue the task instead.

Form — See ScreenMan Forms.

FORUM — The central E-mail system within DHCP. It is used by developers to communicate at a national level about programming and other issues. FORUM is located at the Washington, DC ISC (162-2).

Free Text — A type of data field whose permissible values are any combination of numbers, letters, and symbols.

Go-home Jump — A menu jump that returns the user to the Primary menu presented at sign-on. It is specified by entering two up-arrows (^^) at the menu's select prompt. It resembles the rubber band jump but without an option specification after the up-arrows.

Help Frames — Entries in the HELP FRAME file that may be distributed with application packages to provide on-line documentation. Frames may be linked with other related frames to form a nested structure.

Help Processor — A Kernel module that provides a system for creating and displaying on-line documentation. It is integrated within the menu system so that help frames associated with options can be displayed with a standard query at the menu's select prompt.

Help Prompt — Computer assistance available to you at your terminal screen. The Help function assists you with menus and describes options so you can make the proper choice. To get "help" in DHCP, enter 1 to 4 question marks in response to a prompt. The level of help you get increases with the number of question marks you enter.

Hook or Link — Non-specific terms referring to ways in which files may be related (via pointer links) or can be accessed (via hooks).

Host File Server (HFS)	**A procedure available on layered systems whereby a file on the host system can be identified to receive output. It is implemented by the Device Handler's HFS device type.**
Hunt Group	**An attribute of an entry in the DEVICE file that allows several devices to be used interchangeably; useful for sending network mail or printing reports. If the first hunt group member is busy, another member may stand in as a substitute.**
IDCU	**Integrated Data Communications Utility; the telecommunications network used to interconnect computers among VA facilities.**
Index (%INDEX)	**A Kernel utility used to verify routines and other MUMPS code associated with a package. Checking is done according to current ANSI MUMPS standards and DHCP programming standards (see SAC). This tool can be invoked through an option or from direct mode (>D ^%INDEX).**
Init	**Initialization of an application package. INIT* routines are built by VA FileMan's DIFROM and, when run, recreate a set of files and other package components.**
Internal Entry Number (IEN)	**The number used to identify an entry within a file. Every record has a unique internal entry number.**
IRM	**Information Resource Management. A service at VA medical centers responsible for computer management and system security.**
ISC	**Information Systems Center.**
ISO	**Information Security Officer. Person responsible for information security at each VA Medical Center. Works in conjunction with Regional Security Officers (RISOs).**
Jump	**In DHCP applications, the Jump command allows you to go from a particular field within an option to another field within that same option. You may also Jump from one menu option to another menu option without having to respond to all the prompts in between. To jump, type an up-arrow (^) -- which is your shift key plus the 6 key -- and then type the name of the field or option you wish to jump to. See also Go-home, Phantom, Rubber Band, or Up-arrow jump.**

Jump Start	**A logon procedure whereby the user enters the "access code;verify code;option" to go immediately to the target option, indicated by its menu text or synonym. The jump syntax can be used to reach an option within the menu trees by entering "access;verify;^option".**
Kermit	**A standard file transfer protocol. It is supported by the Kernel and can be set up as an alternate editor.**
Kernel	**The DHCP package that enables DHCP application packages to coexist in a standard operating-system-independent computing environment.**
Laygo Access	**A DHCP user's authorization to create a new entry when editing a computer file. (Learn As You GO, the ability to create new entries).**
Link or Hook	**Non-specific terms referring to ways in which files may be related (via pointer links) or can be accessed (via hooks).**
Logon	**The process of gaining access to a computer system.**
Logoff	**The process of exiting from a computer system.**
M	**A programming language recognized by the American National Standards Institute. Alternately know as MUMPS; the acronym MUMPS stands for Massachusetts General Hospital Utility Multiprogramming System.**
Mail Message	**An entry in the MESSAGE file. The DHCP electronic mail system (MailMan) supports local and remote networking of messages.**
MailMan	**The Kernel module that provides a mechanism for handling electronic communication, whether it's user-oriented mail messages, automatic firing of bulletins, or initiation of server-handled data transmissions.**
Manager Account	**A UCI that can be referenced by non-manager accounts such as production accounts. Like a library, the MGR UCI holds percent routines and globals (e.g., ^%ZOSF) for shared use by other UCIs.**
MAS	**Medical Administration Service.**
Menu	**A list of choices for computing activity. A menu is a type of option designed to identify a series of items (other options) for presentation to the user for selection.**

Menu Cycle — The process of first visiting a menu option by picking it from a menu's list of choices and then returning to the menu's select prompt. Menu Manager keeps track of information, such as the user's place in the menu trees, according to the completion of a cycle through the menu system.

Menu Manager — The Kernel module that controls the presentation of user activities such as menu choices or options. Information about each user's menu choices is stored in the Compiled Menu System, the ^XUTL global, for easy and efficient access.

Menu System — The overall Menu Manager logic as it functions within the Kernel framework.

Menu Template — An association of options as pathway specifications to reach one or more final destination options. The final options must be executable activities and not merely menus for the template to function. Any user may define user-specific menu templates via the corresponding Common option.

Menu Text — The descriptive words that appear when a list of option choices is displayed. Specifically, the Menu Text field of the OPTION file. For example, User's Toolbox is the menu text of the XUSERTOOLS option. The option's synonym is TBOX.

Menu Trees — The menu system's hierarchical tree-like structures that can be traversed or navigated, like pathways, to give users easy access to various options.

MIRMO — Medical Information Resources Management Office.

MIS — Management Information System.

Modem — A device for connecting a terminal to a telephone line, allowing it to communicate with another modem.

Monitor — The device on which images generated by the computer are displayed. The term usually refers to a video display and its housing. See also CRT, VDT, Terminal.

Multiple — A multiple-valued field; a subfile. In many respects, a multiple is structured like a file.

MUMPS — See M.

Namespacing	**The convention of using a unique 2-4 character prefix for package components like options and routines. The DBA assigns unique character strings for package developers to use in naming routines, options, and other package elements so that packages may coexist. Namespacing includes "number spacing" whereby the files of a package stay within a pre-defined range of numbers.**
Node	**In a tree structure, a point at which subordinate items of data originate. A MUMPS array element is characterized by a name and a unique subscript. Thus, the terms node, array element, and subscripted variable are synonymous. In a global array, each node might have specific fields or "pieces" reserved for data attributes.**
Numeric Field	**A data field whose permissible values are limited to numeric characters of a restricted number of digits.**
Online	**A device is online when it is connected and capable of responding to the computer.**
Operating System	**A basic program that runs on the computer, controls the peripherals, allocates computing time to each user, and communicates with terminals.**
Option	**An entry in the OPTION file. As an item on a menu, an option provides an opportunity for users to select it thereby invoking the associated computing activity. Options may also be scheduled to run in the background, non-interactively, by TaskMan.**
Option Name	**The NAME field in the OPTION file. For example, XUMAINT for the option that has the menu text "Menu Management". Options are namespaced according to DHCP conventions monitored by the DBA.**
PAC	**Programmer Access Code. An optional user attribute that may function as a second level password into programmer mode.**
Package	**The set of programs, files, documentation, help prompts, and installation procedures required for a given software application. For example, Laboratory, Pharmacy, and MAS are packages.**
Part 3 of the Kernel Init	**See File Access Security system.**

Password	**A user's confidential sequence of keyboard characters, which must be entered at the beginning of each computer session to provide the user's identity.**
Patch	**An update to a package. Patches can include code updates, documentation updates, and information updates. Patches are applied to the programs on your DHCP system by IRM Service.**
Pattern Match	**A preset formula used to test strings of data. Refer to your system's M Language Manuals for information on Pattern Match operations.**
Peripheral Device	**Any hardware device other than the computer itself (central processing unit plus internal memory). Typical examples include card readers, printers, CRT units, and disk drives.**
Phantom Jump	**Menu jumping in the background. Used by the menu system to check menu pathway restrictions.**
Pointer	**Allows entries in one VA FileMan file to be the field values of another file; this is accomplished by use of a pointer field.**
Primary Menus	**The list of options presented at sign-on. Each user must have a primary menu in order to sign-on and reach Menu Manager. Users are given primary menus by IRM. This menu should include most of the computing activities the user will need.**
Production Account	**The UCI where users log on and carry out their work, as opposed to the manager, or library, account.**
Programmer Access	**Privilege to become a programmer on the system and work outside many of the security controls of Kernel. Accessing programmer mode from Kernel's menus requires having the programmer's at-sign security code, which sets the variable DUZ(0)=@.**
Prompt	**A question or message issued interactively and requiring a response.**
Protocol	**An entry in the PROTOCOL file. Used by the Order Entry/Results Reporting (OE/RR) package to support the ordering of medical tests and other activities. The Kernel includes several protocol-type options for enhanced menu displays within the OE/RR package.**

Queuing	**Requesting that a job be processed in the background rather than in the foreground within the current session. The Kernel's Task Manager handles the queuing of tasks.**
Queuing Required	**An option attribute that specifies that the option must be processed by TaskMan (the option can only be queued). The option may be invoked and the job prepared for processing, but the output can only be generated during the specified time periods.**
Read Access	**A user's authorization to read information stored in a computer file.**
Record	**A set of related data treated as a unit. An entry in a VA FileMan file constitutes a record.**
Required Field	**A mandatory field, one that must not be left blank. The prompt for such a field will be repeated until the user enters a valid response.**
Resource	**A method that enables sequential processing of tasks. The processing is accomplished with a RES device type designed by the application programmer and implemented by IRM. The process is controlled via the RESOURCE file.**
Return	**On the computer keyboard, the key located where the carriage return is on an electric typewriter. It is used in DHCP to terminate "reads". Symbolized by <RET>.**
RISO	**Regional Information Security Officer. Regional representative of VA Medical Center Information Security Officers (ISOs).**
Routine	**A program or sequence of computer instructions that may have some general or frequent use. M routines are groups of program lines that are saved, loaded, and called as a single unit via a specific name.**
Rubber Band Jump	**A menu jump used to go out to an option and then return, in a bouncing motion. The syntax of the jump is two up-arrows followed by an option's menu text or synonym (e.g., ^^Print Option File). If the two up-arrows are not followed by an option specification, the user is returned to the primary menu (see Go-home Jump).**

SAC	**Standards and Conventions (maintained by the SACC, setting guidelines to be followed by DHCP application programmers).**
SACC	**Standards and Conventions Committee of DHCP. This committee is responsible for maintaining the SAC.**
Scheduling Options	**A way of ordering TaskMan to run an option at a designated time with a specified rescheduling frequency, such as once per week.**
ScreenMan Forms	**A screen-oriented display of fields, for editing or simply for reading. VA FileMan's Screen Manager is used to create forms that are stored in the FORM file and exported with a package. Forms are composed of blocks (stored in the BLOCK file) and can be regular, full screen pages or smaller, pop-up pages for multiples.**
Scroll/No Scroll	**The Scroll/No Scroll button (also called Hold Screen) allows the user to "stop" (No Scroll) the terminal screen when large amounts of data are displayed too fast to read and "restart" (Scroll) when the user wishes to continue.**
Secondary Menus	**Options assigned to individual users to tailor their menu choices. If a user needs a few options in addition to those available on the Primary menu, the options can be assigned as secondary options. To facilitate menu jumping, secondary menus should be specific activities, not elaborate and deep menu trees.**
Secure Menu Delegation (SMD)	**A controlled system whereby menus and keys can be allocated by people other than IRM staff, such as application coordinators, who have been so authorized. SMD is a part of Menu Manager.**
Server	**An entry in the OPTION file. An automated mail protocol that is activated by sending a message to the server with the "S.server" syntax. A server's activity is specified in the OPTION file and can be the running of a routine or the placement of data into a file.**
Set of Codes	**Usually a one- or two-character preset code that is a permissible value for a data field. Almost always, the set of codes data fields require capital letters as a response (e.g., M for male and F for female). If anything other than the acceptable code is entered, the computer will reject the response.**

Sign-on/Security — The Kernel module that regulates access to the menu system. It performs a number of checks to determine whether access can be permitted at a particular time. A log of sign-ons is maintained.

Site Manager/IRM Chief — At each DHCP site, the individual who is responsible for managing computer systems, installing and maintaining new modules, and serving as liaison to the ISCs.

Software — The set of instructions and data required to operate the computer. One type is called operating system software -- that is, fundamental computer software that supports other software. The second type is called applications software -- in other words, customized programs that tell the computer how to run applications (e.g., Pharmacy, Laboratory).

Special Queuing — An option attribute indicating that TaskMan should automatically run the option whenever the system reboots.

Spooler — An entry in the DEVICE file. It uses the associated operating system's spool facility, whether it's a global, device, or host file. The Kernel manages spooling so that the underlying OS mechanism is transparent. In any environment, the same method can be used to send output to the spooler. The Kernel will subsequently transfer the text to a global for subsequent despooling (printing).

Subscript — In MUMPS, a numeric or string value that is enclosed in parentheses, appended to the name of a local or global variable, and used to identify a specific node within an array.

Synonym — A field in the OPTION file. Options may be selected by their menu text or synonym (see Menu Text).

TaskMan — The Kernel module that schedules and processes background tasks (also called Task Manager).

Templates — In VA FileMan, a way of associating fields in a file or in related files for later reference. Edit sequences are stored in the INPUT TEMPLATE file, print specifications are stored in the PRINT TEMPLATE file, and search or sort specifications are stored in the SORT TEMPLATE file.

Terminal — A device consisting of a video adapter, a monitor, and a keyboard. A terminal does little or no computer processing on its own; instead, it is connected to a computer by a communications link. See also Monitor and CRT.

Timed-read — The amount of time the Kernel will wait for a user response to an interactive read command before starting to halt the process.

Trigger — A type of VA FileMan cross reference. Often used to update values in the database given certain conditions (as specified in the trigger logic). For example, whenever an entry is made in a file, a trigger could automatically enter the current date into another field holding the creation date.

Type-ahead — A buffer used to store characters that are entered before the corresponding prompt appears. Type-ahead is a shortcut for experienced users who can anticipate an expected sequence of prompts.

UCI — User Class Identification, a computing area. The MGR UCI is typically the manager's account, while VAH or ROU may be production accounts.

Up-arrow Jump — In the menu system, entering an up-arrow (^) followed by an option name accomplishes a jump to the target option without needing to take the usual steps through the menu pathway.

User Interface — The way the package is presented to the user -- issuing of prompts, help messages, menu choices, etc. A standard user interface can be achieved by using VA FileMan for data manipulation, the menu system to provide option choices, and VA FileMan's Reader, the ^DIR utility, to present interactive dialogue.

VA FileMan — DHCP's Database Management System (DBMS). The central component of the Kernel that defines the way standard DHCP files are structured and manipulated.

VAX — Virtual Address Extension; a computer series manufactured by Digital Equipment Corporation. One of the types of computers used by DHCP.

VDT — Video Display Terminal. (See CRT, Terminal, Monitor.)

Verification — **A process of DHCP package review carried out by technical staff not directly involved in the development of the package. Any violations of SAC policy should be identified and corrected.**

Verify Code — **A secret password used along with the access code to provide secure user access. The Kernel's Sign-on/Security system uses the verify code to validate the user's identity.**

Write Access — **A user's authorization to write/update/edit information stored in a computer file.**

Z Editor (^%Z) — **A Kernel tool used to edit routines or globals. It can be invoked with an option, or from direct mode after loading a routine with >X ^%Z.**

ZOSF Global (^%ZOSF) — **The Operating System File -- a manager account global distributed with the Kernel to provide an interface between DHCP application packages and the underlying operating system. This global is built during Kernel installation when running the manager setup routine (ZTMGRSET). The nodes of the global are filled-in with operating system-specific code to enable interaction with the operating system. Nodes in the ^%ZOSF global may be referenced by application programmers so that separate versions of the package need not be written for each operating system.**

Index

$Y special variable 261
^%ZIS 205-214
^%ZISC 218
^%ZISL global 270
^%ZOSF nodes 483-486
    ACTJ 483
    AVJ 483
    BRK 483
    DEL 483
    EOFF 483
    EON 483
    EOT 483
    ERRTN 483
    ETRP 483
    GD 483
    JOBPARAM 483
    LABOFF 484
    LOAD 484
    LPC 484
    MAGTAPE 484
    MAXSIZ 484
    MGR 484
    MTBOT 484
    MTERR 484
    MTONLINE 484
    MTWPROT 484
    NBRK 484
    NO-PASSALL 484
    NO-TYPE-AHEAD 484
    OS 484
    PASSALL 484
    PRIINQ 485
    PRIORITY 485
    PROD 485
    PROGMODE 485
    RD 485
    RESJOB 485
    RM 485
    RSEL 485
    RSUM 485
    SAVE 485
    SIZE 485
    SS 485
    TEST 485
    TMK 485
    TRAP 485
    TRMOFF 485
    TRMON 485
    TRMRD 485
    TYPE-AHEAD 485
    UCI 485
    UCICHECK 486
    UPPERCASE 486
    VOL 486
    XY 486
    ZD 486
^%ZTBKC 481
^%ZTER 183
^%ZTER global 177, 179
^%ZTLOAD 271, 288, 369, 378-383
^%ZTSCH global 292, 351-354
^%ZTSK global 292, 354-355
486 configuration 311
Abbreviated Menu Diagrams 89
ABS^XLFMTH 516
Access code 13, 29, 34
    Purging 44
ACOSDEG^XLFMTH 516
ACOSH^XLFHYPER 512
ACOS^XLFMTH 516
ACOTDEG^XLFMTH 516
ACOTH^XLFHYPER 512
ACOT^XLFMTH 516
ACSCDEG^XLFMTH 517
ACSCH^XLFHYPER 512
ACSC^XLFMTH 516
ACTION^XQALERT 147
ACTION^XQH4 176
Add Error Screens (Task Manager) 350
Adding new users 29-31, 50
    Add a New User to the System 29
    Grant Access by Profile 30
    NEW PERSON IDENTIFIERS 29
    Primary menu 29
    Security forms 30, 31
    SSN 29, 30
    XUMGR key 29, 30
    XUSPF200 key 29
ADD^XPDMENU 112
ADD^XUSERNEW 50
Agency 26
AGENCY file 26
AK.keyname cross reference 120
ALERT file 138, 141
Alert identifier 142
Alert management 138-140
ALERT TRACKING file 138
Alerts 17, 135-156
    ACTION^XQALERT 147
    ALERT file 138
    Alert identifier 142
    ALERT TRACKING file 138
    DELETE^XQALERT 147, 148

FORWARD^XQALFWD 152
Glossary 156
Make an Alert on the Fly 140
Management 138-140
NOTIPURG^XQALBUTL 149
Package identifier 142
PATIENT^XQALERT 150
Processing 136-137
Programmer tools 141-153
PTPURG^XQALBUTL 151
Purge Alerts for a User 140
Purging 139
RECIPURG^XQALBUTL 151
SETUP^XQALERT 143
USER^XQALERT 154
All keys a user needs 116
Allocation of Security Keys 116
Allowed to Use Spooler 36
Alpha/Beta Install Tracking Menu Options
Alpha/Beta Test Option Usage Menu 476
Print Alpha/Beta Errors (Date/Site/Num/Rou/Err) 476
Alpha/beta tracking 414-416, 475-477
Actual Usage of Alpha/Beta Test Options 414
Alpha/Beta Test Option Usage Menu 414
Local option counting 416
Low Usage Alpha/Beta Test Options 414
Print Alpha/Beta Errors (Date/Site/Num/Rou/Err) 414
Purging of the Option Counts 416
Send Alpha/Beta Usage to Developers 414
Sending a Summary Message 415
ALTERNATE EDITOR file 35
Always Show Secondaries 37
Answerback message 24
ASECDEG^XLFMTH 517
ASECH^XLFHYPER 512
ASEC^XLFMTH 517
ASINDEG^XLFMTH 517
ASINH^XLFHYPER 512
ASIN^XLFMTH 517
Ask Device Type at Sign-On 18, 36
ASK PARAMETERS 197
ATANDEG^XLFMTH 517
ATANH^XLFHYPER 512
ATAN^XLFMTH 517
Audit privileges 64
Audited Options Purge 88
Audits
Failed Access Attempts 43
Option use 88
Sign-on 43
Auto-despooling 246, 252
AUTO-GENERATE ACCESS CODES 25
Auto-menu 18, 26, 36, 78, 87, 109
Automatic deactivation of users 41
Balance state (Task Manager) 362
BASE^XLFUTL 525
BMES^XPDUTL 460, 478
Browser device 256-259
BSA^XLFMSMT 513
Build a New Menu 124
Build checklists (KIDS) 531-538
Build entries 428
Build entry (defined) 393
BUILD file 394, 418, 422, 425, 428
Purging 422
Build File Print 418
CCD^XLFUTL 525
Change Device's Terminal Type 198
Change user's allocated keys to delegated keys 116
Check Taskman's Environment 335
Checksums (KIDS) 404, 425
CHGA^XGF 489
CJ^XLFSTR 522
Clean Error Log Over Range of Dates (Task Manager) 348
Clean old Job Nodes in XUTL 96
Clean Task File 339
Cleanup Task List 329
CLEAN^XGF 491
Clear all users at startup 28
Clear Electronic signature code 56
CLEAR^XGF 492
CLOSE^%ZISH 234
CLOSE^%ZISUTL 225
Closest printer 206, 212
CNV^XLFUTL 525
COMCP^XPDUTL 469, 478
Common menu 79, 82, 92
Component (defined) 393
Compute server job list 293, 333, 351
Computer Account Notification 31
Continue option 15, 82
Convert Loaded Package (KIDS) 420
Copy Build to Build 429
Copy Everything About an Option to a New Option 124
Copy One Users Menus and Keys to others 124
Copy the compiled menus from the print server option 97
COSDEG^XLFMTH 518

COSH^XLFHYPER 512
COS^XLFMTH 518
COTDEG^XLFMTH 518
COTH^XLFHYPER 512
COT^XLFMTH 518
CPU/Service/User/Device Stats 43
Create a Build Using Namespace 428-429
Create a Set of Options To Mark Out-Of-Order 94
CSCDEG^XLFMTH 518
CSCH^XLFHYPER 512
CSC^XLFMTH 518
CURCP^XPDUTL 469, 478
CURRENT^%ZIS 216
DA Return Code 201
DA Return Code Edit 201
DA RETURN CODES file 25, 193
Dangling pointers (OPTION file) 93
Date functions (XLF) (see XLF Function Library)
DCL context (Task Manager) (see Task Manager (DCL context))
De-allocation of Security Keys 116
Deactivate a User option 39
Deactivating users (see Terminating users)
DECDMS^XLFMTH 518
DEC^XLFUTL 525
Default Institution 26, 35
DEFAULT MULTIPLE SIGN-ON 24
DEL^%ZISH 235
Delegate keys option 116
Delegate's menu 124
Delegates (see Secure menu delegation)
Delete Error Log (Task Manager) 349
Delete Old (>14 d) Alerts 139
Delete Tasks 329
Delete Unreferenced Options 93
DELETEA^XQALERT 148
DELETE^XQALERT 147
Dequeue Tasks 328
Device allocation list 332, 351
DEVICE file 193-197, 294
  $I 194
  ASK HFS I/O OPERATION 230
  ASK HOST FILE 230
  ASK PARAMETERS 230
  Cross-references 203
  NAME 194
  OPEN PARAMETERS 197
  OpenVMS-Specific DEVICE Fields 197
  POST-CLOSE EXECUTE 194, 197
  PRE-OPEN EXECUTE 194, 197
  PRIORITY AT RUN TIME 318
  QUEUING 196
  RESOURCE SLOTS 270
  SIGN-ON/SYSTEM DEVICE 196
  SUBTYPE 194, 196
  TaskMan header page? 318
  TYPE 194, 318
  USE PARAMETERS 197
  VOLUME SET (CPU) 194, 318
Device Handler 34
  $I 208
  ^%ZIS 205-214
  ^%ZISC 218
  Alternate syntax 191
  Browser device 256-259
  CLOSE^%ZISUTL 225
  DA Return Codes 201
  DEVICE file 193-197
  Device type 211
  ENDR^%ZISS 220
  ENS^%ZISS 221
  Form feeds 260-263
  GKILL^%ZISS 224
  GSET^%ZISS 224
  Help frames 214, 215
  HLP1^%ZIS 214
  HLP2^%ZIS 215
  Home device 199
  HOME^%ZIS 215
  Host files (see Host files)
  Hunt groups 264-265
  KILL^%ZISS 225
  Loopback test 202
  Magtape devices 266
  Multiple Devices and ^%ZIS 214
  Network Channel Devices 267-269
  OPEN^%ZISUTL 226
  Out of service devices 202
  Page length 188
  PKILL^%ZISP 219
  Queuing 189
  Resource devices 270-271
  REWIND^%ZIS 217
  Right margin 188
  SDP devices 272-274
  Security (devices) 197
  Selecting devices 187-192, 201
  Slave printers 275-278
  Spool documents 191
  Spooling (see Spooling)
  Subtype (See Subtype)
  Subtypes 190
  Test pattern 202
  Troubleshooting 202
  USE^%ZISUTL 228
  Virtual terminals 200

Device IO list 332
Device security 197
Device waiting list 352
Devices (See Device Handler)
DE^XUSHSHP 57
Diagram Menus 89
Dialog entries (with KIDS) 444
DIALOG file 444
DIFROM 396
DIFROM variable 453, 459
Direct Mode Utilities 182, 340
Disable User 39
Display attributes 24
Display nodes 100
Display nodes (^XUTL) 102-103
Display User Characteristics 17, 20
Distribution (defined) 393
Distributions (see KIDS)
DISUSER field 36, 39
^DISV global 67
Division 35
Division multiple 26
DLAYGO 73
DLAYGO When Navigating to Files 73
DMSDEC^XLFMTH 519
Domain 47
Double quote jump 83
DOW^XLFDT 507
DQ^%ZTLOAD 383
DTIME 26, 36
DTR^XLFMTH 519
DT^XLFDT 507
DUZ (description) 66
DUZ(0) 34, 61, 62, 64, 67, 73
DUZ(2) 26, 35, 45
DUZ("AG") 26
DUZ("AG"), 45
DUZ("AUTO") 26
EC^%ZOSV 183
Edit a Build 430-464
  Components 441-445
    Dialog entries 444
    Forms 444
    Options 442
    Protocols 442
    Routines 443
    Templates 445
  File list 432-440
    Data options 437
    DD (full or partial) 434
    DD cleanup 440
    DD update 432
    Limited resolution of pointers 439
    Matching incoming entries 438
    Re-indexing files 439
    Sending data 436-440
    Sending security codes 432
Edit a User's Options 124
Edit an Existing User 32-37
  Access code 34
  Allowed to Use Spooler 36
  Always Show Secondaries 37
  ASK DEVICE TYPE AT SIGN-ON 36
  Auto-menu 36
  DISUSER 36
  Division multiple 35
  File Manager Access Code 34
  File Range 36
  Initial 32
  Mail Code 33
  Multiple sign-on 36
  Name 32
  Nick Name 32
  PREFERRED EDITOR 35
  Primary menu 33
  Prohibited Times for Sign-on 37
  Secondary Menu 33
  SERVICE/SECTION 35
  SSN 33
  Termination date 37
  Timed-read 36
  Title 32
  Type-Ahead 36
  Verify code 34
Edit Electronic Signature code 17, 55
Edit Error Screens (Task Manager) 350
Edit options 86, 108
Edit User Characteristics 17, 18-19, 38
  Form and template 38
Edit User Characteristics option 38
Edit User's Spooler Access 249
Electronic Signature Block Edit 56
Electronic signatures 55-58
  DE^XUSHSHP 57
  EN^XUSHSHP 58
  HASH^XUSHSHP 58
  SIG^XUSEIG 57
EN1^XQH 176
ENDR^%ZISS 220
ENS^%ZISS 221
Enter/Edit Kernel Site Parameters 22
Enter/Edit of Security Keys 116
Entry action (options) 109
Environment check (see KIDS environment check)
EN^XQH 175
EN^XUA4A71 527
EN^XUSHSHP 58

ERROR LOG file 179, 347
ERROR MESSAGES file 179
Error processing 16, 177-184
  ^%ZTER 183
  Clean Error Trap option 179
  EC^%ZOSV 183
  Enhanced error processing 180
  Error Trap Display 180
  Interactive Print of Error Messages option 181
  P1 Print 1 Occurrence of Each Error for T-1 (QUEUE) option 178
  P2 Print 2 Occurrences of Errors for T-1 (QUEUE) 178
  ^XTER 180, 182
  ^XTERPUR 179, 182
Error Screens (Task Manager) 293, 349, 351
Error Trap Display 180
Errors Logged in Alpha/Beta Test (QUEUED) option 476
Establish System Audit Parameters 88
Example 53
Exit action (options) 109
E^XLFMTH 519
Exploding key 119
EXP^XLFMTH 519
Extended help 170
FAILED ACCESS ATTEMPTS LOG 43
FAILED ACCESS ATTEMPTS LOG file 44
File Access Security 34, 65
File Access Security system 59-74
  Conversion 67-72
  File access 66
  Management 61-66
  Programmer tools 73-74
  User Interface 59-60
File Manager access code 34, 64
File Manager Options, Limited (Build) 127
FileMan line editor 18, 35
Find a User 42
Fix Help Frame File Pointers 172
Fix Option File Pointers 93
FMADD^XLFDT 507
FMDIFF^XLFDT 508
FMTE^XLFDT 508
FMTH^XLFDT 509
Forced queuing 213
Form feeds 210, 218, 260-263
  Guidelines for issuing 261-263
  SUPPRESS FORM FEED AT CLOSE 260
Forms (KIDS) 444
FORWARD^XQALFWD 152
FTG^%ZISH 236
GATF^%ZISH 237
GET1^DID 458
GETENV^%ZOSV 299, 482
GKILL^%ZISS 224
Global block count 481
Global distribution 396
Go home jump 82
Grant Access by Profile 29, 30
Graphic variables 224, 225
GSET^%ZISS 224
GTF^%ZISH 238
HADD^XLFDT 509
Halt option 15, 82
HASH^XUSHSHP 58
HDIFF^XLFDT 510
Header (options) 109
Header page (TaskMan) 318
Help frames
  Keywords 173
Help processor 169-176
  ACTION^XQH4 176
  Creating help frames 173-174
  Deleting help frames 172
  Editors 172
  EN1^XQH 176
  EN^XQH 175
  Help frame layout 173
  Help system actions 170
  Menu system 170, 174
HLP1^%ZIS 214
HLP2^%ZIS 215
HOLIDAY global 526
Home device 187, 206, 209, 216
HOME^%ZIS 88, 215
Host file devices 209
Host File Server Device Edit option 230
Host files 229-242
  CLOSE^%ZISH 234
  DEL^%ZISH 235
  DSM for OpenVMS devices 231
  FTG^%ZISH 236
  GATF^%ZISH 237
  GTF^%ZISH 238
  LIST^%ZISH 239
  MSM-DOS devices 232
  MV^%ZISH 240
  OPEN^%ZISH 241
  Programmer tools 233-242
  PWD^%ZISH 242
  STATUS^%ZISH 242
HTE^XLFDT 510
HTFM^XLFDT 510
Hunt groups 206, 210, 264-265
H^XUS 45

Hyperbolic trigonometric functions (XLF) (see XLF Function Library)
INHIBIT LOGONS? 28
Initial 18
Inquire 89
INSTALL file 395, 406, 418, 422
Purging 422
Install File Print 418
Installation questions (see KIDS)
Installations (KIDS) 397, 416
INSTALL^XPDCPU 411, 423
Institution 47
Interactive Print of Error Messages 181
Introductory text 21
INVERT^XLFSTR 522
IO list 332
IONOFF variable 218, 260, 261
IOP 212
ISQED^%ZTLOAD 384
Job list 293, 352
Jump nodes 100
Jump Nodes (^XUTL) 104-105
Jump start (Sign-on) 15
Jumping to options 81
Kernel Management Menu [XUKERNEL] 416
KERNEL SYSTEM PARAMETER file 47
KERNEL SYSTEM PARAMETERS file 50, 416
Kernel Toolkit 313, 320
Key delegation levels 120
Key Management 116
Key variables 453, 459
Servers 166
Tasks 368
Keys for a given menu tree 116
KIDS 393-416, 417-425, 427-449, 451-478
Alpha/beta tracking 475-477
BMES^XPDUTL 478
Build entries 428
Build entry (defined) 393
BUILD file 394, 418, 425, 428
Build File Print 418
Checksums 404, 425
COMCP^XPDUTL 478
Component (defined) 393
Convert Loaded Package 420
Copy Build to Build 429
Create a Build Using Namespace 428-429
CURCP^XPDUTL 478
Distribution (defined) 393
Distributions 396-398, 446-449
Global 396, 412
Split across diskettes 400
Standard 396
Edit a Build (see Edit a Build)
INSTALL file 395, 406, 418
Install File Print 418
Installations 397-416
Installation sequence 397-398
Moving routines 410
Progress 408
Queued 406
Restarting 409
INSTALL^XPDCPU 411
Loading a Distribution 399-401, 404
MES^XPDUTL 478
MOVE^XPDCPU 411
NEWCP^XPDUTL 478
Package (defined) 393
PACKAGE file 395
PACKAGE file link 473
Package revision data node 458
PARCP^XPDUTL 478
PKG^XPDUTL 461, 478
Purging 412
Purging KIDS files 412
ROUTINE file 424
RTNUP^XPDUTL 478
Track package nationally 474
Transport global
Checksums 404
Comparing 403
Printing 402
Transport global (defined) 393
Transporting a distribution 446, 449
Efficient builds 449
UPCP^XPDUTL 478
Update Routine File 424
Utilities 417-425
VERCP^XPDUTL 478
Verify a Build 425
Verify Package Integrity 425
VERSION^XPDUTL 461, 478
VER^XPDUTL 461, 478
^XPDCPU 410
KIDS build checklists 531-538
KIDS environment check 452-457
Aborting installations 454
DIFROM variable 453
Disable options/protocols 455
Key variables 453
Move routines to other CPUs 455
Routine install options 453
RTNUP^XPDUTL 457
Run twice 452
Sample routine 456

Version numbers 453
XPDENV variable 453
XPDNM variable 453
KIDS installation questions 462-464
Answering 405
M code 463
Questions and answers 463
Re-answering 406
Skipping 463
Subscripts 462
Where asked 464
KIDS pre- and post-install 458-472
Aborting installations 458
BMES^XPDUTL 460
Checkpoint parameter node 466
Checkpoints 465-472
Checkpoints with call backs 465-467
Checkpoints without call backs 468
COMCP^XPDUTL 469
CURCP^XPDUTL 469
DIFROM variable 459
Key variables 459
MES^XPDUTL 460
NEWCP^XPDUTL 470
PARCP^XPDUTL 471
Sample routine 467
UPCP^XPDUTL 472
VERCP^XPDUTL 472
XPDNM variable 459
ZTQUEUED variable 459
KIDS utilities 417-425
KILL^%ZISS 225
KILL^%ZTLOAD 369, 371, 385
Killing a task 333
KILL^XUSCLEAN 49
KSP^XUPARAM 47
LENGTH^XLFMSMT 513
LGR^%ZOSV 482
LIFETIME OF VERIFY CODE 25
Limited File Manager Options (Build) 124
Limited Resolution of Pointers 439
Line Editor 35
Link list 293, 352
List Defined Option Sets 94
List Error Screens (Task Manager) 349
List Options by Parents and Use 89
List Terminal Types 198
List Users 42
List users holding a certain key 116
LIST^%ZISH 239
LJ^XLFSTR 522
LN^XLFMTH 519
Load list 352
Loading a Distribution (KIDS) 401
Loading a Standard Distribution (KIDS) 399
Lock-out times 22
LOG RESOURCE USAGE? 352
LOGIN menu template 14, 84
Logoff 15-16
Logon (see Sign-on)
LOG^XLFMTH 519
LOW^XLFSTR 522
Magtape devices 266
Mail code 33
Make an Alert on the fly 139
Manage User File 119
Manager (Task Manager) 287, 289
Manager (TaskMan) 289
Managing the Display Attributes (DA) Return Codes 201
Mark Option Set Out-Of-Order 94
Math functions (XLF) (see XLF Function Library)
MAX SIGNON ALLOWED 21, 22
MAX SPOOL DOCUMENT LIFE-SPAN 250
MAX SPOOL DOCUMENTS PER USER 250
MAX SPOOL LINES PER USER 250
MAX^XLFMTH 519
Measurement functions (XLF) (see XLF Function Library)
Menu Diagrams (with Entry/Exit Actions) 89
Menu Manager 90
ADD^XPDMENU 112
Auto-menu 78
Choosing options 77-84
Creating options 86, 107
Diagramming options 89
Displaying options 90
Double quote jump 83
Fixing option file pointers 93
Go home jump 82
Local modifications 92
LOGIN menu template 14
Menu jumping 81
Menu jumping error messages 98-99
Menu tree rebuilding 97
NEXT^XQ92 113
OPTION file fields 87
Option scheduling 96-97
Option types 107
OP^XQCHK 114
Out-of-order option sets 94
OUT^XPDMENU 112
Primary menu 78
Rebuilding menu trees 97
RENAME^XPDMENU 113

Restricting option usage 95
Rubber band jump 82
Templates 83-84
LOGIN menu template 84
Up-arrow jump 81
Variables 106
Variables for programmer use 109-110
^XQ1 111
^XQDATE 114
XQMM("A") variable 109
XQMM("B") variable 109
XQMM("J") variable 110
XQMM("N") variable 110
XQUIT variable 109
^XUTL global 100-105
Menu Templates 17
Menu trees (rebuilding) 97
MES^XPDUTL 460, 478
MIN^XLFMTH 520
Monitor Taskman
Monitor actions 334
Run node 330
Mounted volume sets 296, 314
MOVE^XPDCPU 411, 423
Multiple copies (spooling) 243
Multiple Sign-On 36
MV^%ZISH 240
Network Channel Device Edit option 268
Network Channel Devices 267-269
NEW PERSON IDENTIFIERS 29
NEWCP^XPDUTL 470, 478
NEXT^XQ92 113
Nick Name 18
NOTIPURG^XQALBUTL 149
NOW^XLFDT 511
OBJECT file 175
OLD ACCESS AND VERIFY CODES file 44
One-time option queue 346
OPEN PARAMETERS field 197, 207, 210, 232, 273
OPEN^%ZISH 241
OPEN^%ZISUTL 226
Option Access By User 90
Option Audit Display 88
Option description 89
OPTION file 85, 416
Dangling pointers 93
Entry action 109
Exit action 109
Fields 108
Header 109
Option name 79
Option restrictions 79
Option Scheduling 88, 341-346
Deleting and requeuing 342
List Background Options 341
One-time option queue 346
Problems 346
Queuing an option 342
Rescheduling frequency 344
Scheduling code formats 345
Task parameters 344
OPTION SCHEDULING file 97, 288, 294
Option sets (out-of-order) 94
Options
Creating 107, 108
Diagramming 89
Displaying 90
Locked 116
Scheduling 88
Unreferenced 93
Options (with KIDS) 442
Options in the Option File that are Out-of-Order 94
OP^XQCHK 114
OUT OF ORDER MESSAGE 95
Out-Of-Order Set Management 94
OUT^XPDMENU 112
Package (defined) 393
PACKAGE file (KIDS) 395, 473
Package identifier (alerts) 142
Package revision data node 458
Package-wide variables, protecting 49
Page length 211
Parameters, site 47
PARCP^XPDUTL 466, 471, 478
Parent of Queuable Options 28, 341
Part 3 of Kernel Install (see File Access Security system)
Partition size 300
PATIENT^XQALERT 138, 150
Performance monitor 313, 320
PERMITTED DEVICES multiple 95
Phantom jump 110
PI^XLFMTH 520
PKG^XPDUTL 461, 478
PKILL^%ZISP 219
Place Taskman in a WAIT State 337
POP output variable 205, 211
Post sign-in Text Edit 27
Post-execution commands - ZTREQ 372
PRD^DILFD 458
Preferred Editor 19, 35
Primary menu 14, 21, 29, 91
Print Option File 89
Print servers 311
Print Sign-on Log 42
Priority (for tasks) 300

Priority of interactive users 24
Programmer Access Code (PAC) 36
Programmer Mode option 36
PROHIBITED TIMES FOR SIGN-ON 22, 37
Protocols (with KIDS) 442
Protocols Marked Out-of-Order in Protocol File 94
Provider key 118
PSET^%ZISP 219
PTPURG^XQALBUTL 151
Purge Alerts for a User 139
Purge Error Log Of Type of Error (Task Manager) 348
Purge Inactive Users' Attributes option 39
Purge Log of Old Access and Verify Codes 44
Purge old spool documents option 250
Purging
  Alerts 139, 140
  Alpha/beta tracking 416
  Audited options 88
  Error trap 179
  Failed Access Attempts Log Purge 44
  Inactive users' attributes 41
  KIDS files 412, 422-423
  Old access and verify codes 44
  Old job nodes in XUTL 96
  Options (unreferenced) 93
  Sign-On Log 43
  SIGN-ON LOG file 44
  Sign-on nodes 96
  Spool documents 245, 250
  TaskMan error log 348, 349, 350
  Tasks 339, 354
  ^UTILITY 96
  ^XTMP 96
PWD^%ZISH 242
PWR^XLFMTH 520
Queuable Task Log Cleanup 339
Queuers 287, 288, 366-367
  ^%ZTLOAD 366
  Non-interactive 382
  Scheduled options 366
Queuing 206, 208, 209, 281
  Forced queuing 196
Queuing (to the spooler) 243, 253
Queuing an option 342
Queuing Required Flag 95
Qume 102 24
Re-Indexing Files (KIDS) 439
Reactivate a User option 39
Rebuilding Primary Menu Trees 97
RECIPURG^XQALBUTL 151
Recover Deleted Option Set 94
Reindex the users key's 119
Release User 42
Remove delegated keys 116
Remove Error Screens (Task Manager) 350
Remove Out-Of-Order Messages from a Set of Options 94
Remove Taskman from WAIT State 338
RENAME^XPDKEY 121
RENAME^XPDMENU 113
REPEAT^XLFSTR 523
Replacement volume set 305
REPLACE^XLFSTR 523
Reprint Access agreement letter 31
REQ^%ZTLOAD 386
Requeue Tasks 328
Required volume set 306
Resource Device Edit option 271
Resource devices 270-271
  RESOURCE file 270
  SYNC FLAGs 270, 271
RESOURCE file 270
Response time 24
Restart Session option 15, 82
Restart Task Manager 322, 337
Restart TaskMan 340
Restarting KIDS installations 409
RESTART^ZTMB 340
Restrict Availability of Options 95
RESTRICT DEVICES flag 95
Reverse Locks 119
REVERSE/NEGATIVE LOCK 119
REWIND^%ZIS 217
Rewinding devices 217
Right margin 205, 210, 213
RJ^XLFSTR 523
ROUTINE file 424
Routines (with KIDS) 443
RTD^XLFMTH 520
RTNUP^XPDUTL 457, 478
Rubber band jump 82
RUN^ZTMKU 340
S^%ZTLOAD 369, 370, 387
SAC 1
SAYU^XGF 503
Schedule file 292, 293, 351-354
Schedule list 293, 332, 351
Scheduling options (see Option scheduling)
SCHEDULING RECOMMENDED field 88
SCH^XLFDT 511
Screen editor 35
Screen handling variables 220, 221
SDP Device Edit option 273
SDP devices 272-274
SECDEG^XLFMTH 520
SECH^XLFHYPER 512

Secondary Menu 33, 79, 91
Secure menu delegation 123-134
  Acting as a delegate 124-128
  Build a New Menu 126
  Copy Everything About an Option to a New Option 126
  Copy One Users Menus and Keys to Others 126
  Delegate's menu 124
  Delegating keys 130
  Delegating Options
    Select Options to be Delegated 129
  Delegation levels 131
  Edit a User's Options 125
  Limited File Manager Options (Build) 127
  Managing delegates 129-134
  Menu prefix 133
  Options too sensitive to delegate 132
  Remove Options Previously Delegated 132, 133
  Replicate or replace a delegate 132
  Reports 133
Security forms 30, 31
Security keys 115-121
  Allocating keys 117
  Creating 118
  De-allocating keys 117
  Delegating keys 117
  Delegation levels 120
  Deleting 119
  Exploding key 119
  Person lookup 118, 120
  Programmer tools 120
  RENAME^XPDKEY 121
  Reverse Locks 119
  Subordinate keys 118
SEC^XLFMTH 520
SELECTABLE AT SIGN-ON 25
Send Alpha/Beta Usage to Developers option 415
Sequential Disk Block Processor device 273
Sequential disk processor (see SDP devices)
SERVER BULLETIN 160
Servers 157-165
  Denying server requests 158
  Errors and Warnings 164
  Key variables 166
  Programmer tools 166-168
  Server bulletin 168
  Server request 157
  Setting up a server 159-161
  Testing 162
  XQSCHK utility 162
  XQSPING utility 162
SERVICE/SECTION 32, 35
SETUP^XQALERT 143
SET^XUS1A 48
Show Error Log (Task Manager) 347
Show the keys of a particular user 116
Show Users with Selected Primary Menu 90
Sign-Off 15-16
Sign-on 13-15, 21-28
  Enabling/disabling logons 28
  Flow Chart 23
  Jump start 15
  Lock-out times 22
  SIGN-ON LOG file 43
  Statistics 43
SIGN-ON LOG file 44
SIG^XUSESIG 57
SINDEG^XLFMTH 521
SINH^XLFHYPER 512
SIN^XLFMTH 520
Site parameters 21, 47, 299
  Spooler 250
Site parameters (Task Manager) 298-314
Slave printers 209, 275-278
Soundex ($$EN^XUA4A71) 527
Specifying Right Margin and Page Length 188
SPOOL DATA file 248
SPOOL DOCUMENT file 248
Spool Documents
  Making into mail messages 36
Spooler Menu 17
Spooler Site Parameters Edit 250
Spooling 243-253
  Auto-despooling 252
  Creating spool documents 243-244
  Document names (generated) 252
  Generating names 252
  Making into mail messages 246
  Management 249-252
  Naming 243
  Printing spool documents 246
  Privileges 36, 249
  Programmer tools 253
  Purging spool documents 250
  Retrieving spool documents 245
  Site parameters 47
  Spool device 209
  Spool Device Edit option 252
  Spool device types 251
  Spool document name 191, 198, 213
  Spool documents storage 248
  Spool Management menu 249
  Spooler menu 245

Storage overflows 248
System defaults 250
Viewing spool documents 245
SQRT^XLFMTH 521
SSN 29, 30, 33
Standard distribution 396
Startup list 353
Status List 293, 331, 353
STATUS^%ZISH 242
Stop node 353
Stop Task Manager 338
Stopping tasks 387
STOP^ZTMKU 340
String functions (XLF) (see XLF Function Library)
STRIP^XLFSTR 524
Sub node 354
Submanager 290
Submanagers 287, 290
Subordinate keys 118
Subtype 199, 211
SUPPRESS FORM FEED AT CLOSE 260
Switch Identities 93
SYNC FLAGs 270, 271, 338
SYNC FLAGs to Control Sequences of Tasks 374
Synonym 79, 87
System parameters 21
TANDEG^XLFMTH 521
TANH^XLFHYPER 512
TAN^XLFMTH 521
Task file cleanup 339
Task list 293, 354
Task Manager 281
^%ZTLOAD 271, 378-383
^%ZTSCH global 292, 351-354
^%ZTSK global 292, 354-355
API 288
Cleanup Task List 329
Configuration (see Task Manager Configuration, 298
Creating tasks 281, 282
Defining environments 297
Delete Tasks 329
Dequeue Tasks 328
Direct mode utilities 340
RESTART^ZTMB 340
RUN^ZTMKU 340
Starting 340
Stopping 338
STOP^ZTMKU 340
WAIT^ZTMKU 340
^ZTMB 340
^ZTMCHK 340
^ZTMON 340
DQ^%ZTLOAD 383
error log (see Taskman error log
ISQED^%ZTLOAD 384
Job limit 301
KILL^%ZTLOAD 385
List Tasks 326
Load balancing 316-317
Manager 287, 289
Manager Startup 315
Managers (running multiple) 316-317
Operation and management 325
Option scheduling (see Option scheduling)
Overview 287-296
Programmer tools 365-389
Queuers 287, 288
Queuing an option 342
Queuing output 281
REQ^%ZTLOAD 386
Requeue Tasks 328
Restarting 337
S^%ZTLOAD 387
Schedule file 292, 293
Sequences of tasks 270
Starting 340
Startup 315
STAT^%ZTLOAD 388
States 331, 362
Stopping 338
Stopping tasks 285
Submanagers 287, 290
SYNC FLAGs 270, 271
Task rejection messages 360-361
Task status 388
Task status codes 356-359
Taskman error log (See Taskman error log) 347
Taskman Management Menu 325-330
Taskman Management Utilities 330-339
TaskMan's reach 298
Tasks (See Tasks)
TASKS file 292, 294
Terminology 295
TM^%ZTLOAD 389
Troubleshooting 351-364
Working with Tasks 283
ZTMQ key 296
Task Manager (DCL context) 320-324, 379
ACL setup 320, 324
Batch queues 321, 323
Logicals (VMS) 321
Restarting 321, 322
Site parameters 321

Startup 321
TASKMAN account 320, 323
TASKMAN Queue 323
ZTMSWDCL.COM 322
ZTMWDCL.COM 322
Task Manager Configuration 297-314, 324
DCL context 320-324
DEVICE file 318
DSM for OpenVMS 310
Load balancing 316-317
Mixed Systems 314
MSM-DOS 311-313
Multiple managers 316-317
TASKMAN SITE PARAMETERS file 299
UCI ASSOCIATION file 308-309
VOLUME SET file 304-307, 344
Taskman Error Log 293, 347-350, 351
Add Error Screens 350
Clean Error Log 348
Delete Error Log 349
Edit Error Screens 350
Error screens 349
List Error Screens 349
Purge Error Log 348
Remove Error Screens 350
Show Error Log 347
Taskman Management Menu 325, 330
Cleanup Task List 329
Delete Tasks 329
Dequeue Tasks 328
List Tasks 326
Requeue Tasks 328
Taskman Management Utilities 330-339
Check Taskman's Environment 335
Clean Task File 339
Monitor Taskman 330
Place Taskman in a WAIT State 337
Queuable Task Log Cleanup 339
Remove Taskman from a WAIT State 338
Restart Task Manager 337
Stop Task Manager 338
SYNC flag file control 338
TASKMAN SITE PARAMETERS 298
TASKMAN SITE PARAMETERS file 294, 310, 311, 312
BOX-VOLUME PAIR 299
DEFAULT TASK PRIORITY 300
Load balance routine 316
LOG TASKS? 299
Mode of TaskMan 302
SUBMANAGER RETENTION TIME 301
TASK PARTITION SIZE 300
TASKMAN HANG BETWEEN JOBS 301
TASKMAN JOB LIMIT 301
VAX DSM ENVIRONMENT FOR DCL 303
TaskMan User menu 17, 283
Editing tasks 285
Listing tasks 286
Stopping tasks 285
Task status 284
Tasks 368-375
^%ZIS call within a task 373
^%ZTLOAD call within a task 373
Destination 369
Device 369
DT variable 368
DUZ array 368
Error Trap 369
Guaranteed Environment 368-370
IO* array 368
Key Variables 368-370
Post-execution commands 372
Priority 300, 369
Purging the task record 371
Queuing with no I/O device 383
S^%ZTLOAD 370
Saved Variables 369
Stop requests 370
SYNC FLAGs 374
Tools 369
Two-step tasks 373
ZTDESC variable 368
ZTDTH variable 368
ZTIO variable 369
ZTQUEUED variable 369, 371
ZTREQ 372
ZTREQ variable 371
ZTSK variable 369
ZTSTAT variable 374
ZTSTOP variable 370
TASKS file 292, 294, 354-355
TBOX (see User's Toolbox)
Templates (with KIDS) 445
TEMP^XLFMSMT 514
Terminal server 206, 210
Terminal Type Edit 198
TERMINAL TYPE file 193, 198-200, 260
BACK SPACE 198
CLOSE EXECUTE 198
FORM FEED 198
NAME 198
Naming Conventions 198
OPEN EXECUTE 198

PAGE LENGTH 198
RIGHT MARGIN 198
SELECTABLE AT SIGN-ON 198, 201
Terminal type prompt 16
Terminating users 39, 53
Termination Date 37, 39
Text Terminator 18
TIED ROUTINE 21
Time option 82
Timed-read 26, 36
TM^%ZTLOAD 389
Toggle options/protocols on and off 94
Toolbox (see User's Toolbox)
Toolkit Queuable Options menu
Errors Logged in Alpha/Beta Test (QUEUED) option 476
Tracking option usage 414
Transfer Lines from Another Document option 60
Transport global
Comparing 403
defined 393
Printing 402
Transporting a distribution 446, 449
Type-Ahead 18, 26, 36
UCI 295
UCI ASSOCIATION 298
UCI ASSOCIATION file 294, 308, 310, 311, 312
FROM UCI 308
FROM VOLUME SET 309
TO UCI 309
TO VOLUME SET 309
UCI association table (see UCI ASSOCIATION file)
Up-arrow jump 77, 81, 97
UPCP^XPDUTL 466, 472, 478
Update node 354
Update Routine File 424
UP^XLFSTR 524
USE PARAMETERS 197, 207
USE PARAMETERS field 210
USE^%ZISUTL 228
User Attributes (see Edit an Existing User)
User Help 17
User Inquiry 42
User stacks 100
User stacks (^XUTL) 100-101
User Status Report 42
USER TERMINATE ROUTINE 53
USER TERMINATE TAG 53
User's Guide to Computing 2
User's Toolbox 17-20, 82, 83, 286
Display User Characteristics 20
Edit User Characteristics 18-19
Electronic signatures 55
TaskMan User (see TaskMan User menu
Users
Adding new (see Add new users)
Terminating (see Terminating users)
USER^XQALERT 138, 154
Utility functions (XLF) (see XLF Function Library)
Variables
Menu Manager 106
VAX/Alpha Performance Monitor 320
VCD^XLFUTL 525
VERCP^XPDUTL 472, 478
Verify a Build 425
Verify code 13, 14, 19, 25, 29, 34
Purging 44
Verify Package Integrity 425
VERSION^%ZOSV 482
VERSION^XPDUTL 461, 478
VER^XPDUTL 461, 478
View Alerts 17, 82, 135, 136
Virtual terminals 200
Volume set 295, 298
VOLUME SET file 294, 304, 310, 311, 312, 313, 314
DAYS TO KEEP OLD TASKS 307
INHIBIT LOGONS? 305
LINK ACCESS 305
OUT OF SERVICE? 305
REPLACEMENT VOLUME SET 306
REQUIRED VOLUME SET? 306
TASKMAN FILES UCI 306
TASKMAN FILES VOLUME SET 306
TYPE 304
VOLUME SET 304
Volume sets (mounted) 314
Volume sets (replacement) 305
VOLUME^XLFMSMT 514
VPM 320
Wait node 354
WAIT state 354
Waiting list 293
WAIT^ZTMKU 340
WEIGHT^XLFMSMT 515
Where am I? option 82
Workday calculation 526
XGF Function Library 487-506
$$READ^XGF 497
CHGA^XGF 489
CLEAN^XGF 491
CLEAR^XGF 492
Demo program 488

FRAME^XGF 493
INITKB^XGF 494
IOXY^XGF 495
PREP^XGF 496
RESETKB^XGF 499
RESTORE^XGF 500
SAVE^XGF 500
SAYU^XGF 503
SAY^XGF 501
SETA^XGF 504
System requirements 488
WIN^XGF 505
^XGFDEMO 488
^XGFDEMO 488
XLF Function Library 507-527
Date functions 507-511
Hyperbolic trigonometric functions 512
Math functions 516-521
Measurement functions 513-515
String functions 522-524
Utility functions 525
^XLFDT (Date functions) 507-511
^XLFHYPER (Hyperbolic trigonometric functions) 512
^XLFMSMT (Measurement functions) 513-515
^XLFMTH (Math functions) 516-521
^XLFSTR (String functions) 522-524
^XLFUTL (Utility functions) 525
^XMB global 416
^XPDCPU 410
XPDENV variable 453
XPDNM variable 453, 459
^XQ1 111
XQABTST variable 416
^XQDATE 114
XQSMDFM key 127
XQUIT variable 109
^XTER 182
^XTERPUR 179, 182
XTSPING utility 162
XU USER SIGN-ON 52
XU USER SIGN-ON extended action 28, 48
XU USER SIGN-ON option
Package-Specific Sign-On Actions 52
XU USER TERMINATE extended action 53
XUAUTHOR key 173
XUCOMMAND option 92
XUMGR key 29, 30, 33, 56
^XUP 45
^XUP routine 111
XUPROG key 36, 93, 115, 116
XUPROGMODE key 36, 179
^XUS 45
^XUSCLEAN 46
^XUSEC global 120
XUSPF200 key 29, 30, 33, 50
^XUTL global 96, 100-105
Display Nodes 102
Jump Nodes 104
User Stacks 100
XUTM QCLEAN 307, 354
^XUVERIFY 51
^XUWORKDY 526
ZSTOP 370
^ZTMB 340
^ZTMCHK 340
^ZTMGRSET 481
^ZTMON 340
ZTMQ key 296, 328, 329
ZTMSWDCL.COM 320, 322
ZTMWDCL.COM 320, 322
ZTQUEUED variable 369, 371, 459
ZTREQ 372
ZTREQ variable 369, 371
ZTSTAT variable 374
ZTSTOP variable 358, 369, 370
^ZU 28, 46

Option Index

[EVE] 42, 85
[XPD BUILD NAMESPACE] 427, 428
[XPD COPY BUILD] 427, 429
[XPD DISTRIBUTION MENU] 427
[XPD EDIT BUILD] 427, 430-464
[XPD PRINT BUILD] 417
[XPD PRINT INSTALL FILE] 417
[XPD ROUTINE UPDATE] 417
[XPD TRANSPORT PACKAGE] 427
[XPD UTILITY] 417
[XPD VERIFY BUILD] 417
[XQ UNREF'D OPTIONS] 93
[XQ XUTL $J NODES] 96
[XQAB ACTUAL OPTION USAGE] 414
[XQAB AUTO SEND] 414
[XQAB ERR DATE/SITE/NUM/ROU/ERR] 414, 476
[XQAB LIST LOW USAGE OPTS] 414
[XQAB MENU] 414, 476
[XQALERT BY USER DELETE] 139
[XQALERT DELETE OLD] 139
[XQALERT MAKE] 139
[XQALERT MGR] 139
[XQALERT] 17
[XQBUILDTREEQUE] 97
[XQBUILDTREE] 97
[XQCOPYOP] 124
[XQDISPLAY OPTIONS] 89
[XQHELP-ASSIGN] 171
[XQHELP-DEASSIGN] 171
[XQHELP-DISPLAY] 171
[XQHELP-LIST] 171
[XQHELP-MENU] 171
[XQHELP-UPDATE] 171
[XQHELP-XREF] 171
[XQHELPFIX] 171
[XQKEYALTODEL] 116
[XQKEYDEL] 116
[XQKEYRDEL] 116
[XQLISTKEY] 116
[XQLOCK1] 116
[XQLOCK2] 116
[XQOOFF] 94
[XQOOMAIN] 94
[XQOOMAKE] 94
[XQOON] 94
[XQOOREDO] 94
[XQOOSHOFIL] 94
[XQOOSHOPRO] 94
[XQOOSHOW] 94
[XQOOTOG] 94
[XQOPTFIX] 93
[XQRESTRICT] 95
[XQSHOKEY] 116
[XQSMD BUILD MENU] 124
[XQSMD COPY USER] 124
[XQSMD EDIT OPTIONS] 124
[XQSMD LIMITED FM OPTIONS] 124
[XQSMD USER MENU] 124
[XU DA EDIT] 201
[XU FINDUSER] 42
[XU OPTION QUEUE] 346
[XU-486 MENU COPY] 97
[XU-SPL-MGR] 249
[XU-SPL-PURGE] 250
[XU-SPL-SITE] 250
[XU-SPL-USER] 249
[XUADISP] 88
[XUAUDIT MAINT] 88
[XUAUDIT] 88
[XUAUTODEACTIVATE] 41
[XUCHANGE] 198
[XUCOMMAND] 15, 85
[XUCONTINUE] 15
[XUDEVEDITCHAN] 268
[XUDEVEDITHFS] 230
[XUDEVEDITMT] 266
[XUDEVEDITRES] 271
[XUDEVEDITSDP] 273
[XUDEVEDITSPL] 252
[XUEDITOPT] 86, 108
[XUERRS] 178
[XUERTRAP] 178, 180
[XUERTRP AUTO CLEAN] 179
[XUERTRP CLEAN] 178
[XUERTRP PRINT ERRS] 178
[XUERTRP PRINT T-1 1 ERR] 178
[XUERTRP PRINT T-1 2 ERR] 178
[XUFILEACCESS] 65
[XUFILECOPY] 65
[XUFILEDELETE] 65
[XUFILEGRANT] 65
[XUFILEINQUIRY] 65
[XUFILELIST] 65
[XUFILEPRINT] 65
[XUFILERANGEASSIGN] 65
[XUFILEREMOVEALL] 65
[XUFILESETDELETE] 65
[XUFILESINGLEADD] 65
[XUHALT] 15
[XUINQUIRE] 89
[XUKERNEL] 416
[XUKEYALL] 116

[XUKEYDEALL] 116
[XUKEYEDIT] 116
[XUKEYMGMT] 116
[XULIST] 198
[XUOPTDISP] 88
[XUOPTPURGE] 88
[XUOPTUSER] 42
[XUOPTWHO] 90
[XUPRINT] 89
[XURELOG] 15
[XUSC LIST] 42
[XUSER FILE MGR] 119
[XUSER KEY RE-INDEX] 119
[XUSER-CLEAR-ALL] 28
[XUSERAOLD] 44
[XUSERBLK] 29
[XUSERDEACT] 39
[XUSEREDIT] 32
[XUSERINQ] 42
[XUSERINT] 21
[XUSERLIST] 42
[XUSERNEW] 29
[XUSERPOST] 27
[XUSERPURGEATT] 39
[XUSERREACT] 39
[XUSERREL] 42
[XUSERREPRINT] 31
[XUSESIG BLOCK] 56
[XUSESIG CLEAR] 56
[XUSESIG] 55
[XUSITEMGR] 42, 414, 476
[XUSITEPARM] 22
[XUSTAT] 43
[XUTERM] 198
[XUTIO] 198, 201
[XUTM BACKGROUND PRINT] 341
[XUTM BVPAIR] 299
[XUTM CHECK ENV] 335
[XUTM CLEAN] 339
[XUTM DEL] 329
[XUTM DQ] 328
[XUTM ERROR DELETE] 349
[XUTM ERROR LOG CLEAN 348
[XUTM ERROR PURGE TYPE] 348
[XUTM ERROR SCREEN ADD] 350
[XUTM ERROR SCREEN EDIT] 350
[XUTM ERROR SCREEN LIST] 349
[XUTM ERROR SCREEN REMOVE] 350
[XUTM ERROR SHOW] 347
[XUTM INQ] 326
[XUTM REQ] 328
[XUTM RESTART] 337
[XUTM RUN] 338
[XUTM SCHEDULE] 342
[XUTM STOP] 338
[XUTM SYNC] 338
[XUTM UCI] 308
[XUTM VOLUME] 304
[XUTM WAIT] 337
[XUTM ZTMON] 330
[XUUSERACC1] 89
[XUUSERACC2] 89
[XUUSERACC] 89
[XUUSERSTATUS] 42
[XUXREF-2] 90
[XUXREF] 89
[ZTMQUEUABLE OPTIONS] 28, 41, 85, 97, 341
[ZTMUSER] 283

Department of Veterans Affairs
Decentralized Hospital Computer Program

KERNEL
SYSTEMS MANUAL

Version 8.0

July 1995

Information Systems Center
San Francisco, California

Preface

This manual provides descriptive information about the Kernel for use by Information Resource Management (IRM) staff, end users, Automated Data Processing Application Coordinators (ADPACs), and application programmers.

This manual assumes that the reader is familiar with the computing environment of the Decentralized Hospital Computer Program (DHCP), and understands VA FileMan data structures and terminology. Some understanding of the M programming language is helpful for some parts of the manual. No attempt is made to explain how the overall DHCP programming system is integrated and maintained; such methods and procedures are documented elsewhere. This manual does, however, provide an explanation of the Kernel utilities, describing how they can be used to establish a standard user interface, monitor and manage the computer system, customize the environment according to local site needs, and define new areas of computing activities for users.

Table of Contents

Introduction 1

Orientation 5

Package Management 9

Part 1: Sign-On/Security 11

Chapter 1 Sign-On/Security: User Interface 13
- **Signing On** 13
- **Escaping from a Jumbled Screen** 17
- **Alerts** 17
- **User's Toolbox** 17
- **Edit User Characteristics** 18
- **Display User Characteristics** 20

Chapter 2 Sign-On/Security: System Management 21
- **The Sign-On Process** 21
- **Adding New Users** 29
- **Editing Existing Users** 32
- **Deactivating and Reactivating Users** 39
- **User Management Options on the Operations Menu** 42
- **Sign-On Audits** 43

Chapter 3 Sign-On/Security: Programmer Tools 45
- **Direct Mode Utilities** 45
- **Callable Entry Points** 47
- **XU USER SIGN-ON Option** 52
- **XU USER TERMINATE Option** 53

Chapter 4 Electronic Signature Codes 55
- **User Interface** 55
- **System Management** 56
- **Programmer Tools** 57

Chapter 5 File Access Security 59
- **User Interface** 59
- **System Management** 61
- **Programmer Tools** 73

Part 2: Menu Manager 75

Chapter 6 Menu Manager: User Interface 77
Navigating Kernel's Menus 77
Menu Templates 83

Chapter 7 Menu Manager: System Management 85
Kernel's Menus 85
Creating Menus and Options 86
Displaying Menus and Options 89
Managing Menus and Options 91
Restricting Option Usage 95
Menu Manager Options that Should Be Scheduled 96
Error Messages During Menu Jumping 98
The ^XUTL Global: Structure and Function 100
Menu Manager Variables (Troubleshooting) 106

Chapter 8 Menu Manager: Programmer Tools 107
Creating Options 107
Variables for Programmer Use 109
Direct Mode Utilities 111
Callable Entry Points 112

Chapter 9 Security Keys 115
User Interface 115
System Management 116
Programmer Tools 120

Chapter 10 Secure Menu Delegation 123
User Interface: Acting as a Delegate 124
System Management: Managing Delegates 129

Chapter 11 Alerts 135
User Interface 135
System Management 138
Programmer Tools 141
Glossary of Terms for Alerts 156

Chapter 12 Servers 157
System Management 157
Programmer Tools 166

Chapter 13 Help Processor 169
User Interface 169

System Management 171
Programmer Tools 175
Chapter 14 Error Processing 177
User Interface 177
System Management 177
Programmer Tools 183
Part 3: Device Handler 185
Chapter 15 Device Handler: User Interface 187
Printing to Devices 187
Queuing 189
Specifying a Special Subtype 190
Alternate Syntax for Device Specification 191
Chapter 16 Device Handler: System Management 193
DEVICE File 193
TERMINAL TYPE File 198
Troubleshooting 202
Device Identification and Cross-references 203
Chapter 17 Device Handler: Programmer Tools 205
Callable Entry Points 205
Chapter 18 Host Files 229
User Interface 229
System Management 230
Programmer Tools 233
Chapter 19 Spooling 243
User Interface 243
System Management 248
Programmer Tools 253
Chapter 20 Special Device Issues 255
Browser Device 256
Form Feeds 260
Hunt Groups 264
Magtape 266
Network Channel Devices 267
Resources 270
SDP 272
Slaved Printers 275

Part 4: Task Manager 279

Chapter 21 Task Manager: User Interface 281
Creating Tasks 281
Working with Tasks 283

Chapter 22 Task Manager System Management: Overview 287
Task Manager's Division of Labor 287
Task Manager's Files 292
System Configuration Terminology 295
Task Manager Security Key 296

Chapter 23 Task Manager System Management: Configuration 297
Defining Task Manager Environments 297
Configuring Task Manager 298
Manager Startup 315
Multiple Managers and Load Balancing 316
Device Handler's Influence on TaskMan 318
Running TaskMan with a DCL Context 320

Chapter 24 Task Manager System Management: Operation 325
TaskMan Management Menu 325
TaskMan Direct Mode Utilities 340
Scheduling Options 341
The Taskman Error Log 347
Troubleshooting 351

Chapter 25 Task Manager: Programmer Tools 365
How to Write Code to Queue Tasks 365
Callable Entry Points 376

Part 5: KIDS 391

Chapter 26 KIDS System Management: Installations 393
KIDS = Distribution and Installation 394
Build Entries and the BUILD File 394
The INSTALL File 395
Changes in the Role of the PACKAGE File 395
The New Transport Mechanism: Distributions 396
What Happens to DIFROM? 396
Installing Standard Distributions 397
Installing Global Distributions 412
Purging the BUILD and INSTALL Files 412

Alpha/Beta Tracking 414
Chapter 27 KIDS Utilities 417
Build File Print 418
Install File Print 418
Convert Loaded Package for Redistribution 420
Purging Build and Install Files 422
Update Routine File 424
Verify a Build 425
Verify Package Integrity 425
Chapter 28 KIDS Programmer Tools: Creating Builds 427
Build Entries 428
Create a Build Using Namespace 428
Copy Build to Build 429
Edit a Build 430
Transporting a Distribution 446
Creating Transport Globals that Install Efficiently 449
Chapter 29 KIDS Programmer Tools: Advanced Build Techniques 451
Environment Check Routine 452
Pre- and Post-Install Routines: Special Features 458
Obtaining Package Name and Version Information 461
How to Ask Installation Questions 462
Using Checkpoints (Pre- and Post-Install Routines) 465
PACKAGE FILE LINK 473
Track Package Nationally 474
Alpha/Beta Tracking 475
KIDS Callable Entry Point Summary 478
Part 6: Other Tools 479
Chapter 30 Operating System Interface 481
System Management 481
Programmer Tools 482
Chapter 31 XGF Function Library 487
System Management 488
Programmer Tools 488
Chapter 32 XLF Function Library 507
Date Functions—XLFDT 507
Measurement Functions—XLFMSMT 513

Math Functions—XLFMTH 516
String Functions—XLFSTR 522
Utility Functions—XLFUTL 525
Other Functions 526

Appendices 529

Appendix A KIDS Build Checklists 531

Glossary 539

Index 557

Option Index 571