# KERNEL

# PROGRAMMER MANUAL

# (DRAFT)

Version 8.0

July 1995

Revised June 2006

Department of Veterans Affairs
VistA Health Systems Design & Development (HSD&D)
Infrastructure and Security Services (ISS)

# Revision History

## Documentation Revisions

The following table displays the revision history for this document. Revisions to the documentation are based on patches and new versions released to the field.

| Date | Revision | Description | Author |
|---|---|---|---|
| 07/95 | 1.0 | Initial Kernel V. 8.0 software and documentation release | San Francisco, CA Office of Information Field Office (OIFO):<br>• Project Manager: Hans Von Blanckensee<br>• Developers: Kernel Development Team<br>• Technical Writer: Kyle Clarke |
| 12/24/03 | 2.0 | Kernel V. 8.0 documentation reformatting/revision.<br><br>This is a DRAFT of the new *Kernel Programmer Manual*. This manual was created by extracting all programmer-specific content from the *Kernel Systems Manual* (original release date of July 1995).<br><br>The new *Kernel Programmer Manual* Includes added/updated API information (e.g., Reference Type, Category, Integration Agreement number). It also includes APIs for previous Kernel APIs never before documented (i.e., includes APIs that were previously only documented in patch descriptions, Integration Agreements, or separate supplemental documentation).<br><br>**NOTE:** This manual also includes some Kernel Toolkit APIs.<br><br>Due to time constraints, not all released Kernel patches with programmer-related content changes have been added at this time. Also, there is known missing information that will be added/updated at a future date. We wanted to get a new baseline document published so that in the future we can more easily update this new *Kernel Programmer Manual*. | Oakland, CA OIFO:<br>• Project Manager: Jack Schram<br>• Developers: Kernel Development Team<br>• Technical Writer: Thom Blom |

| | | | |
|---|---|---|---|
| | | As time allows, we will be updating this new manual with all released patch information that affects its content. Because of the chapter-numbering scheme, future additions can be made with minimal disruption to the entire manual page flow.<br>Thanks for your patience! | |
| 12/09/04 | 2.1 | Updated various APIs based on developer feedback. Also making minor edits as I begin populating the HTML versions of the APIs. | Oakland, CA OIFO:<br>• Project Manager: Jack Schram<br>• Developers: Wally Fort<br>• Technical Writer: Thom Blom |
| 12/20/04 | 2.2 | Reviewed document and edited for the "Data Scrubbing" and the "PDF 508 Compliance" projects.<br>**Data Scrubbing—**Changed all patient/user TEST data to conform to HSD&D standards and conventions as indicated below:<br>• The first three digits (prefix) of any Social Security Numbers (SSN) start with "000" or "666."<br>• Patient or user names are formatted as follows: KRNPATIENT,[N] or KRNUSER,[N] respectively, where the N is a number written out and incremented with each new entry (e.g., KRNPATIENT, ONE, KRNPATIENT, TWO, etc.).<br>• Other personal demographic-related data (e.g., addresses, phones, IP addresses, etc.) were also changed to be generic.<br>**PDF 508 Compliance—**The final PDF document was recreated and now supports the minimum requirements to be 508 compliant (i.e., accessibility tags, language selection, alternate text for all images/icons, fully functional Web links, successfully passed Adobe Acrobat Quick Check). | Oakland, CA OIFO:<br>• Project Manager: Jack Schram<br>• Technical Writer: Thom Blom |

| | | | |
|---|---|---|---|
| 02/10/05 | 2.3 | Updated the ^%ZTLOAD, REQ^%ZTLOAD and added three new XUTMDEVQ APIs (Kernel Patch XU*8.0*275). | Oakland, CA OIFO:<br>• Project Manager: Jack Schram<br>• Developers: Gary Beuschel and Wally Fort<br>• Technical Writer: Thom Blom |
| 03/02/05 | 2.4 | Corrected various APIs. Reordered all APIs under each category: 1) by routine name and 2) by tag name. | Oakland, CA OIFO:<br>• Project Manager: Jack Schram<br>• Technical Writer: Thom Blom |
| 04/14/05 | 2.5 | Recategorized CRC XLF functions into their own category (i.e., "CRC" vs. "Other"). | Oakland, CA OIFO:<br>• Project Manager: Jack Schram<br>• Technical Writer: Thom Blom |
| 09/22/05 | 2.6 | Updated the SETUP^XQALERT, SETUP^XUSRB, OWNSKEY^XUSRB, DQ^%ZTLOAD, ISQED^%ZTLOAD, KILL^%ZTLOAD, PCLEAR^%ZTLOAD, and STAT^%ZTLOAD APIs.<br>Added the ASKSTOP^%ZTLOAD, DESC^%ZTLOAD, JOB^%ZTLOAD, OPTION^%ZTLOAD, $$PSET^%ZTLOAD, RTN^%ZTLOAD, $$S^%ZTLOAD and ZTSAVE^%ZTLOAD APIs. | Oakland, CA OIFO:<br>• Project Manager: Jack Schram<br>• Developer: Wally Fort and Joel Ivey<br>• Technical Writer: Thom Blom |
| 09/28/05 | 2.7 | Added the $$HANDLE^XUSRB4 and REQQ^XUTMDEVQ APIs. | Oakland, CA OIFO:<br>• Project Manager: Jack Schram<br>• Developer: Wally Fort<br>• Technical Writer: Thom Blom |
| 10/19/05 | 2.8 | Updated the SETUP^XQALERT API based on feedback from user community and developer. | Oakland, CA OIFO:<br>• Project Manager: Jack Schram<br>• Developers: Wally Fort and Joel Ivey<br>• Technical Writer: Thom Blom |

| | | | |
|---|---|---|---|
| 12/15/05 | 2.9 | Added the following APIs (via patches currently not yet released):<br>• $$CREATE^XUSAP (XU*8.0*361)<br>• $$SENTCASE^XLFSTR (XU*8.0*400)<br>• $$TITLE^XLFSTR (XU*8.0*400)<br>• Changed Job^%ZTLOAD to $$JOB^%ZTLOAD | Oakland, CA OIFO:<br>• Project Manager: Jack Schram<br>• Developer: Wally Fort<br>• Technical Writer: Thom Blom |
| 01/23/06 | 3.0 | Updated the following APIs:<br>• $$QQ^XUTMDEVQ, updated description (XU*8.0*389)<br>• Changed REQQ^XUTMDEVQ to $$REQQ^XUTMDEVQ; updated description (XU*8.0*389)<br>• Updated REQ^%ZTLOAD and ^%ZTLOAD APIs.<br>• Changed $$SENTCASE^XLFSTR to $$SENTENCE^XLFSTR (XU*8.0*400) | Oakland, CA OIFO:<br>• Project Manager: Jack Schram<br>• Developers: Gary Beuschel and Wally Fort<br>• Technical Writer: Thom Blom |
| 06/20/06 | 3.1 | Updates:<br>• Corrected output array subscript in the F4^XUAF4 API from "STATION NUMER" to "STATION NUMBER (Remedy #HD0000000147298)<br>• Updated document format to follow latest ISS Guidelines and SOP. | Oakland, CA OIFO:<br>• Project Manager: Jack Schram<br>• Technical Writer: Thom Blom |

**Table i. Documentation revision history**

# Patch Revisions

For the current patch history related to this software, please refer to the Patch Module on FORUM.

# Contents

Revision History ..... iii
Figures and Tables ..... xxiii
Orientation ..... xxvii

**1. Introduction ..... 1-1**

**2. Address Hygiene: Programmer Tools ..... 2-1**

Application Program Interface (API) ..... 2-1
CCODE^XIPUTIL(): FIPS Code Data ..... 2-1
$$FIPS^XIPUTIL(): FIPS Code for ZIP Code ..... 2-2
$$FIPSCHK^XIPUTIL(): Check for FIPS Code ..... 2-3
POSTAL^XIPUTIL(): ZIP Code Information ..... 2-4
POSTALB^XIPUTIL(): Active ZIP Codes ..... 2-6

**3. Alerts: Programmer Tools ..... 3-1**

Package Identifier versus Alert Identifier ..... 3-2
Package Identifier Conventions ..... 3-2
Glossary of Terms for Alerts ..... 3-3
Application Program Interface (API) ..... 3-4
AHISTORY^XQALBUTL(): Get Alert Tracking File Information ..... 3-4
ALERTDAT^XQALBUTL(): Get Alert Tracking File Information ..... 3-6
DELSTAT^XQALBUTL(): Get User Information and Status for Recent Alert ..... 3-7
NOTIPURG^XQALBUTL(): Purge Alerts Based on Code ..... 3-8
$$PENDING^XQALBUTL(): Pending Alerts for a User ..... 3-9
$$PKGPEND^XQALBUTL(): Pending Alerts for a User in Specified Software ..... 3-10
PTPURG^XQALBUTL(): Purge Alerts Based on Patient ..... 3-11
RECIPURG^XQALBUTL(): Purge User Alerts ..... 3-12
USERDATA^XQALBUTL(): Get User Information for an Alert ..... 3-13
USERLIST^XQALBUTL(): Get Recipient Information for an Alert ..... 3-14
ACTION^XQALERT(): Process an Alert ..... 3-15
DELETE^XQALERT(): Clear Obsolete Alerts ..... 3-16
DELETEA^XQALERT(): Clear Obsolete Alerts ..... 3-17
GETACT^XQALERT(): Return Alert Variables ..... 3-18
PATIENT^XQALERT(): Get Alerts for a Patient ..... 3-19

SETUP^XQALERT: Send Alerts ..... 3-21
$$SETUP1^XQALERT: Send Alerts ..... 3-25
USER^XQALERT(): Get Alerts for a User ..... 3-30
FORWARD^XQALFWD(): Forward Alerts ..... 3-31
$$CURRSURO^XQALSURO(): Get Current Surrogate for Alerts ..... 3-33
$$GETSURO^XQALSURO(): Get Current Surrogate Information ..... 3-34
REMVSURO^XQALSURO(): Remove Surrogates for Alerts ..... 3-35
SETSURO1^XQALSURO(): Establish a Surrogate for Alerts ..... 3-36

**4. Device Handler: Programmer Tools ..... 4-1**

Application Program Interface (API) ..... 4-2
DEVICE^XUDHGUI(): GUI Device Lookup ..... 4-2
$$RES^XUDHSET(): Set Up Resource Device ..... 4-5
^%ZIS: Standard Device Call ..... 4-6
HLP1^%ZIS: Display Brief Device Help ..... 4-17
HLP2^%ZIS: Display Device Help Frames ..... 4-18
HOME^%ZIS: Reset Home Device IO Variables ..... 4-19
$$REWIND^%ZIS(): Rewind Devices ..... 4-21
^%ZISC: Close Device ..... 4-22
PKILL^%ZISP: Kill Special Printer Variables ..... 4-23
PSET^%ZISP: Set Up Special Printer Variables ..... 4-24
ENDR^%ZISS: Set Up Specific Screen Handling Variables ..... 4-25
ENS^%ZISS: Set Up Screen-handling Variables ..... 4-26
GKILL^%ZISS: KILL Graphic Variables ..... 4-33
GSET^%ZISS: Set Up Graphic Variables ..... 4-34
KILL^%ZISS: KILL Screen Handling Variables ..... 4-35
CALL^%ZISTCP: Make TCP/IP Connection (Remote System) ..... 4-36
CLOSE^%ZISTCP: Close TCP/IP Connection (Remote System) ..... 4-37
CLOSE^%ZISUTL(): Close Device with Handle ..... 4-38
OPEN^%ZISUTL(): Open Device with Handle ..... 4-39
RMDEV^%ZISUTL(): Delete Data Given a Handle ..... 4-41
SAVDEV^%ZISUTL(): Save Data Given a Handle ..... 4-42
USE^%ZISUTL(): Use Device Given a Handle ..... 4-43
Special Device Issues ..... 4-44
Form Feeds ..... 4-44

Resources ..... 4-47

**5. Domain Name Service (DNS): Programmer Tools ..... 5-1**

Application Program Interface (API) ..... 5-1

$$ADDRESS^XLFNSLK(): Convert Domain Name to IP Addresses ..... 5-1

MAIL^XLFNSLK(): Get IP Addresses for a Domain Name ..... 5-2

**6. Electronic Signatures: Programmer Tools ..... 6-1**

Application Program Interface (API) ..... 6-1

^XUSESIG: Set Up Electronic Signature Code ..... 6-1

SIG^XUSESIG(): Verify Electronic Signature Code ..... 6-2

$$CHKSUM^XUSESIG1(): Build Checksum for Global Root ..... 6-2

$$CMP^XUSESIG1(): Compare Checksum to $Name_Value ..... 6-3

$$DE^XUSESIG1(): Decode String ..... 6-4

$$EN^XUSESIG1(): Encode Esblock ..... 6-4

$$ESBLOCK^XUSESIG1(): E-Sig Fields Required for Hash ..... 6-5

DE^XUSHSHP(): Decrypt Data String ..... 6-6

EN^XUSHSHP(): Encrypt Data String ..... 6-7

HASH^XUSHSHP(): Hash Electronic Signature Code ..... 6-8

**7. Error Processing: Programmer Tools ..... 7-1**

Application Program Interface (API) ..... 7-1

$$EC^%ZOSV: Get Error Code ..... 7-1

^%ZTER: Kernel Standard Error Recording Routine ..... 7-2

$$NEWERR^%ZTER: Verify Support of Standard Error Trapping (Obsolete) ..... 7-4

UNWIND^%ZTER: Quit Back to Calling Routine ..... 7-5

**8. Field Monitoring: Programmer Tools ..... 8-1**

Application Program Interface (API) ..... 8-1

OPKG^XUHUI(): Monitor New Style Cross-referenced Fields ..... 8-1

**9. File Access Security: Programmer Tools ..... 9-1**

Field Level Protection ..... 9-1

File Navigation ..... 9-1

Use of DLAYGO When Navigating to Files ..... 9-2

Use of DLAYGO in ^DIC Calls ..... 9-2

Use of DIDEL in ^DIE Calls ..... 9-3

**10. Help Processor: Programmer Tools ..... 10-1**

Entry and Exit Execute Statements ..... 10-1
Link to the OBJECT File ..... 10-1
Application Program Interface (API) ..... 10-2
EN^XQH(): Display Help Frames ..... 10-2
EN1^XQH(): Display Help Frames ..... 10-3
ACTION^XQH4(): Print Help Frame Tree ..... 10-4

**11. Host Files: Programmer Tools ..... 11-1**

Application Program Interface (API) ..... 11-1
CLOSE^%ZISH(): Close Host File ..... 11-2
$$DEFDIR^%ZISH(): Get Default Host File Directory ..... 11-3
$$DEL^%ZISH(): Delete Host File ..... 11-4
$$FTG^%ZISH(): Load Host File into Global ..... 11-5
$$GATF^%ZISH(): Copy Global to Host File ..... 11-7
$$GTF^%ZISH(): Copy Global to Host File ..... 11-8
$$LIST^%ZISH(): List Directory ..... 11-9
$$MV^%ZISH(): Rename Host File ..... 11-10
OPEN^%ZISH(): Open Host File ..... 11-11
$$PWD^%ZISH: Get Current Directory ..... 11-12
$$STATUS^%ZISH: Return End-of-File Status ..... 11-13

**12. Institution File: Programmer Tools ..... 12-1**

Application Program Interface (API) ..... 12-1
$$ACTIVE^XUAF4(): Institution Active Facility (True/False) ..... 12-1
CDSYS^XUAF4(): Coding System Name ..... 12-2
CHILDREN^XUAF4(): List of Child Institutions for a Parent ..... 12-2
$$CIRN^XUAF4(): Institution CIRN-enabled Field Value ..... 12-4
F4^XUAF4(): Institution Data for a Station Number ..... 12-4
$$ID^XUAF4(): Institution Identifier ..... 12-6
$$IDX^XUAF4(): Institution IEN (Using Coding System & ID) ..... 12-7
$$IEN^XUAF4(): IEN for Station Number ..... 12-8
$$LEGACY^XUAF4(): Institution Realigned/Legacy (True/False) ..... 12-9
$$LKUP^XUAF4(): Institution Lookup ..... 12-10
LOOKUP^XUAF4(): Look Up Institution Identifier ..... 12-11
$$MADD^XUAF4(): Institution Mailing Address ..... 12-12

$$NAME^XUAF4(): Institution Official Name ..... 12-13
$$NNT^XUAF4(): Institution Station Number, Name, and Type ..... 12-14
$$NS^XUAF4(): Institution Name and Station Number ..... 12-14
$$O99^XUAF4(): IEN of Merged Station Number ..... 12-15
$$PADD^XUAF4(): Institution Physical Address ..... 12-16
PARENT^XUAF4(): Parent Institution Lookup ..... 12-17
$$PRNT^XUAF4(): Institution Parent Facility ..... 12-18
$$RF^XUAF4(): Realigned From Institution Information ..... 12-19
$$RT^XUAF4(): Realigned To Institution Information ..... 12-20
SIBLING^XUAF4(): Sibling Institution Lookup ..... 12-21
$$STA^XUAF4(): Station Number for IEN ..... 12-22
$$TF^XUAF4(): Treating Facility (True/False) ..... 12-22
$$WHAT^XUAF4(): Institution Single Field Information ..... 12-23
$$IEN^XUMF(): Institution IEN (Using IFN, Coding System, & ID) ..... 12-24
MAIN^XUMFI(): HL7 Master File Message Builder ..... 12-25
MAIN^XUMFP(): Master File Parameters ..... 12-27

**13. KIDS: Programmer Tools ..... 13-1**

KIDS Build-related Options ..... 13-1
Creating Builds ..... 13-1
Build Entries ..... 13-2
Create a Build Using Namespace ..... 13-3
Copy Build to Build ..... 13-4
Edit a Build ..... 13-4
Transporting a Distribution ..... 13-20
Creating Transport Globals that Install Efficiently ..... 13-25
Advanced Build Techniques ..... 13-26
Environment Check Routine ..... 13-26
PRE-TRANSPORTATION ROUTINE field (#900) ..... 13-31
Pre- and Post-Install Routines: Special Features ..... 13-32
Edit a Build—Screen 4 ..... 13-34
How to Ask Installation Questions ..... 13-35
Using Checkpoints (Pre- and Post-Install Routines) ..... 13-38
Required Builds ..... 13-42
Package File Link ..... 13-43

Track Package Nationally ..... 13-45
Alpha/Beta Tracking ..... 13-45
Application Program Interface (API) ..... 13-49
EXIT^XPDID(): Restore Screen, Clean Up Variables, and Display Text ..... 13-49
INIT^XPDID: Initialize Device and Draw Box Borders ..... 13-50
TITLE^XPDID(): Display Title Text ..... 13-50
UPDATE^XPDID(): Update Install Progress Bar ..... 13-51
EN^XPDIJ(): Task Off KIDS Install ..... 13-52
$$PKGPAT^XPDIP(): Update Patch History ..... 13-53
BMES^XPDUTL(): Output a Message with Blank Line ..... 13-54
$$COMCP^XPDUTL(): Complete Checkpoint ..... 13-55
$$CURCP^XPDUTL(): Get Current Checkpoint Name/IEN ..... 13-56
$$LAST^XPDUTL(): Last Software Patch ..... 13-57
MES^XPDUTL(): Output a Message ..... 13-58
$$NEWCP^XPDUTL(): Create Checkpoint ..... 13-59
$$OPTDE^XPDUTL(): Disable/Enable an Option ..... 13-60
$$PARCP^XPDUTL(): Get Checkpoint Parameter ..... 13-61
$$PATCH^XPDUTL(): Verify Patch Installation ..... 13-62
$$PKG^XPDUTL(): Parse Software Name From Build Name ..... 13-62
$$PRODE^XPDUTL(): Disable/Enable a Protocol ..... 13-63
$$RTNUP^XPDUTL(): Update Routine Action ..... 13-64
$$UPCP^XPDUTL(): Update Checkpoint ..... 13-65
$$VER^XPDUTL(): Parse Version from Build Name ..... 13-66
$$VERCP^XPDUTL(): Verify Checkpoint ..... 13-67
$$VERSION^XPDUTL(): Package File Current Version ..... 13-68

**14. Menu Manager: Programmer Tools ..... 14-1**

Creating Options ..... 14-1
Option Types ..... 14-1
Creating Options (Edit Options) ..... 14-1
Variables for Programmer Use ..... 14-2
Direct Mode Utilities ..... 14-4
^XQ1: Test an Option ..... 14-4
Application Program Interface (API) ..... 14-5
$$ADD^XPDMENU(): Add Option to Menu ..... 14-5

DELETE^XPDMENU(): Delete Menu Item ..... 14-6
LKOPT^XPDMENU(): Look Up Option IEN ..... 14-7
OUT^XPDMENU(): Edit Option's Out of Order Message ..... 14-8
RENAME^XPDMENU(): Rename Option ..... 14-9
NEXT^XQ92(): Restricted Times Check ..... 14-9
$$ACCESS^XQCHK(): User Option Access Test ..... 14-10
OP^XQCHK(): Current Option Check ..... 14-12

**15. Miscellaneous: Programmer Tools ..... 15-1**

Application Program Interface (API) ..... 15-1
^XQDATE: Convert $H to VA FileMan Format (Obsolete) ..... 15-1
$$EN^XUA4A71(): Convert String to Soundex ..... 15-2
^XUWORKDY: Workday Calculation (Obsolete) ..... 15-3

**16. Name Standardization: Programmer Tools ..... 16-1**

Application Program Interface (API) ..... 16-1
$$BLDNAME^XLFNAME(): Build Name from Component Parts ..... 16-1
$$CLEANC^XLFNAME(): Name Component Standardization Routine ..... 16-4
$$FMNAME^XLFNAME(): Convert HL7 Formatted Name to Name ..... 16-6
$$HLNAME^XLFNAME(): Convert Name to HL7 Formatted Name ..... 16-9
NAMECOMP^XLFNAME(): Component Parts from Standard Name ..... 16-12
$$NAMEFMT^XLFNAME(): Formatted Name from Name Components ..... 16-13
STDNAME^XLFNAME(): Name Standardization Routine ..... 16-18
DELCOMP^XLFNAME2(): Delete Name Components Entry ..... 16-23
UPDCOMP^XLFNAME2(): Update Name Components Entry ..... 16-24

**17. Operating System Interface: Programmer Tools ..... 17-1**

Application Program Interface (API) ..... 17-1
^%ZOSF(): Operating System-dependent Logic Global ..... 17-1
$$ACTJ^%ZOSV: Number of Active Jobs ..... 17-4
$$AVJ^%ZOSV: Number of Available Jobs ..... 17-4
DOLRO^%ZOSV: Display Local Variables ..... 17-5
GETENV^%ZOSV: Current System Information ..... 17-6
$$LGR^%ZOSV: Last Global Reference ..... 17-6
LOGRSRC^%ZOSV(): Record Resource Usage (RUM) ..... 17-7
$$OS^%ZOSV: Get Operating System Information ..... 17-8

SETENV^%ZOSV: Set VMS Process Name (Caché/OpenVMS Systems) ..... 17-9
SETNM^%ZOSV(): Set VMS Process Name (Caché/OpenVMS Systems) ..... 17-10
T0^%ZOSV: Start RT Measure (Obsolete) ..... 17-11
T1^%ZOSV: Stop RT Measure (Obsolete) ..... 17-12
$$VERSION^%ZOSV(): Get OS Version Number or Name ..... 17-13

**18. Security Keys: Programmer Tools ..... 18-1**

Key Lookup ..... 18-1
Person Lookup ..... 18-1
Application Program Interface (API) ..... 18-2
DEL^XPDKEY(): Delete Security Key ..... 18-2
$$LKUP^XPDKEY(): Look Up Security Key Value ..... 18-3
$$RENAME^XPDKEY(): Rename Security Key ..... 18-4
OWNSKEY^XUSRB(): Verify Security Keys Assigned to a User ..... 18-5

**19. Server Options: Programmer Tools ..... 19-1**

Tools for Processing Server Requests ..... 19-1
Key Variables When a Server Option is Running ..... 19-1
Appending Text to a Server Request Bulletin or Mailman Reply ..... 19-1
Customizing a Server Request Bulletin ..... 19-2

**20. Signon/Security: Programmer Tools ..... 20-1**

Direct Mode Utilities ..... 20-1
^XUP: Programmer Signon ..... 20-1
^XUS: User Signon: No Error Trapping ..... 20-1
H^XUS: Programmer Halt ..... 20-2
^XUSCLEAN: Programmer Halt ..... 20-2
^ZU: User Signon ..... 20-2
XU USER SIGN-ON Option ..... 20-3
XU USER SIGN-ON: Package-specific Signon Actions ..... 20-3
XU USER TERMINATE Option ..... 20-4
Discontinuation of USER TERMINATE ROUTINE ..... 20-4
Creating a Package-specific User Termination Action ..... 20-4
Application Program Interface (API) ..... 20-6
$$GET^XUPARAM(): Get Parameters ..... 20-6
$$KSP^XUPARAM(): Return Kernel Site Parameter ..... 20-7
$$LKUP^XUPARAM(): Look Up Parameters ..... 20-8

SET^XUPARAM(): Set Parameters....20-8
$$PROD^XUPROD(): Production Vs. Test Account....20-9
H^XUS: Programmer Halt....20-10
SET^XUS1A(): Output Message During Signon....20-10
AVHLPTXT^XUS2: Get Help Text....20-11
$$CREATE^XUSAP: Create Application Proxy User....20-12
KILL^XUSCLEAN: Clear all but Kernel Variables....20-14
$$ADD^XUSERNEW(): Add New Users....20-15
$$CHECKAV^XUSRB(): Check Access/Verify Codes....20-17
CVC^XUSRB: VistALink—Change User's Verify Code....20-18
$$INHIBIT^XUSRB: Check if Logons Inhibited....20-18
INTRO^XUSRB: VistALink—Get Introductory Text....20-19
LOGOUT^XUSRB: VistALink—Log Out User From M....20-19
SETUP^XUSRB(): VistALink—Set Up User's Partition in M....20-20
VALIDAV^XUSRB(): VistALink—Validate User Credentials....20-21
$$DECRYP^XUSRB1(): Decrypt String....20-21
$$ENCRYP^XUSRB1(): Encrypt String....20-22
$$HANDLE^XUSRB4(): Return Unique Session ID String....20-23
^XUVERIFY(): Verify Access and Verify Codes....20-24
$$CHECKAV^XUVERIFY(): Check Access/Verify Codes....20-25
WITNESS^XUVERIFY(): Return IEN of Users with A/V Codes & Security Keys....20-26
GETPEER^%ZOSV: VistALink—Get IP Address for Current Session....20-26

**21. Spooling: Programmer Tools....21-1**

Application Program Interface (API)....21-2
DSD^ZISPL: Delete Spool Data File Entry....21-2
DSDOC^ZISPL: Delete Spool Document File Entry....21-2

**22. TaskMan: Programmer Tools....22-1**

How to Write Code to Queue Tasks....22-1
Queuers....22-1
Tasks....22-3
Application Program Interface (API)....22-11
$$DEV^XUTMDEVQ(): Force Queueing—Ask for Device....22-11
EN^XUTMDEVQ(): Run a Task (Directly or Queued)....22-13
$$NODEV^XUTMDEVQ(): Force Queueing—No Device Selection....22-15

$$QQ^XUTMDEVQ(): Double Queue—Direct Queuing in a Single Call .....22-17
$$REQQ^XUTMDEVQ(): Schedule Second Part of a Task .....22-20
DISP^XUTMOPT(): Display Option Schedule.....22-22
EDIT^XUTMOPT(): Edit an Option's Scheduling .....22-22
OPTSTAT^XUTMOPT(): Obtain Option Schedule .....22-23
RESCH^XUTMOPT(): Set Up Option Schedule.....22-24
EN^XUTMTP(): Display HL7 Task Information .....22-25
^%ZTLOAD: Queue a Task.....22-26
$$ASKSTOP^%ZTLOAD: Stop TaskMan Task .....22-34
DESC^%ZTLOAD(): Find Tasks with a Description.....22-35
DQ^%ZTLOAD: Unschedule a Task.....22-35
ISQED^%ZTLOAD: Return Task Status.....22-36
$$JOB^%ZTLOAD(): Return a Job Number for a Task.....22-38
KILL^%ZTLOAD: Delete a Task.....22-39
OPTION^%ZTLOAD(): Find Tasks for an Option.....22-40
PCLEAR^%ZTLOAD(): Clear Persistent Flag for a Task.....22-40
$$PSET^%ZTLOAD(): Set Task as Persistent .....22-41
REQ^%ZTLOAD: Requeue a Task .....22-42
RTN^%ZTLOAD(): Find Tasks that Call a Routine.....22-47
$$S^%ZTLOAD(): Check for Task Stop Request .....22-48
STAT^%ZTLOAD: Task Status .....22-49
$$TM^%ZTLOAD: Check if TaskMan is Running.....22-51
ZTSAVE^%ZTLOAD(): Build ZTSAVE Array.....22-51

**23. Unwinder: Programmer Tools.....23-1**

Application Program Interface (API).....23-1
EN^XQOR(): Navigating Protocols .....23-1
EN1^XQOR(): Navigating Protocols .....23-2
MSG^XQOR(): Enable HL7 Messaging.....23-3
EN^XQORM(): Menu Item Display and Selection.....23-4
XREF^XQORM(): Force Menu Recompile.....23-5
DISP^XQORM1(): Display Menu Selections From Help Code .....23-5

**24. User: Programmer Tools.....24-1**

Application Program Interface (API).....24-1
$$CODE2TXT^XUA4A72(): Get HCFA Text.....24-1

$$GET^XUA4A72(): Get Specialty and Subspecialty for a User....24-2
$$IEN2CODE^XUA4A72(): Get VA Code....24-3
$$DTIME^XUP(): Reset DTIME for USER....24-4
$$ACTIVE^XUSER(): Status Indicator....24-6
$$DEA^XUSER(): Get DEA Number....24-8
DIV4^XUSER(): Get User Divisions....24-11
$$LOOKUP^XUSER(): New Person File Lookup....24-12
$$NAME^XUSER(): Get Name of User....24-14
$$PROVIDER^XUSER(): Providers in New Person File....24-15
$$KCHK^XUSRB(): Check If User Holds Security Key....24-16
DIVGET^XUSRB2(): Get Divisions for Current User....24-17
DIVSET^XUSRB2(): Set Division for Current User....24-18
USERINFO^XUSRB2(): Get Demographics for Current User....24-19

**25. XGF Function Library: Programmer Tools....25-1**

Direct Mode Utilities....25-1
^XGFDEMO: Demo Program....25-1
Application Program Interface (API)....25-2
CHGA^XGF(): Screen Change Attributes....25-2
CLEAN^XGF: Screen/Keyboard Exit and Cleanup....25-4
CLEAR^XGF(): Screen Clear Region....25-5
FRAME^XGF(): Screen Frame....25-6
INITKB^XGF(): Keyboard Setup Only....25-7
IOXY^XGF(): Screen Cursor Placement....25-8
PREP^XGF(): Screen/Keyboard Setup....25-9
$$READ^XGF(): Read Using Escape Processing....25-10
RESETKB^XGF: Exit XGF Keyboard....25-12
RESTORE^XGF(): Screen Restore....25-13
SAVE^XGF(): Screen Save....25-14
SAY^XGF(): Screen String....25-15
SAYU^XGF(): Screen String with Attributes....25-17
SETA^XGF(): Screen Video Attributes....25-19
WIN^XGF(): Screen Text Window....25-21

**26. XLF Function Library: Programmer Tools....26-1**

Application Program Interface (API)....26-1

CRC Functions—XLFCRC ..... 26-1
$$CRC16^XLFCRC(): Cyclic Redundancy Code 16 ..... 26-1
$$CRC32^XLFCRC(): Cyclic Redundancy Code 32 ..... 26-3
Date Functions—XLFDT ..... 26-4
$$%H^XLFDT(): Convert Seconds to $H ..... 26-4
$$DOW^XLFDT(): Day of Week ..... 26-5
$$DT^XLFDT: Current Date (VA FileMan Date Format) ..... 26-6
$$FMADD^XLFDT(): VA FileMan Date Add ..... 26-6
$$FMDIFF^XLFDT(): VA FileMan Date Difference ..... 26-7
$$FMTE^XLFDT(): Convert VA FileMan Date to External Format ..... 26-9
$$FMTH^XLFDT(): Convert VA FileMan Date to $H ..... 26-13
$$FMTHL7^XLFDT(): Convert VA FileMan Date to HL7 Date ..... 26-14
$$HADD^XLFDT(): $H Add ..... 26-15
$$HDIFF^XLFDT(): $H Difference ..... 26-16
$$HL7TFM^XLFDT(): Convert HL7 Date to VA FileMan Date ..... 26-17
$$HTE^XLFDT(): Convert $H to External Format ..... 26-19
$$HTFM^XLFDT(): Convert $H to VA FileMan Date Format ..... 26-21
$$NOW^XLFDT: Current Date and Time (VA FileMan Format) ..... 26-22
$$SCH^XLFDT(): Next Scheduled Runtime ..... 26-23
$$SEC^XLFDT(): Convert $H/VA FileMan date to Seconds ..... 26-25
$$TZ^XLFDT: Time Zone Offset (GMT) ..... 26-26
$$WITHIN^XLFDT(): Checks Dates/Times Within Schedule ..... 26-27
Hyperbolic Trigonometric Functions—XLFHYPER ..... 26-28
$$ACOSH^XLFHYPER(): Hyperbolic Arc-cosine ..... 26-28
$$ACOTH^XLFHYPER(): Hyperbolic Arc-cotangent ..... 26-29
$$ACSCH^XLFHYPER(): Hyperbolic Arc-cosecant ..... 26-29
$$ASECH^XLFHYPER(): Hyperbolic Arc-secant ..... 26-30
$$ASINH^XLFHYPER(): Hyperbolic Arc-sine ..... 26-31
$$ATANH^XLFHYPER(): Hyperbolic Arc-tangent ..... 26-31
$$COSH^XLFHYPER(): Hyperbolic Cosine ..... 26-32
$$COTH^XLFHYPER(): Hyperbolic Cotangent ..... 26-33
$$CSCH^XLFHYPER(): Hyperbolic Cosecant ..... 26-33
$$SECH^XLFHYPER(): Hyperbolic Secant ..... 26-34
$$SINH^XLFHYPER(): Hyperbolic Sine ..... 26-35
$$TANH^XLFHYPER(): Hyperbolic Tangent ..... 26-36

Mathematical Functions—XLFMTH....26-37
$$ABS^XLFMTH(): Absolute Value....26-37
$$ACOS^XLFMTH(): Arc-cosine (Radians)....26-38
$$ACOSDEG^XLFMTH(): Arc-cosine (Degrees)....26-38
$$ACOT^XLFMTH(): Arc-cotangent (Radians)....26-39
$$ACOTDEG^XLFMTH(): Arc-cotangent (Degrees)....26-40
$$ACSC^XLFMTH(): Arc-cosecant (Radians)....26-40
$$ACSCDEG^XLFMTH(): Arc-cosecant (Degrees)....26-41
$$ASEC^XLFMTH(): Arc-secant (Radians)....26-42
$$ASECDEG^XLFMTH(): Arc-secant (Degrees)....26-42
$$ASIN^XLFMTH(): Arc-sine (Radians)....26-43
$$ASINDEG^XLFMTH(): Arc-sine (Degrees)....26-44
$$ATAN^XLFMTH(): Arc-tangent (Radians)....26-44
$$ATANDEG^XLFMTH(): Arc-tangent (Degrees)....26-45
$$COS^XLFMTH(): Cosine (Radians)....26-46
$$COSDEG^XLFMTH(): Cosine (Degrees)....26-46
$$COT^XLFMTH(): Cotangent (Radians)....26-47
$$COTDEG^XLFMTH(): Cotangent (Degrees)....26-48
$$CSC^XLFMTH(): Cosecant (Radians)....26-48
$$CSCDEG^XLFMTH(): Cosecant (Degrees)....26-49
$$DECDMS^XLFMTH(): Convert Decimals to Degrees:Minutes:Seconds....26-50
$$DMSDEC^XLFMTH(): Convert Degrees:Minutes:Seconds to Decimal....26-51
$$DTR^XLFMTH(): Convert Degrees to Radians....26-51
$$E^XLFMTH(): e—Natural Logarithm....26-52
$$EXP^XLFMTH(): e—Natural Logarithm to the Nth Power....26-53
$$LN^XLFMTH(): Natural Log (Base e)....26-53
$$LOG^XLFMTH(): Logarithm (Base 10)....26-54
$$MAX^XLFMTH(): Maximum of Two Numbers....26-55
$$MIN^XLFMTH(): Minimum of Two Numbers....26-56
$$PI^XLFMTH(): PI....26-56
$$PWR^XLFMTH(): X to the Y Power....26-57
$$RTD^XLFMTH(): Convert Radians to Degrees....26-58
$$SD^XLFMTH(): Standard Deviation....26-59
$$SEC^XLFMTH(): Secant (Radians)....26-60
$$SECDEG^XLFMTH(): Secant (Degrees)....26-60

$$SIN^XLFMTH(): Sine (Radians) ....26-61
$$SINDEG^XLFMTH(): Sine (Degrees) ....26-62
$$SQRT^XLFMTH(): Square Root ....26-62
$$TAN^XLFMTH(): Tangent (Radians) ....26-63
$$TANDEG^XLFMTH(): Tangent (Degrees) ....26-64
Measurement Functions—XLFMSMT ....26-65
$$BSA^XLFMSMT(): Body Surface Area Measurement ....26-65
$$LENGTH^XLFMSMT(): Convert Length Measurement ....26-66
$$TEMP^XLFMSMT(): Convert Temperature Measurement ....26-67
$$VOLUME^XLFMSMT(): Convert Volume Measurement ....26-68
$$WEIGHT^XLFMSMT(): Convert Weight Measurement ....26-70
String Functions—XLFSTR ....26-72
$$CJ^XLFSTR(): Center Justify String ....26-72
$$INVERT^XLFSTR(): Invert String ....26-73
$$LJ^XLFSTR(): Left Justify String ....26-73
$$LOW^XLFSTR(): Convert String to Lowercase ....26-74
$$REPEAT^XLFSTR(): Repeat String ....26-75
$$REPLACE^XLFSTR(): Replace Strings ....26-76
$$RJ^XLFSTR(): Right Justify String ....26-77
$$SENTENCE^XLFSTR(): Convert String to Sentence Case ....26-78
$$STRIP^XLFSTR(): Strip a String ....26-78
$$TITLE^XLFSTR(): Convert String to Title Case ....26-79
$$TRIM^XLFSTR(): Trim String ....26-80
$$UP^XLFSTR(): Convert String to Uppercase ....26-81
Utility Functions—XLFUTL ....26-82
$$BASE^XLFUTL(): Convert Between Two Bases ....26-82
$$CCD^XLFUTL(): Append Check Digit ....26-83
$$CNV^XLFUTL(): Convert Base 10 to Another Base ....26-84
$$DEC^XLFUTL(): Convert Another Base to Base 10 ....26-85
$$VCD^XLFUTL(): Verify Integrity ....26-85

**27. XML: Programmer Tools ....27-1**

Application Program Interface (API) ....27-1
$$ATTRIB^MXMLDOM(): XML—Get Attribute Name ....27-1
$$CHILD^MXMLDOM(): XML—Get Child Node ....27-2

$$CMNT^MXMLDOM(): XML—Extract Comment Text (True/False) .....27-3
CMNT^MXMLDOM(): XML—Extract Comment Text (True/False) .....27-4
DELETE^MXMLDOM(): XML—Delete Document Instance.....27-5
$$EN^MXMLDOM(): XML—Initial Processing, Build In-memory Image .....27-6
$$NAME^MXMLDOM(): XML—Get Element Name.....27-7
$$PARENT^MXMLDOM(): XML—Get Parent Node.....27-8
$$SIBLING^MXMLDOM(): XML—Get Sibling Node .....27-9
$$TEXT^MXMLDOM(): XML—Get Text (True/False) .....27-10
TEXT^MXMLDOM(): XML—Get Text (True/False) .....27-11
$$VALUE^MXMLDOM(): XML—Get Attribute Value.....27-12
EN^MXMLPRSE(): XML—Event Driven API.....27-13
$$SYMENC^MXMLUTL(): XML—Encoded Strings in Messages .....27-17
$$XMLHDR^MXMLUTL: XML—Message Headers .....27-18
Glossary .....Glossary-1
Appendix A—KIDS Build Checklists ..... A-1
Index .....Index-1

# Figures and Tables

Table i. Documentation revision history ..... vi
Table ii. Documentation symbol descriptions ..... xxvii
Figure 3-1. Alerts: Creating an alert for User #14 ..... 3-1
Figure 3-2. Alerts: Checking that the alert was sent ..... 3-1
Table 3-1. Alerts: Related terms and definitions ..... 3-3
Figure 3-3. Alerts: Call to send an alert sample ..... 3-24
Figure 3-4. Alerts: Resulting alert, from View Alerts option ..... 3-24
Figure 3-5. Alerts: Call to send an alert sample ..... 3-29
Figure 3-6. Alerts: Resulting alert, from View Alerts option ..... 3-29
Figure 3-7. FORWARD^XQALFWD: Example ..... 3-33
Figure 4-1. ^%ZIS: Example ..... 4-15
Figure 4-2. GSET^%ZISS: Example ..... 4-35
Figure 4-3. OPEN^%ZISUTL: Example ..... 4-41
Figure 4-4. Device Handler: Issuing form feeds following current guidelines ..... 4-46
Figure 4-5. Device Handler: Alternate approach following current guidelines ..... 4-46
Figure 7-1. Error Trap: Example ..... 7-3
Figure 7-2. UNWIND^%ZTER: Main code example ..... 7-5
Figure 7-3. UNWIND^%ZTER: Usage ..... 7-5
Figure 8-1. OPKG^XUHUI: Example of creating New Style Cross-references ..... 8-2
Figure 8-2. OPKG^XUHUI API: Sample scenario ..... 8-3
Figure 8-3. OPKG^XUHUI: Example of internal results ..... 8-4
Figure 9-1. File Access Security: Setting DLAYGO in a template ..... 9-2
Figure 11-1. Host Files: Opening a Host file using the ^%ZIS API ..... 11-1
Figure 11-2. CLOSE^%ZISH: Example ..... 11-3
Figure 11-3. Host Files: Overflow lines in a Host file sample ..... 11-5
Figure 11-4. OPEN^%ZISH: Example ..... 11-12
Figure 11-5. $$STATUS^%ZISH: Example ..... 11-13
Figure 13-1. KIDS Edits and Distribution menu options ..... 13-1
Table 13-1. KIDS: Options supporting software application builds and exports ..... 13-2
Figure 13-2. KIDS: Choosing a build type sample ..... 13-3
Table 13-2. KIDS: Kernel V. 8.0 component types (listed alphabetically) ..... 13-3
Figure 13-3. KIDS: Populating a build entry by namespace ..... 13-4

Figure 13-4. KIDS: Copying a build entry ....13-4
Table 13-3. KIDS: Functional layout, Edit a Build ....13-6
Figure 13-5. KIDS: Screen 1 of Edit a Build sample....13-7
Figure 13-6. KIDS: Screen 2 of Edit a Build: Selecting files ....13-9
Figure 13-7. KIDS: Data dictionary and data settings ....13-9
Figure 13-8. KIDS: Data dictionary settings screen ....13-11
Figure 13-9. KIDS: Partial DD—Choosing DD levels (top level and Multiple) to send....13-11
Figure 13-10. KIDS: Settings for sending data....13-12
Table 13-4. KIDS: Data installation actions ....13-13
Figure 13-11. KIDS: Screen 3 of Edit a Build: Components....13-17
Table 13-5. KIDS: Option and protocol installation actions....13-18
Figure 13-12. KIDS: Choosing routines ....13-19
Figure 13-13. KIDS: Selecting templates ....13-20
Figure 13-14. KIDS: Transport a Distribution option—Creating a distribution sample user dialogue .13-21
Figure 13-15. KIDS: Transport a Distribution option—Sending via network (PackMan message) sample user dialogue ....13-22
Figure 13-16. KIDS: Splitting a distribution across diskettes (1 of 2)....13-22
Figure 13-17. KIDS: Splitting a distribution across diskettes (2 of 2)....13-23
Figure 13-18. KIDS: Multi-package builds sample ....13-24
Figure 13-19. KIDS: Exporting global distributions sample ....13-25
Table 13-6. KIDS: Key variables during the environment check....13-27
Table 13-7. KIDS: Actions based on environment check conclusions....13-29
Figure 13-20. KIDS: Dialog when the XPDNOQUE variable is set to disable queuing....13-29
Figure 13-21. KIDS: "DISABLE" default prompt during installations....13-29
Table 13-8. KIDS: Installation—XPDDIQ array sample ....13-30
Figure 13-22. KIDS :"MOVE routines" default prompt during installations ....13-30
Table 13-9. KIDS: Environment Check—XPDDIQ array sample ....13-30
Figure 13-23. KIDS: Environment Check routine sample....13-31
Figure 13-24. KIDS: PRE-TRANSPORTATION ROUTINE field sample ....13-32
Table 13-10. KIDS: Key variables during the pre- and post-install routines....13-33
Figure 13-25. KIDS: Screen 4 of Edit a Build sample....13-35
Table 13-11. KIDS: DIR input values for KIDS install questions....13-35
Figure 13-26. KIDS: Pre-install question (setting up) sample ....13-37
Figure 13-27. KIDS: Appearance of question during installation ....13-38
Table 13-12. KIDS: Functions using checkpoints *with* callbacks....13-39

Figure 13-28. KIDS: Using checkpoints *with* callbacks: combined pre- and post-install routine .....13-40
Table 13-13. KIDS: Functions using checkpoints *without* callbacks .....13-41
Figure 13-29. KIDS: Required builds sample.....13-42
Table 13-14. KIDS: Required builds installation actions .....13-43
Table 13-15. KIDS: National PACKAGE file field updates .....13-44
Figure 13-30. KIDS: Patch application history sample .....13-44
Table 13-16. KIDS: Alpha/Beta testing options for KIDS edits.....13-46
Figure 13-31. KIDS: Errors Logged in Alpha/Beta Test option.....13-46
Figure 13-32. KIDS: Print Alpha/Beta Errors (Date/Site/Num/Rou/Err) option.....13-47
Figure 14-1. Menu Manager: Edit options [XUEDITOPT].....14-1
Table 17-1. ^%ZOSF: Global nodes.....17-3
Figure 19-1. XQSERVER: Default bulletin .....19-2
Figure 20-1. ZZTALK: Protocol.....20-4
Figure 20-2. $$ADD^XUSERNEW: Example of adding a new user.....20-16
Figure 21-1. Spooling: Sending output to the spooler (and pre-defining ZTIO) .....21-1
Figure 21-2. Spooling: Allowing output to go the spooler (*without* pre-defining ZTIO).....21-1
Table 22-1. TaskMan: ZTREQ piece and equivalent REQ^ZTLOAD variable.....22-7
Figure 22-1. $$DEV^XUTMDEVQ example—Sample code .....22-12
Figure 22-2. EN^XUTMDEVQ: Sample report .....22-14
Figure 22-3. $$NODEV^XUTMDEVQ example—Sample code .....22-16
Figure 22-4. $$QQ^XUTMDEVQ example—Sample code .....22-19
Figure 22-5. $$REQQ^XUTMDEVQ example—Sample code .....22-21
Figure 22-6. TaskMan: Print queuer sample code .....22-30
Figure 22-7. ^%ZTLOAD example—Sample code.....22-33
Figure 22-8. ^%ZTLOAD example—Sample code execution .....22-33
Figure 22-9. ^%ZTLOAD example—Sample output.....22-34
Figure 22-10. REQ^%ZTLOAD example—Sample code.....22-46
Figure 22-11. ^%ZTLOAD example—Sample code execution .....22-46
Figure 22-12. ^%ZTLOAD example—Sample output.....22-47
Figure 24-1. $$LOOKUP^XUSER: Example showing confirmation prompt.....24-13
Figure 24-2. $$LOOKUP^XUSER: Example suppressing confirmation prompt .....24-13
Figure 24-3. $$LOOKUP^XUSER: Example of a terminated user.....24-13
Table 25-1. XGF Function Library: Demo functional division .....25-1
Table 25-2. XGF Function Library: Mnemonics for keys that terminate READs.....25-10
Figure 25-1. SAY^XGF: Example to READ a name .....25-11

Figure 25-2. $$READ^XGF: Example to accept only Up-Arrow and Down-Arrow keys .....25-11
Table 26-1. $$LENGTH^XLFMSMT: Valid units .....26-66
Table 26-2: $$TEMP^XLFMSMT: Valid units .....26-67
Table 26-3. $$VOLUME^XLFMSMT: Valid units .....26-68
Table 26-4. $$WEIGHT^XLFMSMT: Valid units .....26-70

# Orientation

## How to Use this Manual

Throughout this manual, advice and instruction are offered about Kernel Application Programming Interfaces (APIs), Direct Mode Utilities, and other programmer-related information that Kernel V. 8.0 provides for overall Veterans Health Information Systems and Technology Architecture (VistA) application developers.

**CAUTION: To protect the security of VistA systems, distribution of this software for use on any other computer system by VistA sites is prohibited. All requests for copies of Kernel for non-VistA use should be referred to the VistA site's local Office of Information Field Office (OIFO).**

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

This manual uses several methods to highlight different aspects of the material:

- Various symbols are used throughout the documentation to alert the reader to special information. The following table gives a description of each of these symbols:

| Symbol | Description |
|---|---|
| | **NOTE/REF:** Used to inform the reader of general information including references to additional reading material. |
| | **CAUTION or DISCLAIMER:** Used to caution the reader to take special notice of critical information. |

**Table ii. Documentation symbol descriptions**

- Descriptive text is presented in a proportional font (as represented by this font).
- Conventions for displaying TEST data in this document are as follows:
  - The first three digits (prefix) of any Social Security Numbers (SSN) will begin with either "000" or "666".
  - Patient and user names will be formatted as follows: [Application Name]PATIENT,[N] and [Application Name]USER,[N] respectively, where "Application Name" is defined in the Approved Application Abbreviations document and "N" represents the first name as a number spelled out and incremented with each new entry. For example, in Kernel (KRN) test patient and user names would be documented as follows: KRNPATIENT,ONE; KRNPATIENT,TWO; KRNPATIENT,THREE; etc.

- Sample HL7 messages, "snapshots" of computer online displays (i.e., roll-and-scroll screen or character-based screen captures/dialogues) and computer source code, if any, are shown in a *non*-proportional font and enclosed within a box.
  - User's responses to online prompts will be boldface.
  - The "**<Enter>**" found within these snapshots indicate that the user should press the Enter key on their keyboard. Other special keys are represented within angle brackets (< >). For example, pressing the **PF1** key can be represented as pressing **<PF1>**.
  - Author's comments, if any, are displayed in italics or as "callout" boxes.

    **NOTE:** Callout boxes refer to labels or descriptions usually enclosed within a box, which point to specific areas of a displayed image.

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

- The following conventions will be used with regards to APIs:
  - Headings for programmer API descriptions (e.g., supported for use in applications and on the Database Integration Committee [DBIC] list) include the routine tag (if any), the caret ("^") used when calling the routine, and the routine name. The following is an example:

    ```
    EN1^XQH
    ```

  - For APIs that take input parameter, the input parameter will be labeled "required" when it is a required input parameter and labeled "optional" when it is an optional input parameter.
  - For APIs 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:

    ```
    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.
  - All APIs are categorized by function. This categorization is subjective and subject to change based on feedback from the development community. Also, some APIs could fall under multiple categories; however, they are only listed once under a chosen category.

    APIs within a category are first sorted alphabetically by Routine name and then within routine name are sorted alphabetically by Tag reference. The "**$$**", "^", or "**^%**" prefixes on APIs is ignored when alphabetizing.

- All uppercase is reserved for the representation of M code, variable names, or the formal name of options, field/file names, and security keys (e.g., the XUPROGMODE key).

**NOTE:** Other software code (e.g., Delphi/Pascal and Java) variable names and file/folder names can be written in lower or mixed case.

# How to Obtain Technical Information Online

Exported VistA M Server-based software file, routine, and global documentation can be generated through the use of Kernel, MailMan, and VA FileMan utilities.

**NOTE:** Methods of obtaining specific technical information online will be indicated where applicable under the appropriate topic.

**REF:** Please refer to the *Kernel Technical Manual* for further information.

### Help at Prompts

VistA M Server-based software provides online help and commonly used system default prompts. Users are encouraged to enter question marks at any response prompt. At the end of the help display, you are immediately returned to the point from which you started. This is an easy way to learn about any aspect of the software.

### Obtaining Data Dictionary Listings

Technical information about VistA M Server-based files and the fields in files is stored in data dictionaries (DD). You can use the List File Attributes option on the Data Dictionary Utilities submenu in VA FileMan to print formatted data dictionaries.

**REF:** For details about obtaining data dictionaries and about the formats available, please refer to the "List File Attributes" chapter in the "File Management" section of the *VA FileMan Advanced User Manual*.

# Assumptions About the Reader

This manual is written with the assumption that the reader is familiar with the following:

- VistA computing environment:
  - Kernel—VistA M Server software
  - VA FileMan data structures and terminology—VistA M Server software
- Microsoft Windows environment
- M programming language

This manual provides an overall explanation of Kernel and the functionality contained in Kernel V. 8.0. However, no attempt is made to explain how the overall VistA programming system is integrated and maintained. Such methods and procedures are documented elsewhere. We suggest you look at the various VA home pages on the World Wide Web (WWW) and VA Intranet for a general orientation to VistA. For example, go to the Veterans Health Administration (VHA) Office of Information (OI) Health Systems Design & Development (HSD&D) Home Page at the following Intranet Web address:

http://vista.med.va.gov/

## Reference Materials

Readers who wish to learn more about Kernel should consult the following:

- *Kernel Release Notes*
- *Kernel Installation Guide*
- *Kernel Systems Manual*
- *Kernel Programmer Manual* (this manual)
- *Kernel Technical Manual*
- *Kernel Security Tools Manual*
- Kernel Home Page at the following Web address:

  http://vista.med.va.gov/kernel/index.asp

  This site contains other information and provides links to additional documentation.

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 VistA policy are:

- *VA Programming Standards and Conventions (SAC)*
- *MIRMO/OIFO Operations Document*

VistA documentation is made available online in Microsoft Word format and in Adobe Acrobat Portable Document Format (PDF). The PDF documents *must* be read using the Adobe Acrobat Reader (i.e., ACROREAD.EXE), which is freely distributed by Adobe Systems Incorporated at the following Web address:

http://www.adobe.com/

**REF:** For more information on the use of the Adobe Acrobat Reader, please refer to the *Adobe Acrobat Quick Guide* at the following Web address:

http://vista.med.va.gov/iss/acrobat/index.asp

VistA documentation can be downloaded from the Health Systems Design and Development (HSD&D) VistA Documentation Library (VDL) Web site:

http://www.va.gov/vdl/

VistA documentation and software can also be downloaded from the Enterprise VistA Support (EVS) anonymous directories:

- Albany OIFO ftp.fo-albany.med.va.gov
- Hines OIFO ftp.fo-hines.med.va.gov
- Salt Lake City OIFO ftp.fo-slc.med.va.gov
- Preferred Method download.vista.med.va.gov

  This method transmits the files from the first available FTP server.

**DISCLAIMER: The appearance of external hyperlink references in this manual does *not* constitute endorsement by the Department of Veterans Affairs (VA) of this Web site or the information, products, or services contained therein. The VA does *not* exercise any editorial control over the information you may find at these locations. Such links are provided and are consistent with the stated purpose of this VA Intranet Service.**

# 1. Introduction

This manual provides descriptive information about Kernel for use by application programmers. Kernel provides programmers with a number of tools. These tools include Application Program 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.

This manual assumes that the reader is familiar with the computing environment of the VA's Veterans Health Information Systems and Technology Architecture (VistA), and understands VA FileMan data structures and terminology. Understanding of the M programming language is required for this manual. No attempt is made to explain how the overall VistA programming system is integrated and maintained; such methods and procedures are documented elsewhere.

**REF:** For a chapter breakdown of the API and Direct Mode Utility categories, please refer to the "How to Use this Manual" topic in the "Orientation" section of this manual.

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

Kernel provides programmers with a number of tools. These tools include Application Program 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 Kernel Programmer Manual is divided into chapters, based on the following functional API/Direct Mode Utility categories within Kernel (listed alphabetically):

- Address Hygiene
- Alerts
- Device Handler
- Domain Name Service (DNS)
- Electronic Signature Codes
- Error Processing
- Field Monitoring
- File Access Security
- Help Processor
- Host Files
- Institution File
- Kernel Installation and Distribution System (KIDS)
- Menu Manager
- Miscellaneous
- Name Standardization
- Operating System (OS) Interface
- Public Key Infrastructure (PKI)
- Security Keys
- Server Options
- Signon/Security
- Spooling
- TaskMan
- Unwinder
- User
- XGF Function Library

- XLF Function Library
  - Date Functions
  - Hyperbolic Trigonometric Functions
  - Math Functions
  - Measurement Functions
  - String Functions
  - Utility Functions
- XML

**REF:** For general user information and system manager information , please refer to the *Kernel Systems Manual*.

Instructions for installing Kernel are provided in the *Kernel Installation Guide*. This guide also includes information about software application management (e.g., recommended settings for site parameters and scheduling time frames for tasked options).

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*.

Information about managing computer security, which includes a detailed description of techniques that can be used to monitor and audit computing activity, is presented in the *Kernel Security Tools Manual*.

# 2. Address Hygiene: Programmer Tools

## Application Program Interface (API)

Several APIs are available for programmers to work with address hygiene. These APIs are described below.

### CCODE^XIPUTIL(): FIPS Code Data

**Reference Type** Supported

**Category** Address Hygiene

**IA #** 3618

**Description** This API returns all the data associated for a FIPS code.

**Format** `CCODE^XIPUTIL(fips,.xipc)`

**Input Parameters** fips: (required) FIPS Code.

**Output Parameters** xipc: An array containing the following:

XIPC("COUNTY")—County associated with this FIPS code
XIPC("FIPS CODE")—5-digit FIPS county code
XIPC("INACTIVE DATE")—Date the FIPS code was inactivated
XIPC("LATITUDE")—Estimated Latitude of the county
XIPC("LONGITUDE")—Estimated Longitude of the county
XIPC("STATE")—State associated with this FIPS code
XIPC("STATE POINTER")—Pointer to the state in the STATE file (#5)

XIPC("ERROR")—Errors encountered during lookup

**Example**

```
>S ZFIPS=54041

>S ZTMP=""

>D CCODE^XIPUTIL(ZFIPS,.ZTMP)

>ZW ZTMP,ZFIPS
ZFIPS=54041
ZTMP=
ZTMP("COUNTY")=LEWIS
ZTMP("FIPS CODE")=54041
ZTMP("INACTIVE DATE")=
```

```
ZTMP("LATITUDE")=39:00N
ZTMP("LONGITUDE")=80:28W
ZTMP("STATE")=WEST VIRGINIA
ZTMP("STATE POINTER")=54
```

## $$FIPS^XIPUTIL(): FIPS Code for ZIP Code

**Reference Type** Supported

**Category** Address Hygiene

**IA #** 3618

**Description** This extrinsic function returns the Federal Information Processing Standard (FIPS) Code associated with the Postal Code.

**Format** `$$FIPS^XIPUTIL(pcode)`

**Input Parameters** pcode: (required) Postal Code for which the FIPS Code is returned.

**Output** returns: Returns the FIPS Code.

**Example**

```
>S X=$$FIPS^XIPUTIL("26452")

>W X
54041
```

## $$FIPSCHK^XIPUTIL(): Check for FIPS Code

**Reference Type** Supported

**Category** Address Hygiene

**IA #** 3618

**Description** This extrinsic function answers the question as to whether or not a Federal Information Processing Standard (FIPS) code exists. It returns the following:

- IEN—Internal Entry Number, if the FIPS code exists.
- Zero (0)—FIPS Code does *not* exist.

**Format** `$$FIPSCHK^XIPUTIL(fips)`

**Input Parameters** fips: (required) FIPS Code.

**Output** returns: Returns:

IEN—Internal Entry Number, if the FIPS code exists.

Zero (0)—FIPS Code does *not* exist.

**Example 1**

```
>S X=$$FIPSCHK^XIPUTIL("54041")

>W X
335
```

**Example 2**

```
>S X=$$FIPSCHK^XIPUTIL("54999")

>W X
0
```

# POSTAL^XIPUTIL(): ZIP Code Information

**Reference Type** Supported

**Category** Address Hygiene

**IA #** 3618

**Description** This API returns United States Postal Service (USPS)-related data/information in an output array (see Output) for the preferred (default) ZIP Code.

**Format** `POSTAL^XIPUTIL(pcode,.xip)`

**Input Parameters** pcode: (required) Postal Code for which data is returned.

**Output Parameters** .xip: An array containing the following:

XIP("CITY")—City that the USPS assigned to this PCODE.
XIP("CITY ABBREVIATION")—USPS's assigned abbreviation.
XIP("CITY KEY")—USPS's assigned city key.
XIP("COUNTY")—County associated with this PCODE.
XIP("COUNTY POINTER")—Pointer to the county in the COUNTY CODE file (#5.13).
XIP("FIPS CODE")—5-digit FIPS code associated with the county.
XIP("INACTIVE DATE")—Date FIPS Code inactive.
XIP("LATITUDE")—Latitude.
XIP("LONGITUDE")—Longitude.
XIP("POSTAL CODE")—Value used to look up postal data.
XIP("PREFERRED CITY KEY")—USPS's preferred (DEFAULT) city key.
XIP("STATE")—State associated with this PCODE.
XIP("STATE POINTER")—Pointer to the state in the STATE file (#5).
XIP("UNIQUE KEY")—Unique lookup value.

XIP("ERROR")—Errors encountered during lookup.

**Example 1**

```
>S ZCODE=94114

>S ZTMP=""

>D POSTAL^XIPUTIL(ZCODE,.ZTMP)

>ZW ZTMP,ZCODE
ZCODE=94114
ZTMP=
```

```
ZTMP("CITY")=SAN FRANCISCO
ZTMP("CITY ABBREVIATION")=
ZTMP("CITY KEY")=Z22802
ZTMP("COUNTY")=SAN FRANCISCO
ZTMP("COUNTY POINTER")=2910
ZTMP("FIPS CODE")=06075
ZTMP("INACTIVE DATE")=
ZTMP("LATITUDE")=39:00N
ZTMP("LONGITUDE")=80:28W
ZTMP("POSTAL CODE")=94114
ZTMP("PREFERRED CITY KEY")=Z22802
ZTMP("STATE")=CALIFORNIA
ZTMP("STATE POINTER")=6
ZTMP("UNIQUE KEY")=94114Z22802
```

### Example 2

```
>S ZCODE=94612

>S ZTMP=""

>D POSTAL^XIPUTIL(ZCODE,.ZTMP)

>ZW ZTMP,ZCODE
ZCODE=94612
ZTMP=
ZTMP("CITY")=OAKLAND
ZTMP("CITY ABBREVIATION")=
ZTMP("CITY KEY")=Z22296
ZTMP("COUNTY")=ALAMEDA
ZTMP("COUNTY POINTER")=2912
ZTMP("FIPS CODE")=06001
ZTMP("INACTIVE DATE")=
ZTMP("POSTAL CODE")=94612
ZTMP("PREFERRED CITY KEY")=Z22296
ZTMP("STATE")=CALIFORNIA
ZTMP("STATE POINTER")=6
ZTMP("UNIQUE KEY")=94612Z22296
```

## POSTALB^XIPUTIL(): Active ZIP Codes

**Reference Type** Supported

**Category** Address Hygiene

**IA #**

**Description** This API returns all of the active ZIP Codes for a single ZIP Code.

**Format** `POSTALB^XIPUTIL(pcode,.xip)`

**Input Parameters** pcode: (required) Postal Code for which the data is being requested.

**Output Parameters** .xip(n): The number of primary subscripts in an array:

XIP(n,"CITY")—City that the USPS assigned to this PCODE. An asterisk "*" indicates which city is PREFERRED (DEFAULT).
XIP(n"CITY KEY")—USPS's assigned city key.
XIP(n,"CITY ABBREVIATION")—USPS's assigned abbreviation.
XIP(n,"COUNTY")—County associated with this PCODE.
XIP(n,"COUNTY POINTER")—Pointer to the county in COUNTY CODE file (#5.13).
XIP(n,"FIPS CODE")—5-digit FIPS code associated with the county
XIP(n,"POSTAL CODE")—Value used to look up postal data
XIP(n,"PREFERRED CITY KEY")—USPS's preferred (DEFAULT) city key.
XIP(n,"STATE")—State associated with this PCODE.
XIP(n,"STATE POINTER")—Pointer to the state in the STATE file (#5).
XIP(n,"UNIQUE KEY")—Unique lookup value.

XIP("ERROR")—Errors encountered during lookup.

**Example**

```
>S ZCODE=26452

>S ZTMP=""

>D POSTALB^XIPUTIL(ZCODE,.ZTMP)

>ZW ZTMP,ZCODE
ZCODE=26452
ZTMP=2
ZTMP(1,"CITY")=WESTON*
ZTMP(1,"CITY ABBREVIATION")=
ZTMP(1,"CITY KEY")=X29362
```

```
ZTMP(1,"COUNTY")=LEWIS
ZTMP(1,"COUNTY POINTER")=335
ZTMP(1,"FIPS CODE")=54041
ZTMP(1,"POSTAL CODE")=26452
ZTMP(1,"PREFERRED CITY KEY")=X29362
ZTMP(1,"STATE")=WEST VIRGINIA
ZTMP(1,"STATE POINTER")=54
ZTMP(1,"UNIQUE KEY")=26452X29362
ZTMP(2,"CITY")=VALLEY CHAPEL
ZTMP(2,"CITY ABBREVIATION")=
ZTMP(2,"CITY KEY")=X2A444
ZTMP(2,"COUNTY")=LEWIS
ZTMP(2,"COUNTY POINTER")=335
ZTMP(2,"FIPS CODE")=54041
ZTMP(2,"POSTAL CODE")=26452
ZTMP(2,"PREFERRED CITY KEY")=X29362
ZTMP(2,"STATE")=WEST VIRGINIA
ZTMP(2,"STATE POINTER")=54
ZTMP(2,"UNIQUE KEY")=26452X2A444
```

# 3. Alerts: Programmer Tools

An application 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 SETUP^XQALERT API is used to create an alert.

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) to whom you sent the alert.

```
; send alert
S XQA(14)="",XQAMSG="Enter progress note",XQAOPT="ZZNOTES"
D SETUP^XQALERT
```

**Figure 3-1. Alerts: Creating an alert for User #14**

```
>D Q^DI

Select OPTION: INQUIRE TO FILE ENTRIES

OUTPUT FROM WHAT FILE: ALERT
Select ALERT RECIPIENT: `14 <Enter> EXAMPLE,PERSON
ANOTHER ONE: <Enter>
STANDARD CAPTIONED OUTPUT? YES// <Enter>
Include COMPUTED fields:  (N/Y/R/B): NO// <Enter> - 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
```

**Figure 3-2. Alerts: Checking that the alert was sent**

## Package Identifier versus Alert Identifier

**Package Identifier:** The software application identifier for an alert is defined as the original value of the XQAID input variable when the alert is created via the SETUP^XQALERT: Send Alerts API. Typically, the software application identifier should begin with the software application namespace.

**Alert Identifier:** The alert identifier consists of three semicolon pieces:

pkgid_";"_duz_";"_time

where pkgid is the original software application 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 [#8992.1]).

The distinction between software application identifier and alert identifier is important. More than one alert can share the same software application identifier, but the alert identifier is unique. Some Alert Handler APIs ask for a software application identifier (and act on multiple alerts), while other APIs ask for an alert identifier (and act on a single alert).

## Package Identifier Conventions

The Computerized Patient Record System (CPRS) software uses a convention for the format of the software application identifier consisting of three comma-delimited pieces:

namespace_","_dfn_","_notificationcode

where namespace is the software application namespace, DFN is the internal entry number of the patient whom the alert concerns in the PATIENT file (#2), and notification code is a code maintained by the CPRS software describing the type of alert.

**NOTE:** This three-comma-piece software application identifier is still only the first semicolon piece of an alert identifier.

Several Alert Handler APIs make use of these software application identifier conventions:

- PATIENT^XQALERT returns an array of alerts for a particular patient, based on the second comma-piece of alerts' software application identifiers.
- PTPURG^XQALBUTL purges alerts for a particular patient, based on the second comma-piece of alerts' software application identifiers.
- NOTIPURG^XQALBUTL purges alerts with a particular notification code, based on the third comma-piece of alerts' software application identifiers.

# Glossary of Terms for Alerts

| Term | Definition |
|---|---|
| ALERT | An alert notifies one or more users of a matter requiring immediate attention. Thus, alerts function as brief notices that are distinct from mail messages or triggered bulletins.<br><br>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 can also include an alert action, software application identifier, alert flag, and alert data. Alerts are stored in the ALERT file (#8992). |
| ALERT ACTION | The computing activity that can be associated with an alert (i.e., an option [XQAOPT input variable] or routine [XQAROU input variable]). |
| ALERT DATA | An optional string that the programmer can define when creating the alert. This string is restored in the XQADATA input 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 input 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 (described 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 input variable). |
| PACKAGE IDENTIFIER | An optional identifier that the programmer can use to identify the alert for such purposes as subsequent lookup and deletion (XQAID input variable). |
| PURGE INDICATOR | Checked by the Alert Handler (in the XQAKILL input 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. |

**Table 3-1. Alerts: Related terms and definitions**

# Application Program Interface (API)

Several APIs are available for programmers to work with alerts. These APIs are described below.

## AHISTORY^XQALBUTL(): Get Alert Tracking File Information

**Reference Type** Supported

**Category** Alerts

**IA #** 2788

**Description** This API returns information from the ALERT TRACKING file (#8992.1) for alerts with xqaid (input parameter) as its alert ID. The data is returned descendent from the closed root passed in the root input parameter. Usually, xqaid is known based on alert processing.

**Format** `AHISTORY^XQALBUTL(xqaid,root)`

**Input Parameters**

xqaid: (required) This is the value xqaid that is the alert identifier. It is passed to the routine or option that is run when the alert is selected. It can also be obtained from a listing of all of the xqaid values for a specified user and/or patient.

root: (required) This parameter is a closed reference to a local or global root. The information associated with the desired entry in the ALERT TRACKING file (#8992.1) is returned descendent from the specified root.

**NOTE:** A more user (developer) friendly call would be the ALERTDAT^XQALBUTL(): API, which returns the data in an array with the field numbers and names as the subscripts and the internal and external (if different) values as the value.

**Output**

returns: The data returned reflects the global structure of the ALERT TRACKING file (#8992.1).

### Example

The following example illustrates the use of this API and the format of the data returned.

```
>S XQAID="NO-ID;20;2990212.11294719"

>D AHISTORY^XQALBUTL(XQAID,"XXXROOT")
```

```
>ZW XXXROOT

XXXROOT(0)=NO-ID;20;2990212.11294719^2990212.112947^NO-ID^^20
XXXROOT(1)=TEST MESSAGE (ROUTINE) 20^^^XM
XXXROOT(20,0)=^8992.11^20^1
XXXROOT(20,1,0)=20^2990212.112954^2990212.145609^2990212.145621^2990212.145621
XXXROOT(20,"B",20,1)=
```

This is in the basic structure of the nodes taken from the global for this entry, which can be seen from a global map view of the ALERT TRACKING file (#8992.1):

```
^XTV(8992.1,D0,0)= (#.01) NAME [1F] ^ (#.02) DATE CREATED [2D]^ (#.03) PKG
     ==>ID [3F] ^ (#.04) PATIENT [4P] ^ (#.05)
GENERATED BY [5P] ^
     ==>(#.06) GENERATED WHILE QUEUED [6S] ^ (#.07)
STATUS [7S] ^
     ==>(#.08) RETENTION DATE [8D] ^

^XTV(8992.1,D0,1)= (#1.01) DISPLAY TEXT [1F] ^ (#1.02) OPTION FOR PROCESSING
     ==>[2F] ^ (#1.03) ROUTINE TAG [3F] ^ (#1.04)
ROUTINE FOR
     ==>PROCESSING [4F] ^

^XTV(8992.1,D0,2)= (#2) DATA FOR PROCESSING [E1,245F] ^

^XTV(8992.1,D0,20,0)=^8992.11PA^^  (#20) RECIPIENT

^XTV(8992.1,D0,20,D1,0)= (#.01) RECIPIENT [1P] ^ (#.02) ALERT FIRST DISPLAYED
     ==>[2D] ^ (#.03) FIRST SELECTED ALERT [3D] ^ (#.04)
     ==>PROCESSED ALERT [4D] ^ (#.05) DELETED ON [5D] ^
     ==>(#.06) AUTO DELETED [6D] ^ (#.07) FORWARDED BY [7P]
     ==>^ (#.08) DATE/TIME FORWARDED [8D] ^ (#.09) DELETED
     ==>BY USER [9P] ^
```

**NOTE:** A more user (developer) friendly API would be the ALERTDAT^XQALBUTL(): API, which returns the data in an array with the field numbers and names as the subscripts and the internal and external (if different) values as the value.

# ALERTDAT^XQALBUTL(): Get Alert Tracking File Information

**Reference Type** Supported

**Category** Alerts

**IA #** 2788

**Description** This API returns information from the ALERT TRACKING file (#8992.1) for alerts with xqaid input parameter as its alert ID in the array specified by the root input parameter. If root is not specified, then the data is returned in an XQALERTD array. If the specified alert is not present, the root array is returned with a NULL value.

**Format** `ALERTDAT^XQALBUTL(xqaid[,root])`

**Input Parameters**

xqaid: (required) This is the value xqaid that is the alert identifier. It is passed to the routine or option that is run when the alert is selected. It can also be obtained from a listing of all of the xqaid values for a specified user and/or patient.

root: (optional) This parameter is a closed reference to a local or global root. If root is not specified, then the data is returned in an XQALERTD array.

**Output**

returns: Returns:

ALERT TRACKING File Entry—The information associated with the desired entry in the ALERT TRACKING file (#8992.1) descendent from the specified root.

NULL—If the specified alert is not present, the array root is returned with a NULL value.

**Example**

```
>S XQAID="NO-ID;20;2990212.11294719"

>D ALERTDAT^XQALBUTL(XQAID,$NA(^TMP($J,"A")))

>D ^%G Global ^TMP($J,"A"
    TMP($J,"A"
^TMP(000056198,"A",.01) = NO-ID;20;2990212.11294719
^TMP(000056198,"A",.01,"NAME") =
^TMP(000056198,"A",.02) = 2990212.112947^FEB 12, 1999@11:29:47
^TMP(000056198,"A",.02,"DATE CREATED") =
^TMP(000056198,"A",.03) = NO-ID ^TMP(000056198,"A",.03,"PKG ID") =
^TMP(000056198,"A",.04) =
^TMP(000056198,"A",.04,"PATIENT") = ^TMP(000056198,"A",.05) = 20^USER,XXX
^TMP(000056198,"A",.05,"GENERATED BY") =
^TMP(000056198,"A",.06) = ^TMP(000056198,"A",.06,"GENERATED WHILE QUEUED") =
^TMP(000056198,"A",.07) =
^TMP(000056198,"A",.07,"STATUS") =
```

```
^TMP(000056198,"A",.08) =
^TMP(000056198,"A",.08,"RETENTION DATE") =
^TMP(000056198,"A",1.01) = TEST MESSAGE (ROUTINE) 20
^TMP(000056198,"A",1.01,"DISPLAY TEXT") =
^TMP(000056198,"A",1.02) = ^TMP(000056198,"A",1.02,"OPTION FOR PROCESSING") =
^TMP(000056198,"A",1.03) =
^TMP(000056198,"A",1.03,"ROUTINE TAG") =
^TMP(000056198,"A",1.04) = XM ^TMP(000056198,"A",1.04,"ROUTINE FOR PROCESSING") =
^TMP(000056198,"A",2) =
^TMP(000056198,"A",2,"DATA FOR PROCESSING") =
```

The data elements at the top level of the ACTIVITY TRACKING file are returned subscripted by the field numbers. This subscript is sufficient to obtain the data. The values are shown as internal^external if the internal and external forms are different. The next subscript after the field number will provide the field names if they are desired.

## DELSTAT^XQALBUTL(): Get User Information and Status for Recent Alert

**Reference Type** Supported

**Category** Alerts

**IA #** 3197

**Description** This API obtains information on the recipients of the most recent alert with a specified alert ID and the status of whether the alert has been deleted or not for those recipients.

**Format** `DELSTAT^XQALBUTL(xqaidval,.values)`

**Input Parameters**

xqaidval: (required) This input parameter is a value that has been used as the xwaid value for generating an alert by a software application. This value is used to identify the most recent alert generated with this xqaid value and that alert is used to generate the responses in terms of recipients and deletion status of the alert for each of the recipients.

**Output Parameters**

.values: This parameter is passed by reference and is returned as an array. The value of the values array indicates the number of entries in the array. The entries are then ordered in numerical order in the values array. The array contains the DUZ for users along with an indicator of whether or not the alert has been deleted.

**NOTE:** The contents of the array are KILLed prior to building the list.

For example:

DUZ^1—If alert deleted.

DUZ^0—If alert *not* deleted.

**Example**

```
>D DELSTAT^XQALBUTL("OR;14765;23",.VALUE)
```

The value of VALUE indicates the number of entries in the array. The entries are then ordered in numerical order in the VALUE array:

```
VALUE = 3
VALUE(1) = "146^0"   User 146 - not deleted
VALUE(2) = "297^1"   User 297 - deleted
VALUE(3) = "673^0"   User 673 - not deleted
```

## NOTIPURG^XQALBUTL(): Purge Alerts Based on Code

**Reference Type** Supported

**Category** Alerts

**IA #** 3010

**Description** This API 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).

**Format** `NOTIPURG^XQALBUTL(notifnum)`

**Input Parameters**

`notifnum:` (required) 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

**Example**

## $$PENDING^XQALBUTL(): Pending Alerts for a User

**Reference Type** Supported

**Category** Alerts

**IA #** 2788

**Description** This extrinsic function returns whether or not the user specified has the alert indicated by the xqaid input parameter as pending. It returns either of the following:

1—YES, alert is pending.

0—NO, alert is *not* pending.

**Format** `$$PENDING^XQALBUTL(xqauser,xqaid)`

**Input Parameters**

| | |
|---|---|
| xqauser: | (required) This is the Internal Entry Number (IEN, DUZ value) in the NEW PERSON file (#200) for the desired user. |
| xqaid | (required) This is the value xqaid that is the alert identifier. It is passed to the routine or option that is run when the alert is selected. It can also be obtained from a listing of all of the xqaid values for a specified user and/or patient. |

**Output**

| | |
|---|---|
| returns: | Returns:<br>1—YES, alert is pending.<br>0—NO, alert is *not* pending. |

### Example 1

The following is an example of an alert *not* pending:

```
>S XQAID="NO-ID;20;2990212.11294719"

>W $$PENDING^XQALBUTL(20,XQAID)
0
```

### Example 2

The following is an example of an alert pending:

```
>S XQAID="NO-ID;20;2990212.15540723"

>W $$PENDING^XQALBUTL(20,XQAID)
1
```

## $$PKGPEND^XQALBUTL(): Pending Alerts for a User in Specified Software

**Reference Type** Supported

**Category** Alerts

**IA #** 2788

**Description** This extrinsic function returns whether or not the user specified has an alert with XQAID containing the first "**;**"-piece (software/package identifier) indicated by the xqapkg input parameter pending. It returns either of the following:

1—YES, indicates one *or more* alerts pending for the specified user containing the software/package identifier.

0—NO, alerts *not* pending.

**Format** `$$PENDING^XQALBUTL(xqauser,xqapkg)`

**Input Parameters**

xqauser: (required) This is the Internal Entry Number (IEN, DUZ value) in the NEW PERSON file (#200) for the desired user.

xqapkg: (required) This is the software/package identifier portion of the alert identifier (XQAID). It is a textual identifier for the software that created the alert and is the first "**;**"-piece of XQAID. It can be used in this context to determine whether or not the user specified by xqauser has any alerts pending containing the specified software identifier. The software identifier used can be a complete software identifier (e.g., XU-TSK) or more general (e.g., XU) to find users with any XU software alerts.

**Output**

returns: Returns:

1—YES, indicates one *or more* alerts pending for the specified user containing the software/package identifier string in the package part of XQAID.

0—NO, alerts *not* pending.

### Example 1

The following is an example of an alert *not* pending:

```
>S XQAPKG="XU"

>W $$PKGPEND^XQALBUTL(20,XQAPKG)
0
```

**Example 2**

The following is an example of an alert pending (one or more):

```
>S XQAPKG="XU"

>W $$PKGPEND^XQALBUTL(20,XQAPKG)
1
```

## PTPURG^XQALBUTL(): Purge Alerts Based on Patient

**Reference Type** Supported

**Category** Alerts

**IA #** 3010

**Description** This API 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).

**Format** `PTPURG^XQALBUTL(dfn)`

**Input Parameters** dfn: (required) Internal entry number (DFN in the PATIENT file [#2]) for which alerts are deleted.

**Output** none

**Example**

## RECIPURG^XQALBUTL(): Purge User Alerts

**Reference Type** Supported

**Category** Alerts

**IA #** 3010

**Description** This API deletes all alerts that have been sent to the user in the NEW PERSON file (#200), as indicated by the duz parameter.

**Format** `RECIPURG^XQALBUTL(duz)`

**Input Parameters**

duz: (required) Internal Entry Number (IEN in the NEW PERSON file [#200]) of the user for whom received alerts are deleted.

**Output** none

**Example**

## USERDATA^XQALBUTL(): Get User Information for an Alert

**Reference Type** Supported

**Category** Alerts

**IA #** 2788

**Description** This API returns recipients of the alert with the xqaid input parameter as its alert ID from the ALERT TRACKING file (#8992.1) in the array specified by the root input parameter. If root is *not* specified, then the data is returned in the XQALUSER array. If the specified alert is *not* present, the root array is returned with a NULL value.

**Format** `USERDATA^XQALBUTL(xqaid,xqauser,root)`

**Input Parameters**

xqaid: (required) This is the value xqaid that is the alert identifier. It is passed to the routine or option that is run when the alert is selected. It can also be obtained from a listing of all of the xqaid values for a specified user and/or patient.

xqauser: (required) This is the Internal Entry Number (IEN, DUZ value) in the NEW PERSON file (#200) for the desired user.

root: (optional) This parameter is a closed reference to a local or global root. If root is *not* specified, then the data is returned in the XQALUSER array.

**Output**

returns: Returns:

ALERT TRACKING File Entry—The information associated with the desired entry in the ALERT TRACKING file (#8992.1) descendent from the specified root.

NULL—If the specified alert is *not* present, the array root is returned with a NULL value.

**Example**

```
>D USERDATA^XQALBUTL(XQAID,20,"XXX")

>ZW XXX

XXX(.01)=20^USER,XXX XXX(.01,"RECIPIENT")=
XXX(.02)=2990212.112954^FEB 12, 1999@11:29:54 XXX(.02,"ALERT FIRST DISPLAYED")=
XXX(.03)=2990212.145609^FEB 12, 1999@14:56:09 XXX(.03,"FIRST SELECTED ALERT")=
XXX(.04)=2990212.145621^FEB 12, 1999@14:56:21 XXX(.04,"PROCESSED ALERT")=
XXX(.05)=2990212.145621^FEB 12, 1999@14:56:21 XXX(.05,"DELETED ON")=
XXX(.06)= XXX(.06,"AUTODELETED")=
XXX(.07)= XXX(.07,"FORWARDED BY")=
XXX(.08)= XXX(.08,"DATE/TIME FORWARDED")=
XXX(.09)= XXX(.09,"DELETED BY USER")=
```

## USERLIST^XQALBUTL(): Get Recipient Information for an Alert

**Reference Type** Supported

**Category** Alerts

**IA #** 2788

**Description** This API returns recipients of the alert with the xqaid input parameter as its alert ID from the ALERT TRACKING file (#8992.1) in the array specified by the root input parameter. If root is *not* specified, then the data is returned in the XQALUSRS array. If the specified alert is *not* present, the root array is returned with a NULL value.

**Format** `USERLIST^XQALBUTL(xqaid,root)`

**Input Parameters**

xqaid: (required) This is the value xqaid that is the alert identifier. It is passed to the routine or option that is run when the alert is selected. It can also be obtained from a listing of all of the xqaid values for a specified user and/or patient.

root: (optional) This parameter is a closed reference to a local or global root. If root is *not* specified, then the data is returned in the XQALUSRS array.

**Output**

returns: Returns:

ALERT TRACKING File Entry—The information associated with the desired entry in the ALERT TRACKING file (#8992.1) descendent from the specified root.

NULL—If the specified alert is not present, the array root is returned with a NULL value.

**Example**

```
>D USERLIST^XQALBUTL(XQAID)

>ZW XQALUSRS XQALUSRS(1)=20^USER,XXX
```

## ACTION^XQALERT(): Process an Alert

**Reference Type** Supported

**Category** Alerts

**IA #** 10081

**Description** This API processes 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.

**Format** `ACTION^XQALERT(alertid)`

**Input Parameters**

alertid: (required) Alert Identifier of the alert to process (same as ALERT ID field in ALERT file [#8992]). This contains three semicolon-delimited pieces, the first being the original software application 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

**Example**

## DELETE^XQALERT(): Clear Obsolete Alerts

**Reference Type** Supported

**Category** Alerts

**IA #** 10081

**Description** This API 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 zero (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.

**REF:** For more information on alert identifiers, please refer to the "Package Identifier versus Alert Identifier" topic in this chapter.

**Format**

```
DELETE^XQALERT(xqaid[,xqakill])
```

**Input Parameters**

xqaid: (required) 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 zero (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 software application 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

**Example**

# DELETEA^XQALERT(): Clear Obsolete Alerts

**Reference Type** Supported

**Category** Alerts

**IA #** 10081

**Description** This API deletes (clears) all alerts with the same software application 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 zero (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 API to delete any unprocessed alerts associated with the condition. It deletes *all* alerts whose software application identifiers match the software application identifier you pass in the xqaid input parameter (multiple alerts can potentially share the same software application identifier).

**REF:** For more information on software application identifiers, please refer to the "Package Identifier versus Alert Identifier" topic earlier in this chapter.

**Format**

```
DELETEA^XQALERT(xqaid[,xqakill])
```

**Input Parameters**

xqaid: (required) All alerts whose software application identifier matches the value of this input parameter will be deleted, for the alert recipients designated by the xqakill input parameter.

The form of xqaid can be exactly as initially set when creating the alert. Alternatively, it can contain the two additional semicolon pieces added by the Alert Handler (the full alert identifier). The two additional semicolon pieces are ignored, however; this API only requires the original software application identifier.

If the alert identifier you specify is "NO-ID", however, (the generic software application id assigned to alerts with no original software application identifier), this API will not delete matching alerts.

xqakill: (optional) If xqakill is undefined or zero (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

**Example**

## GETACT^XQALERT(): Return Alert Variables

**Reference Type** Supported

**Category** Alerts

**IA #** 10081

**Description** This API returns to the calling routine the required variables to act on a specific alert.

**Format** `GETACT^XQALERT(alertid)`

**Input Parameters**

alertid: (required) This is the alert identifier in the ALERT TRACKING file (#8992.1).

**Output Variables**

XQAID: This is the full alert identifier.

XQADATA: The XQADATA variable stores any software application-specific data string that was passed at the time the alert was generated.

XQAOPT: Indicates a non-menu type option on the user's primary, secondary or common menu to be run if *not* null.

XQAROU: Indicates the routine or tag^routine to run when the alert is processed. It can can have three values:

- Null—A null value indicates no routine to be used (XQAOPT contains option name to be run).
- ^<space>—A value of ^<space> indicates that the alert is information only (no routine or option action involved).
- ^ROUTINE or TAG^ROUTINE—The name of the routine as ^ROUTINE or TAG^ROUTINE.

**Example**

## PATIENT^XQALERT(): Get Alerts for a Patient

**Reference Type** Supported

**Category** Alerts

**IA #** 10081

**Description** This API allows you to return an array of all alerts for a particular patient that are either:

a. Open.

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 CPRS software application 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 (#2).

**REF:** For information on CPRS conventions for the format of the Package Identifier, please refer to the "Package Identifier versus Alert Identifier" topic earlier in this chapter.

**Format**

```
PATIENT^XQALERT(root,dfn[,startdate][,enddate])
```

**Input Parameters**

root: (required) Fully resolved global or local reference in which to return a list of matching alerts.

dfn: (required) Internal entry number (DFN in the PATIENT file [#2]) of the patient for whom alerts are returned.

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 Parameters**

root: All alerts matching the request are returned in the input parameter you specified in root, in the following 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:

1. The original software application identifier (value of XQAID).
2. The DUZ of the alert creator.
3. The VA FileMan date and time the alert was created.

**Example**

## SETUP^XQALERT: Send Alerts

**Reference Type** Supported

**Category** Alerts

**IA #** 10081

**Description** This API sends alerts to users. The preferred call now is $$SETUP1^XQALERT: Send Alerts.

To send an information-only alert, make sure that XQAOPT and XQAROU input variables are *not* defined. To send an alert that takes an action, specify either the XQAOPT (to run an option) or XQAROU (to run a routine) input variables.

**Format** `SETUP^XQALERT`

Make sure to perform the following steps before calling SETUP^XQALERT:

- NEW all of the following input variables.
- Set the input variables you want changed.
- Call SETUP^XQALERT.

If you do not follow these steps, the input variables could unintentionally assume the values of the variables of the current running task.

**Input Variables**

XQA: (required) 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 mail group names to send to users in mail groups:

```
>S XQA(USERDUZ)=""
>S XQA("G.MAILGROUP")=""
```

XQAARCH: (optional) Number of days to wait before Delete Old (>14d) Alerts option forwards alert to recipient's supervisor on the basis of Service/Section, if alert is unprocessed by recipient. Can be a number from 1 to 30.

XQADATA: (optional) Use this to store a software application-specific data string, in any format. It will be restored in the XQADATA input variable 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 input variable, including the caret. You can use it to make data such as patient number, lab accession, or cost center available to your software application-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.

XQAGUID: (optional) As of Kernel Patch XU*8.0*207, the GUID FOR GUI adds an interface GUID (a 32 character string containing hexadecimal digits in a specific format within curly braces) to permit a program on the client to process the alert. The presence of a GUID in the variable indicates that the alert can be processed within a GUI environment, and is used to open the correct application to process the alert within the GUI environment.

**NOTE:** Unfortunately, this functionality has never been implemented by CPRS or other GUI applications.

XQAID: (optional) Package identifier for the alert, typically a software application 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.

**REF:** For information on CPRS conventions for the format of the Package Identifier, please refer to the "Package Identifier versus Alert Identifier" topic in this chapter.

XQAMSG: (required) 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 cannot 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.

XQASUPV: (optional) Number of days to wait before Delete Old (>14d) Alerts option forwards alert to recipient's supervisor on the basis of Service/Section, if alert is unprocessed by recipient. Can be a number from 1 to 30.

XQASURO: (optional) Number of days to wait before Delete Old (>14d) Alerts option forwards alert to recipient's MailMan surrogates (if any), if alert is unprocessed by recipient. Can be a number from 1 to 30.

XQATEXT: (optional) As of Kernel Patch XU*8.0*207, this variable permits informational text of any length to be passed with an alert. When the alert is selected, the contents of this variable will be displayed in a ScreenMan form within the roll & scroll environment.

**NOTE:** It was also intended to be displayed within a text display box within the GUI environment. However, CPRS has never implemented this functionality, so it can only be viewed in the roll & scroll environment.

**Output** none

**Details—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 following input variables:
   - XQADATA—If originally set when alert was created.
   - XQAID—If originally set when alert was created.
   - XQAKILL—The purge indicator. It is always set to 1 by the Alert Handler.

   If you associated a software application identifier, XQAID, with the alert, it is restored along with two additional semicolon pieces:
   - Current user number.
   - Current date/time.

   With the two additional semicolon pieces, the software application identifier becomes the alert identifier. If you did not define XQAID when creating the alert, Alert Handler sets XQAID input variable 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 input variables listed above (i.e., 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 zero (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:
     – Original Package Identifier.
     – Alert sender.
     – Date/Time the alert was sent.

     It purges them so that other users need not be notified of an obsolete alert.

     **NOTE:** 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.

### Example

```
;send an alert
;assume DFN is for patient KRNPATIENT,ONE
N XQA,XQAARCH,XQADATA,XQAFLG,XQAGUID,XQAID,XQAMSG,XQAOPT,XQAROU,XQASUPV,XQASURO,
XQATEXT,XQALERR
S XQA(161)="" ; recipient is user `161
S XQAMSG="Elevated CEA for "_$$GET1^DIQ(2,DFN_",",.01)_"
("_$E($$GET1^DIQ(2,DFN_",",9),6,9)_") Schedule follow-up exam in Surgical Clinic."
D SETUP^XQALERT
Q
```

**Figure 3-3. Alerts: Call to send an alert sample**

```
Select Systems Manager Menu Option: "VA



 1.I  Elevated CEA for KRNPATIENT,ONE (5345).  Schedule follow-up exam in Surgical
Clinic.
          Select from 1 to 1
          or enter ?, A, I, P, M, R, or ^ to exit:
```

**Figure 3-4. Alerts: Resulting alert, from View Alerts option**

## $$SETUP1^XQALERT: Send Alerts

**Reference Type** Supported

**Category** Alerts

**IA #** 10081

**Description** This API sends alerts to users. This is the preferred call rather than SETUP^XQALERT.

To send an information-only alert, make sure that XQAOPT and XQAROU input variables are *not* defined. To send an alert that takes an action, specify either the XQAOPT (to run an option) or XQAROU (to run a routine) input variables.

**Format** `$$SETUP1^XQALERT`

Make sure to perform the following steps before calling $$SETUP1^XQALERT:

- NEW all of the following input variables.
- Set the input variables you want changed.
- Call $$SETUP1^XQALERT.

If you do not follow these steps, the input variables could unintentionally assume the values of the variables of the current running task.

**Input Variables**

XQA: (required) 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 mail group names to send to users in mail groups:

```
>S XQA(USERDUZ)=""
>S XQA("G.MAILGROUP")=""
```

XQAARCH: (optional) Number of days that alert tracking information for this alert should be retained in the ALERT TRACKING file (#8992.1). Default time period is 30 days. Users can specify a different number of days using this input variable. To retain information forever, a value of 100000 is recommended (good for approximately 220 years).

XQADATA: (optional) Use this to store a software application-specific data string, in any format. It will be restored in the XQADATA input variable 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 input variable, including the caret. You can use it to make data such as patient number, lab accession, or cost center available to your software application-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.

XQAGUID: (optional) As of Kernel Patch XU*8.0*207, the GUID FOR GUI adds an interface GUID (a 32 character string containing hexadecimal digits in a specific format within curly braces) to permit a program on the client to process the alert. The presence of a GUID in the variable indicates that the alert can be processed within a GUI environment, and is used to open the correct application to process the alert within the GUI environment.

**NOTE:** Unfortunately, this functionality has never been implemented by CPRS or other GUI applications.

XQAID: (optional) Package identifier for the alert, typically a software application 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.

**REF:** For information on CPRS conventions for the format of the Package Identifier, please refer to the "Package Identifier versus Alert Identifier" topic in this chapter.

XQAMSG: (required) 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 cannot 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.

XQASUPV: (optional) Number of days to wait before Delete Old (>14d) Alerts option forwards alert to recipient's supervisor on the basis of Service/Section, if alert is unprocessed by recipient. Can be a number from 1 to 30.

XQASURO: (optional) Number of days to wait before Delete Old (>14d) Alerts option forwards alert to recipient's MailMan surrogates (if any), if alert is unprocessed by recipient. Can be a number from 1 to 30.

XQATEXT: (optional) As of Kernel Patch XU*8.0*207, this variable permits informational text of any length to be passed with an alert. When the alert is selected, the contents of this variable will be displayed in a ScreenMan form within the roll & scroll environment.

**NOTE:** It was also intended to be displayed within a text display box within the GUI environment. However, CPRS has never implemented this functionality, so it can only be viewed in the roll & scroll environment.

**Output**

returns: Returns:

- 1—The alert was sent successfully.
- 0—The alert was *not* sent successfully, in which case the XQALERR variable will contain a text string indicating the reason that the alert was not sent.

### Details—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 following input variables:

   - XQADATA—If originally set when alert was created.
   - XQAID—If originally set when alert was created.
   - XQAKILL—The purge indicator. It is always set to 1 by the Alert Handler.

   If you associated a software application identifier, XQAID, with the alert, it is restored along with two additional semicolon pieces:

   - Current user number.
   - Current date/time.

   With the two additional semicolon pieces, the software application identifier becomes the alert identifier. If you did not define XQAID when creating the alert, Alert Handler sets XQAID input variable 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 input variables listed above (i.e., 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 zero (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:

     – Original Package Identifier.

     – Alert sender.

     – Date/Time the alert was sent.

     It purges them so that other users need not be notified of an obsolete alert.

     **NOTE:** 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.

### Example

```
;send an alert
;assume DFN is for patient KRNPATIENT,ONE
N
XQA,XQAARCH,XQADATA,XQAFLG,XQAGUID,XQAID,XQAMSG,XQAOPT,XQAROU,XQASUPV,XQASURO,XQATE
XT,XQALERR
S XQA(161)="" ; recipient is user `161
S XQAMSG="Elevated CEA for "_$$GET1^DIQ(2,DFN_",",.01)_"
("_$E($$GET1^DIQ(2,DFN_",",9),6,9)_") Schedule follow-up exam in Surgical Clinic."
S VAR=$$SETUP1^XQALERT I ‘XQALERR W !,"ERROR IN ALERT: ",XQALERR
Q
```

**Figure 3-5. Alerts: Call to send an alert sample**

```
Select Systems Manager Menu Option: "VA



 1.I  Elevated CEA for KRNPATIENT,ONE (5345).  Schedule follow-up exam in Surgical
Clinic.
          Select from 1 to 1
          or enter ?, A, I, P, M, R, or ^ to exit:
```

**Figure 3-6. Alerts: Resulting alert, from View Alerts option**

## USER^XQALERT(): Get Alerts for a User

**Reference Type** Supported

**Category** Alerts

**IA #** 10081

**Description** This API returns a list of alerts for a given user. You can return a list of all alerts for a particular user that are either:

- Open.
- Within a given time range (open and closed).

**Format**

```
USER^XQALERT(root[,duz][,startdate][,enddate])
```

**Input Parameters**

root: (required) Fully resolved global or local reference in which to return a list of matching alerts.

duz: (optional) DUZ number of the user for whom the alert list is returned. 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 Parameters** root: All alerts matching the request are returned in the input parameter you specified in root, in the following 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:

1. The original software application identifier (value of XQAID).
2. The DUZ of the alert creator.
3. The VA FileMan date and time the alert was created.

**Example**

```
>D USER^XQALERT("ZZALRT",ZZDUZ,2900101)

>ZW ZZALRT
ZZALRT=1
ZZLART(1)="I  Test Message^NO-ID;92;2940729.10312"
```

## FORWARD^XQALFWD(): Forward Alerts

**Reference Type** Supported

**Category** Alerts

**IA #** 3009

**Description** This API can be used to forward alerts (in most cases, for the current user only). It is a silent (no screen input or output) API, and so can be used for windowed applications.

**Format** `FORWARD^XQALFWD([.]alerts,[.]users,type,comment)`

**Input Parameters**

[.]alerts: (required) 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 (#8992). Current user is identified by the value of the XQADUZ input variable). Use the $$SETUP1^XQALERT: Send Alerts API to obtain alert identifiers for a user's current open alerts.

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: (required) Users to forward alert to. For forwarding as an alert or as a mail message (when the type parameter is A or M), the input parameter can specify one or more users, and/or mail groups. For users, specify by IEN (in the NEW PERSON file [#200]). 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 mail group, 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 (#3.5). 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: (required) 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: (required) A character string to use as a comment to accompany the alert when it is forwarded.

**Output** none

**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 mail group
.K A6AUSER S A6AUSER="G.MAS CLERKS"
.D FORWARD^XQALFWD(.A6AALRT1,A6AUSER,"A","Forwarded Alert")
Q
```

**Figure 3-7. FORWARD^XQALFWD: Example**

## $$CURRSURO^XQALSURO(): Get Current Surrogate for Alerts

**Reference Type** Supported

**Category** Alerts

**IA #** 2790

**Description** This extrinsic function obtains the current surrogate for alerts (if any for the user with DUZ specified by the xqauser input parameter.

**Format** `$$CURRSURO^XQALSURO(xqauser)`

**Input Parameters** xqauser: (required) This is the Internal Entry Number (IEN, DUZ value) in the NEW PERSON file (#200) for the specified user with the surrogate.

**Output** returns: Returns:

DUZ—Internal Entry Number (IEN) of the surrogate.

-1—if there is no surrogate specified.

**Example**

## $$GETSURO^XQALSURO(): Get Current Surrogate Information

**Reference Type** Supported

**Category** Alerts

**IA #** 3213

**Description** This extrinsic function returns the following string of information on the current surrogate for the user with XQAUSER as his or her Internal Entry Number (IEN) in the NEW PERSON file (#200):

```
ien^NAME^FM_STARTDATE^FM_ENDDATE
```

If there is no surrogate, the result will be:

```
^^^
```

If either of the date/times are not specified, they return a null value for that piece of the return string.

**REF:** For a description of each piece of information separated by the caret ("^"), please refer to the "Output" section below.

**Format** `$$GETSURO^XQALSURO(xqauser)`

**Input Parameters**

xqauser: (required) This is the Internal Entry Number (IEN) in the NEW PERSON file (#200) of the user for whom the alert surrogate information is to be returned.

**Output**

returns: Returns the following string of information, each piece separated by a caret ("^"):

IEN^NAME^FM_STARTDATE^FM_ENDDATE

- IEN—Internal Entry Number (IEN) of the SURROGATE in the NEW PERSON file (#200).
- NAME—Contents of the .01 field for the SURROGATE.
- FM_STARTDATE—Starting date/time for the SURROGATE in internal VA FileMan format.
- FM_ENDDATE—Ending date/time for the SURROGATE in internal VA FileMan format.

**Example**

```
>S X=$$GETSURO^XQALSURO(124)

>W X

2327^KRNUSER,FOUR^3000929.1630^3001006.0800
```

This indicates that user #2327 (Four Krnuser) will become active as surrogate at 4:30 PM 9/29/00 and will remain surrogate until 8:00 am on 10/06/00.

## REMVSURO^XQALSURO(): Remove Surrogates for Alerts

**Reference Type** Supported

**Category** Alerts

**IA #** 2790

**Description** This API removes any surrogates for alerts for the specified user.

**Format** `REMVSURO^XQALSURO(xqauser)`

**Input Parameters** xqauser: (required) This is the Internal Entry Number (IEN, DUZ value) in the NEW PERSON file (#200) for the specified user.

**Output** none

**Example**

## SETSURO1^XQALSURO(): Establish a Surrogate for Alerts

**Reference Type** Supported

**Category** Alerts

**IA #** 3213

**Description** This API establishes a surrogate for alerts. It should be used instead of the SETSURO^XQALSURO API. The SETSURO1^XQALSURO API also tests for cyclic relationships (such that the user eventually would become the surrogate). SETSURO1 does these tests and therefore has the possibility of failure. It returns either of the following values:

- 1 (True)—If the surrogate was created successfully.
- Text String (False)—Text explaining why the surrogate was not created.

Previously, the SETSURO^XQALSURO API returned no value and, as long as both a user and surrogate were specified, would simply store the values. This left open the possibility that the user was specified as the surrogate or that a chain of surrogates ended up pointing again at the user, cases that could result in a very tight, non-ending, loop being generated if an alert was sent. These possibilities have been tested for in the interactive specification of surrogates, and is tested for non-interactive usage in the SETSURO1^XQALSURO API.

**NOTE:** The SETSURO1^XQALSURO API should be used instead of the SETSURO^XQALSURO API (i.e., IA# 2790).

**Format**

```
SETSURO1^XQALSURO(xqauser,xqalsuro[,xqalstrt][,xqalend])
```

**Input Parameters**

`xqauser`: (required) User's DUZ number (i.e., Internal Entry Number in the NEW PERSON file [#200]) for which the surrogate should act in receiving alerts.

`xqalsuro`: (required) Surrogate's DUZ number (i.e., Internal Entry Number in the NEW PERSON file [#200]) for the user who will receive and process alerts for XQAUSER.

xqalstrt: (optional) The start date/time or the surrogate activity, in VA FileMan internal format. If the start date/time is *not* specified, the surrogate relationship begins immediately.

xqalend: (optional) The end date/time for the end of the surrogate relationship, in VA FileMan internal format. If the end date/time is *not* specified, the surrogate remains active until another surrogate is specified or the surrogate is deleted.

**Output** returns: Returns:

- 1 (True)—If the surrogate was created successfully.
- Text String (False)—Text explaining why the surrogate was not created.

**Example**

```
>S XQAUSER=DUZ

>S XQASURRO=45

>S XQASTART=3001004.1630

>S XQAEND=3001008.1630

>S X=$$SETSURO1^XQALSURO(XQAUSER,XQASURRO,XQASTART,XQAEND)

>I 'X W !,"Could not activate surrogate",!,?5,X Q
```

# 4. 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.

# Application Program Interface (API)

Several APIs are available for programmers to work with devices. These APIs are described below.

## DEVICE^XUDHGUI(): GUI Device Lookup

**Reference Type** Supported

**Category** Device Handler

**IA #** 3771

**Description** This API allows VistA Graphical User Interface (GUI)-based applications to look up devices. This API retrieves the first 20 devices that meet the specifications passed. This API was made available with Kernel Patch XU*8.0*220.

**Format**

```
DEVICE^XUDHGUI(.list,starting_point[,direction]
[,right_margin_range])
```

**Input Parameters**

.list: (required) Named array to store output.

starting_point: (required) This parameter indicates where to start the $ORDERing of the Global. "P" will only return devices whose name starts with "P"; "P*" will return up to 20 devices the first starting with "P".

direction: (optional) This parameter indicates whether to $ORDER up or down from the starting_point parameter. The acceptable values are 1 and -1:

1—Up.

-1—Down.

right_margin_range: (optional) This parameter is used to specify a width range of devices:

- Exact Width (e.g., "132-132")
- At Least Width (e.g., "132")
- Range (e.g., "80-132")

**Output Parameters**

.list: The data is returned in this named array. Data is returned in the following format:

```
IEN^NAME^DISPLAY NAME^LOCATION^RIGHT
MARGIN^PAGE LENGTH
```

### Example 1

In this example, we want to store/display a list of all devices that begin with "P" in an array (e.g., DEVICES), without passing a direction or right margin range parameter:

```
>K DEVICES
D DEVICE^XUDHGUI(.DEVICES,"P")
```

The DEVICES array displays the following results:

```
>ZW DEVICES
DEVICES(1)=358^P-MESSAGE-HFS^P-MESSAGE-HFS^HFS FILE=>MESSAGE^255^256
DEVICES(2)=350^P-MESSAGE-HFS-MSM^P-MESSAGE-HFS-MSM^HFS FILE==> MESSAGE^80^999
DEVICES(3)=348^P-MESSAGE-HFS-ONT^P-MESSAGE-HFS-ONT^HFS FILE==> MESSAGE^80^999
DEVICES(4)=274^P-MESSAGE-HFS-VXD^P-MESSAGE-HFS-VXD^HFS FILE==> MESSAGE^80^256
DEVICES(5)=292^P-RESMON^P-RESMON^IRM^132^64
DEVICES(6)=310^P-WINDOC^P-WINDOC^MWAPI WINDOW DOCUMENT BOX^80^256
```

### Example 2

In this example, we want to store/display a list of all devices that begin with "P" in an array (e.g., DEVICES), without passing a direction parameter but including those devices with a right margin of an exact width of 80:

```
>K DEVICES
>D DEVICE^XUDHGUI(.DEVICES,"P",,"80-80")
```

The DEVICES array displays the following results:

```
>ZW DEVICES
DEVICES(1)=350^P-MESSAGE-HFS-MSM^P-MESSAGE-HFS-MSM^HFS FILE==> MESSAGE^80^999
DEVICES(2)=348^P-MESSAGE-HFS-ONT^P-MESSAGE-HFS-ONT^HFS FILE==> MESSAGE^80^999
DEVICES(3)=274^P-MESSAGE-HFS-VXD^P-MESSAGE-HFS-VXD^HFS FILE==> MESSAGE^80^256
DEVICES(4)=310^P-WINDOC^P-WINDOC^MWAPI WINDOW DOCUMENT BOX^80^256
```

### Example 3

In this example, we want to store/display a list of all devices that begin with "P" in an array (e.g., DEVICES), without passing a direction parameter but including those devices with a right margin width range of 80-132:

```
>K DEVICES
>D DEVICE^XUDHGUI(.DEVICES,"P",,"80-132")
```

The DEVICES array displays the following results:

```
>ZW DEVICES
DEVICES(1)=350^P-MESSAGE-HFS-MSM^P-MESSAGE-HFS-MSM^HFS FILE==> MESSAGE^80^999
DEVICES(2)=348^P-MESSAGE-HFS-ONT^P-MESSAGE-HFS-ONT^HFS FILE==> MESSAGE^80^999
DEVICES(3)=274^P-MESSAGE-HFS-VXD^P-MESSAGE-HFS-VXD^HFS FILE==> MESSAGE^80^256
DEVICES(4)=292^P-RESMON^P-RESMON^IRM^132^64
DEVICES(5)=310^P-WINDOC^P-WINDOC^MWAPI WINDOW DOCUMENT BOX^80^256
```

### Example 4

In this example, we want to store/display a list of up to 20 devices, the first of which starts with "P," in an array (e.g., DEVICES), without passing a direction or right margin range parameter:

```
>K DEVICES
>D DEVICE^XUDHGUI(.DEVICES,"P*")
```

The DEVICES array displays the following results:

```
>ZW DEVICES
DEVICES(1)=358^P-MESSAGE-HFS^P-MESSAGE-HFS^HFS FILE=>MESSAGE^255^256
DEVICES(2)=350^P-MESSAGE-HFS-MSM^P-MESSAGE-HFS-MSM^HFS FILE==> MESSAGE^80^999
DEVICES(3)=348^P-MESSAGE-HFS-ONT^P-MESSAGE-HFS-ONT^HFS FILE==> MESSAGE^80^999
DEVICES(4)=274^P-MESSAGE-HFS-VXD^P-MESSAGE-HFS-VXD^HFS FILE==> MESSAGE^80^256
DEVICES(5)=292^P-RESMON^P-RESMON^IRM^132^64
DEVICES(6)=310^P-WINDOC^P-WINDOC^MWAPI WINDOW DOCUMENT BOX^80^256
DEVICES(7)=202^C6_SDD_MX3 ROUTINE^ROUTINE <C6_SDD_MX3 ROUTINE>^Next to Jean's
Office^80^59
DEVICES(8)=428^SDD DUPLEX P10^SDD DUPLEX P10^SSD DUPLEX PRINTER NEXT TO JACK^80^60
DEVICES(9)=429^SDD P10^SDD P10^Printer next to Jack.^80^60
DEVICES(10)=329^C6_SDD_MX3 P10^SS10 <C6_SDD_MX3 P10>^Near Jean's Office^80^59
DEVICES(11)=330^C6_SDD_MX3 P12^SS12 <C6_SDD_MX3 P12>^Near Jean's Office^96^57
DEVICES(12)=331^C6_SDD_MX3 P16^SS16 <C6_SDD_MX3 P16>^Near Jean's Office^255^58
DEVICES(13)=349^C6_SDD_MX3 P16P8L^SS16P8L <C6_SDD_MX3 P16P8L>^Near Jean's
Office^117^79
DEVICES(14)=202^C6_SDD_MX3 ROUTINE^SSR <C6_SDD_MX3 ROUTINE>^Next to Jean's
Office^80^59
DEVICES(15)=427^SUP$PRT TEST^SUP$PRT TEST^DISK FILE^132^58
DEVICES(16)=283^SYS$INPUT^SYS$INPUT^SYS$INPUT;^132^64
DEVICES(17)=198^VMS FILE^VMS FILE^DISK^80^64
DEVICES(18)=349^C6_SDD_MX3 P16P8L^VPM <C6_SDD_MX3 P16P8L>^Near Jean's Office^117^79
DEVICES(19)=291^VTB255^VTB255^RMS FILE^255^99999
DEVICES(20)=288^ZBROWSE^ZBROWSE^RMS FILE^255^99999
```

## $$RES^XUDHSET(): Set Up Resource Device

**Reference Type** Supported

**Category** Device Handler

**IA #** 2232

**Description** This extrinsic function sets up a Resource device. It returns:

- Error: -1^text
- Successful: IEN^device name

**Format**

```
$$RES^XUDHSET(device_name[,resource_name],slot_count,description,
subtype)
```

**Input Parameters**

| | |
|---|---|
| device_name: | (required) The name of the resource device. |
| resource_name: | (optional) The resource name if not the same as the device name. |
| slot_count: | (required) The number of concurrent jobs that can use this device. It defaults to 1. |
| `description:` | (required) The device description. It defaults to "Resource Device". |
| `subtype:` | (required) The subtype to use. It defaults to P-OTHER. |

**Output**

returns: Returns:

Error: -1^text

Successful: IEN^device name

**Example**

## ^%ZIS: Standard Device Call

**Reference Type** Supported

**Category** Device Handler

**IA #** 10086

**Description** This API allows you to select a device.

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 POP input variable as a positive number.

Device selection can be done as shown in the example that follows.

**REF:** For a discussion of form feeds, please refer to the "Form Feeds" topic in the "Special Device Issues" chapter in this manual.

**Format** `^%ZIS`

**Input Variables**

%ZIS (optional) The %ZIS input 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 can be included in the string are described below. If the %ZIS input 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 (obsolete)—CLOSEST PRINTER: The closest printer, if one has been defined in the DEVICE file (#3.5), will be presented at the default response to the device prompt.

Q—QUEUING ALLOWED: The job can be queued to run at a later time. There is no automatic link between the Device Handler and the TaskMan. If queuing is allowed, just before the Device Handler is called, the application routine must set the %ZIS input variable to a string that includes the letter "Q".

For example:

```
>S %ZIS="MQ" D ^%ZIS
```

If the user selects queuing, the Device Handler will define the IO("Q") input variable 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.

**REF:** For further details on how to call the TaskMan interface, please refer to the "TaskMan: Programmer Tools" chapter in this manual.

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 (e.g., 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", which may not work in all environments, represents READ/WRITE access, "R" represents READ Only access, "W" represents WRITE access, and "A" represents Append mode. For example:

```
>S  %ZIS("HFSMODE")="R"
```

%ZIS("HFSNAME"): (optional) Use to pass the name of a Host file to %ZIS. For example:

```
>S  %ZIS("HFSNAME")="MYFILE.DAT"
```

%ZIS("IOPAR"): (optional) Use this input variable to pass open command variables to the Device Handler. If defined, the value of this input variable is used instead of any value specified in the OPEN PARAMETERS field of the DEVICE file (#3.5) (#3.5). The Device Handler uses the data from either this input variable or from the OPEN PARAMETERS field whether or not the device type is TRM.

On some M systems, Right Margin is an OPEN PARAMETERS. Therefore, any value for Right Margin in the DEVICE file (#3.5), TERMINAL TYPE file (#3.2), or user response can be ignored when this input 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:** The specific variables you pass may not be functional for all operating systems. Use of this feature should be limited to local development efforts.

%ZIS("IOUPAR"): (optional) Use this input variable in the same way as %ZIS("IOPAR"), but for variables to the USE (rather than OPEN) command. Any USE PARAMETERS specified in the DEVICE file (#3.5) will be overridden. For 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 input 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 input variables like ION or IOT. As with VA FileMan, the variable "Y" can be used in the screen to refer to the Internal Entry Number (IEN) 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:

```
>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 (#3.5), 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 can 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:

```
>S IOP="`202" D ^%ZIS
```

Using the IEN rather than device name can be useful when applications have the desired device stored as a pointer to the DEVICE file (#3.5) rather than as FREE TEXT.

**Output Variables**

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.

**CAUTION: 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 Kernel.

IO("CLOSE")

Device closed.

IO("DOC"):

SPOOL DOCUMENT NAME—If output has been sent to the spool device, this output variable holds the name of the spool document that was selected.

**NOTE:** This variable is KILLed when a call is made to ^%ZIS or HOME^%ZIS.

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 input variable should have the same value as that stored in the IO output variable. 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.

**NOTE:** This variable is KILLed when a call is made to ^%ZIS or HOME^%ZIS.

IO("P") This variable holds data about the new syntax requested.

IO("Q"): OUTPUT WAS QUEUED—If queuing is allowed (%ZIS["Q") and an output device for queuing is selected, this output variable is returned with a value of 1: IO("Q")=1. Otherwise, it will be undefined.

**NOTE:** This variable is KILLed when a call is made to ^%ZIS or HOME^%ZIS.

IO("S"): SLAVED DEVICE—When a slaved printer is selected, the Device Handler uses this output variable to save the subtype specification for the home device so that the appropriate close printer logic can be executed with X ^%ZIS("C").

IO("SPOOL"): SPOOLER WAS USED—The existence of this output variable indicates that output was sent to the spool device. It will exist temporarily, during spooling, and is KILLed upon normal exit.

**NOTE:** This variable is KILLed when a call is made to ^%ZIS or HOME^%ZIS.

IO("T") TaskMan call.

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 output variable. This code is used to WRITE a backspace with W @IOBS. |
| IOCPU: | CPU INDICATOR—If the selected device is on another CPU, this output variable is returned with the other CPU reference, obtained from the VOLUME SET (CPU) field in the DEVICE file (#3.5). The IOCPU input variable is used by TaskMan as an indicator of where the job should ultimately be run. |
| IOF: | FORM FEED—This output 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 returns 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 (#3.5). |
| 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 input 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 input variable has not been defined, IOUPAR is returned as NULL. |
| IOS: | DEVICE NUMBER—The DEVICE file (#3.5) Internal Entry Number (IEN) 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 (#3.2). |

IOST(0): SUBTYPE NUMBER—This variable returns the Internal Entry Number (IEN) of the selected device's subtype as recorded in the TERMINAL TYPE file (#3.2).

IOT: TYPE OF DEVICE—The DEVICE file (#3.5) holds an indication of Type for all devices. IOT returns the value of the device type (e.g., TRM for terminal, VTRM for virtual terminal, and HFS for Host File Server).

IOXY: CURSOR POSITIONING—This output 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 output 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 POP variable.

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 a caret ("^") or times out at the device prompt. Exceeding the TIMED READ at the right margin or address/variables 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 (#3.5). 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 (#3.5) 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:** 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.

### Example 1

This following is a simplified example; the process of issuing form feeds is not shown.

```
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
```

**Figure 4-1. ^%ZIS: Example**

### Example 2

The IOP variable can 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]]
```

### Example 3

If the SPOOL DOCUMENT NAME is included, then the RIGHT MARGIN and PAGE LENGTH are ignored. Therefore, use the following format if a spool device is desired:

```
>S IOP=[Q[;]][DEVICE NAME][;SUBTYPE][;SPOOL DOCUMENT NAME]
```

### Example 4

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
```

### Example 5

When setting the IOP variable, you can include the right margin:

```
>S IOP=ION_";"_IOM or S IOP=";120"
```

or

```
>S IOP="RXPRINTER;120"
```

In this example, ION is the local variable that contains the name of the device to be opened and the IOM variable contains the value of the desired right margin.

### Example 6

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
```

### Example 7

A *spool document name* can be passed to the Device Handler:

```
>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"
```

**REF:** For more information, please refer to the "Spooling" chapter in the *Kernel Systems Manual*.

In this example, 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.

### Example 8

Finally, the IOP variable can 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 API 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 APIs to support using more than one additional device simultaneously:

- OPEN^%ZISUTL(): Open Device with Handle
- USE^%ZISUTL(): Use Device Given a Handle
- CLOSE^%ZISUTL(): Close Device with Handle

These "multiple device" APIs are described later in this chapter.

### Host Files and ^%ZIS

Although it is possible to use the ^%ZIS API to manipulate Host files, the Host file API (in ^%ZISH) offers more robust Host file functionality.

**REF:** For more information on using the Host file API, please refer to the "Host Files" chapter in this manual.

## HLP1^%ZIS: Display Brief Device Help

**Reference Type** Supported

**Category** Device Handler

**IA #** 10086

**Description** This API displays brief help about device selection. There are no input parameters.

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.

**Format**

```
HLP1^%ZIS
```

**Input Parameters** none

**Output** none

**Example**

## HLP2^%ZIS: Display Device Help Frames

**Reference Type** Supported

**Category** Device Handler

**IA #** 10086

**Description** This API allows you to display extended help about device selection . The Help Processor is invoked to display a series of help frames. There are no input parameters.

While invoking the Help Processor involves a straightforward call in the production account (the ACTION^XQH4(): Print Help Frame Tree or EN1^XQH(): Display Help Frames APIs), it is a more complex matter in the Manager account where ^%ZIS resides. Hence, this call is provided.

**Format** `HLP2^%ZIS`

**Input Parameters** none

**Output** none

**Example**

# HOME^%ZIS: Reset Home Device IO Variables

**Reference Type** Supported

**Category** Device Handler

**IA #** 10086

**Description** This API sets the key IO variables to match the characteristics of the home device. The HOME^%ZIS API performs the same function as the obsolete CURRENT^%ZIS API. 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.

**Format** `HOME^%ZIS`

Make sure to perform the following steps before calling HOME^%ZIS:

- NEW all of the following output variables.
- Call HOME^%ZIS.

If you do not follow these steps, the variables could unintentionally assume the values of the variables of the current running task.

**Input Variables** none

**Output Variables**

| | |
|---|---|
| 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 (#3.5). |
| IOS: | Internal Entry Number (IEN) of last selected input/output device from the DEVICE file (#3.5). |
| IOSL: | Screen or page length. |
| IOST: | Subtype of the selected device. |
| IOST(0): | Subtype Internal Entry Number (IEN). |
| IOT: | Type of device, such as TRM for terminal. |
| IOXY: | Executable M code for cursor control. |

**Example**

## $$REWIND^%ZIS(): Rewind Devices

**Reference Type** Supported

**Category** Device Handler

**IA #** 10086

**Description** This extrinsic function rewinds special devices. These devices may be of the following types:

- Magtape
- Sequential Disk Processor
- Host File Server

**Format**

```
$$REWIND^%ZIS(io,iot,iopar)
```

**Input Parameters**

io: (required) The $IO representation of the device to be rewound, in the same format as IO, which is returned by ^%ZIS.

iot: (required) The "Type" of device to be rewound, in the same format as IOT, which is returned by ^%ZIS.

iopar: (required) The "Open Parameters" for the selected device, in the same format as IOPAR which is returned by ^%ZIS.

**Output**

returns: Returns:

1—Device was rewound successfully.

0—Device was *not* rewound successfully.

**Example**

```
>S Y=$$REWIND^%ZIS(IO,IOT,IOPAR)
```

## ^%ZISC: Close Device

**Reference Type** Supported

**Category** Device Handler

**IA #** 10089

**Description** This API closes a device opened with a call to the ^%ZIS API and restores the home device.

Do *not* issue a form feed when calling ^%ZISC. The Device Handler takes care of issuing a form feed if necessary (i.e., 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 input variable before calling ^%ZISC.

Before the ^%ZISC API 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 API 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").

**Format** `^%ZISC`

**Input Varia bles** various: For a list of input variables, please refer to the normal device output variables from the ^%ZIS: Standard Device Call API.

**Output Varia bles** various: For a list of output variables, please refer to the normal device output variables from the ^%ZIS: Standard Device Call API.

**Example**

```
>D ^%ZISC
```

# PKILL^%ZISP: Kill Special Printer Variables

**Reference Type** Supported

**Category** Device Handler

**IA #** 3172

**Description** This API KILLs printer-specific Device Handler variables. All output parameters defined by the PSET^%ZISP: Set Up Special Printer Variables API are KILLed.

**Format** `PKILL^%ZISP`

**Input Parameters** none

**Output** none

**Example**

## PSET^%ZISP: Set Up Special Printer Variables

**Reference Type** Supported

**Category** Device Handler

**IA #** 3172

**Description** This API defines a set of variables that toggle special printer modes. The corresponding fields in the TERMINAL TYPE file (#3.2) entry for the terminal type in question must be correctly set up, however; that is where PSET^%ZISP retrieves its output values.

**Format** `PSET^%ZISP`

Make sure to perform the following steps before calling PSET^%ZISP:

- NEW all of the following variables.
- Set the input variable.
- Call PSET^%ZISP.

If you do not follow these steps, the variables could unintentionally assume the values of the variables of the current running task.

| | | |
|---|---|---|
| **Input Variables** | IOST(0) | (required) Pointer to the TERMINAL TYPE entry for the printer in question, as set up by the Device Handler. |
| **Output Variables** | 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.

**Example**

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

**Reference Type** Supported

**Category** Device Handler

**IA #** 10088

**Description** This API sets up specific screen-handling variables and other terminal type attributes. Unlike the ENS^%ZISS: Set Up Screen-handling Variables API, which sets up all screen-handling variables, you specify which ones to set up with ENDR^%ZISS.

**Format** `ENDR^%ZISS`

Make sure to perform the following steps before calling ENDR^%ZISS:

- NEW all of the following variables.
- Set the input variables you want changed.
- Call ENDR^%ZISS.

If you do not follow these steps, the variables could unintentionally assume the values of the variables of the current running task.

**Input Variables**

IOST(0): (required) Internal entry number (IEN) of the selected device's subtype as recorded in the TERMINAL TYPE file (#3.2).

X: (required) Use this input 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 x variable, make two or more calls to ENDR^%ZISS, each with a partial list of the variable 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
```

For example:

```
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 Variables**

A subset of the output variables returned by ENS^%ZISS: Set Up Screen-handling Variables API are returned by ENDR^%ZISS, depending on what screen-handling variables are requested in the x input variable.

**Example**

## ENS^%ZISS: Set Up Screen-handling Variables

**Reference Type** Supported

**Category** Device Handler

**IA #** 10088

**Description** This API is used for screen management. This API sets up screen handling variables and other terminal type attributes.

**Format** `ENS^%ZISS`

Make sure to perform the following steps before calling ENS^%ZISS:

- NEW all of the following variables.
- Set the input variables you want changed.
- Call ENS^%ZISS.

If you do not follow these steps, the variables could unintentionally assume the values of the variables of the current running task.

**Input Variables**

IOST(0): (required) Internal entry number of the selected device's subtype as recorded in the TERMINAL TYPE file (#3.2).

%ZIS: (optional) If you define %ZIS = "I", the output array IOIS (mapping escape codes sent by input keys to input keys) is created.

**REF:** For a description of the IOIS nodes created, please refer to the Outputs that follow.

**Output Variables**

The output variables are listed below.

**NOTE:** Not all characteristics are possible on all terminal types. Also note that two of the variables 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/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, such as, 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, such as, 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: This 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 |

**Example**

## GKILL^%ZISS: KILL Graphic Variables

**Reference Type** Supported

**Category** Device Handler

**IA #** 10088

**Description** This API is used for screen management. It KILLs graphic variables used in screen handling. All output parameters set up by the GSET^%ZISS: Set Up Graphic Variables API are KILLed.

**Format** `GKILL^%ZISS`

**Input Parameters** none

**Output** none

**Example**

## GSET^%ZISS: Set Up Graphic Variables

**Reference Type** Supported

**Category** Device Handler

**IA #** 10088

**Description** This API is used for screen management. It 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.

**Format** `GSET^%ZISS`

Make sure to perform the following steps before calling GSET^%ZISS:

- NEW the input variable.
- Set the input variable.
- Call GSET^%ZISS.

If you do not follow these steps, the variables could unintentionally assume the values of the variables of the current running task.

| | | |
|---|---|---|
| **Input Variables** | IOST(0): | (required) Terminal Type. |
| **Output Variables** | 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.

**Example**

```
; write a horizontal line
D GSET^%ZISS
W IOG1
F I=1:1:20 W IOHL
W IOG0
D GKILL^%ZISS
```

**Figure 4-2. GSET^%ZISS: Example**

## KILL^%ZISS: KILL Screen Handling Variables

**Reference Type** Supported

**Category** Device Handler

**IA #** 10088

**Description** This API is used for screen management. It KILLs graphic variables used in screen handling. Only the output parameters set up by the ENS^%ZISS: Set Up Screen-handling Variables and ENDR^%ZISS: Set Up Specific Screen Handling Variables APIs are KILLed by this call.

**Format** `KILL^%ZISS`

**Input Parameters** none

**Output** none

**Example**

## CALL^%ZISTCP: Make TCP/IP Connection (Remote System)

**Reference Type** Supported

**Category** Device Handler

**IA #** 2118

**Description** This API is used to make a TCP/IP connection to a remote system.

**Format** `CALL^%ZISTCP`

Make sure to perform the following steps before calling CALL^%ZISTCP:

- NEW all of the following variables.
- Set the input variables you want changed.
- Call CALL^%ZISTCP.

If you do not follow these steps, the input variables could unintentionally assume the values of the variables of the current running task.

**Input Variables**

IPADDRESS: (required) This is the Internet Protocol (IP) address of the Host system to connect to. It must be in the IP format of four numbers separated by dots (e.g., 99.99.9.999).

SOCKET: (required) This is the socket to connect to on the remote host. It is an integer from 1-65535. Values below 5000 are reserved for standard Internet services (e.g., SMTP mail).

TIMEOUT: (optional) This is the timeout to apply to the Open.

**Output Variables**

IO: If the connection is made then IO variable will hold the implementation value that is used to reference the connection.

POP: This output variable reports the connection status:

- Successful—A value of zero (0) means the connection was successful.
- Unsuccessful—A positive value means the connection failed.

It works the same as a call to ^%ZIS: Standard Device Call.

**Example**

# CLOSE^%ZISTCP: Close TCP/IP Connection (Remote System)

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Device Handler | |
| **IA #** | 2118 | |
| **Description** | This API is used to close the connection opened with the CALL^%ZISTCP: Make TCP/IP Connection (Remote System) API. It works like a call to the ^%ZISC: Close Device API. | |
| **Format** | `CLOSE^%ZISTCP` | |
| **Input Variables** | various: | For a list of input variables, please refer to CALL^%ZISTCP: Make TCP/IP Connection (Remote System) API. |
| **Output Variables** | various | For a list of output variables, please refer to CALL^%ZISTCP: Make TCP/IP Connection (Remote System) API. |

**Example**

# CLOSE^%ZISUTL(): Close Device with Handle

**Reference Type** Supported

**Category** Device Handler

**IA #** 2119

**Description** This API closes a device opened with the OPEN^%ZISUTL(): Open Device with Handle API. 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. One of three functions that support using multiple devices at the same time.

**REF:** See also: OPEN^%ZISUTL(): Open Device with Handle and USE^%ZISUTL(): Use Device Given a Handle APIs.

**Format**

```
CLOSE^%ZISUTL(handle)
```

**Input Parameters**

| | |
|---|---|
| handle: | (required) The handle of a device opened with the OPEN^%ZISUTL(): Open Device with Handle API. |

**Output** none

**Example**

## OPEN^%ZISUTL(): Open Device with Handle

**Reference Type** Supported

**Category** Device Handler

**IA #** 2119

**Description** This API is used when you expect to be using multiple output devices. This API, as well as its two companion APIs: RMDEV^%ZISUTL(): Delete Data Given a Handle and CLOSE^%ZISTCP: Close TCP/IP Connection (Remote System), makes use of handles to refer to a device. A handle is a unique string identifying the device.

The three ^%ZISUTL APIs are essentially wrappers around the ^%ZIS API. They provide enhanced management of IO variables and the current device, especially when working with multiple open devices. One of three functions that support using multiple devices at the same time.

**REF:** See also: CLOSE^%ZISTCP: Close TCP/IP Connection (Remote System) and RMDEV^%ZISUTL(): Delete Data Given a Handle APIs.

**Format**

```
OPEN^%ZISUTL(handle[,valiop][,.valzis])
```

**Input Parameters**

handle: (required) 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: Standard Device Call API. 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: Standard Device Call API. Must be passed by reference.<br><br>**REF:** For more information, please refer to the ^%ZIS function documentation. |
| **Output Variables** | IOF: | OPEN^%ZISUTL returns all the same output variables as the ^%ZIS: Standard Device Call API. OPEN^%ZISUTL serves as a "wrapper" around the ^%ZIS: Standard Device Call API, providing additional management of IO output variables that ^%ZIS does not (principally to support opening multiple devices simultaneously).<br><br>**REF:** For more information on these input parameter, please refer to the ^%ZIS documentation. |
| | 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

**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
```

**Figure 4-3. OPEN^%ZISUTL: Example**

## RMDEV^%ZISUTL(): Delete Data Given a Handle

**Reference Type** Supported

**Category** Device Handler

**IA #** 2119

**Description** This API deletes the data associated with the handle. It does *not* change any of the IO* variables.

**Format** `RMDEV^%ZISUTL(handle)`

**Input Parameters** handle: (required) A unique Free Text name to associate with a device that you want to delete.

**Output** none

**Example**

## SAVDEV^%ZISUTL(): Save Data Given a Handle

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Device Handler | |
| **IA #** | 2119 | |
| **Description** | This API saves the current device IO* variables under the handle name. | |
| **Format** | `SAVDEV^%ZISUTL(handle)` | |
| **Input Parameters** | handle: | (required) A unique Free Text name to associate with a device that you want to save. |
| **Output** | none | |

**Example**

# USE^%ZISUTL(): Use Device Given a Handle

**Reference Type** Supported

**Category** Device Handler

**IA #** 2119

**Description** This API restores the IO variables for a device saved with the OPEN^%ZISUTL(): Open Device with Handle or SAVDEV^%ZISUTL(): Save Data Given a Handle APIs. It then does a USE of the device if it is open. The same as:

```
>DO USE^%ZISUTL(handle) U IO
```

**REF:** See also: OPEN^%ZISUTL(): Open Device with Handle and CALL^%ZISTCP: Make TCP/IP Connection (Remote System) APIs.

**Format**

```
USE^%ZISUTL(handle)
```

**Input Parameters**

| | |
|---|---|
| handle: | (required) A unique Free Text name to associate with the device that was opened with the OPEN^%ZISUTL(): Open Device with Handle API. |

**Output Variables**

| | |
|---|---|
| IO*: | Standard IO variables. |

**Example**

# Special Device Issues

This topic discusses the following special devices and device issues:

- Form Feeds
- Resources

## Form Feeds

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 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. Applications, 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 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 Kernel.

Device Handler also checks for the existence of the ionoff parameter when closing the device. Thus, application programmers can use the ionoff parameter to suppress form feeds by setting it just before calling ^%ZISC: Close Device API 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 **<Enter>** to continue.
6. A time-out at the READ or a caret ("^") response to the continue prompt represents a request to terminate the display. GO TO STEP #9.
7. If the user presses **<Enter>** 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.

```
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)
```

**Figure 4-4. Device Handler: Issuing form feeds following current guidelines**

```
ROU      ;SAMPLE ROUTINE
         S IOP="DEVNAM" D ^%ZIS G EXIT:POP
         I $D(IO("Q")) S ZTRTN="DQ^ROU",ZTDESC="SAMPLE REPORT" D ^%ZTLOAD,HOME^%ZIS
Q
.
.
.
DQ       ;SAMPLE REPORT
         S (END,PAGE)=0
         U IO F  Q:END  D
         .D HDR:$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)
```

**Figure 4-5. Device Handler: Alternate approach following current guidelines**

# Resources

## 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 parameter 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.

**REF:** For more information on using SYNC FLAGs, please refer to the "TaskMan: Programmer Tools" chapter in this manual.

# 5. Domain Name Service (DNS): Programmer Tools

## Application Program Interface (API)

Several APIs are available for programmers to work with Domain Name Service (DNS). These APIs are described below.

### $$ADDRESS^XLFNSLK(): Convert Domain Name to IP Addresses

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Domain Name Service (DNS) | |
| **IA #** | 3056 | |
| **Description** | This extrinsic function calls the Domain Name Service (DNS) to convert a domain name into its IP addresses. The IP addresses of the DNS being called are in the DNS IP field (#8989.3,51) in the KERNEL SYSTEM PARAMETERS file (#8989.3). | |
| **Format** | `$$ADDRESS^XLFNSLK(domain_name[,type])` | |
| **Input Parameters** | domain_name: | (required) This is the fully qualified domain name (e.g., FORUM.VA.GOV). |
| | type: | (optional) This input parameter is from the set A: address (the default), CNAME: alias. |
| **Output** | returns: | Returns a comma-separated list of IP addresses that are associated with the input domain. |

**Example**

```
>S X=$$ADDRESS^XLFNSLK("FORUM.VA.GOV")

>W X
10.2.29.130
```

## MAIL^XLFNSLK(): Get IP Addresses for a Domain Name

**Reference Type** Supported

**Category** Domain Name Service (DNS)

**IA #** 3056

**Description** This API calls the Domain Name Service (DNS) to get the MX records for a domain name with its IP addresses.

**Format** `MAIL^XLFNSLK(.return,domain_name)`

**Input Parameters**

| | |
|---|---|
| .return: | (required) A local variable passed by reference to hold the return array. |
| domain_name: | (required) This parameter is a fully qualified domain name (e.g., FORUM.VA.GOV). |

**Output Parameters**

| | |
|---|---|
| .return: | Returns data in the array passed in by reference. The data is subscripted by priority. domain_name is a full domain name (e.g., FORUM.VA.GOV). |

**Example**

```
>K ZX D MAIL^XLFNSLK(.ZX,"ISC-SF.MED.VA.GOV") ZW ZX
ZX=2
ZX(5)=a2.ISC-SF.MED.VA.GOV.^10.6.21.15
ZX(10)=a1.ISC-SF.MED.VA.GOV.^10.6.21.14
```

# 6. Electronic Signatures: Programmer Tools

## Application Program Interface (API)

Several APIs are available for programmers to work with electronic signatures. These APIs are described below.

### ^XUSESIG: Set Up Electronic Signature Code

**Reference Type** Controlled Subscription

**Category** Electronic Signatures

**IA #** 936

**Description** This API, when called from the top, allows the user to set up a personal electronic signature code. It is used within application code to allow the user immediate on-the-fly access to set up the electronic signature, rather than force the user to leave the application and enter a different option to do the same.

**Format** `^XUSESIG`

**Input Parameters** none

**Output** none

**Example**

## SIG^XUSESIG(): Verify Electronic Signature Code

**Reference Type** Supported

**Category** Electronic Signatures

**IA #** 10050

**Description** This API requests and verifies the electronic signature code of the current user.

**Format** `SIG^XUSESIG(duz,x1)`

**Input Parameters**

| | |
|---|---|
| duz: | (required) User number. |

**Output Parameters**

| | |
|---|---|
| x1: | If the user entered the correct electronic signature code, the encrypted electronic signature code as stored in the NEW PERSON file (#200) is returned in x1. Otherwise, x1 is returned as NULL. |

**Example**

## $$CHKSUM^XUSESIG1(): Build Checksum for Global Root

**Reference Type** Supported

**Category** Electronic Signatures

**IA #** 1557

**Description** This extrinsic function takes a global root ($name_value) and builds a checksum for all data in the root.

**NOTE:** The flag input parameter is no longer used. Previously, It was used when there was more than one checksum algorithm.

**Format** `$$CHKSUM^XUSESIG1($name_value[,flag])`

**Input Parameters**

| | |
|---|---|
| $name_value: | (required) This is a global root as would be returned from $NAME. |
| flag: | (obsolete) Not used at this time. |

**Output**

| | |
|---|---|
| returns | Returns the checksum for the global root. |

**Example**

## $$CMP^XUSESIG1(): Compare Checksum to $Name_Value

**Reference Type** Supported

**Category** Electronic Signatures

**IA #** 1557

**Description** This extrinsic function compares the checksum passed in to the calculated value from the $NAME_VALUE. It Returns the following:

1—Match.

0—No match.

**Format** `$$CMP^XUSESIG1(checksum,$name_value)`

**Input Parameters**

| | |
|---|---|
| checksum: | (required) The output from the $$CHKSUM^XUSESIG1(): Build Checksum for Global Root API. |
| $name_value: | (required) This is a global root as would be returned from $NAME. |

**Output**

| | |
|---|---|
| returns: | Returns:<br>1—Match.<br>0—No match. |

**Example**

## $$DE^XUSESIG1(): Decode String

**Reference Type** Supported

**Category** Electronic Signatures

**IA #** 1557

**Description** This extrinsic function decodes the input string using the checksum as the key.

**Format** `$$DE^XUSESIG1(checksum,encoded_string)`

**Input Parameters**

| | |
|---|---|
| checksum: | (required) The output from the $$CHKSUM^XUSESIG1(): Build Checksum for Global Root API. |
| encoded_string: | (required) The output from the $$EN^XUSESIG1(): Encode Esblock API. |

**Output** returns: Returns the decoded string.

**Example**

## $$EN^XUSESIG1(): Encode Esblock

**Reference Type** Supported

**Category** Electronic Signatures

**IA #** 1557

**Description** This extrinsic function encodes the ESBLOCK using the checksum as the key.

**Format** `$$EN^XUSESIG1(checksum,esblock)`

**Input Parameters**

| | |
|---|---|
| checksum: | (required) A number that reveals if the data in the root has been changed. |
| esblock: | (optional) This should be the data returned from the $$ESBLOCK^XUSESIG1(): E-Sig Fields Required for Hash API. |

**Output** returns: Returns encoded ESBLOCK.

**Example**

## $$ESBLOCK^XUSESIG1(): E-Sig Fields Required for Hash

**Reference Type** Supported

**Category** Electronic Signatures

**IA #** 1557

**Description** This extrinsic function returns the set of fields from the NEW PERSON file (#200) that are needed as part of the hash for an acceptable electronic signature (E-Sig). These fields include the following:

- E-Sig Block
- E-Sig Title
- Degree
- Current Date/Time

If the Internal Entry Number (IEN) is *not* passed in then the DUZ is used.

**Format** `$$ESBLOCK^XUSESIG1([ien])`

**Input Parameters**

| | |
|---|---|
| ien: | (optional) This is the Internal Entry Number (IEN) of the NEW PERSON file (#200) entry for which data is requested. The default is to use the DUZ of the current user. |

**Output**

| | |
|---|---|
| returns: | Returns the following fields:<br>E-Sig Block<br>E-Sig Title<br>Degree<br>Current Date/Time |

**Example**

## DE^XUSHSHP(): Decrypt Data String

**Reference Type** Supported

**Category** Electronic Signatures

**IA #** 10045

**Description** This API decrypts a string encrypted by a call to the EN^XUSHSHP(): Encrypt Data String API. Typically, this API would be used to decrypt strings when printing a document containing encrypted strings.

**Format** `DE^XUSHSHP(x,x1,x2)`

**Input Parameters**

| | |
|---|---|
| x: | (required) Encrypted string generated by a call to EN^XUSHSHP(): Encrypt Data String. |
| x1: | (required) Identification number used as the x1 input parameter in the EN^XUSHSHP(): Encrypt Data String call. |
| x2: | (required) Number used as the x2 input parameter in the EN^XUSHSHP(): Encrypt Data String call. |

**Output Parameters** x: The decrypted string (can be printed).

**Example**

## EN^XUSHSHP(): Encrypt Data String

**Reference Type** Supported

**Category** Electronic Signatures

**IA #** 10045

**Description** This API 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 Data String API, with the encrypted string, identification number, and document number as input parameters. Typically, this API would be used to encrypt strings within a document.

**Format** `EN^XUSHSHP(x,x1,x2)`

**Input Parameters**

| | |
|---|---|
| x: | (required) The string to be encrypted (such as the contents of the SIGNATURE BLOCK PRINTED NAME field in the NEW PERSON file [#200]). |
| x1: | (required) An identification number such as DUZ. |
| x2: | (required) A document number (or the number one). |

**Output Parameters** x: Encrypted string.

**Example**

## HASH^XUSHSHP(): Hash Electronic Signature Code

**Reference Type** Supported

**Category** Electronic Signatures

**IA #** 10045

**Description** This API 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 [#200]).

**Format** `HASH^XUSHSHP(x)`

**Input Parameters**

| | |
|---|---|
| x: | (required) Electronic Signature code as entered by the user. |

**Output Parameters**

| | |
|---|---|
| x: | Hashed form of the electronic signature code submitted as input to function. |

**Example**

# 7. Error Processing: Programmer Tools

## Application Program Interface (API)

Several APIs are available for programmers to work with error processing. These APIs are described below.

### $$EC^%ZOSV: Get Error Code

| | |
|---|---|
| **Reference Type** | Supported |
| **Category** | Operating System Interface |
| **IA #** | 10097 |
| **Description** | This extrinsic function returns the most recent error message recorded by the operating system. |
| **Format** | `$$EC^%ZOSV` |
| **Input Parameters** | none |
| **Output** | returns: Returns the most recent error code/message. |

**Example**

```
>S X=$$EC^%ZOSV
```

## ^%ZTER: Kernel Standard Error Recording Routine

**Reference Type** Supported

**Category** Error Processing

**IA #** 1621

**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 (#3.075).

The application-specific error trap routine, when it is called as a result of an error, can then use the ^%ZTER API to record error information in the ERROR LOG file (#3.075) 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 (#3.075).

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.

**Format** ^%ZTER

**Symbol Table** %ZT (optional) The %ZT array can be used to identify a global node whose descendents 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. It is not a parameter, but if it is in the symbol table it is used:

```
>S %ZT("^TMP($J)")=""
>D ^%ZTER
```

**Output** %ZTERROR Calls to the error recorder always return this parameter. 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).

### Example

The following is an example of the Error Trap:

```
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.
         ;
         N $ESTACK,$ETRAP S $ETRAP="D ERR^ZXGAPP"
         ;
         ; 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" S $ECODE="" G FILECLOS ; continue if EOF error
         D ^%ZTER ; record the error if anything other than EOF
         D UNWIND^%ZTER ; unwind the stack, return to caller.
         Q
         ;
```

**Figure 7-1. Error Trap: Example**

## $$NEWERR^%ZTER: Verify Support of Standard Error Trapping (Obsolete)

**NOTE:** This API is obsolete, because all VA systems support the standard error trapping.

**Reference Type** Supported

**Category** Error Processing

**IA #** 1621

**Description** This extrinsic function reports if the current platform supports the standard error trapping. It returns:

1—If the standard error trapping is supported.

0—For all other cases.

**Format** `$$NEWERR^%ZTER`

**Input Parameters** none

**Output** returns: Returns:

1—If the standard error trapping is supported.

0—For all other cases.

**Example**

# UNWIND^%ZTER: Quit Back to Calling Routine

**Reference Type** Supported

**Category** Error Processing

**IA #** 1621

**Description** This API is used after a package error trap to quit back to the calling routine. Control returns to the level above the one that NEWED $ESTACK.

**Format** `UNWIND^%ZTER`

**Input Parameters** none

**Output** none

**Example**

Main:

```
S X=1 D SUB
W X
Q SUB    N $ESTACK,$ETRAP S $ETRAP="D ERROR"
S X=1/0
Q
```

**Figure 7-2. UNWIND^%ZTER: Main code example**

Usage:

```
D ^%ZTER ;This will record the error info and clear $ECODE
S ^XXX="Incomplete record"
G UNWIND^%ZTER
```

**Figure 7-3. UNWIND^%ZTER: Usage**

# 8. Field Monitoring: Programmer Tools

## Application Program Interface (API)

One API is available for programmers to work with field monitoring. This API is described below.

### OPKG^XUHUI(): Monitor New Style Cross-referenced Fields

**Reference Type** Supported

**Category** Field Monitoring

**IA #** 3589

**Description** This API allows other packages to task an Option or Protocol from a New Style cross-reference. This API can be used to monitor any field or fields in any file using a New Style cross-reference.

**Format** `OPKG^XUHUI([xuhuiop,]xuhuinm[,xuhuia],xuhuixr)`

**Input Parameters**

xuhuiop (optional) This parameter is a set of Numeric codes that tells the Unwinder whether to use the PROTOCOL file (#101) or the OPTION file (#19). If this parameter is null, the default value will be used (i.e., "101"):

- 101 (default)—PROTOCOL file (#101) will be used.
- 19—The OPTION file (#19) will be used.

xuhuinm (required) This parameter is the NAME (#.01) value of the Protocol or Option that is to be launched.

xuhuia (optional) This parameter is a Set of Codes. If this input parameter is null, the default value will be used (i.e., "S"):

- S (default)—The data being passed is from the SETting of the cross-reference.
- K—The data being passed is from the KILLing of the cross-reference.

xuhuixr (required) This parameter is the name of the cross-reference.

**Output** See Example Monitored fields with a New Style cross-reference.

### Example

The Hui Project needs to monitor the following fields at the top level of the NEW PERSON file (#200) for changes in value, in the order listed:

- NAME (#.01)
- TERMINATION DATE (#9.2)
- DOB (#5)
- SSN (#9)

### Create New Style Cross-references

Create a MUMPS New Style cross-reference for the fields that are to be monitored for value changes, as shown below:

```
Index Name: AXUHUI (#n)

Short Description: Hui Project Top File Cross-reference

      Description: This MUMPS New Style cross-reference is on non-multiple
                   fields in the NEW PERSON file (#200) that the Hui Project
                   needs to monitor for changes in value.  The following fields
                   are being monitored in the order listed:

                          .01 (NAME)
                           9.2 (TERMINATION DATE)
                           5 (DOB)
                           9 (SSN)
                   For details on how this cross-reference processes changes,
                   please refer to the patch description for Kernel Patch XU*8*236.
                   For more detailed information about the MUMPS New Style
                   cross-reference, please refer to the "VA FileMan V. 22.0 Key
                   and Index Tutorial" at the following web address:


http://vista.med.va.gov/fileman/fileMan_training/online_pres/FMtut_frm.htm
            (see Lessons #5 and #6)

               Type: MUMPS

          EXECUTION: RECORD

                Use: ACTION

      Set Logic:  D OPKG^XUHUI("","XUHUI FIELD CHANGE EVENT","","AXUHUI") Q
     Kill Logic:  Q
     Whole Kill:  Q
           X(1):  NAME  (200,.01)  (forwards)
           X(2):  TERMINATION DATE  (200,9.2)  (forwards)
           X(3):  DOB  (200,5)  (forwards)
           X(4):  SSN  (200,9)  (forwards)
```

**Figure 8-1. OPKG^XUHUI: Example of creating New Style Cross-references**

### Sample Scenario

Change a monitored (cross-referenced) field(s) value in the NEW PERSON file (#200), as shown below:

```
INPUT TO WHAT FILE: NEW PERSON// <Enter>
EDIT WHICH FIELD: ALL// DOB
THEN EDIT FIELD: SSN
THEN EDIT FIELD: <Enter>

Select NEW PERSON NAME: KRNUSER <Enter>    KRNUSER,ONE     OK
DOB: JUL 4,1950// 12.24.49 <Enter> (DEC 24, 1949)
SSN: 000220000// 000558888
```

**Figure 8-2. OPKG^XUHUI API: Sample scenario**

Here we have changed ONE KRNUSER's Date of Birth (DOB) from 07/04/50 to 12/24/49 and changed his Social Security Number (SSN) from 000-22-0000 to 000-55-8888. Since these fields are being monitored (i.e., MUMPS New Style cross-reference, see the "Create Cross-references" previous topic), we should see this data passed to the "XUHUI FIELD CHANGE EVENT" protocol (see the "Internal Results for Developers" topic that follows).

### Internal Results for Developer

The following data is passed to the "XUHUI FIELD CHANGE EVENT" Protocol via the Kernel OPKG^XUHUI API that is called in the AXUHUI cross-reference (see the "Create Cross-references" previous topic).

```
-----------------------------------------------------------------------
If executing the Kill logic, then the 'X' array will be equal to the 'X1'
array.  If executing the Set logic, then the 'X' array will be equal to
the 'X2' array.
-----------------------------------------------------------------------
X=KRNUSER,ONE
X(1)=KRNUSER,ONE
X(2)=
X(3)=2491224
X(4)=000558888
-----------------------------------------------------------------------
X1=KRNUSER,ONE
X1(1)=KRNUSER,ONE
X1(2)=
X1(3)=2500704
X1(4)=000220000
-----------------------------------------------------------------------
X2=KRNUSER,ONE
X2(1)=KRNUSER,ONE
X2(2)=
X2(3)=2491224
X2(4)=000558888
-----------------------------------------------------------------------
XUHUIA=S
XUHUIDA=70
XUHUIFIL=200
XUHUIFLD=
XUHUINM=XUHUI FIELD CHANGE EVENT
XUHUIOP=101
XUHUIX=KRNUSER,ONE
XUHUIX(1)=KRNUSER,ONE
XUHUIX(2)=
XUHUIX(3)=2491224
XUHUIX(4)=000558888
XUHUIX1=KRNUSER,ONE
XUHUIX1(1)=KRNUSER,ONE
XUHUIX1(2)=
XUHUIX1(3)=2500704
XUHUIX1(4)=000220000
XUHUIX2=KRNUSER,ONE
XUHUIX2(1)=KRNUSER,ONE
XUHUIX2(2)=
XUHUIX2(3)=2491224
XUHUIX2(4)=000558888
XUHUIXR=AXUHUI
```

**Old values are in this array.**

**New values are in this array.**

**"S" = Set Logic is being executed,**
**"K" = Kill logic being executed.**

**"DA" array, File number,& Field number(s) if available.**

**Name of Extended Action entry in File #101 or in File #19.**

**File number of where to find the Extended Action.**

**The "X" array.**

**The "X1" array.**

**The "X2" array.**

**Name of cross-reference being executed by DIK.**

**Figure 8-3. OPKG^XUHUI: Example of internal results**

# 9. File Access Security: Programmer Tools

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

**REF:** For an overview of the functionality provided by the File Access Security system, please refer to the "File Access Security" chapter in the *Kernel Systems Manual*.

## 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 DLAYGO variable 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. However, if navigation to a second file is involved, LAYGO is not automatically granted. One of the three conditions mentioned above must be met to allow navigation to the second file:

- LAYGO access is granted.
- DUZ(0)=**@.**
- DLAYGO variable 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
EDIT WHICH FIELD: TRANSACTION NUMBER
THEN EDIT FIELD: DATE RENTED
THEN EDIT FIELD: S DLAYGO=800265
THEN EDIT FIELD: LINE ITEM:
   By 'LINE ITEM', do you mean the LINE ITEM File,
       pointing via its 'RENTAL TRANSACTION' Field? YES// Y <Enter> (YES)
WILL TERMINAL USER BE ALLOWED TO SELECT PROPER ENTRY IN 'LINE ITEM' FILE? YES//
<Enter> (YES)
DO YOU WANT TO PERMIT ADDING A NEW 'LINE ITEM' ENTRY? NO// Y <Enter> (YES)
WELL THEN, DO YOU WANT TO **FORCE** ADDING A NEW ENTRY EVERY TIME? NO// <Enter>
(NO)
DO YOU WANT AN 'ADDING A NEW LINE ITEM' MESSAGE? NO// N <Enter> (NO)
   EDIT WHICH LINE ITEM FIELD: LINE ITEM
   THEN EDIT LINE ITEM FIELD: RENTAL TRANSACTION
   THEN EDIT LINE ITEM FIELD: K DLAYGO
   THEN EDIT LINE ITEM FIELD:
```

**A file pointed-to by the Line Item file.**

**Set DLAYGO to the number of the file to be navigated-to via backward pointing.**

**KILL DLAYGO upon exit.**

**Figure 9-1. File Access Security: Setting DLAYGO in a template**

# 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.

**REF:** For more information on DLAYGO as used in ^DIC calls, please refer to 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.

**REF:** For more information on DIDEL as used in ^DIE calls, please refer to the *VA FileMan Programmer Manual*.

# 10. Help Processor: Programmer Tools

## Entry and Exit Execute Statements

The HELP FRAME file (#9.2) 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 (#9.2) contains a pointer to the OBJECT file (#2005), a file that is maintained by the Washington DC OIFO. It has been established so that images (e.g., cardiac catheterization films) can be integrated within an educational help system on multimedia workstations.

# Application Program Interface (API)

Several APIs are available for programmers to work with help processing. These APIs are described below.

## EN^XQH(): Display Help Frames

**Reference Type** Supported

**Category** Help Processor

**IA #** 10074

**Description** This API displays a help frame. It immediately clears the screen and displays the help frame (unlike the EN1^XQH(): Display Help Frames API, which does *not* clear the screen and offers the user a choice of whether to load the help frame).

**Format** `EN^XQH(xqh)`

**Input Parameters**

| | |
|---|---|
| xqh: | (required) Help Frame name (the .01 value from the HELP FRAME file [#9.2]). |

**Output** none

**Example**

## EN1^XQH(): Display Help Frames

**Reference Type** Supported

**Category** Help Processor

**IA #** 10074

**Description** This API displays a help frame as ACTION^XQH4(): Print Help Frame Tree 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 **<Enter>**, the help frame is displayed.

**Format** `EN1^XQH(xqh)`

**Input Parameters**

| | |
|---|---|
| xqh: | (required) Help Frame name (the .01 value from the HELP FRAME file [#9.2]). |

**Output** none

**Example**

## ACTION^XQH4(): Print Help Frame Tree

**Reference Type** Supported

**Category** Help Processor

**IA #** 10080

**Description** This API 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 parameter.

**Format**

```
ACTION^XQH4(xqhfy[,xqfmt])
```

**Input Parameters**

xqhfy: (required) Help frame name, equal to the .01 field of the desired entry in the HELP FRAME file (#9.2). 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 xwfmt 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

**Example**

# 11. Host Files: Programmer Tools

## Application Program Interface (API)

Several APIs are available for programmers to work with Host files. These APIs are described below.

The traditional method of working with Host File System (HFS) files prior to Kernel V. 8.0 was to use the device handler API (^%ZIS). Using several input parameters, you could open a Host file (given a Host file device entry in the DEVICE file [#3.5]). For example:

```
S %ZIS("HFSNAME")="ARCHIVE.DAT"
S %ZIS("HFSMODE")="W"
S IOP="HFS" D ^%ZIS Q:POP
U IO D...
```

**Figure 11-1. Host Files: Opening a Host file using the ^%ZIS API**

Kernel V. 8.0 provides a set of APIs for working with Host files. The Host file APIs 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.
- $$LIST^%ZISH — Get a list of files in a directory.
- $$MV^%ZISH — Rename Host file.
- OPEN^%ZISH — Open Host file (bypass device handler).
- $$PWD^%ZISH — Get name of current directory.
- $$STATUS^%ZISH — Return end-of-file status.

The following definitions apply for the Host file APIs:

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 READs only.

"W" WRITE, use the file for writing. If the file exists, it is truncated to a length of zero (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

**Reference Type** Supported

**Category** Host Files

**IA #** 2320

**Description** This API closes a Host file that was opened with the OPEN^%ZISH(): Open Host File API.

**Format** `CLOSE^%ZISH(handle)`

**Input Parameters** handle: (required) Handle used when file was opened with the OPEN^%ZISH(): Open Host File API.

**Output** none

**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")
```

**Figure 11-2. CLOSE^%ZISH: Example**

## $$DEFDIR^%ZISH(): Get Default Host File Directory

**Reference Type** Supported

**Category** Host Files

**IA #** 2320

**Description** This extrinsic function gets the default Host file directory. It has two modes:

- NULL/Missing Parameter—If it is called with a NULL/missing parameter, it returns the "default directory for HFS files" from the KERNEL SYSTEM PARAMETERS file (#8989.3).
- Directory Parameter—If it is called with a parameter, it must be the directory for a file. This parameter is checked to see that it is in the correct format for the operating system in question.

**Format** `$$DEFDIR^%ZISH([df])`

**Input Parameters**

| | |
|---|---|
| df: | (optional) This is the directory path upon which a simple format check is made. For the NT operating system it changes "/" to "\" and makes sure that there is a trailing "\". There is no error response. |

**Output**

| | |
|---|---|
| returns: | Returns the default Host file directory. |

**Example**

## $$DEL^%ZISH(): Delete Host File

**Reference Type** Supported

**Category** Host Files

**IA #** 2320

**Description** This extrinsic function deletes 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.

**Format** `$$DEL^%ZISH(path,arrname)`

**Input Parameters**

path: (required) Full path, up to but not including the filename.

arrname: (required) Fully resolved array name containing the files to delete as subscripts at the next descendent 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**

returns: Returns:

1—Success for all deletions.

0—Failure on at least one deletion.

### Example

```
>K FILESPEC
>S FILESPEC("TMP.DAT")=""
>S Y=$$DEL^%ZISH("\MYDIR\",$NA(FILESPEC))
```

# $$FTG^%ZISH(): Load Host File into Global

**Reference Type** Supported

**Category** Host Files

**IA #** 2320

**Description** This extrinsic function loads 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."
```

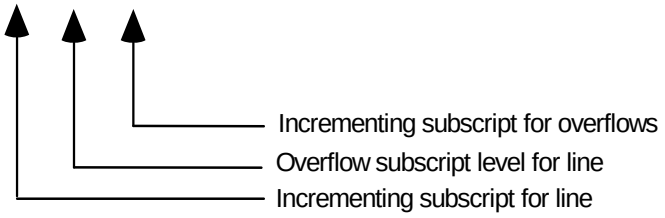


**Figure 11-3. Host Files: Overflow lines in a Host file sample**

**Format**

```
$$FTG^%ZISH(path,filename,global_ref,inc_subscr[,ovfsub])
```

**Input Parameters**

path: (required) Full path, up to but not including the filename.

filename: (required) Name of the file to open.

global_ref: (required) 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: (required) 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, such as ^TMP(115,1,x), but will WRITE nodes at the full global reference, such as ^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** | returns: | Returns:<br>1—Success.<br>0—Failure. |

**Example**

```
>S Y=$$FTG^%ZISH("USER$:[COMMON]","MYFILE.DAT",$NA(^MYGLOBAL(612,1,0)),2)
```

## $$GATF^%ZISH(): Copy Global to Host File

**Reference Type** Supported

**Category** Host Files

**IA #** 2320

**Description** This extrinsic function is used in the same way as the $$GTF^%ZISH(): Copy Global to Host File. The one difference is that if the file already exists, $$GATF^%ZISH appends global nodes to the existing file rather than truncating the existing file first.

**REF:** For more information, please refer to the $$GTF^%ZISH(): Copy Global to Host File API description.

**Format**

```
$$GATF^%ZISH(global_ref,inc_subscr,path,filename)
```

**Input Parameters**

| | |
|---|---|
| global_ref: | (required) Global to READ lines from, fully resolved in closed root form. |
| inc_subscr: | (required) 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, $$GATF 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: | (required) Full path, up to but not including the filename. |
| filename: | (required) Name of the file to open. |

**Output**

returns: Returns:

1—Success.

0—Failure.

**Example**

## $$GTF^%ZISH(): Copy Global to Host File

**Reference Type** Supported

**Category** Host Files

**IA #** 2320

**Description** This extrinsic function WRITEs 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 zero (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.

**Format** `$$GTF^%ZISH(global_ref,inc_subscr,path,filename)`

**Input Parameters**

| | |
|---|---|
| global_ref: | (required) Global to READ lines from, fully resolved in closed root form. |
| inc_subscr: | (required) 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: | (required) Full path, up to but not including the filename. |
| filename: | (required) Name of the file to open. |

**Output**

| | |
|---|---|
| returns: | Returns:<br>1—Success.<br>0—Failure. |

**Example**

```
>S Y=$$GTF^%ZISH($NA(^MYGLOBAL(612,1,0)),2,"USER$:[COMMON]","MYFILE.DAT")
```

## $$LIST^%ZISH(): List Directory

**Reference Type** Supported

**Category** Host Files

**IA #** 2320

**Description** This extrinsic function returns a list of file names in the current directory. The list is returned in an array in the variable named by the third parameter.

**Format**

```
$$LIST^%ZISH(path,arrname,retarrnam)
```

**Input Parameters**

path: (required) Full path, up to but not including any filename. For current directory, pass the NULL string.

arrname: (required) Fully resolved array name containing file specifications to list at the next descendent 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: (required) 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 Parameters**

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.)

**Output**

returns: Returns:

1—Success.

0—Failure.

**Example**

```
>K FILESPEC,FILE
>S FILESPEC("L*")="",FILESPEC("P*")=""
>S Y=$$LIST^%ZISH("","FILESPEC","FILE")
```

## $$MV^%ZISH(): Rename Host File

**Reference Type** Supported

**Category** Host Files

**IA #** 2320

**Description** This extrinsic function renames 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.

**Format** `$$MV^%ZISH(path1,filename1,path2,filename2)`

**Input Parameters**

| | |
|---|---|
| path1: | (required) Full path of the original file, up to but not including the filename. |
| filename1: | (required) Name of the original file. |
| path2: | (required) Full path of renamed file, up to but not including the filename. |
| filename2: | (required) Name of the renamed file. |

**Output**

returns: Returns:

1—Success.

0—Failure.

**Example**

```
>S Y=$$MV^%ZISH("","TMP.DAT","","ZXG"_I_".DAT")
```

## OPEN^%ZISH(): Open Host File

**Reference Type** Supported

**Category** Host Files

**IA #** 2320

**Description** This API 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 API with the handle you used to open the file.

**Format**

```
OPEN^%ZISH([handle][,path,]filename,mode[,max][,subtype])
```

**Input Parameters**

| | |
|---|---|
| handle: | (optional) Unique name you supply to identify the opened device. |
| path: | (optional) Full directory path, up to but not including the filename. If not supplied, the default HFS directory will be used. |
| filename: | (required) Name of the file to open. |
| mode: | (required) Mode to open file:<br>"W"—WRITE<br>"R"—READ<br>"A"—APPEND<br>"B"—BLOCK (fixed record size). |
| max | (optional) Maximum record size for a new file. |
| subtype | (optional) File subtype. |

**Output Variables**

| | |
|---|---|
| POP: | A value of zero (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. |

**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")
```

**Figure 11-4. OPEN^%ZISH: Example**

## $$PWD^%ZISH: Get Current Directory

**Reference Type** Supported

**Category** Host Files

**IA #** 2320

**Description** This extrinsic function returns the name of the current working directory.

**Format** `$$PWD^%ZISH`

**Input Parameters** none

**Output** returns: Returns:

String—The string representing the current directory specification, including device if any.

NULL—If a problem occurs while retrieving the current directory.

**Example**

```
>S Y=$$PWD^%ZISH()
```

## $$STATUS^%ZISH: Return End-of-File Status

**Reference Type** Supported

**Category** Host Files

**IA #** 2320

**Description** This extrinsic function returns the current end-of-file status. If end-of-file has been reached, $$STATUS^%ZISH returns 1. Otherwise, it returns 0.

**Format** `$$STATUS^%ZISH`

**Input Parameters** none

**Output** returns: Returns:

1—End-of-file (EOF) has been reached.

0—End-of-file (EOF) has *not* been reached.

**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")
```

**Figure 11-5. $$STATUS^%ZISH: Example**

# 12. Institution File: Programmer Tools

## Application Program Interface (API)

Several APIs are available for programmers to work with the INSTITUTION file (#4). These APIs are described below.

### $$ACTIVE^XUAF4(): Institution Active Facility (True/False)

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This extrinsic function, given the Internal Entry Number (IEN) in the INSTITUTION file (#4), returns the Boolean value for the question—is this an active facility? It checks to see if the INACTIVE FACILITY FLAG field (#101) is *not* set.

**Format** `$$ACTIVE^XUAF4(ien)`

**Input Parameters**

ien: (required) Internal Entry Number (IEN) of the institution in question.

**Output**

returns: Returns a Boolean value:

True (non-zero)—Station Number is an active facility.

False (zero)—Station Number is *not* an active facility. The INACTIVE FACILITY FLAG field (#101) has a value indicating it is inactive.

**Example**

## CDSYS^XUAF4(): Coding System Name

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This API returns the Coding System name.

**Format** `CDSYS^XUAF4(y)`

**Input Parameters** y: (required) Pass by reference, returns:

```
Y(coding_system) =
$D_of_local_system^ coding_system name
```

**Output Parameters** y: Passed by reference, returns:

```
Y(coding_system) =
$D_of_local_system^ coding_system name
```

**Example**

## CHILDREN^XUAF4(): List of Child Institutions for a Parent

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

Description This API returns a list of all institutions that make up a given Veterans Integrated Service Network (VISN), parent institution entered in the "parent" input parameter.

**Format** `CHILDREN^XUAF4(array,parent)`

**Input Parameters**

array — (required) $NAME reference to store the list of institutions that make up the parent VISN institution for the "parent" input parameter.

parent — (required) Parent (VISN) institution lookup value, any of the following:

- Internal Entry Number (IEN), will have the ` in front of it.
- Station Number
- Station Name

**Output**

returns: Returns the array populated with the list of institutions that make up the parent VISN.

```
Variable array
("c",ien)=station_name^station_number
```

**Example**

## $$CIRN^XUAF4(): Institution CIRN-enabled Field Value

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This extrinsic function returns the value of the CIRN-enabled field from the INSTITUTION file (#4).

**Format** `$$CIRN^XUAF4(inst[,value])`

**Input Parameters**

inst: (required) Institution lookup value, any of the following:

- Internal Entry Number (IEN), will have the ` in front of it.
- Station Number
- Station Name

value: (optional) Restricted to use by CIRN. This input parameter allows the setting of the field to a new value.

**Output**

returns: Returns the CIRN-enabled field value.

**Example 1**

**Example 2**

## F4^XUAF4(): Institution Data for a Station Number

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This API returns the Internal Entry Number (IEN) and other institution data, including historical information, for a given STATION NUMBER (#99) in the INSTITUTION file (#4).

**Format** `F4^XUAF4(sta,[.]array[,flag][,date])`

| | | |
|---|---|---|
| **Input Parameters** | sta: | (required) Station Number. |
| | [.]array: | (required) $NAME reference for return values. |
| | flag: | (optional) Flags that represent the Station Number Status. Possible values are:<br>• A—Active entries only.<br>• M—Medical treating facilities only. |
| | date: | (optional) Return name on this VA FileMan internal date. |
| **Output** | array | IEN or "0^error message" |
| | array("NAME") | Name |
| | array("VA NAME") | Official VA Name |
| | array("STATION NUMBER") | Station Number |
| | array("TYPE") | Facility Type Name |
| | array("INACTIVE") | Inactive Date (0=not inactive)<br>**NOTE:** If inactive date *not* available, then 1. |
| | array("REALIGNED TO") | IEN^station number^date |
| | array("REALIGNED FROM") | IEN^station number^date |
| | array("MERGE",IEN") | Merged Records |

### Example

```
>D F4^XUAF4("528A8",.ARRAY)

>ZW ARRAY
ARRAY=7020
ARRAY("INACTIVE")=0
ARRAY("NAME")=ALBANY
ARRAY("REALIGNED FROM")=500^500^3000701
ARRAY("STATION NUMBER")=528A8
ARRAY("TYPE")=VAMC
ARRAY("VA NAME")=VA HEALTHCARE NETWORK UPSTATE NEW YORK SYSTEM VISN 2 - ALBANY
DIVISION
```

## $$ID^XUAF4(): Institution Identifier

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This extrinsic function returns the Identifier (ID) of an INSTITUTION file (#4) entry for a given Coding System and Internal Entry Number (IEN).

**Format** `$$ID^XUAF4(cdsys,ien)`

**Input Parameters**

cdsys: (required) CDSYS is an existing CODING SYSTEM field (#.01) in the INDENTIFIER field (#9999) multiple of the INSTITUTION file (#4). To see existing coding systems in the file:

```
>D CDSYS^XUAF4(.Y)
```

ien: (required) Internal Entry Number (IEN) of the institution in question.

**Output**

returns: Returns the INSTITUITION file (#4) Identifier (ID) associated with the given Coding System and IEN.

**Example**

## $$IDX^XUAF4(): Institution IEN (Using Coding System & ID)

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This extrinsic function returns the Internal Entry Number (IEN) of an INSTITUTION file (#4) entry for a given Coding System and Identifier (ID) pair.

**Format** `$$IDX^XUAF4(cdsys,id)`

**Input Parameters**

cdsys: (required) CDSYS is an existing CODING SYSTEM field (#.01) in the INDENTIFIER field (#9999) multiple of the INSTITUTION file (#4). To see existing coding systems in the file:

```
>D CDSYS^XUAF4(.Y)
```

id: (required) ID is the ID filed (#.02) in the INDENTIFIER field (#9999) multiple of the INSTITUTION file (#4) that corresponds to the Coding System input (i.e., cdsys) as the first parameter.

**Output** returns: Returns the INSTITUTION file (#4) Internal Entry Number (IEN) associated with the given Coding System and Identifier (ID).

**Example**

## $$IEN^XUAF4(): IEN for Station Number

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This extrinsic function returns the Internal Entry Number (IEN) of the entry for a given STATION NUMBER field (#99) in the INSTITUTION file (#4).

**Format** `$$IEN^XUAF4(sta)`

**Input Parameters** sta: (required) Station Number.

**Output** returns: Returns:

IEN—Internal Entry Number.

NULL—Error.

**Example**

```
>S X=$$IEN^XUAF4("528A5")

>W X
532
```

## $$LEGACY^XUAF4(): Institution Realigned/Legacy (True/False)

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This extrinsic function, given the STATION NUMBER field (#99) in the INSTITUTION file (#4), returns the Boolean value for the question—has this station number been realigned? Is it a legacy Station Number?

**Format** `$$LEGACY^XUAF4(sta)`

**Input Parameters**

sta: (required) The STATION NUMBER field (#99) value in the INSTITUTION file (#4) for the Station Number in question.

**Output**

returns: Returns a Boolean value:

True (non-zero)—Station Number has been realigned; it is a legacy Station Number.

False (zero)—Station Number has *not* been realigned; it is *not* a legacy Station Number.

**Example**

## $$LKUP^XUAF4(): Institution Lookup

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This extrinsic function returns the IEN or zero when doing a lookup on the INSTITUTION file (#4).

**Format** `$$LKUP^XUAF4(inst)`

**Input Parameters**

inst: (required) Institution lookup value, any of the following:

- Internal Entry Number (IEN), will have the ` in front of it.
- Station Number
- Station Name

**Output**

returns: Returns:

IEN—Internal Entry Number.

Zero (0).

**Example**

## LOOKUP^XUAF4(): Look Up Institution Identifier

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This API lookup utility allows a user to select an Institution by Coding System and ID. It prompts a user for a Coding System and then prompts for an Identifier—it's an IX^DIC API call on a New Style cross-reference of the ID field (#.02) of the IDENTIFIER field (#9999) multiple of the INSTITUTION file (#4).

**Format** `LOOKUP^XUAF4()`

**Input Parameters**

**REF:** For input information, please refer to the IX^DIC documentation in the *VA FileMan Programmer Manual*.

**Output**

**REF:** For output information, please refer to the IX^DIC documentation in the *VA FileMan Programmer Manual*.

**Example**

```
Select INSTITUTION CODING SYSTEM: DMIS
                              ID: 0037
DMIS  0037  WALTER REED        DC  USAH      688CN
```

## $$MADD^XUAF4(): Institution Mailing Address

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This extrinsic function returns the mailing address information for an institution in a caret-delimited string (i.e., streetaddr^city^state^zip) for a given Internal Entry Number (IEN) in the INSTITUTION file (#4).

**Format** `$$MADD^XUAF4(ien)`

**Input Parameters**

| | |
|---|---|
| ien: | (required) Internal Entry Number (IEN) of the institution in question. |

**Output**

| | |
|---|---|
| returns: | Returns the institution mailing address in a caret-delimited string: `streetaddr^city^state^zip` |

**Example**

## $$NAME^XUAF4(): Institution Official Name

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This extrinsic function returns the OFFICIAL NAME field (#100) value in the INSTITUTION file (#4) for an institution given its Internal Entry Number (IEN). However, If Field #100 is null, the NAME field (#.01) in the INSTITUTION file (#4) is returned.

**Format** `$$NAME^XUAF4(ien)`

**Input Parameters** ien: (required) Internal Entry Number (IEN) of the institution in question.

**Output** returns: Returns either of the following:

- OFFICIAL NAME field (#100) value in the INSTITUTION file (#4)—If Field #100 is *not* null.
- NAME field (#.01) value in the INSTITUTION file (#4)—If Field #100 is null.

**Example**

## $$NNT^XUAF4(): Institution Station Number, Name, and Type

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This extrinsic function returns the station information for an institution in a caret-delimited string (i.e., station_number^station_name^station_type) for a given Internal Entry Number (IEN) in the INSTITUTION file (#4).

**Format** `$$NNT^XUAF4(ien)`

**Input Parameters** ien: (required) Internal Entry Number (IEN) of the institution in question.

**Output** returns: Returns the institution station information in a caret-delimited string:

```
station_number^station_name^station_type
```

**Example**

## $$NS^XUAF4(): Institution Name and Station Number

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This extrinsic function returns the institution information in a caret-delimited string (i.e., institution_name^station_number) for a given Internal Entry Number (IEN) in the INSTITUTION file (#4).

**Format** `$$NS^XUAF4(ien)`

**Input Parameters** ien: (required) Internal Entry Number (IEN) of the institution in question.

**Output** returns: Returns the institution information in a caret-delimited string:

```
institution_name^station_number
```

**Example**

## $$O99^XUAF4(): IEN of Merged Station Number

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This extrinsic function returns the Internal Entry Number (IEN) of the valid STATION NUMBER in the INSTITUTION file (#4), if this entry was merged during the INSTITUTION file (#4) cleanup process (e.g., due to a duplicate STATION NUMBER field [#99]). This function may be used by application developers to re-point their INSTITUTION file (#4) references to a valid entry complete with Station Number.

**Format** `$$O99^XUAF4(ien)`

**Input Parameters** ien: (required) Internal Entry Number (IEN) of the institution in question.

**Output** returns: Returns the Internal Entry Number (IEN) of the INSTITUTION file (#4) entry with a valid STATION NUMBER filed (#99)—the Station Number deleted from the input IEN during the cleanup process (i.e., Kernel Patch XU*8.0*206).

**Example**

```
>S NEWIEN=$$O99^XUAF4(6538)

>W NEWIEN
6164
>W ^DIC(4,6164,99)
519HB^^^
```

## $$PADD^XUAF4(): Institution Physical Address

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This extrinsic function returns the physical address information for an institution in a caret-delimited string (streetaddr^city^state^zip) for a given Internal Entry Number (IEN) in the INSTITUTION file (#4).

**Format** `$$PADD^XUAF4(ien)`

**Input Parameters**

| | |
|---|---|
| ien: | (required) Internal Entry Number (IEN) of the institution in question. |

**Output**

| | |
|---|---|
| returns: | Returns the institution physical address in a caret-delimited string: `streetaddr^city^state^zip` |

**Example**

## PARENT^XUAF4(): Parent Institution Lookup

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This API returns a list of all institutions that make up a given Veterans Integrated Service Network (VISN), parent institution entered in the "lookup" input parameter.

**Format**

```
PARENT^XUAF4(array,lookup[,type])
```

**Input Parameters**

array: (required) $NAME reference to store the list of the parent (VISN) institution for the "lookup" input parameter institution.

lookup: (required) Parent (VISN) institution lookup value, any of the following:

- Internal Entry Number (IEN), will have the ` in front of it.
- Station Number
- Station Name

type: (optional) Type of institution from the INSTITUTION ASSOCIATION TYPES file (#4.05, default is VISN).

**Output**

returns: Returns the array populated with the list of parent (VISN) institutions.

```
Variable array
("p",pien)=station_name^station_number
```

**NOTE:** With the business rule that institutions can only have one parent per type, if you specify the input parameter type, you will get an array that will only have one pien in it. If the type parameter is left blank, it finds all parents for the institution and lists then in the array.

**Example**

## $$PRNT^XUAF4(): Institution Parent Facility

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This extrinsic function returns the parent facility institution information in a caret-delimited string (ien^station_number^name) for a given child facility STATION NUMBER filed (#99) in the INSTITUTION file (#4).

**Format**

```
$$PRNT^XUAF4(sta)
```

**Input Parameters**

sta: (required) The STATION NUMBER field (#99) value in the INSTITUTION file (#4) for the child facility whose parent facility information is being requested.

**Output**

returns: Returns the parent facility institution information in a caret-delimited string:

```
ien^station_number^name
```

**Example**

## $$RF^XUAF4(): Realigned From Institution Information

**Reference Type** Supported

**Category** Institution File

**IA #**

**Description** This extrinsic function returns the information that is pointed to in the REALIGNED FROM field (#.06) in the HISTORY field (#999) multiple in a caret-delimited string (ien^station_number^effective_date) for a given Internal Entry Number (IEN) in the INSTITUTION file (#4).

**Format** `$$RF^XUAF4(ien)`

**Input Parameters**

ien: (required) Internal Entry Number (IEN) of the institution in question.

**Output**

returns: Returns the realigned from institution information in a caret-delimited string:

```
ien^station_number^effective_date
```

**Example**

```
>S IEN=$$RF^XUAF4(7020)

>W IEN
500^500^3000701
```

## $$RT^XUAF4(): Realigned To Institution Information

**Reference Type** Supported

**Category** Institution File

**IA #**

**Description** This extrinsic function returns the information that is pointed to in the REALIGNED TO field (#.05) in the HISTORY field (#999) multiple in a caret-delimited string (ien^station_number^effective_date) for a given Internal Entry Number (IEN) in the INSTITUTION file (#4).

**Format** `$$RT^XUAF4(ien)`

**Input Parameters**

ien: (required) Internal Entry Number (IEN) of the institution in question.

**Output**

returns: Returns the realigned to institution information in a caret-delimited string:

```
ien^station_number^effective_date
```

**Example**

```
>S IEN=$$RT^XUAF4(500)

>W IEN
7020^528A8^3000701
```

## SIBLING^XUAF4(): Sibling Institution Lookup

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This API returns a list of all institutions that make up a given Veterans Integrated Service Network (VISN), parent institution entered in the "child" input parameter.

**Format** `SIBLING^XUAF4(array,child[,type])`

**Input Parameters**

array: (required) $NAME reference to store the list of all institutions of a parent (VISN) institution for the "child" input parameter institution.

child: (required) Child institution lookup value, any of the following:

- Internal Entry Number (IEN), will have the ` in front of it.
- Station Number
- Station Name

type: (optional) Type of institution from the INSTITUTION ASSOCIATION TYPES file (#4.05, default is VISN).

**Output**

returns: Returns the array populated with the list of all institutions of the parent (VISN) institution.

```
Variable array
("p",pien, "c",cien)=station_name^station_number
```

**NOTE:** With the business rule that institutions can only have one parent per type, if you specify the input parameter type, you will get an array that will only have one pien in it. If the type parameter is left blank, it finds all parents for the institution and lists then in the array. Also, the input site (i.e., "child" input parameter) is included in the list.

**Example**

## $$STA^XUAF4(): Station Number for IEN

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This extrinsic function returns the STATION NUMBER (#99) for the entry of a given Internal Entry Number (IEN) in the INSTITUTION file (#4).

**Format** `$$STA^XUAF4(ien)`

**Input Parameters** ien: (required) Internal Entry Number (IEN) of the institution in question.

**Output** returns: Returns the Station Number.

**Example**

```
>S STA=$$STA^XUAF4(7020)

>W STA
528A8
```

## $$TF^XUAF4(): Treating Facility (True/False)

**Reference Type** Supported

**Category** Institution File

**IA #** 2171

**Description** This extrinsic function, given the Internal Entry Number (IEN) in the INSTITUTION file (#4), returns the Boolean value for the question—is this a medical treating facility?

**Format** `$$TF^XUAF4(ien)`

**Input Parameters** ien: (required) Internal Entry Number (IEN) of the institution in question.

**Output** returns: Returns a Boolean value:

True (non-zero)—Treating facility.

False (zero)—Not a Treating facility.

**Example**

```
>S TF=$$TF^XUAF4(7020)

>W TF
1
```

## $$WHAT^XUAF4(): Institution Single Field Information

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Institution File | |
| **IA #** | 2171 | |
| **Description** | This extrinsic function returns the data from a single field given the Internal Entry Number (IEN) and the specific field requested in the INSTITUTION file (#4). | |
| **Format** | `$$WHAT^XUAF4(ien,field)` | |
| **Input Parameters** | ien: | (required) Internal Entry Number (IEN) of the institution in question (pointer value to the INSTITUTION file (#4). |
| | field: | (required) field number of the field in question. |
| **Output** | returns: | Returns the value in the specified field. |

**Example**

## $$IEN^XUMF(): Institution IEN (Using IFN, Coding System, & ID)

**Reference Type** Supported

**Category** Institution File

**IA #** 3795

**Description** This extrinsic function returns the Internal Entry Number (IEN) for a given Internal File Number (IFN), Coding System, and Identifier (ID).

**Format** `$$IEN^XUMF(ifn,cdsys,id)`

**Input Parameters**

| | |
|---|---|
| ifn: | (required) Internal File Number (IFN). |
| cdsys: | (required) Coding System. |
| id: | (required) Identifier. |

**Output**

| | |
|---|---|
| returns: | Returns the Internal Entry Number (IEN) of the institution requested. |

**Example**

## MAIN^XUMFI(): HL7 Master File Message Builder

**Reference Type** Controlled Subscription

**Category** Institution File

**IA #** 2171

**Description** This API implements an HL7 Master File Message Builder Interface that dynamically maps a VA FileMan field to an HL7 Master File sequence within a segment. The interface implements functionality to build Master File Notification (MFN), Master File Query (MFQ), and Master File Response (MFR) segments. The interface calls applicable VISTA HL7 GENERATE and GENACK interfaces to send/reply/broadcast an appropriate HL7 Master File message.

**Format** `MAIN^XUMFI(ifn,ien,type,param,error)`

**Input Parameters**

**REF:** For a description of the Input parameters for this API, please refer to the "MAIN^XUMFP(): Master File Parameters" API.

**Output Parameters & Output**

**REF:** For a description of the Output Parameters and Output for this API, please refer to the "MAIN^XUMFP(): Master File Parameters" API.

**Details**

This interface should be called after the Master File Parameter API. The Master File Parameter API sets up the required parameters in the PARAM array.

The Institution File Redesign (IFR) patch (i.e., XU*8.0*206) implements several Application Programmer Interfaces (APIs). After the IFR patch has been installed and the Cleanup performed, the STATION NUMBER field (#99) will be a unique key to the INSTITUTION file (#4).

### Example

```
>D MAIN^XUMFI(4,18723,1,.PARAM,.ERROR)
```

From the HL7 MESSAGE TEXT file (#772), you would see the following:

```
DATE/TIME ENTERED: JAN 12, 2001@09:17:29
 SERVER APPLICATION: XUMF MFN          TRANSMISSION TYPE: OUTGOING
 MESSAGE ID: 0259                      PARENT MESSAGE: JAN 12, 2001@09:17:29
 PRIORITY: DEFERRED                    RELATED EVENT PROTOCOL: XUMF MFN
 MESSAGE TYPE: SINGLE MESSAGE
MESSAGE TEXT:
MFI^Z04^MFS^REP^20010112091729^20010112091729^NE
MFE^MUP^^19001011^631GD~STATION NUMBER~D
ZIN^GREENFIELD^631GD^National^CBOC~FACILITY TYPE~VA^^^MASSACHUSETTS^^^^^^
 STATUS: SUCCESSFULLY COMPLETED
 DATE/TIME PROCESSED: JAN 12, 2001@09:17:29
 NO. OF CHARACTERS IN MESSAGE: 161     NO. OF EVENTS IN MESSAGE: 1
```

## MAIN^XUMFP(): Master File Parameters

**Reference Type** Controlled Subscription

**Category** Institution File

**IA #** 2171

**Description** This API sets up required parameters used by the HL7 Master File Message Builder Interface and the HL7 Master File message handler. The interface defines required parameters and serves as a common interface for parameter initialization. This interface is the enabling component of the Master File Server (MFS) mechanism allowing VA FileMan Master Files to be maintained by the server, including files with multiple fields and extended references.

The programmer may set any PARAM parameter before or after the interface call and override the default value.

**Format** `MAIN^XUMFP(ifn,ien,type,param,error)`

**Input Parameters**

ifn: (required) Internal File Number (IFN).

ien: (required) Internal Entry Number (IEN).

Single entry (pass by value).

Example:

- IEN=1

Multiple entries (pass by reference).

Example:

- IEN(1)=""
- IEN(2)=""

ALL national entries (pass by value).

Example:

- IEN="ALL"

type: (required) Message TYPE. Possible values are:

0—MFN: Unsolicited update.

1—MFQ: Query particular record and file.

3—MFQ: Query particular record in array.

5—MFQ: Query group records file.

7—MFQ: Query group records array.

11—MFR: Query response particular record file.

13—MFR: Query response particular record array.

15—MFR: Query response group records file.

17—MFR: Query response group records array.

**Output Parameters**

| | |
|---|---|
| param("PROTOCOL") | IEN Protocol file (#101). |
| param("BROADCAST") | Broadcast message to all VistA sites. |
| param("LLNK") | Logical link in HLL("LINKS",n) format. |

**Output**

| | |
|---|---|
| error | `1^Error message text` |

**Details**

| QRD -- Query definition | HL7 Sequence | HL7 Data Type |
|---|---|---|
| param("QDT") | Query Date/Time | TS |
| param("QFC") | Query Format Code | ID |
| param("QP") | Query Priority | ID |
| param("QID") | Query ID | ST |
| param("DRT") | Deferred Response Type | ID |
| param("DRDT") | Deferred Response Date/Time | TS |
| param("QLR") | Quantity Limited Request | CQ |
| param("WHO") | Who Subject Filter | XCN |
| param("WHAT") | What Subject Filter | CE |
| param("WDDC") | What Department Data Code | CE |
| param("WDCVQ") | What Data Code Value Qual. | CM |
| param("QRL") | Query Results Level | ID |

XCN data type of QRD WHO parameter

| | | |
|---|---|---|
| 1ST component | | One of the following: |
| NAME | | Value of NAME field (#.01) for Internal Entry Number (IEN). |
| ALL | | String represents all national entries. |
| IEN ARRAY | | String represents entries passed in IEN array. |
| 9th component | D | Source table (VA FileMan cross-reference). |
| 10th component | 045A4 | Assigning authority. |

CE data type of QRD WHAT parameter

| | | |
|---|---|---|
| 1ST component | 4 | Identifier |
| 2nd component | IFN | Text |
| 3rd component | VA FM | Name of Coding System |

MFI—Master File Identification

| | |
|---|---|
| PARAM("MFI") | Master File Identifier |
| PARAM("MFAI") | Master File Application Identifier |
| PARAM("FLEC") | File-Level Event Code |
| PARAM("ENDT") | Entered Data/Time |
| PARAM("MFIEDT") | Effective Date/Time |
| PARAM("RLC") | Response Level Code |

MFE—Master File Entry

| | |
|---|---|
| PARAM("RLEC") | Record-Level Event Code |
| PARAM("MFNCID") | MFN Control ID |
| PARAM("MFEEDT") | Effective Date/Time |
| PARAM("PKV") | Primary Key Value |

[Z...] segment(s) parameters

| | |
|---|---|
| PARAM("SEG",SEG)="" | HL7 segment name |
| PARAM("SEG",SEG,"SEQ",SEQ,FLD#) | segment sequence # and field |

[Z...] segment(s) parameters

**NOTE:** If any special processing is required, in addition to the external value passed by VA FileMan, set the FLD# node equal to a formatting function "n^$$TAG^RTN(X)".

- "**n**" being the component sequence number.
- "**X**" representing the external value from VA FileMan.

$P(segment_sequence,HLCS,n)=FM_external_value.

Files involving sub-records and/or extended reference

| | |
|---|---|
| PARAM("SEG",SEG,"SEQ",SEQ,"FILE") | See VA FileMan documentation. |
| PARAM("SEG",SEG,"SEQ",SEQ,"IENS") | $$GET1^DIQ() for value. |
| PARAM("SEG",SEG,"SEQ",SEQ,"FIELD") | of FILE, IENS, & FIELD. |
| PARAM("SEG",SEG,"SEQ",SEQ,"KEY") | .01 value. |
| PARAM("SEG",SEG,"SEQ",SEQ,"FORMAT") | format non ST data types. |

**NOTE:** Query group records store PARAM in the ^TMP global with the following root: ^TMP("XUMF MFS",$J,"PARAM",IEN).

Example: MFE PKV node is ^TMP("XUMF MFS",$J,"PARAM",IEN,"PKV")

**Example**

The following example is a query (MFQ) for a group records array:

```
>D MAIN^XUMFP(4,"ALL",7,.PARAM,.ERROR)
```

Since query group records store PARAM in the ^TMP global, display the ^TMP global to see the PARAM values:

```
>D ^%G

Global ^TMP("XUMF MFS",$J
        TMP("XUMF MFS",$J
^TMP("XUMF MFS",539017563,"PARAM","DRDT") =
^TMP("XUMF MFS",539017563,"PARAM","DRT") =
^TMP("XUMF MFS",539017563,"PARAM","ENDT") =
^TMP("XUMF MFS",539017563,"PARAM","FLEC") = UPD
^TMP("XUMF MFS",539017563,"PARAM","MFAI") =
^TMP("XUMF MFS",539017563,"PARAM","MFEEDT") = 20010212110654
^TMP("XUMF MFS",539017563,"PARAM","MFI") = Z04
^TMP("XUMF MFS",539017563,"PARAM","MFIEDT") =
^TMP("XUMF MFS",539017563,"PARAM","MFNCID") =
^TMP("XUMF MFS",539017563,"PARAM","POST") = POST^XUMFP4C
^TMP("XUMF MFS",539017563,"PARAM","PRE") = PRE^XUMFP4C
^TMP("XUMF MFS",539017563,"PARAM","PROTOCOL") = 2233
^TMP("XUMF MFS",539017563,"PARAM","QDT") = 20010212110654
^TMP("XUMF MFS",539017563,"PARAM","QFC") = R
```

```
^TMP("XUMF MFS",539017563,"PARAM","QID") = Z04 ARRAY
^TMP("XUMF MFS",539017563,"PARAM","QLR") = RD~999
^TMP("XUMF MFS",539017563,"PARAM","QP") = I
^TMP("XUMF MFS",539017563,"PARAM","QRL") =
^TMP("XUMF MFS",539017563,"PARAM","RLC") = NE
^TMP("XUMF MFS",539017563,"PARAM","RLEC") = MUP
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",1,.01) = ST
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",2,99) = ST
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",3,11) = ID
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",4,13) = CE^~FACILITY TYPE~VA
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",5,100) = ST
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",6,101) = ST
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",7,.02) = ST
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",8,"DTYP") = CE^~VISN~VA
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",8,"FIELD") = 1
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",8,"FILE") = 4.014
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",8,"IENS") = 1,?+1,
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",9,"DTYP") = ST
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",9,"FIELD") = 1:99
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",9,"FILE") = 4.014
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",9,"IENS") = 2,?+1,
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",10,"DTYP") = DT
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",10,"FIELD") = .01
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",10,"FILE") = 4.999
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",11,"DTYP") = ST
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",11,"FIELD") = .06:99
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",11,"FILE") = 4.999
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",12,"DTYP") = DT
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",12,"FIELD") = .01
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",12,"FILE") = 4.999
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",13,"DTYP") = ST
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",13,"FIELD") = .05:99
^TMP("XUMF MFS",539017563,"PARAM","SEG","ZIN","SEQ",13,"FILE") = 4.999
^TMP("XUMF MFS",539017563,"PARAM","SEGMENT") = ZIN
^TMP("XUMF MFS",539017563,"PARAM","WDCVQ") =
^TMP("XUMF MFS",539017563,"PARAM","WDDC") = INFRASTRUCTURE~INFORMATION INFRASTRUCTURE
~VA TS
^TMP("XUMF MFS",539017563,"PARAM","WHAT") = 4~IFN~VA FM
^TMP("XUMF MFS",539017563,"PARAM","WHO") = ALL~~~~~~~~D~045A4
```

# 13. KIDS: Programmer Tools

## KIDS Build-related Options

To get to the KIDS: Kernel Installation & Distribution System menu [XPD MAIN] (locked with the XUPROG security key) choose the Programmer Options menu option [XUPROG] on the Kernel Systems Manager Menu [EVE], as shown below:

```
Select Systems Manager Menu Option: Programmer Options

   KIDS   Kernel Installation & Distribution System ...                  [XPD MAIN]
             **> Locked with XUPROG
   NTEG   Build an 'NTEG' routine for a package
   PG     Programmer mode
          ALS MENU TEXT SAMPLE ...
          Calculate and Show Checksum Values
          Delete Unreferenced Options
          Error Processing ...
          Global Block Count
          List Global
          Map Pointer Relations
          Number base changer
          Routine Tools ...
          Test an option not in your menu
          Verifier Tools Menu ...

Select Programmer Options Option: kids <Enter>  Kernel Installation & Distribution
    System

          Edits and Distribution ...                        [XPD DISTRIBUTION MENU]
          Utilities ...                                               [XPD UTILITY]
          Installation ...                                  [XPD INSTALLATION MENU]
             **> Locked with XUPROGMODE
Select Kernel Installation & Distribution System Option: Edits and Distribution


          Create a Build Using Namespace
          Copy Build to Build
          Edit a Build
          Transport a Distribution
          Old Checksum Update from Build
          Old Checksum Edit
          Routine Summary List
          Version Number Update

Select Edits and Distribution Option:
```

**Figure 13-1. KIDS Edits and Distribution menu options**

## Creating Builds

KIDS introduces significant revisions to the process of exporting software applications over the previous export mechanism, DIFROM.

**REF:** For an introduction to KIDS and a description of the KIDS installation and utility options, please refer to the "KIDS: System Management—Installations" and "KIDS: System Management—Utilities" chapters in the *Kernel Systems Manual*.

A functional listing of the KIDS options supporting software application (package) export is shown below:

| Task Category | Option Name | Option Text |
|---|---|---|
| Create Build Entry | XPD BUILD NAMESPACE | Create a Build using Namespace |
| | XPD COPY BUILD | Copy Build to Build |
| | XPD EDIT BUILD | Edit a Build |
| Create a Distribution | XPD TRANSPORT PACKAGE | Transport a Distribution |

**Table 13-1. KIDS: Options supporting software application builds and exports**

This chapter covers each of these tasks, describing how to accomplish the tasks using KIDS options.

## Build Entries

KIDS stores the definition of a software application in the BUILD file (#9.6). Individual entries in the BUILD file (#9.6) are called build entries, or builds for short. To export a software application, you must first define a build entry for it in the BUILD file (#9.6).

Unlike DIFROM, where you re-used the same PACKAGE file (#9.4) entry each time you exported a new version of a software application, with KIDS you create a new BUILD file (#9.6) entry each time you export a software application version. One advantage of having one BUILD entry per software application version is that you have a complete history of each version of your software application, which makes it easier to compare previous versions of a software application with the current version.

After you create the build name, KIDS give you the option to choose the type of build you are creating. There are three types from which to choose:

- Single
- Multi-Package
- Global

```
Select Edits and Distribution Option: Edit a Build
Select BUILD NAME: TEST 5.0
  Are you adding 'TEST 5.0' as a new BUILD (the 104TH)? Y <Enter> (Yes)
   BUILD PACKAGE FILE LINK: RET
   BUILD TYPE: SINGLE PACKAGE// ?
     Choose from:
       0        SINGLE PACKAGE
       1        MULTI-PACKAGE
       2        GLOBAL PACKAGE
   BUILD TYPE: SINGLE PACKAGE// GLOBAL <Enter> GLOBAL PACKAGE
```

**Figure 13-2. KIDS: Choosing a build type sample**

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.

| | | |
|---|---|---|
| Bulletin | HL Logical Link | Print Template |
| Dialog | HL Lower Level Protocol | Protocol |
| Form | Input Template | Remote Procedure |
| Function | List Template | Routine |
| Help Frame | Mail Group | Security Key |
| HL7 Application Parameter | Option | Sort Template |

**Table 13-2. KIDS: Kernel V. 8.0 component types (listed alphabetically)**

```
Select Edits and Distribution Option: Create a Build Using Namespace

Select BUILD NAME: ZXGY 1.0
  Are you adding 'ZXGY 1.0' as a new BUILD (the 14th)? YES
   BUILD PACKAGE FILE LINK: <Enter>


Namespace: ZXG
Namespace: -ZXGI
Namespace: <Enter>

NAMESPACE  INCLUDE                EXCLUDE
           -------                -------
           ZXG                    ZXGI

OK to continue? YES// <Enter>
...SORRY, LET ME THINK ABOUT THAT A MOMENT...

    ...Done.
```

**Figure 13-3. KIDS: Populating a build entry by namespace**

```
Select Edits and Distribution Option: Copy Build to Build

Copy FROM what Package: ZXG TEST 1.0
Copy TO what Package: ZXG TEST 1.1
  ARE YOU ADDING 'ZXG TEST 1.1' AS A NEW BUILD (THE 5TH)? Y <Enter> (YES)
   BUILD PACKAGE FILE LINK: <Enter>

OK to continue? YES// <Enter>
...HMMM, LET ME PUT YOU ON 'HOLD' FOR A SECOND...    ...Done.
```

**Figure 13-4. KIDS: Copying a build entry**

## Copy Build to Build

You can create a new build entry based on a previous entry using the Copy Build to Build option. With KIDS, you must create a new build entry for each new version of a software application. 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 topics 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 Appendix A—KIDS Build Checklists. They are designed in conjunction with the Edit a Build option to help you plan your build entries.

| Screen | Build Section | Build Sub-Section |
|---|---|---|
| Screen 1 | Build Name<br>Date Distributed<br>Description<br>Environment Check Routine<br>Pre-Install Routine<br>Post-Install Routine<br>Pre-Transportation Routine | |
| Screen 2 | Files and Data | Partial DD Definition<br>Send Data Definition |
| Screen 3 | Build Components | Print Template<br>Sort Template<br>Input Template<br>Form<br>Function<br>Dialog<br>Bulletin<br>Mail Group<br>Help Frame<br>Routine<br>Option<br>Security Key<br>Protocol<br>List Template<br>HL7 Application Parameter<br>HL Lower Level Protocol<br>HL Logical Link<br>Remote Procedure |
| Screen 4 | Install Questions<br>Required Builds<br>Package File Link<br>Package Tracking | |

**Table 13-3. KIDS: Functional layout, Edit a Build**

## 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 software application settings:

- Name
- Date Distributed
- Description
- Environment Check Routine
- Pre-Install Routine
- Post-Install Routine
- Pre-Transportation Routine

### Build Name

The name of a build entry is where KIDS stores both the software application's name and version number. The build name must be a software application name, followed by a space and then followed by a version number. This means that every version of a software application requires a separate entry in the BUILD file (#9.6). One way that this is an advantage is that you have a record of the contents of every version of a software application that you export.

```
                            Edit a Build                        PAGE 1 OF 4
Name: ZXG TEST 1.0                        TYPE: SINGLE PACKAGE
---------------------------------------------------------------------------


                       Name: ZXG DEMO 1.0

          Date Distributed: AUG 29,2004

               Description:

 Environment Check Routine:

       Pre-Install Routine: ZXGPRE

      Post-Install Routine: ZXGPOS

Pre-Transportation Routine:
___________________________________________________________________________



COMMAND:                            Press <PF1>H for help      Insert
```

**Figure 13-5. KIDS: Screen 1 of Edit a Build sample**

## Edit a Build: Files

The second screen of Edit a Build is where you enter all the files to export with your software application. 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.

```
                            Edit a Build                        PAGE 2 OF 4
Name: ZXG DEMO 1.0                        Type: SINGLE PACKAGE
----------------------------------------------------------------------------
                           File List (Name or Number)

     NEW PERSON


COMMAND:                            Press <PF1>H for help     Insert
```

**Figure 13-6. KIDS: Screen 2 of Edit a Build: Selecting files**

```
                            Edit a Build                        PAGE 2 OF 4
Name: ZXG DEMO 1.0                        TYPE: SINGLE PACKAGE
----------------------------------------------------------------------------
                            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
```

**Figure 13-7. KIDS: Data dictionary and data settings**

### 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 subdata dictionary number of a Multiple (also known as a subfile). You can export any Multiple, no matter how deep (every Multiple'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 at the selected data dictionary level and all descendent Multiples.
- **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:** 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.

```
                              Edit a Build                    PAGE 2 OF 4
Name: ZXG DEMO 1.0                        TYPE: SINGLE PACKAGE
-----------------------------------------------------------------------
                              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
```

**Figure 13-8. KIDS: Data dictionary settings screen**

```
                              Edit a Build                    PAGE 2 OF 4
Name: ZXG DEMO 1.0                        TYPE: SINGLE PACKAGE
-----------------------------------------------------------------------
                              File List (Name or Number)
                            DD Export Options
                            Data Dictionary Number
   NEW PERSON  (File-top level)
   DMMS UNITS  (sub-file)
   ALIAS  (sub-file)
   DEFINED FORMATS FOR LM  (sub-file)

COMMAND:                              Press <PF1>H for help      Insert
```

**Figure 13-9. KIDS: Partial DD—Choosing DD levels (top level and Multiple) 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 (IEN) 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.

```
                            Edit a Build                        PAGE 2 OF 4
Name: ZXG DEMO 1.0                        TYPE: SINGLE PACKAGE
-------------------------------------------------------------------------
                            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
```

**Figure 13-10. KIDS: Settings for sending data**

### 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:

| Data Installation Action | Description |
|---|---|
| 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.<br><br>KIDS does not send out cross-references with the data. When you merge the data, however, KIDS re-indexes and creates new cross-references. Also, when you merge the data, KIDS does *not* delete the old cross-references for that data. |
| 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.<br><br>KIDS does not send out cross-references with the data. When you replace the data, however, KIDS re-indexes and creates new cross-references. Also, when you replace the data, KIDS deletes any old cross-references for that data.<br><br>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. |

**Table 13-4. KIDS: Data installation actions**

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 (#.7). 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 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 Multiples, nor fields that point to other pointer fields. KIDS can resolve pointers, however, for all other pointer fields in a given file or Multiple.

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, 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 the file to which the pointed-to entry was pointing, 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 software application 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 asterisk (*****) wildcard and the–exclusion sign.

With most component types, the permissible installation actions are:

- SEND TO SITE
- DELETE AT SITE

Some component types, however, have additional installation actions available; the special cases are discussed on the following pages.

**REF:** For a list of Kernel component types, please refer to Table 13-2 in this chapter.

```
                            Edit a Build                        PAGE 3 OF 4
Name: ZXG 1.0                             TYPE: SINGLE PACKAGE
----------------------------------------------------------------------------
                            Build Components

PRINT TEMPLATE              (0)
SORT TEMPLATE               (0)
INPUT TEMPLATE              (0)
FORM                        (0)
FUNCTION                    (0)
DIALOG                      (0)
BULLETIN                    (0)
MAIL GROUP                  (0)
HELP FRAME                  (0)
ROUTINE                     (0)
OPTION                      (0)
SECURITY KEY                (0)
PROTOCOL                    (0)
LIST TEMPLATE               (0)
HL7 APPLICATION PARAMETE    (0)
HL LOWER LEVEL PROTOCOL     (0)
HL LOGICAL LINK             (0)
REMOTE PROCEDURE            (0)


COMMAND:                              Press <PF1>H for help    Insert
```

**Figure 13-11. KIDS: Screen 3 of Edit a Build: Components**

**NOTE:** This is an expanded view of this screen in order to show you all of the currently available component types. You will have to scroll through the list in order to see all of the available types.

## 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:** Beginning with Kernel V. 8.0, you can no longer send out an option with an attached scheduling frequency. Scheduling of options was moved out of the OPTION file (#19) and into the OPTION SCHEDULING file (#19.2). 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.

| Option/Protocol 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. KIDS does *not* disable options and protocols that have an Action of USE AS LINK FOR MENU ITEMS. |
| 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.<br><br>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. |

**Table 13-5. KIDS: Option and protocol installation actions**

## Edit a Build: Routines

Routine selection is done based on pointers to entries in the ROUTINE file (#9.8), 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 (default)
- 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 software applications 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. KIDS will display various status messages while each CPU is updated. 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.

```
                          Edit a Build                      PAGE 3 OF 4
NAME: XU 99.0                           TYPE: SINGLE PACKAGE
----------------------------------------------------------------------
                          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
```

**Figure 13-12. KIDS: Choosing routines**

## Edit a Build: Dialog Entries (DIALOG File [#.84])

VA FileMan supports the capability for other software applications to store their dialog in the VA FileMan DIALOG file. Some advantages to using the DIALOG file (#.84) 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 (#.84), it can be re-used.
- Easily generating software application 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 software application maintains in the DIALOG file (#.84). Simply select which DIALOG entries you want to include in your software application, as you would for any other software application component, and choose an installation action for each item (the default is SEND TO SITE, the other permissible choice is DELETE AT SITE).

**REF:** For more information on using the DIALOG file (#.84), please refer to the *VA FileMan Programmer Manual*.

### 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).

```
                            Edit a Build                      PAGE 3 OF 4
Name: KERNEL 8.0T14                      TYPE: SINGLE PACKAGE
-----------------------------------------------------------------------
                            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
```

**Figure 13-13. KIDS: Selecting templates**

## 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 software application. 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.
- A PackMan message (to be sent over the network).
- The ^XTMP global on your local system.

If you choose to transport the distribution via a Host file enter HF after the "Transport through (HF)Host File or (PM)PackMan:" prompt and enter a Host file name after the "Enter a Host File" prompt. The option creates transport globals and puts them in the distribution (HFS file) that you specify.

```
Select Edits and Distribution Option: Transport 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 DEMO 1.0
Another Package Name: ZXG TEST 1.0
Another Package Name: <Enter>

ORDER   PACKAGE
  1    ZXG DEMO 1.0
  2    ZXG TEST 1.0


OK to continue? NO// YES
Transport through (HF)Host File or (PM)PackMan: HF <Enter> Host File

Enter a Host File: ZXG_EXPT.DAT
Header Comment: export of ZXG package


     ZXG DEMO 1.0...
     ZXG TEST 1.0...

Package Transported Successfully

Select Edits and Distribution Option:
```

**Figure 13-14. KIDS: Transport a Distribution option—Creating a distribution sample user dialogue**

If you don't enter a Host file name, 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 software application 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.

If you want the distribution sent via a PackMan message enter PM after the "Transport through (HF)Host File or (PM)PackMan:" prompt. You can only send *one* transport global per PackMan message, however.

```
Select Edits and Distribution Option: Transport 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: TEST 1.1
Another Package Name: <Enter>

ORDER   PACKAGE
  1    TEST 1.1


OK to continue? NO// YES
Transport through (HF)Host File or (PM)PackMan: PM <Enter> PackMan


     TEST 1.1...
No Package File Link
Subject: TEST
Please enter description of Packman Message


TEST


 Created by KRNUSER,FIVE at KERNEL.ISC-SF.VA.GOV  (KIDS) on MONDAY, 10/07/96 at
15:21
Do you wish to secure this message? No// ?

If you answer yes, this message will be secured to insure that
what you send is what is actually received.
Do you wish to secure this message? No// Y <Enter> (Yes)
Enter the scramble hint: THIS IS A HINT
Enter scramble password:
Securing the message, now.  This may take a while !!!

Send mail to: KRNUSER,FIVE     Last used MailMan: 04 Oct 96 15:28
  Select basket to send to: IN// <Enter>
And send to: <Enter>
```

**The password is *not* echoed back.**

**Figure 13-15. KIDS: Transport a Distribution option—Sending via network (PackMan message) sample user dialogue**

## 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):
```

**Figure 13-16. KIDS: Splitting a distribution across diskettes (1 of 2)**

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:
```

**Figure 13-17. KIDS: Splitting a distribution across diskettes (2 of 2)**

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 software applications 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 software applications is to be installed together, however, and if there are dependencies between the software applications, sending the software applications together in one distribution can give you more control over how the group of software applications is installed. If in some cases only software applications A and B should be installed, and in other situations only software applications A and C should be installed, and you can do the determination yourself (in each software application's environment check routine), sending the group of software applications in a single distribution lets you control which software applications in the distribution actually are installed.

When you are using PackMan messages to send your software application (rather than using a distribution), you are limited to sending only one transport global per PackMan message.

## Multi-Package Builds

Multi-Package builds contain a list of other builds and lists their installation order. A Multi-Package build will transport this list of builds (template or meta-build).

```
                         Edit a Build                          PAGE 1 OF 1
Name: TEST 3.0                        TYPE: MULTI-PACKAGE
---------------------------------------------------------------------------

                     Name: TEST 3.0

         Date Distributed: OCT 9,2004

              Description:


Install Order        Packages or Patches
  1                  TEST 1.0
  2                  TEST 1.1




COMMAND:                              Press <PF1>H for help      Insert
```

**Figure 13-18. KIDS: Multi-package builds sample**

## Exporting Globals with KIDS

KIDS in Kernel V. 8.0 supports the installation of global distributions (distributions that export globals). KIDS now supports the creation of global distributions by developers. Any number of globals can be included in a build. You are given the opportunity to run an environment check before installing the global and post-install routines after installing the globals. You also are given the choice of KILLing globals prior to installing new globals at a site. If you answer NO to this question, the global is merged with any previously installed global at the site.

**REF:** For more information on global distributions, please refer to the "KIDS: System Management—Installations" chapter in the *Kernel Systems Manual*.

```
                          Edit a Build                 PAGE 1 OF 1
Name: TEST 5.0                           TYPE: GLOBAL PACKAGE
-------------------------------------------------------------------

                   Name: TEST 5.0

       Date Distributed: OCT 9,2004

            Description:

 Environment Check Rtn.:         Post-Install Rtn.:

   Globals                       Kill Global Before Install?
   TMP(100)                      NO



___________________________________________________________________



COMMAND:                         Press <PF1>H for help      Insert
```

**Figure 13-19. KIDS: Exporting global distributions sample**

## 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.

# 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 subjects are covered:

- Environment Check Routine
- PRE-TRANSPORTATION ROUTINE
- Pre- and Post-Install Routines: Special Features
- Obtaining Package Name and Version Information
- Edit a Build - Screen 4
- How to Ask Installation Questions
- Using Checkpoints (Pre- and Post-Install Routines)
- Required Builds
- Package File Link
- Track Package Nationally
- Alpha/Beta Tracking

## 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 software application, based on conditions on the installing site's current system or environment.

You are not required to specify an environment check in order for your software application 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, because 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 software application, abort installing the software application, or abort installing all software applications (transport globals) in the distribution.

Although output during the pre-install and post-install should be done with the MES^XPDUTL and BMES^XPDUTL APIs, 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 software applications in the loaded distribution.

The KIDS key variable XPDENV indicates which phase (load or install) the environment check is running in.

**REF:** For more information on XPDENV, please refer to the "Key Variables during Environment Check" topic that follows.

## Key Variables during Environment Check

| Variable | Description |
|---|---|
| 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 software application's BUILD file (#9.6) entry, which is software application 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:<br>1—The environment check is being run by the KIDS Install Package(s) option.<br>0—The environment check is being run by the KIDS Load a Distribution option.<br><br>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 software application. |

**Table 13-6. KIDS: Key variables during the environment check**

## Package Version vs. Installing Version

KIDS provides several functions that you can use during the environment check to compare version numbers of the current software application at the site to the incoming transport global.

**REF:** For more on this subject, please refer to the "Obtaining Package Name and Version Information" topic in this chapter.

## 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.

## Verifying Patch Installation

During the environment check, you can tell KIDS to verify that a particular patch has been installed on a system prior to the installation of your software application.

For example, if your software application is dependent on a particular patch being installed, you can use the $$PATCH^XPDUTL function to have KIDS alert the user that a required patch is not installed on their system.

## 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:

| KIDS Desired Action (Based on Environment Check Conclusions) | How to Tell KIDS to Take Action |
|---|---|
| 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` |

| KIDS Desired Action (Based on Environment Check Conclusions) | How to Tell KIDS to Take Action |
|---|---|
| Abort all transport globals in distribution but leave them in ^XTMP. | `>S XPDABORT=2` |

**Table 13-7. KIDS: Actions based on environment check conclusions**

**NOTE:** It is recommended you use XPDQUIT when you have a distribution that contains multiple builds and you only want to selectively install a portion of it. Use the XPDABORT to abort the entire installation of a distribution.

## Controlling the Queueing of the Install Prompt

By default, KIDS allows the installer to run in the future. It does this by allowing the installer to enter "Q" at the device prompt. If the XPDNOQUE variable is set to 1, then the installer will see the following prompt and not be allowed to enter "Q":

```
Enter the Device you want to print the Install messages.
Enter a '^' to abort the install.

DEVICE: HOME//
```

**Figure 13-20. KIDS: Dialog when the XPDNOQUE variable is set to disable queuing**

## Controlling the Disable Options/Protocols Prompt

By default, KIDS asks the following question during KIDS installations:

```
Want to DISABLE Scheduled Options, Options, and Protocols? YES//
```

**Figure 13-21. KIDS: "DISABLE" default prompt during installations**

You can control the way this question is asked by defining the array XPDDIQ("XPZ1") during the environment check. The environment check runs once during the installation and prompts the user if it should run during the load. 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):

| Array Node | Description |
|---|---|
| XPDDIQ("XPZ1") | (optional) Set to zero (0) to force answer to NO or set to 1 to force answer to YES. When this node is set, the site is not asked the question. |
| XPDDIQ("XPZ1","A") | (optional) Replace the default question prompt with the value of this node. |
| XPDDIQ("XPZ1","B") | (optional) Set to new default answer in external form (YES or NO). |

**Table 13-8. KIDS: Installation—XPDDIQ array sample**

## Controlling the Move Routines to Other CPUs Prompt

By default, KIDS asks the following question during KIDS installations:

```
Want to MOVE routines to other CPUs? NO//
```

**Figure 13-22. KIDS :"MOVE routines" default prompt during installations**

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):

| Array Node | Description |
|---|---|
| XPDDIQ("XPZ2") | (optional) Set to zero (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") | (optional) Replace the default question prompt with the value of this node. |
| XPDDIQ("XPZ2","B") | (optional) Set to new default answer in external form (YES or NO). |

**Table 13-9. KIDS: Environment Check—XPDDIQ array sample**

```
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
```

**Figure 13-23. KIDS: Environment Check routine sample**

## PRE-TRANSPORTATION ROUTINE field (#900)

The PRE-TRANSPORTATION ROUTINE field (#900) in the BUILD file (#9.6) contains a [TAG^]ROUTINE that is run during the transportation process for the Build. This allows developers to populate the transport global using the XPDGREF variable.

Developers can now put information in the KIDS Transport Global, which can be used by the Pre- or Post-install routines. KIDS runs the [TAG^]ROUTINE in the field PRE-TRANSPORTATION ROUTINE during the transport process. This routine can use the XPDGREF variable to set nodes in the transport global. For example, enter the following at the programmer prompt:

```
>S @XPDGREF@("My Namespace",1)="Information I need during install"
```

During the install process, in the Pre- or Post-install routines, the developer can retrieve the data by using the same variable, XPDGREF. Since these nodes are part of the transport global, they are removed when the install is completed.

```
                            Edit a Build                PAGE 1 OF 4
Name: TEST 4.0                          TYPE: SINGLE PACKAGE
-----------------------------------------------------------------

                      Name: TEST 4.0

          Date Distributed: OCT 9,2004

               Description:

 Environment Check Routine:

       Pre-Install Routine:

      Post-Install Routine:

Pre-Transportation Routine: TAG^ROUTINE
_________________________________________________________________



COMMAND                            Press PF1H for help    Insert
```

**Figure 13-24. KIDS: PRE-TRANSPORTATION ROUTINE field sample**

## 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 topic describes how to use pre- and post-install routines with KIDS installations.

Pre- and post-routines are optional; you are not required to specify them in order for your software application 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.

Two functions can be called during the install process to disable or enable an option or protocol:

- $$OPTDE^XPDUTL(): Disable/Enable an Option
- $$PRODE^XPDUTL(): Disable/Enable a Protocol

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 XPDABORT variable to 1 and quitting. This is exactly as if the installing site pressed **<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 software application 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.

**REF:** For more information, please refer to the *VA FileMan Programmer Manual*.

## Key Variables during Pre- and Post-Install Routines

| Variable | Description |
|---|---|
| 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 software application's BUILD file (#9.6) entry, which is software application 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 software application. |
| ZTQUEUED | If the ZTQUEUED variable 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 queueing it. |

**Table 13-10. KIDS: Key variables during the pre- and post-install routines**

## 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 APIs, however. MailMan APIs can terminate prematurely if the DIFROM variable is present because the DIFROM variable has a special meaning within MailMan.

### Update the Status Bar During Pre- and Post-Install Routines

During the installation, 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 resets the status bar prior to the Pre- and Post-install routines.

**REF:** For more information on the status (progress) bar, please refer to the "Installation Progress" topic in the "KIDS Systems Management Installations" chapter in the *Kernel Systems Manual*.

You can provide a similar status bar for users in the Pre- and Post Install by doing the following:

1. SET XPDIDTOT=total number of items.
2. DO UPDATE^XPDID(current number of items). This moves the status bar.

For example, if you were converting 100 records and want to update the user every time you have completed 10% of the records you would enter the following at the programmer prompt:

```
>SET XPDIDTOT=100
>F%=1:1:100 D CONVERT I'(%#10) D UPDATE^XPDID(%)
```

If you wish to display a status bar at various intervals throughout your Pre or Post-install routines, you should reset the status bar. To reset the status bar enter the following at the programmer prompt:

```
>SET XPDIDTOT=0
>D UPDATE^XPDID(0)
```

## Edit a Build—Screen 4

Screen four of the EDIT A BUILD option is where you can set up the install questions, any required builds, PACKAGE file (#9.4) links, and tracking software application information for a build.

```
                          Edit a Build                 PAGE 4 OF 4
Name: TEST 1.0                       TYPE: SINGLE PACKAGE
-----------------------------------------------------------------
                         Install Questions




                         Required Builds




      Package File Link...: TEST

  Track Package Nationally: NO

_________________________________________________________________



 COMMAND:                       Press <PF1>H for help       Insert
```

**Figure 13-25. KIDS: Screen 4 of Edit a Build sample**

## How to Ask Installation Questions

You are not required to ask any installation questions in order for your software application 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 Input Value | Description |
|---|---|
| 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. |

**Table 13-11. KIDS: DIR input values for KIDS install questions**

**REF:** 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

Where "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 that 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 software application (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.

```
                            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
```

**Figure 13-26. KIDS: Pre-install question (setting up) sample**

```
Do you want to run the pre-install conversion? YES// ?

Answer YES to run the pre-install conversion, NO to skip it...

Do you want to run the pre-install conversion? YES//
```

**Figure 13-27. KIDS: Appearance of question during installation**

### Where Questions Are Asked During Installations

KIDS asks the pre- and post-install questions when a site initiates an installation of the software application. 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)

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 software application 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* callbacks
- Checkpoints *without* callbacks.

## Checkpoints *With* Callbacks

The preferred method of using checkpoints is to use checkpoints with callbacks. When you create a checkpoint with a callback, you give the checkpoint an API (the callback routine). That is all you have to do during your pre- or post-install routine, create a checkpoint with a callback. You do not have to execute the callback. At the completion of the pre- or post-install routine, KIDS manages the created checkpoints by calling, running, and completing the checkpoint and its callback routine.

The reason to let KIDS execute checkpoints (by creating checkpoints with callbacks) 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 callbacks of checkpoints that have not yet completed (and completes them).

In this scenario (checkpoints with callback routines), your pre-install and post-install routine should consist only of calls to $$NEWCP^XPDUTL to create checkpoints (with callbacks). Once you create all of the checkpoints for each discrete pre- or post-install task, the pre-install or post-install should quit.

Once the pre- or post-install routine finishes, KIDS executes each created checkpoint (that has a callback) in the order created. 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.

The KIDS checkpoint functions that apply when using checkpoints *with* callbacks are summarized below (listed in alphabetic order):

| Function | Description |
|---|---|
| $$CURCP^XPDUTL | Retrieve current checkpoint name (use during pre- or post-install routine). Useful when using the same tag^routine for multiple callbacks; this is how you determine which callback you're in. |
| $$NEWCP^XPDUTL | Create checkpoint (use during pre- or post-install routine only.) |
| $$PARCP^XPDUTL | Retrieve checkpoint parameter (use within callback routine.) |
| $$UPCP^XPDUTL | Update checkpoint parameter (use within callback routine.) |

**Table 13-12. KIDS: Functions using checkpoints *with* callbacks**

## Checkpoint Parameter Node

You can store how far you have progressed with a task you're performing in the callback by using a checkpoint parameter node. The $$UPCP^XPDUTL API updates the value of a checkpoint's parameter node; the $$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 callback 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.

The pre-install API in this example is PRE^ZZRON2; the post-install API is POST^ZZRON2.

```
ZZRON2      ;RON TEST 1.0 PRE AND POST INSTALL
     ;;1.0
     ;build checkpoints for PRE
PRE  N %
     S %=$$NEWCP^XPDUTL("ZZRON1","PRE1^ZZRON2","C-")
     Q
PRE1 ;check terminal type file
     N DA,UPDATE,NAME
     ;quit if answer NO to question 1
     Q:'XPDQUES("PRE1")
     S UPDATE=XPDQUES("PRE2")
     ;write message to user about task
     D BMES^XPDUTL("Checking Terminal Type File")
     ;get parameter value to initialize NAME
     S NAME=$$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 %=$$NEWCP^XPDUTL("ZZRON2")
     Q
POST1       ;check version multiple
     N DA,VER,%
     ;quit if answer NO to question 1
     Q:'XPDQUES("POST1")
     ;write message to user about task
     D BMES^XPDUTL("Checking Package File")
     ;get parameter value to initialize DA
     S DA=+$$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
```

**Figure 13-28. KIDS: Using checkpoints *with* callbacks: combined pre- and post-install routine**

## Checkpoints *Without* Callbacks (Data Storage)

KIDS ignores checkpoints that don't have callback routines specified. The ability to create checkpoints without a callback 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 callbacks beyond simply storing data. If you want to manage your own checkpoints instead of letting KIDS manage them, you can create checkpoints without callbacks, 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 callback 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* callback routines).

The KIDS checkpoint functions that apply when using checkpoints *without* callbacks are summarized below (listed in alphabetic order):

| Function | Description |
|---|---|
| $$COMCP^XPDUTL | Complete checkpoint (use during pre- or post-install routine). |
| $$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). |

**Table 13-13. KIDS: Functions using checkpoints *without* callbacks**

## Required Builds

In the fourth screen of the EDIT A BUILD option, you can use the Required Builds multiple to enter other builds (i.e., software applications, or patches) that either warn the installer when they are missing or requires that they be installed before this build is installed. Make an entry in the BUILD file (#9.6) for those software applications or patches not installed using KIDS. Include the name and version number in the BUILD file (#9.6) entry.

**REF:** For the action types available, please refer to Table 13-14. KIDS: Required builds installation actions table.

At the installing site, KIDS checks the PACKAGE file (#9.4), VERSION multiple, and PATCH APPLICATION HISTORY multiple to verify that the required build has been installed at that site.

```
                              Edit a Build                      PAGE 4 OF 4
Name: TEST 1.0                          TYPE: SINGLE PACKAGE
----------------------------------------------------------------------------
                        Install Questions




                        Required Builds
  TEST 1.1                                  Don't install, remove global



     Package File Link...: TEST

 Track Package Nationally: NO
____________________________________________________________________________


COMMAND:                                Press <PF1>H for help      Insert
```

**Figure 13-29. KIDS: Required builds sample**

| Installation Action | Description |
|---|---|
| WARNING ONLY | Warns the installer the listed software application/patch is missing at the site but allows the installation to continue. (Displays a **WARNING** to the installer.) |
| DON'T INSTALL, LEAVE GLOBAL | If the listed software application/patch is missing, this action prevents sites from continuing the installation. It does *not* unload the Transport Global. This allows sites to install the missing item and continue with the installation without having to reload the Transport Global. |
| DON'T INSTALL, REMOVE GLOBAL | If the listed software application/patch is missing, this action prevents sites from continuing the installation. It also *unloads* the Transport Global. |

**Table 13-14. KIDS: Required builds installation actions**

## 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 (#9.4). Use this link if you want to update the site's PACKAGE file (#9.4) when the software application 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 (#9.4) entry that is the same name (minus the version number) as the build you're creating.

If you specify a PACKAGE file (#9.4) entry in the PACKAGE FILE LINK field, and the installing site does not have a matching entry in their PACKAGE file (#9.4), KIDS creates a new entry in the installing site's PACKAGE file (#9.4).

KIDS checks for duplicate version numbers and patch names when updating the PACKAGE file (#9.4). When you link to an entry in the PACKAGE file (#9.4), your installation automatically updates the VERSION multiple in the installing site's corresponding PACKAGE file (#9.4) entry. KIDS makes a new entry in the VERSION multiple for the version of the software application 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
- PATCH APPLICATION HISTORY
  - PATCH APPLICATION HISTORY
  - DATE APPLIED
  - APPLIED BY
  - DESCRIPTION

KIDS saves patch names along with their sequence numbers in the PATCH APPLICATION HISTORY multiple (this functionality was added with patch XU*8.0*30). The Patch Application History sample (Figure 13-30) shows a list of patch names with and without sequence numbers. Those patches without sequence numbers were entered prior to patch XU*8.0*30, since no sequence numbers are evident.

In addition, you can choose to update the following fields at the top level of the National PACKAGE file (#9.4):

| PACKAGE File (#9.4) Field Name | Description |
|---|---|
| PRIMARY HELP FRAME | Select the primary help frame for the software application. |
| AFFECTS RECORD MERGE | (multiple) Select files that, if merged, affect this software application. |
| ALPHA/BETA TESTING | YES means that this software application is currently in alpha or beta test and that you want to track option usage and errors relating to this software application at the sites.<br>NO means that you want to discontinue tracking of alpha or beta testing at the sites. |

**Table 13-15. KIDS: National PACKAGE file field updates**

Beyond these fields, KIDS does *not* support maintaining any other information in the PACKAGE file (#9.4).

```
 Select PATCH APPLICATION HISTORY: 48// ?
 Answer with PATCH APPLICATION HISTORY
Choose from:
   27
   39
   41
   42
   48
   45 SEQ #41
   46 SEQ #42
   47 SEQ #43

     You may enter a new PATCH APPLICATION HISTORY, if you wish
     Answer must be 8-15 characters in length.

  Select PATCH APPLICATION HISTORY: 48// <Enter>
    PATCH APPLICATION HISTORY: 48// <Enter>
    DATE APPLIED: SEP 20,1996// <Enter>
    APPLIED BY: KRNUSER,NINETY// <Enter>
    DESCRIPTION:
  1>This contains fixes related to output fixes for the PCMM software
  2>(distributed as SD*5.3*41).
  3>
  4>Both SD*5.3*41 and SD*5.3*45 must be installed prior to loading this
  5>patch.
```

**Patches without sequence numbers means that they were entered prior Kernel Patch XU*8.0*30.**

**Figure 13-30. KIDS: Patch application history sample**

## 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 (#9.4) on FORUM, each time the software application 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 software application, provided the following conditions are met:

- The PACKAGE FILE LINK field in the build APIs to an entry in the PACKAGE file (#9.4).
- The software application is installed at a site that is a primary VA domain.
- The software application 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 software application version is installed.
- Periodic option usage reports.
- Periodic listings of errors in the software application's namespace.

This topic describes how to set up alpha/beta tracking for software applications 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:

| Alpha/Beta Testing Option | Description |
|---|---|
| Installation Message | Answering YES means that an installation message will be sent when this software application 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 software application namespaces to track. |

**Table 13-16. KIDS: Alpha/Beta testing options for KIDS edits**

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]
```

**Figure 13-31. KIDS: Errors Logged in Alpha/Beta Test option**

The Errors Logged in Alpha/Beta Test (QUEUED) option identifies any errors associated with an application that 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 that includes the:

- Type of error.
- Routine involved.
- Date (usually the previous day).
- Option that was being used at the time of the error.
- Number of times the error was logged.
- Volume.
- UCI.

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 (#8991.5).

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]
```

**Figure 13-32. KIDS: Print Alpha/Beta Errors (Date/Site/Num/Rou/Err) option**

## 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 software application, should be turned off when the final version of the software application 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 (#8989.3).

# Application Program Interface (API)

Several APIs are available for programmers to work with KIDS. These APIs are described below.

**NOTE:** For all output during pre- and post-installs, use the MES^XPDUTL(): Output a Message and BMES^XPDUTL(): Output a Message with Blank Line APIs. These functions WRITE output to both the INSTALL file (#9.7) and the output device.

## EXIT^XPDID(): Restore Screen, Clean Up Variables, and Display Text

**Reference Type** Supported

**Category** KIDS

**IA #** 2172

**Description** This API restores the screen to normal, cleans up all variables, and displays the text in the x input parameter.

**Format** `EXIT^XPDID(x)`

**Input Parameters**

| | |
|---|---|
| x: | (required) Text to display on screen after removing box and progress bar. |

**Output** none

**Example**

## INIT^XPDID: Initialize Device and Draw Box Borders

**Reference Type** Supported

**Category** KIDS

**IA #** 2172

**Description** This API Initializes the device, draws the borders for the box, and draws the progress bar.

**Format** `INIT^XPDID`

**Input Parameters** none

**Output** returns: Returns XPDIDVT:

1—If output device supports graphics.

0—If output device does *not* support graphics.

**Example**

## TITLE^XPDID(): Display Title Text

**Reference Type** Supported

**Category** KIDS

**IA #** 2172

**Description** This API displays the text in the x input parameter as a title at the top of the box.

**Format** `TITLE^XPDID(x)`

**Input Parameters** x: (required) Title text to be displayed at the top of the box.

**Output** none

**Example**

## UPDATE^XPDID(): Update Install Progress Bar

**Reference Type** Supported

**Category** KIDS

**IA #** 2172

**Description** This API updates the installation progress bar by "n", current number of items.

**Format** `UPDATE^XPDID(n,xpdidtot)`

**Input Parameters**

| | |
|---|---|
| n: | (required) The current number of items being updated. |
| xpdidtot: | (required) This is the total number of items that is being updated. |

**Output** none

### Example

If you are converting 100 records and want to update the user every time you have completed 10% of the records you would do the following:

```
>Set XPDIDTOT=100
>F%=1:1:100 D CONVERT I'(%#10) D UPDATE^XPDID(%)
```

## EN^XPDIJ(): Task Off KIDS Install

**Reference Type** Controlled Subscription

**Category** KIDS

**IA #** 2243

**Description** This API can be used with XPDA and is defined to task off a KIDS install. This is useful if a large conversion needs to run in the background while users are back on the system. For example, the first KIDS build can install a new version of software, then task off a second cleanup/conversion build. This allows users back onto the system, because the new version install completes and unlocks options and protocols. Meanwhile, the cleanup runs in the background under KIDS and makes use of KIDS checkpoints, restart upon failure, and message logging that can later be accessed in the Install File Print.

**Format** `EN^XPDIJ(xpda)`

**Input Parameters**

| | |
|---|---|
| xpda: | (required) Internal entry number of the build to be tasked in the INSTALL file (#9.7). |

**Output** none

**Example**

## $$PKGPAT^XPDIP(): Update Patch History

**Reference Type** Supported

**Category** KIDS

**IA #** 2067

**Description** This extrinsic function updates the PATCH APPLICATION HISTORY field of the VERSION field multiple in the PACKAGE file (#9.4). This function can be used during the Pre- or Post-Install routine.

**Format**

```
$$PKGPAT^XPDIP(software_ien,version,.x)
```

**Input Parameters**

| | |
|---|---|
| software_ien: | (required) The software file entry Internal Entry Number (IEN) in the PACKAGE file (#9.4). |
| version: | (required) This is the software version number. It must contain a decimal (e.g., 8.0). |
| .x | (required) |

**Output**

returns: Returns:

```
version ien^patch ien
```

**Example**

## BMES^XPDUTL(): Output a Message with Blank Line

**Reference Type** Supported

**Category** KIDS

**IA #** 10141

**Description** This API is used during KIDS installations. It outputs a message string to the installation device. A message is also recorded in the INSTALL file (#9.7) entry for the installation. It is similar to the MES^XPDUTL(): Output a Message API, except that it outputs a blank line before it outputs the message, and it doesn't take arrays.

**Format** `BMES^XPDUTL(msg)`

**Input Parameters**

| | |
|---|---|
| msg: | (required) String to output. |

**Output**

| | |
|---|---|
| returns: | Returns a message string preceded by a blank line to the installation device. |

**Example**

# $$COMCP^XPDUTL(): Complete Checkpoint

**Reference Type** Supported

**Category** KIDS

**IA #** 10141

**Description** This extrinsic function is used 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 callback routines. If the checkpoint has a callback 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 API only for checkpoints with no callback. KIDS completes checkpoints that have a callback.

**Format** `$$COMCP^XPDUTL(name)`

**Input Parameters** name: (required) Checkpoint name.

**Output** returns: Returns:

1—Successfully completed checkpoint.

0—Error completing checkpoint.

**Example**

## $$CURCP^XPDUTL(): Get Current Checkpoint Name/IEN

**Reference Type** Supported

**Category** KIDS

**IA #** 10141

**Description** This extrinsic function is used 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 API for more than one callback. Using this function, you can determine which callback you are in.

Use this API only for checkpoints *with* a callback. It will return the NULL string if you call it when working with a checkpoint with no callback (in which case, you would really be in either the pre- or post-install routine).

**Format**

```
$$CURCP^XPDUTL(format)
```

**Input Parameters**

| | |
|---|---|
| format | (required) Pass as zero (0) to return checkpoint name. Pass as 1 to return checkpoint Internal Entry Number (IEN). |

**Output**

| | |
|---|---|
| returns: | Returns:<br>Checkpoint Name—The current checkpoint name.<br>NULL String—If *not* currently in a checkpoint callback. |

**Example**

## $$LAST^XPDUTL(): Last Software Patch

**Reference Type** Supported

**Category** KIDS

**IA #** 10141

**Description** This extrinsic function returns the last patch and the date it was applied to the software. The patch will also include the Sequence # if the last patch was a released patch.

**NOTE:** This API can be used outside of KIDS.

**Format** `$$LAST^XPDUTL(x[,y])`

**Input Parameters**

x: (required) Software name or software namespace (in quotes).

y: (optional) Software version number. The current version is assumed if this parameter is *not* supplied.

**Output**

returns: Returns the last patch information in a caret-delimited string:

nnn^yyymmdd—Unreleased patch, where "nnn" = patch number and "yyymmdd" = date in VA FileMan format.

nnn Seq #nn^yyymmdd—Released patch, where "nnn" = patch number, "Seq #nn" = sequence number for released patch, and "yyymmdd" = date in VA FileMan format.

-1—If either software or version do *not* exist or no patches have been applied.

**Example**

```
>S X="KERNEL"

>S Y=8

>S Z=$$LAST^XPDUTL(x,y)

>W Z
```

## MES^XPDUTL(): Output a Message

**Reference Type** Supported

**Category** KIDS

**IA #** 10141

**Description** This API is used during KIDS installations. It outputs a message string to the installation device. A message is also recorded in INSTALL file (#9.7) entry for the installation.

**Format** `MES^XPDUTL([.]msg)`

**Input Parameters**

| | |
|---|---|
| [.]msg: | (required) Message string to output, either in a variable or passed by reference as an array of strings. |

**Output**

| | |
|---|---|
| returns: | Returns a message string to the installation device. |

**Example**

# $$NEWCP^XPDUTL(): Create Checkpoint

**Reference Type** Supported

**Category** KIDS

**IA #** 10141

**Description** This extrinsic function is used during KIDS installations. Use this function to create a checkpoint, in pre- or post-install routines. The checkpoint is stored in the INSTALL file (#9.7).

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 callback. You can also store a value for the parameter node, if you wish.

Checkpoints created with callbacks have that callback automatically executed by KIDS during the appropriate phase of the installation. If the checkpoint is created during the pre-install routine, KIDS executes the callback as soon as the pre-install routine completes. If the callback is created during the post-install, KIDS executes the callback as soon as the post-install routine completes. If multiple checkpoints are created during the pre- or post-install routine, KIDS executes the callbacks (and completes the checkpoints) in the order the corresponding checkpoints were created.

Checkpoints created without a callback 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 callback will not be overwritten.

**Format**

```
$$NEWCP^XPDUTL(name[,callback][,par_value])
```

**Input Parameters**

| | |
|---|---|
| name: | (required) Checkpoint name. |
| callback: | (optional) Callback (^routine or tag^routine reference). |
| par_value: | (optional) Value to which the checkpoint parameter is set. |

**Output** returns: Returns:

Internal Entry Number (IEN)—Created checkpoint if newly created or if checkpoint already exists.

Zero (0)—Error occurred while creating checkpoint.

**Example**

## $$OPTDE^XPDUTL(): Disable/Enable an Option

**Reference Type** Supported

**Category** KIDS

**IA #** 10141

**Description** This extrinsic function is used during KIDS installations in a Pre-Init or Post-Init routine. Use this function to disable or enable an option.

**Format** `$$OPTDE^XPDUTL(name,action)`

**Input Parameters** name: (required) Option name.

action: (required)

1—Enable an option.

2—Disable an option.

**Output** returns: Returns:

1—Success.

0—Failure.

**Example**

# $$PARCP^XPDUTL(): Get Checkpoint Parameter

**Reference Type** Supported

**Category** KIDS

**IA #** 10141

**Description** This extrinsic function is used during KIDS installations. It retrieves the current value of a checkpoint's stored parameter. The parameter is stored in the INSTALL file (#9.7).

Use this API for checkpoints both with and without callbacks.

Use the optional second parameter to retrieve a pre-install checkpoint's parameter during a post-install.

**Format** `$$PARCP^XPDUTL(name[,pre])`

**Input Parameters**

| | |
|---|---|
| name: | (required) Checkpoint name. |
| pre: | (optional) To retrieve a parameter from a pre-install checkpoint while in the post-install, set this parameter to "PRE". |

**Output**

| | |
|---|---|
| returns: | Returns the current parameter node for the checkpoint named in the name input parameter. |

**Example**

## $$PATCH^XPDUTL(): Verify Patch Installation

**Reference Type** Supported

**Category** KIDS

**IA #** 10141

**Description** This extrinsic function is used during KIDS installations—during the environment check only. Use this function to verify if a patch has been loaded. You can check for patches with or without sequence numbers.

**Format** `$$PATCH^XPDUTL(patch)`

**Input Parameters** patch: (required) Patch name.

**Output** returns: Returns:

1—Specified patch was loaded on the current system.

0—Specified patch was *not* loaded on the current system.

**Example**

Checking for a patch. Enter the following at the programmer prompt:

```
>I '$$PATCH^XPDUTL("XU*8*28") W !,"You must install patch XU*8*28"
```

## $$PKG^XPDUTL(): Parse Software Name From Build Name

**Reference Type** Supported

**Category** KIDS

**IA #** 10141

**Description** This extrinsic function parses the name of a software application from a software application'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.

**Format** `$$PKG^XPDUTL(buildname)`

**Input Parameters** buildname: Name of build (.01 field of BUILD file [#9.6]).

**Output** returns: Returns the software name.

**Example**

## $$PRODE^XPDUTL(): Disable/Enable a Protocol

**Reference Type** Supported

**Category** KIDS

**IA #** 10141

**Description** This extrinsic function is used during KIDS installations in a Pre-Init or Post-Init routine. Use this function to disable or enable a protocol.

**Format** `$$PRODE^XPDUTL(name,action)`

**Input Parameters**

| | |
|---|---|
| name: | (required) Protocol name. |
| action: | (required)<br>1—Enable a protocol.<br>2—Disable a protocol. |

**Output**

| | |
|---|---|
| returns: | Returns:<br>1—Success.<br>0—Failure. |

**Example**

## $$RTNUP^XPDUTL(): Update Routine Action

**Reference Type** Supported

**Category** KIDS

**IA #** 10141

**Description** This extrinsic function is used during KIDS installations—during the environment check only. Use this function to update the installation action for a routine.

**Format** `$$RTNUP^XPDUTL(routine,action)`

**Input Parameters**

routine: (required) Routine name.

action: (required)

1—Delete at site.

2—Skip installing at site.

**Output**

returns: Returns:

1—Routine found in routine installation list.

0—Routine *not* found in routine installation list.

**Example**

## $$UPCP^XPDUTL(): Update Checkpoint

**Reference Type** Supported

**Category** KIDS

**IA #** 10141

**Description** This extrinsic function is used 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 (#9.7).

Use this API for checkpoints both with and without callbacks.

During the pre-install, you can only update pre-install checkpoints; during the post-install, you can only update post-install checkpoints.

**Format** `$$UPCP^XPDUTL(name[,par_value])`

**Input Parameters**

name: (required) Checkpoint name.

par_value: (optional) Value to set checkpoint parameter to.

**Output**

returns: Returns:

Internal Entry Number (IEN)—Successfully updated checkpoint.

Zero (0)—Error updating checkpoint.

**Example**

## $$VER^XPDUTL(): Parse Version from Build Name

**Reference Type** Supported

**Category** KIDS

**IA #** 10141

**Description** This extrinsic function parses the version of a software application from a software application'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.

**Format**

```
$$VER^XPDUTL(buildname)
```

**Input Parameters**

buildname: (required) Name of build (.01 field of BUILD file [#9.6]).

**Output**

returns: Returns:

Version—The version of the build identified in the buildname input parameter.

NULL—If no match in the BUILD file (#9.6).

**Example**

# $$VERCP^XPDUTL(): Verify Checkpoint

**Reference Type** Supported

**Category** KIDS

**IA #** 10141

**Description** This extrinsic function is used 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 API only for checkpoints with no callback.

During the pre-install, you can only verify pre-install checkpoints; during the post-install, you can only verify post-install checkpoints.

**Format** `$$VERCP^XPDUTL(name)`

**Input Parameters** name: (required) Checkpoint name.

**Output** returns: Returns:

1—Checkpoint has completed.

0—Checkpoint has not completed but exists.

-1—Checkpoint doesn't exist.

**Example**

## $$VERSION^XPDUTL(): Package File Current Version

**Reference Type** Supported

**Category** KIDS

**IA #** 10141

**Description** This extrinsic function obtains the current version of a site's software application.

**Format** `$$VERSION^XPDUTL(package_id)`

**Input Parameters**

package_id: (required) Software application's name or namespace, from its entry in the PACKAGE file (#9.4).

**Output**

returns: Returns:

Version—The current version of the software application at the site, according to the software application's entry in the site's PACKAGE file (#9.4).

NULL—If the software application is *not* matched.

**Example**

# 14. Menu Manager: Programmer Tools

## Creating Options

You can develop applications 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 topic, are used to group other options for presentation to the user at the select prompt.
- Server options 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.

  **REF:** For a complete description, please refer to the "Server Options: Programmer Tools" chapter later in this "Menu Manager" section in this manual.

- Protocol, Protocol Menu, Extended Action, and Limited option types are specific to the XQOR (Unwinder) software application. 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 software application is exported.

  **REF:** For more information, please refer to the Computerized Patient Record System (CPRS) or Unwinder (XQOR) documentation.

### Creating Options (Edit Options)

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

**Figure 14-1. Menu Manager: 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 (#19) 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.

**NOTE:** For a complete description of the meaning of the variables represented by each of the DI fields, please refer to the *VA FileMan Programmer Manual*.

### Options that Should Be Regularly Scheduled

If an option should be regularly scheduled to run through TaskMan, you should set its SCHEDULING RECOMMENDED field (#209) in the OPTION file (#19)) 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 can be defined within applications, such as in an option's Entry Action, Exit Action, or Header. These variables are listed below.

The XQMM variables can 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 can 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 cannot 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 **<Enter>**, 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 can be used to force a menu jump to an option within the user's menu tree. Set it equal to the exact option name (i.e., .01 field of the OPTION file [#19]) to which Menu Manager should jump. For example:

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

This 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 can 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 can 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") can 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

## ^XQ1: Test an Option

The ^XQ1 routine 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:

```
>D ^XQ1
```

This API is also called by ^XUP.

**CAUTION: Programmers are advised to use ^XUP instead of ^XQ1 to enter Kernel from programmer mode, since the ^XUP routine sets up a standard environment and takes care of cleanup activities.**

**REF:** For a description of the ^XUP direct mode utility, please refer to the "Signon/Security: Programmer Tools" chapter in this manual.

**NOTE:** While D ^XQ1 is a direct mode utility, it is *not* a callable API.

# Application Program Interface (API)

Several APIs are available for programmers to work with menu management. These APIs are described below.

## $$ADD^XPDMENU(): Add Option to Menu

**Reference Type** Supported

**Category** Menu Manager

**IA #** 1157

**Description** This extrinsic function adds an option as a new item to an existing menu.

**Format** `$$ADD^XPDMENU(menu,option[,syn][,order])`

**Input Parameters**

| | |
|---|---|
| menu: | (required) Name of the menu to which an option should be added. |
| option: | (required) 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** returns: Returns:

1—Success, option added to menu.

0—Failure option *not* added to menu.

**Example**

## DELETE^XPDMENU(): Delete Menu Item

**Reference Type** Supported

**Category** Menu Manager

**IA #** 1157

**Description** This API deletes an option from the Menu field of another option. It returns the following:

1—If the function succeeded.

0—If it failed.

**Format**

```
DELETE^XPDMENU(menu,option)
```

**Input Parameters**

menu: (required) This is the name of the option from which you want to delete a menu item.

option: (required) This is the name of the option you want to delete from the menu item of the "menu" input parameter.

**Output**

returns: Returns:

1—Success, menu item deleted.

0—Failure, menu item *not* deleted.

**Example**

## LKOPT^XPDMENU(): Look Up Option IEN

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Menu Manager | |
| **IA #** | 1157 | |
| **Description** | This API looks up an option's Internal Entry Number (IEN) using the "B" cross-reference. | |
| **Format** | `LKOPT^XPDMENU(x)` | |
| **Input Parameters** | x: | (required) The name of the input option. |
| **Output** | returns: | Returns the Internal Entry Number (IEN) of the input option in the OPTION file (#19). |

**Example**

## OUT^XPDMENU(): Edit Option's Out of Order Message

**Reference Type** Supported

**Category** Menu Manager

**IA #** 1157

**Description** This API creates or deletes an out of order message for an option; this action effectively puts the option out of order or back in order.

**Format** `OUT^XPDMENU(option,text)`

**Input Parameters**

option: (required) Name of option in which to place an OUT OF ORDER MESSAGE value.

text: (required) 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

**Example**

## RENAME^XPDMENU(): Rename Option

**Reference Type** Supported

**Category** Menu Manager

**IA #** 1157

**Description** This API renames an existing option.

**Format** `RENAME^XPDMENU(old,new)`

**Input Parameters**

| | |
|---|---|
| old: | (required) Current option name (.01 field of OPTION file [#19] entry). Must be an exact match. |
| new: | (required) New name for option. |

**Output** none

**Example**

## NEXT^XQ92(): Restricted Times Check

**Reference Type** Supported

**Category** Menu Manager

**IA #** 10077

**Description** This API 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, the x parameter is returned as NULL and a message is issued regarding the time restriction.

**Format** `NEXT^XQ92(ien,x)`

**Input Parameters**

| | |
|---|---|
| ien: | (required) Internal entry number (IEN) of the option in the OPTION file (#19). |

**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. |

**Example**

## $$ACCESS^XQCHK(): User Option Access Test

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Menu Manager | |
| **IA #** | 10078 | |
| **Description** | This extrinsic function is used to find out if a user has access to a particular option. | |
| **Format** | `$$ACCESS^XQCHK(duz,option)` | |
| **Input Parameters** | duz: | (required) The identification number of the user in question in the NEW PERSON file (#200). |
| | option: | (required) The Internal Entry Number (IEN) or option name of the option in question in the OPTION file (#19). |

**Output** returns: Returns:

-1—No such user in the NEW PERSON file (#200).

-2—User terminated or has no Access code.

-3—No such option in the Option file (#19).

0—No access found in any menu tree the user owns.

4-Piece String—

- access^menu tree IEN^a set of codes^key
- 0^tree^codes^key: No access because of locks (see XQCODES below).
- 1^OpIEN^^: Access allowed through Primary Menu.
- 2^OpIEN^codes^: Access found in the Common Options.
- 3^OpIEN^codes^: Access found in top level of secondary option.
- 4^OpIEN^codes^: Access through a the secondary menu tree OpIEN.

XQCODES can contain the following:

- N—No Primary Menu in the NEW PERSON file (#200, warning only).
- L—Locked and the user does *not* have the key (forces zero [0] in first piece).
- R—Reverse lock and user has the key (forces zero [0] in first piece).

**Example**

## OP^XQCHK(): Current Option Check

**Reference Type** Supported

**Category** Menu Manager

**IA #** 10078

**Description** This API returns the current option or protocol name and menu text in the first and second pieces of the xqopt output parameter. It looks for the local XQY variable, 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:** XQCHK cannot return option/protocol information if the job is a task that did not originate from an option.

**Format**

```
OP^XQCHK(xqopt)
```

**Input Parameters** none

**Output Parameters** xqopt: Returns a 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
```

**Example**

# 15. Miscellaneous: Programmer Tools

## Application Program Interface (API)

The following are miscellaneous APIs available for programmers. These APIs are described below.

### ^XQDATE: Convert $H to VA FileMan Format (Obsolete)

**NOTE:** This API is obsolete. You should use either of the following APIs instead:

- $$FMTE^XLFDT(): Convert VA FileMan Date to External Format
- $$HTFM^XLFDT(): Convert $H to VA FileMan Date Format

| | |
|---|---|
| **Reference Type** | Supported |
| **Category** | Miscellaneous |
| **IA #** | 10079 |
| **Description** | This API converts $H formatted input date to a VA FileMan formatted date in %, and in human readable format (e.g., Jan. 9, 1990 1:37 PM) in %Y variable. |
| **Format** | `^XQDATE` |

Make sure to perform the following steps before calling ^XQDATE:

- NEW all of the following variables.
- Set the input variable, if you want to change it.
- Call ^XQDATE.

If you do not follow these steps, the input variable could unintentionally assume the value of the variables of the current running task.

| | | |
|---|---|---|
| **Input Variable** | XQD1: | (optional) If this variable is not set, the system uses $H. |
| **Output Variables** | %: | Returns the converted $H date in VA FileMan format. |
| | %Y: | Returns the converted $H date, in human readable format. |

**Example**

## $$EN^XUA4A71(): Convert String to Soundex

**Reference Type** Supported

**Category** Miscellaneous

**IA #** 3178

**Description** This extrinsic function converts 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.

**Format** `$$EN^XUA4A71(string)`

**Input Parameters**

| | |
|---|---|
| string: | (required) String to convert into soundex form. |

**Output**

| | |
|---|---|
| returns: | Returns the soundex version of the string. |

**Example**

# ^XUWORKDY: Workday Calculation (Obsolete)

**NOTE:** This API is obsolete. The XUWORKDY routine is maintained for code that might still use it.

**Reference Type** Supported

**Category** Miscellaneous

**IA #** 10046

**Description**

To use the ^XUWORKDY API, you must make sure, on the system that it is being used on, that Kernel's HOLIDAY file (#40.5) 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) ).

**Format**

```
^XUWORKDY
```

**Input Variables**

| | |
|---|---|
| X: | (required) Starting date in the YYYMMDD format (e.g., 2850420). |
| X1: | (required) Ending date in the YYYMMDD format (e.g., 2850707). |

**Output Variables**

| | |
|---|---|
| X: | The number of workdays in the interval. |

## Example

```
>S X=2850420,X1=2850707 D ^XUWORKDY W X

55
```

# 16. Name Standardization: Programmer Tools

## Application Program Interface (API)

Several APIs are available for programmers to work with name standardization. These APIs are described below.

### $$BLDNAME^XLFNAME(): Build Name from Component Parts

**Reference Type** Supported

**Category** Name Standardization

**IA #** 3065

**Description** This extrinsic function takes the component parts of a name and returns the name, truncated if necessary, in the following format:

Family_name,Given_name<space>Middle_name<space>Suffix(es)

**Format** `$$BLDNAME^XLFNAME(`.name[,max]`)`

**Input Parameters** .name (required) The component parts of the name:

NAME("FAMILY") = Family (Last) Name
NAME("GIVEN") = Given (First) Name(s)
NAME("MIDDLE") = Middle Name(s)
NAME("SUFFIX") = Suffix(es)

Alternatively, this array can contain the file number, IENS, and field number of the field that contains the name. If the name has a corresponding entry in the NAME COMPONENTS file (#20), then the name components are obtained from that entry. Otherwise, the name is obtained directly from the file, record, and field specified, and the name components are obtained by making a call to the STDNAME^XLFNAME(): Name Standardization Routine API.

NAME("FILE") = Source file number (required)
NAME("IENS") = IENS of entry in the source file (required)
NAME("FIELD") = Source field number (required)

max: (optional) The maximum length of the Name to be returned (default = 256). See the following section named "Details" for a description of the pruning algorithm.

**Output** returns: Returns the name, truncated if necessary, in the following format:

```
Family_name,Given_name<space>Middle_name<space>
Suffix(es)
```

**Details**

If the MAX input parameter is used, and the resulting name is longer than MAX, the following pruning algorithm is performed to shorten the name:

1. Truncate Middle Name from the right-most position until only the initial character is left;
2. Drop suffix;
3. Truncate Given Name from the right-most position until only the initial character is left;
4. Truncate Family Name from the right-most position;
5. Truncate the name from the right.

**Example 1**

Suppose the MYNAME array contains the following elements:

```
MYNAME("FAMILY")="KRNUSER"
MYNAME("GIVEN")="SIXTY"
MYNAME("MIDDLE")="K."
MYNAME("SUFFIX")="JR"
```

Calls to $$BLDNAME^XLFNAME will return the name as follows:

```
>S X=$$BLDNAME^XLFNAME(.MYNAME)

>W X
KRNUSER,SIXTY K JR
```

"Pruning" the name to 12 characters total:

```
>S X=$$BLDNAME^XLFNAME(.MYNAME,12)

>W X
KRNUSER,SI K
```

**Example 2**

If an entry in the NAME COMPONENTS file (#20) stores the components of a name stored in the NAME field (#.01) of record number 32 in the NEW PERSON file (#200), and the data in the corresponding record in the NAME COMPONENT file (#20) is:

```
FILE=200
FIELD=.01
IENS="32,"
GIVEN NAME="SIXTY"
MIDDLE NAME="K."
FAMILY NAME="KRNUSER"
SUFFIX="JR"
```

you can set:

```
MYNAME("FILE")=200
MYNAME("FIELD")=.01
MYNAME("IENS")="32,"
```

and call $$BLDNAME^XLFNAME as in Example 1, listed previously:

```
>S X=$$BLDNAME^XLFNAME(.MYNAME)

>W X
KRNUSER,SIXTY K JR
```

"Pruning" the name to 12 characters total:

```
>S X=$$BLDNAME^XLFNAME(.MYNAME,12)

>W X
KRNUSER,SI K
```

## $$CLEANC^XLFNAME(): Name Component Standardization Routine

**Reference Type** Supported

**Category** Name Standardization

**IA #** 3065

**Description** This extrinsic function takes a single name component and returns that name in standard format.

**Format** `$$CLEANC^XLFNAME(comp[,flags])`

**Input Parameters**

comp: (required) The name component to be converted to standard format.

flags: (optional) Flag to control processing. Possible values are:

F—If the name component to be converted is the FAMILY (LAST) NAME, pass the "F" flag. With the "F" flag, colons (**:**), semicolons (**;**), and commas (**,**) are converted to hyphens (**-**). Spaces and all punctuation except hyphens are removed. Two or more consecutive spaces or hyphens are replaced with a single space or hyphen. Birth position indicators 1ST through 10TH are changed to their Roman numeral equivalents.

NULL—Without the "F" flag, the component is converted to upper case. Colons (**:**), semicolons (**;**), commas (**,**), and periods (**.**) are converted to spaces. All punctuation except for hyphens and spaces are removed. Two or more consecutive spaces or hyphens are replaced with a single space or hyphen. Birth position indicators 1ST through 10TH are changed to their Roman numeral equivalents.

**Output** returns: Returns the standard formatted name.

### Example 1

Standardize family (last) name:

```
>Set X=$$CLEANC^XLFNAME("KRNUSER-XU U SER","F")

>W X
KRNUSER-XUUSER
```

```
>Set X=$$CLEANC^XLFNAME("KRNUSER-XU U SER 2ND","F")

>W X
KRNUSER-XUUSERII

>Set X=$$CLEANC^XLFNAME("KRNUSER-XU U SER")

>W X
KRNUSER-XU U SER

>Set X=$$CLEANC^XLFNAME("ST. USER","F")

>W X
STUSER
```

### Example 2

Standardize other (non-family) name components:

```
>S X=$$CLEANC^XLFNAME("F.O.")

>W X
F O

>S X=$$CLEANC^XLFNAME("FORTY'")

>W X
FORTY

>S X=$$CLEANC^XLFNAME("FORTY ONE")

>W X
FORTY ONE

>S X=$$CLEANC^XLFNAME("FORTY-ONE")

>W X
FORTY-ONE
```

## $$FMNAME^XLFNAME(): Convert HL7 Formatted Name to Name

**Reference Type** Supported

**Category** Name Standardization

**IA #** 3065

**Description** This extrinsic function converts an HL7 formatted input name to a VistA formatted name.

**Format** `$$FMNAME^XLFNAME([.]name[,flags][,delim])`

**Input Parameters**

[.]name: (required) This is the HL7 name to be converted; it can be passed by reference. If the "C" flag is used, the name components are returned in nodes descendent from this parameter (see "Output" that follows).

flags: (optional) Flags to controls processing. Possible values are:

C—Return name components in the NAME array (see "Output" that follows).

L#—Truncate the returned name to a maximum Length of # characters, where # is an integer between 1 and 256.

M—Return the name in Mixed case, with the first letter of each name component capitalized.

S—Return the name in Standardized form.

delim: (optional) The delimiter used in the HL7 formatted name (default = "^").

**Output Parameters**

name: If the FLAGS input parameter contains a "C", the component parts of the name are returned in the NAME array:

NAME("FAMILY) = Family (Last) Name
NAME("GIVEN") = Given (First) Name(s)
NAME("MIDDLE") = Middle Name(s)
NAME("SUFFIX") = Suffix(es)

### Details

If the L# flag is used, and the resulting name is longer than #, the following pruning algorithm is performed to shorten the name:

1. Truncate Middle Name from the right-most position until only the initial character is left;
2. Drop suffix;
3. Truncate Given Name from the right-most position until only the initial character is left;
4. Truncate Family Name from the right-most position;
5. Truncate the name from the right.

### Example 1

Convert an HL7 formatted name to a VistA name:

```
>S X=$$FMNAME^XLFNAME("KRNUSER^SIXTY^K.^JR^MR.^PHD")

>W X
KRNUSER,SIXTY K. JR

>S X=$$FMNAME^XLFNAME("KRNUSER^SIXTY^K.^JR^MR.^PHD","S")

>W X
KRNUSER,SIXTY K JR

>S X=$$FMNAME^XLFNAME("KRNUSER^SIXTY^K.^JR^MR.^PHD","M")

>W X
Krnuser,Sixty K. Jr

>S X=$$FMNAME^XLFNAME("KRNUSER^SIXTY^K.^JR^MR.^PHD","SL12")

>W X
KRNUSER,SI K
```

### Example 2

Convert an HL7 formatted name where "~" is the delimiter to a standard name:

```
>S X=$$FMNAME^XLFNAME("KRNUSER~SIXTY~K.~JR~MR","S","~")

>W X
KRNUSER,SIXTY K JR
```

### Example 3

Convert an HL7 formatted name to a standard name, and return the components of that name in the MYNAME array:

```
>S MYNAME="KRNUSER^SIXTY^K.^JR^MR.^PHD"

>W $$FMNAME^XLFNAME(.MYNAME,"CS")
KRNUSER,SIXTY K JR

>ZW MYNAME
MYNAME=KRNUSER^SIXTY^K.^JR^MR.^PHD
MYNAME("DEGREE")=PHD
MYNAME("FAMILY")=KRNUSER
MYNAME("GIVEN")=SIXTY
MYNAME("MIDDLE")=K.
MYNAME("PREFIX")=MR.
MYNAME("SUFFIX")=JR
```

## $$HLNAME^XLFNAME(): Convert Name to HL7 Formatted Name

**Reference Type** Supported

**Category** Name Standardization

**IA #** 3065

**Description** This extrinsic function converts an input name to an HL7 formatted name.

**Format** `$$HLNAME^XLFNAME([.]name[,flags][,delim])`

**Input Parameters**

[.]name: (required) The component parts of the name to be converted:

NAME("FAMILY) = Family (Last) Name (required)
NAME("GIVEN") = Given (First) Name(s) (optional)
NAME("MIDDLE") = Middle Name(s) (optional)
NAME("SUFFIX") = Suffix(es) (optional)
NAME("PREFIX") = Prefix (optional)
NAME("DEGREE") = Degree (optional)

Alternatively, this array can contain the file number, IENS, and field number of the field that contains the name. If the name has a corresponding entry in the NAME COMPONENTS file (#20), then the name components are obtained from that entry. Otherwise, the name is obtained directly from the file, record, and field specified, and the name components are obtained by making a call to the STDNAME^XLFNAME(): Name Standardization Routine API.

NAME("FILE") = Source file number (required)
NAME("IENS") = IENS of entry in the source file (required)
NAME("FIELD") = Source field number (required)

Another alternative is to pass in the unsubscripted NAME parameter the name to be converted. $$HLNAME^XLFNAME obtains the components parts of that name by making a call to the STDNAME^XLFNAME(): Name Standardization Routine API. This alternative is recommended only for names that do not have associated entries on the NAME COMPONENTS file (#20).

flags: (optional) Flags to controls processing. Possible values are:

L#—Truncate the returned name to a maximum Length of # characters, where # is an integer between 1 and 256.

S—Return the name components in the HL7 formatted name in Standardized form.

delim: (optional) The delimiter to use in the HL7 string (default = "^").

**Output** returns: Returns the converted name in HL7 format.

**Details**

If the L# flag is used, and the resulting name is longer than #, the following pruning algorithm is performed to shorten the name:

1. Truncate Middle Name from the right-most position until only the initial character is left;
2. Drop suffix;
3. Truncate Given Name from the right-most position until only the initial character is left;
4. Truncate Family Name from the right-most position;
5. Truncate the name from the right.

**Example 1**

Suppose the MYNAME array contains the following elements:

```
MYNAME("PREFIX")="MR."
MYNAME("GIVEN")="SIXTY"
MYNAME("MIDDLE")="K."
MYNAME("FAMILY")="KRNUSER"
MYNAME("SUFFIX")="JR"
MYNAME("DEGREE")="PHD"
```

Then calls to the $$HLNAME^XLFNAME API will return the name as follows:

```
>S X=$$HLNAME^XLFNAME(.MYNAME)

>W X
KRNUSER^SIXTY^K.^JR^MR.^PHD

>S X=$$HLNAME^XLFNAME(.MYNAME,"","~")

>W X
KRNUSER~SIXTY~K.~JR~MR.~PHD

>S X=$$HLNAME^XLFNAME(.MYNAME,"S","~")

>W X
KRNUSER~SIXTY~K~JR~MR~PHD

>S X=$$HLNAME^XLFNAME(.MYNAME,"L12S")

>W X
KRNUSER^SI^K
```

**Example 2**

If an entry in the NAME COMPONENTS stores the components of a name stored in the NAME field (#.01) of record number 32 in the NEW PERSON file (#200), and the data in the corresponding record in the NAME COMPONENT file (#20) is:

```
FILE = 200
FIELD = .01
IENS = "32,"
PREFIX = "MR."
GIVEN NAME = "SIXTY"
MIDDLE NAME = "K."
FAMILY NAME = "KRNUSER"
SUFFIX = "JR"
DEGREE = "PHD"
```

You can set:

```
MYNAME("FILE") = 200
MYNAME("FIELD") = .01
MYNAME("IENS") = "32,"
```

Then call the $$HLNAME^XLFNAME API, as in Example 1, to return the name in various formats.

**Example 3**

Convert a name passed by value to HL7 format:

```
>S X=$$HLNAME^XLFNAME("KRNUSER,SIXTY HOWARD II")

>W X
KRNUSER^SIXTY^HOWARD^II

>S X=$$HLNAME^XLFNAME("KRNUSER,SIXTY HOWARD II","S")

>W X
KRNUSER^SIXTY^HOWARD^II

>S X=$$HLNAME^XLFNAME("KRNUSER,SIXTY HOWARD II","SL10","~")

>W X
KRNUSE~S~H
```

# NAMECOMP^XLFNAME(): Component Parts from Standard Name

**Reference Type** Supported

**Category** Name Standardization

**IA #** 3065

**Description** This API takes a name in standard format and returns in an array the component parts of that name.

**Format** `NAMECOMP^XLFNAME(.name)`

**Input Parameters**

.name: (required) This parameter is the name in standard format to be parsed. NAMECOMP^XLFNAME returns the component parts of the name in nodes descendent from NAME. (See "Output" that follows.)

**Output Parameters**

.name: The component parts of the name are returned in the NAME array passed in.

NAME("FAMILY) = Family (last) Name
NAME("GIVEN") = Given (first) Name
NAME("MIDDLE") = Middle Name
NAME("SUFFIX") = Suffix(es)

**Example**

In this example, the MYNAME variable is set to the standard name, and the NAMECOMP^XLFNAME call is made to return in the MYNAME array the component parts of that name:

```
>S MYNAME="KRNUSER-XUUSER,FORTY ONE S MD"
>D NAMECOMP^XLFNAME(.MYNAME)

>ZW MYNAME
MYNAME=KRNUSER-XUUSER,FORTY ONE S MD
MYNAME("FAMILY")=KRNUSER-XUUSER
MYNAME("GIVEN")=FORTY ONE
MYNAME("MIDDLE")=S
MYNAME("SUFFIX")=MD
```

## $$NAMEFMT^XLFNAME(): Formatted Name from Name Components

**Reference Type** Supported

**Category** Name Standardization

**IA #** 3065

**Description** This extrinsic function returns a name converted to a form useful for display.

**Format** `$$NAMEFMT^XLFNAME(.name[,format][,flags])`

**Input Parameters**

.name: (required) An array that contains the component parts of the name:

NAME("FAMILY) = Family (Last) Name (required)
NAME("GIVEN") = Given (First) Name(s) (optional)
NAME("MIDDLE") = Middle Name(s) (optional)
NAME("SUFFIX") = Suffix(es) (optional)
NAME("PREFIX") = Prefix (optional)
NAME("DEGREE") = Degree (optional)

Alternatively, this array can contain the file number, IENS, and field number of the field that contains the name. If the name has a corresponding entry in the NAME COMPONENTS file (#20), then the name components are obtained from that entry. Otherwise, the name is obtained directly from the file, record, and field specified, and the name components are obtained by making a call to the STDNAME^XLFNAME(): Name Standardization Routine API.

NAME("FILE") = Source file number (required)
NAME("IENS") = IENS of entry in the source file (required)
NAME("FIELD") = Source field number (required)

format: (optional) Controls the general formatting of the output (default = G). Possible values are:

F—Return **F**amily (Last) Name first.

G—Return **G**iven (First) Name first.

O—Return **O**nly the Family (Last) Name.

flags: (optional) Flags to controls processing. Possible values are:

C—If the "F" format is used, return a **C**omma between the Family (Last) and Given (First) Names. Otherwise, the Family (Last) Name and the Given (First) Name are separated by a space. (Ignored if the "F" format is not used.)

D—Return the **D**egree.

Dc—Return the **D**egree preceded by a **c**omma and space.

L#—Truncate the returned name to a maximum Length of # characters, where # is an integer between 1 and 256. See "Details" as follows for a description of the pruning algorithm.

M—Return the name in **M**ixed case, with the first letter of each name component capitalized.

P—Return the **P**refix.

S—**S**tandardize the name components before building formatted name.

Xc—Precede the Suffi**X** with a **c**omma and space.

**Output** returns: Returns the formatted name.

**Details**

If the L# flag is used, and the resulting name is longer than **#**, the following pruning algorithm is performed to shorten the name:

1. Drop Degree;
2. Drop Prefix;
3. Truncate Middle Name from the right-most position until only the initial character is left;
4. Drop suffix;
5. Truncate Given Name from the right-most position until only the initial character is left;
6. Truncate Family Name from the right-most position;
7. Truncate the name from the right.

**Example 1**

Suppose the MYNAME array contains the following elements:

```
MYNAME("PREFIX")="MR."
MYNAME("GIVEN")="SIXTY"
MYNAME("MIDDLE")="K."
MYNAME("FAMILY")="KRNUSER"
MYNAME("SUFFIX")="JR"
MYNAME("DEGREE")="PHD"
```

Then calls to the $$NAMEFMT^XLFNAME API will return the name as follows:

```
>S X=$$NAMEFMT^XLFNAME(.MYNAME,"F")

>W X
KRNUSER SIXTY K. JR

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"F","C")

>W X
KRNUSER,SIXTY K. JR

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"F","CS")

>W X
KRNUSER,SIXTY K JR

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"F","CSD")

>W X
KRNUSER,SIXTY K JR PHD

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"F","CDcXc")

>W X
KRNUSER,SIXTY K., JR, PHD

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"F","CSL12")

>W X
KRNUSER,SI K

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"F","CMD")

>W X
Krnuser,Sixty K. Jr PhD

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G")

>W X
SIXTY K. KRNUSER JR

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G","D")

>W X
SIXTY K. KRNUSER JR PHD
```

```
>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G","Dc")

>W X
SIXTY K. KRNUSER JR, PHD

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G","P")

>W X
MR. SIXTY K. KRNUSER JR

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G","Xc")

>W X
SIXTY K. KRNUSER, JR

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G","PDcXc")

>W X
MR. SIXTY K. KRNUSER, JR, PHD

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G","PDcXcM")

>W X
Mr. Sixty K. Krnuser, Jr, PhD

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G","S")

>W X
SIXTY K KRNUSER JR

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G","SL12")

>W X
SI K KRNUSER

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"O")

>W X
KRNUSER

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"O","S")

>W X
KRNUSER

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"O","M")

>W X
Krnuser

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"O","L3")

>W X
KRN
```

**Example 2**

If an entry in the NAME COMPONENTS stores the components of a name stored in the NAME field (#.01) of record number 32 in the NEW PERSON file (#200), and the data in the corresponding record in the NAME COMPONENT file (#20) is:

```
FILE = 200
FIELD = .01
IENS = "32,"
PREFIX = "MR."
GIVEN NAME = "SIXTY"
MIDDLE NAME = "K."
FAMILY NAME = "KRNUSER"
SUFFIX = "JR"
DEGREE = "PHD"
```

You can set:

```
MYNAME("FILE")=200
MYNAME("FIELD")=.01
MYNAME("IENS")="32,"
```

Then call the $$NAMEFMT^XLFNAME API, as in Example 1, to return the name in various formats.

## STDNAME^XLFNAME(): Name Standardization Routine

**Reference Type** Supported

**Category** Name Standardization

**IA #** 3065

**Description** This API parses a name and converts it into the following standard format:

Family_name,Given_name<space>Middle_name<space>Suffix(es)

A name in standard format is entirely in uppercase, and contains no Arabic numerals. The Family_name (last name) portion of a standard name appears to the left of the comma and contains no spaces and no punctuation except hyphens (-). The other parts of a standard name (the portion to the right of the comma) contain no punctuation except for hyphens and spaces. NMI and NMN are not used for the Middle_name.

STDNAME^XLFNAME optionally returns in an array the component parts of the name. It also optionally returns information in an array about possible problems encountered during the conversion of the name to standard form and the parsing of the name into its component parts.

**Format**

```
STDNAME^XLFNAME(.name[,flags][,.audit])
```

**Input Parameters**

.name (required) NAME is the name to be converted to standard format. It is assumed that the name is in the general format:

Family_name,Given_name(s) Middle_name Suffix(es)

If the "F" flag is not used, and the name contains no comma, it is assumed the name is in the general format:

Given_name(s) Middle_name Family_name Suffix(es)

The standard form of the name is returned in the NAME variable. If the "C" flag is passed in, the components of the name are returned in nodes descendent from NAME. (See "Output" that follows.)

flags: (optional) Flags to control processing. Possible values are:

C—Return name components in the NAME array. (See "Output" that follows.)

F—If the name passed in the NAME input parameter does not contain a comma, assume it is the Family Name only. For example, if the name input is "ST USER", return the name as "STUSER" instead of "USER,ST".

G—Don't return AUDIT("GIVEN") even if the Given Name is missing.

P—Remove text in parentheses **( )**, brackets **[ ]**, or braces **{ }** from the name. If such text is actually removed, return AUDIT("STRIP").

.audit: (optional) If provided, this is an array that STDNAME^XLFNAME returns if there are any ambiguities or possible problems in standardizing the name or parsing the name into component parts. (See "Output" that follows.)

**Output Parameters**

name: This parameter is set to the name that was input converted to standard format.

If the Flags input parameter contains a "C", the component parts of the name are returned in the NAME array:

NAME("FAMILY) = Family (Last) Name
NAME("GIVEN") = Given (First) Name(s)
NAME("MIDDLE") = Middle Name
NAME("SUFFIX") = Suffix(es)

audit: If this parameter is set to the original name that was passed in the Name parameter. In addition, if there were any problems in the interpretation of the Name being standardized, descendents of Audit are set:

AUDIT("*subscript*") = ""

where "*subscript*" can be any one of the following:

AUDIT("FAMILY")—The Family Name starts with ST. (The period and space are removed from the Family Name. For example, the name "ST. USER" is converted to "STUSER".)

AUDIT("GIVEN")—Returned if there is no Given Name and the "G" flag isn't passed in.

AUDIT("MIDDLE")—Returned if there are three or more names between the first comma and

the Suffix(es). (All name parts except the last are assumed to be part of the Given Name. Only the last part is assumed to be the Middle Name.)

AUDIT("NM")—Returned if NMI or NMN appears to be used as the Middle Name. (NMI and NMN are removed from the standard name, and the Middle Name component is returned as null.)

AUDIT("NOTE")—Returned if the name appears to contain a note or flag that may not actually be part of the name. For example, the name starts with "C-" or "EEE," or has "FEE" at the end.

AUDIT("NUMBER")—Returned if a name part (other than a valid numeric Suffix) contains a number.

AUDIT("PERIOD")—Returned if periods were removed.

AUDIT("PUNC")—Returned if punctuation was removed.

AUDIT("SPACE")—Returned if spaces were removed from the Family Name.

AUDIT("STRIP")—Returned if text in parentheses **( )**, brackets **[ ]**, or braces **{ }** were removed from the Name. (This is done only if the "P" flag is passed.)

AUDIT("SUFFIX")—Returned if:

- Suffix(es) are found immediately to the left of the 1st comma.
- I, V, or X, and nothing else except valid suffixes, appear immediately after the Given Name. (It is interpreted as the Middle Name.)
- The name immediately after the Given Name appears to be a non-numeric suffix (except I, V, and X), and everything after that also appear to be suffixes. (It is assumed there are a Given Name and Suffix(es), but no Middle Name.)
- M.D. or M D is found at the end of the name, or before any valid suffixes at the end of the name. (It is assumed that M and D are initials in the Given or Middle Name rather than a Suffix.)
- The name part before any recognizable suffixes is more than one character in length and doesn't contain any vowels or Y. It is interpreted as a suffix.

- Suffix is found between commas immediately after the Family Name.

**Details**

In forming the standard name, the following changes are made:

1. The name is converted to uppercase.
2. In the Family Name:
   a. Semicolons (**;**) and colons (**:**) are converted to hyphens (**-**).

      Spaces and all other punctuation except hyphens are removed.

   b. Spaces and all other punctuation except hyphens are removed.
3. In the other name parts (Given Name, Middle Name, and Suffix).
   a. Semicolon, colons, commas (**,**), and periods (**.**) are converted to spaces.

      Spaces and all other punctuation except hyphens are removed.

   b. All punctuation except hyphens and spaces are removed.
4. Hyphens and spaces at the beginning and end of the name are removed.
5. Two or more consecutive hyphens/spaces are replaced with a single hyphen/space.
6. Any suffixes immediate preceding the comma are moved to the end.
7. The suffixes indicating birth positions 1st, 2nd, 3rd, ..., 10th are converted to their Roman numeral equivalents I, II, III, … X.
8. DR immediately after the comma (or if there is no comma, at the beginning of the name), is assumed to be a suffix and moved to the end of the name.
9. Any suffixes between two commas immediate after the Family Name are moved to the end of the name.
10. NMI or NMN used as a Middle Name is deleted.

In forming the component parts of the name, only the following changes are made:

1. The name component is converted to uppercase.
2. In the Family Name, semicolons (**;**) and colons (**:**) are converted to hyphens (**-**).
3. In the other name parts (Given Name, Middle Name, and Suffix), semicolons, colons, and commas (**,**) are converted to spaces.
4. Hyphens and spaces at the beginning and end of the name are removed.
5. Two or more consecutive hyphens/spaces are replaced with a single hyphen/space.
6. A Middle Name of NMI or NMN is changed to null.
7. Spaces after periods are removed.

8. Accent graves (`) and up-arrows (^) are removed.

In parsing the name into its component parts, if the name contains a comma or the "F" flag is passed, STDNAME^XLFNAME looks for suffixes immediately to the left of the first comma, and at the very end of the name. The suffixes it recognizes are 1ST through 10TH, JR, SR, DR, MD, ESQ, DDS, RN and Roman numerals I through X. If a name part before any recognizable suffixes is more than one character in length, and contains no vowel or 'Y', it is also assumed to be a suffix. The Name Standardization looks for the DR suffix immediately after the first comma, and for any suffix between two commas immediately after the Family Name. The portion of the name to the left of the comma, less any suffixes, is assumed to be the Family Name.

After STDNAME^XLFNAME accounts for all Suffixes, it looks at the portion of the name after the comma. It assumes that the first space-delimited piece is the Given Name. If any other pieces are left, the last one (rightmost) is assumed to be the Middle Name, and anything else is appended to the end of the Given Name.

If the name contains no comma, and the "F" flag is not passed, STDNAME^XLFNAME looks for suffixes at the very end of the name. The last space-delimited piece before any suffixes is assumed to be the Family Name. The first space-delimited piece is assumed to be the Given Name. If any other pieces are left, the last one (rightmost) is assumed to be the Middle Name, and anything else is appended to the end of the Given Name.

### Example

In this example, the MYNAME variable is set to the name to be standardized. The "C" flag indicates that the name components should be returned in the MYNAME array, and the "P" flag indicates that parenthetical text should be removed from the name. STDNAME^XLFNAME sets MYAUD to original name passed in and sets nodes in the MYAUD array to flag changes and possible problems.

```
>S MYNAME="KRNUSER,FIFTY A. B. 2ND (TEST)"
>D STDNAME^XLFNAME(.MYNAME,"CP",.MYAUD)

>ZW MYNAME
MYNAME=KRNUSER,FIFTY A B II
MYNAME("FAMILY")=KRNUSER
MYNAME("GIVEN")=FIFTY A.
MYNAME("MIDDLE")=B.
MYNAME("SUFFIX")=2ND

>ZW MYAUD
MYAUD=KRNUSER,FIFTY A. B. 2ND (TEST)
MYAUD("MIDDLE")=""
MYAUD("PERIOD")=""
MYAUD("SPACE")=""
MYAUD("STRIP")=""
```

STDNAME^XLFNAME returned the standard form of the name in MYNAME as KRNUSER,FIFTY A B II. It interpreted FIFTY A. as the given (first) name and B. as the middle name. Since this may not be correct, MYAUD("MIDDLE") is set. Periods were removed and spaces were removed to form the standard name, therefore MYAUD("PERIOD") and MYAUD("SPACE") were set. Finally, since the parenthetical text (TEST) was removed, MYAUD("STRIP") was set.

## DELCOMP^XLFNAME2(): Delete Name Components Entry

**Reference Type** Controlled Subscription

**Category** Name Standardization

**IA #** 3066

**Description** This API deletes an entry in the NAME COMPONENTS file (#20), and optionally, the value of the pointer in the source file that points to the name components entry.

**NOTE:** This API is designed to be used in the KILL logic for the MUMPS cross-reference mentioned previously in the UPDCOMP^XLFNAME2(): Update Name Components Entry API.

**Format**

```
DELCOMP^XLFNAME2(file,[.]record,field[,ptrfield])
```

**Input Parameters**

| | |
|---|---|
| file: | (required) The number of the file or Multiple (the "source file") that contains the name. |
| [.]record: | (required) The IENS or the Internal Entry Number array (that looks like the DA array) of the record in the source file that contains the name. |
| field: | (required) The number of the field in the source file that contains the name. |
| ptrfield: | (optional) The number of the pointer field in the source file that points to the NAME COMPONENTS file (#20). Only if this parameter is passed will the value of this pointer field be deleted. |

**Output** none Deletes record.

### Example

Suppose that you have a NAME COMPONENTS file (#20) entry that contains the components of a name stored in File #1000, Record #132, Field #.01. Pointer Field #1.1 of that File #1000 is a pointer to the NAME COMPONENTS file (#20). To delete the entry in the NAME COMPONENTS file (#20), and the value of the pointer field, you can do the following:

```
>D DELCOMP^XLFNAME(1000,132,.01,1.1)
```

# UPDCOMP^XLFNAME2(): Update Name Components Entry

**Reference Type** Controlled Subscription

**Category** Name Standardization

**IA #** 3066

**Description** This API updates an entry in the NAME COMPONENTS file (#20). Optionally, the pointer in the source file that points to the name components entry is also updated.

This API is designed to be used in the SET logic of a MUMPS cross-reference on the name field in a source file, to keep the name field and the associated name components in sync. For an example of its use, see the ANAME index in the INDEX file (#.11). The ANAME index is a MUMPS cross-reference on the .01 NAME field of the NEW PERSON file (#200). If an entry's NAME field is edited, the ANAME cross-reference updates the associated entry in the NAME COMPONENTS file (#20). Note that existing MUMPS cross-references on the NAME COMPONENTS file (#20) already exist to update the associated name field on the source file if the components are edited.

**Format**

```
UPDCOMP^XLFNAME2(file,[.]record,field,[.]name[,ptrfield]
[,ptrval])
```

**Input Parameters**

file: (required) The number of the file or Multiple (the "source file") that contains the name.

[.]record: (required) The IENS or the Internal Entry Number array (that looks like the DA array) of the record in the source file that contains the name.

field: (required) The number of the field in the source file that contains the name.

[.]name: (required) An array that contains the component parts of the name to store in the NAME COMPONENTS file (#20) entry:

NAME("FAMILY) = Family Name (required)
NAME("GIVEN") = Given Name(s) (optional)
NAME("MIDDLE") = Middle Name(s) (optional)
NAME("SUFFIX") = Suffix(es) (optional)
NAME("PREFIX") = Prefix (optional)
NAME("NOTES") = optional free text string

Alternatively, a name in standard format can be passed in the NAME input parameter. If the NAME input parameter has no descendents (that is, $D(NAME)=1), UPDCOMP^XLFNAME2 will make a call to the NAMECOMP^XLFNAME(): Component Parts from Standard Name API to build the NAME array for you.

| | | |
|---|---|---|
| | ptrfield: | (optional) The number of the pointer field in the source file that points to the NAME COMPONENTS file (#20). Only if this parameter is passed will the value of this pointer field be updated with the entry number of the record in the NAME COMPONENTS file (#20) that was added or edited. |
| | ptrval: | (optional) The current value of the pointer field specified by the PTRFIELD input parameter. This parameter can be used to save processing time. If both PTRFIELD and PTRVAL are passed, the pointer field will be updated only if this value is different from the entry number of the record in the NAME COMPONENTS file (#20) that was added or edited. |
| **Output** | returns: | Updated entry in the NAME COMPONENTS file (#20). |

**Example**

Suppose the .01 field of File #1000 contains a person's name, and the component parts of the name in entry 132 should be updated as follows:

Family (last) name: KRNUSER
Given (first) name: FIFTY HENRY
Middle name: A.
Suffix: JR.

Field #1.1 is defined as a pointer to the NAME COMPONENTS file (#20) and has a value of 42, the IEN of a record in the NAME COMPONENTS file (#20). To update the NAME COMPONENTS file (#20) with this name, you can do the following:

```
>S MYNAME("FAMILY")="KRNUSER"
>S MYNAME("GIVEN")="FIFTY HENRY"
>S MYNAME("MIDDLE")="A."
>S MYNAME("SUFFIX")="JR."

>D UPDCOMP^XLFNAME2(1000,132,.01,.MYNAME,1.1,42)
```

If there is an entry in the NAME COMPONENTS file (#20) that corresponds to File #1000, Field #.01, IEN #132, that entry is updated with the name components passed in the MYNAME array. Otherwise, a new entry is added to the name components with this information.

If the entry in the name components that was updated or added is record #42, no change is made to the value of the pointer field #1.1, since 42 was passed in the 6th parameter.

MUMPS cross-references on the NAME COMPONENTS file (#20) updates the name in the Field #.01 of File #1000 to "KRNUSER,FIFTY HENRY A JR" if it doesn't already contain that name.

# 17. Operating System Interface: Programmer Tools

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 can 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.

## Application Program Interface (API)

Several APIs are available for programmers to work with the operating system. These APIs are described below.

### ^%ZOSF(): Operating System-dependent Logic Global

The ^%ZOSF global holds operating system-dependent logic so that application programs can 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 Table 17-1, however, are reference values. For example, to WRITE the operating system, use:

```
W ^%ZOSF("OS")
```

The nodes prefaced with two asterisks in Table 17-1 should be used with the DO command, as in the following:

```
>D @^%ZOSF("ERRTN")
```

**Table Key:**

***** indicates those nodes that hold reference values.

****** indicates those nodes that are invoked with a DO statement (D).

| Node | Description |
|---|---|
| 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. |

| Node | Description |
|---|---|
| EOT | Returns 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):<br>`>D @^%ZOSF("ERRTN")`<br>To initially set the trap:<br>`>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:<br>`>W @%MT("BS")`<br>The full list of functions are:<br>"BS"—Back Space<br>"FS"—Forward Space<br>"WTM"—WRITE Tape Mark<br>"WB"—WRITE Block<br>"REW"—Rewind<br>"RB"—READ Block<br>"REL"—READ Label<br>"WHL"—WRITE HDR Label<br>"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 online. |
| 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. |

| Node | Description |
|---|---|
| *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 $I width to X characters. If X=0, then the line in set to no wrap. |
| RSEL | Returns the user's selection of routines as follows:<br>`^UTILITY($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:<br>`>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 can 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. |

**Table 17-1. ^%ZOSF: Global nodes**

## $$ACTJ^%ZOSV: Number of Active Jobs

**Reference Type** Supported

**Category** Operating System Interface

**IA #** 10097

**Description** This extrinsic function returns the number of active jobs in the scope of this process. It is the same as ^%ZOSF("ACTJ").

**Format** `$$ACTJ^%ZOSV`

**Input Parameters** none

**Output** returns: Returns the number of active jobs.

**Example**

## $$AVJ^%ZOSV: Number of Available Jobs

**Reference Type** Supported

**Category** Operating System Interface

**IA #** 10097

**Description** This extrinsic function returns a best effort on the number of available jobs (i.e., number of new jobs that could be started). It is the same as ^%ZOSF("AVJ").

**Format** `$$AVJ^%ZOSV`

**Input Parameters** none

**Output** returns: Returns the number of available jobs.

**Example**

# DOLRO^%ZOSV: Display Local Variables

**Reference Type** Controlled Subscription

**Category** Operating System Interface

**IA #** 3883

**Description** This API is used to save all local variables. It stores all local variables in the global storage location specified by the "X" input variable.

**Format** `DOLRO^%ZOSV`

Make sure to perform the following steps before calling DOLRO^%ZOSV:

- NEW the input variable.
- Set the input variable.
- Call DOLRO^%ZOSV.

If you do not follow these steps, the input variable could unintentionally assume the value of the variables of the current running task.

| | | |
|---|---|---|
| **Input Variable** | X: | (required) When this variable is set to an open global reference, (e.g., '^XTMP("ZZHL",25,'), all local variables existent when DOLRO^%ZOSV is called are stored in the location specified by the open global reference. These variables, now stored in the X-specified global location, can be listed and examined by application developers. |
| **Output** | returns: | Local variables are stored in the global specified by the X input variable. |

**Example**

```
>S X="^%ZTSK(ZTSKm.3," D DOLRO^%ZOSV
```

## GETENV^%ZOSV: Current System Information

**Reference Type** Supported

**Category** Operating System Interface

**IA #** 10097

**Description** This API returns environment information about the current system.

**Format** `GETENV^%ZOSV`

**Input** none

**Output Variable** Y: Returns a string in the following format:

```
UCI^VOL/DIR^NODE^BOX LOOKUP
```

**Example**

## $$LGR^%ZOSV: Last Global Reference

**Reference Type** Supported

**Category** Operating System Interface

**IA #** 10097

**Description** This extrinsic function returns the last global reference.

**Format** `$$LGR^%ZOSV`

**Input Parameters** none

**Output** returns: Returns the string set to the last full global reference.

**Example**

```
>S X=$$LGR^%ZOSV
```

## LOGRSRC^%ZOSV(): Record Resource Usage (RUM)

**Reference Type** Supported

**Category** Operating System Interface

**IA #** 10097

**Description** This API records resource usage in ^XTMP("KMPR" via the Resource Usage Monitor (RUM) software.

**Format** `LOGRSRC^%ZOSV(opt,type,status)`

**Input Parameters**

opt: (required) Name of Option, Protocol, Remote Procedure Call (RPC) or Health Level Seven (HL7). This is a Free Text parameter.

type: (required) Type of option:

- 0—Option
- 1—Protocol
- 2—Remote Procedure Call (RPC)
- 3—Health Level Seven (HL7)

status: (optional) Reserved for future use.

**Output**

returns: This API saves RUM-related data for each option/type into a file. This file is then downloaded weekly to the Capacity Planning National Database. The data is then available to all sites via the Capacity Planning Service Web site located at the following Web address:

http://vista.med.va.gov/capman/

**Example**

# $$OS^%ZOSV: Get Operating System Information

**Reference Type** Supported

**Category** Operating System Interface

**IA #** 3522

**Description** This extrinsic function returns the underlying operating system (e.g., VMS on OpenVMS, NT on Windows, Unix on Linux). It is only available under Caché/OpenVMS M systems.

**Format** `$$OS^%ZOSV`

**Input Parameters** none

**Output** returns: Returns the underlying operating system information (e.g., VMS on OpenVMS, NT on Windows, Unix on Linux).

**Example**

```
I ^%ZOSF("OS")["OpenM" S Y=$$OS^%ZOSV
```

## SETENV^%ZOSV: Set VMS Process Name (Caché/OpenVMS Systems)

**Reference Type** Supported

**Category** Operating System Interface

**IA #** 10097

**Description** This API sets the VMS process name. It only has meaning on Caché/OpenVMS systems, otherwise it just quits.

**Format** `SETENV^%ZOSV`

Make sure to perform the following steps before calling SETENV^%ZOSV:

- NEW the input variable.
- Set the input variable.
- Call SETENV^%ZOSV.

If you do not follow these steps, the input variable could unintentionally assume the value of the variables of the current running task.

**Input Variable** X: (required) This is a 1-15 character name to be given to the process at the VMS level.

**Output** none

**Example**

## SETNM^%ZOSV(): Set VMS Process Name (Caché/OpenVMS Systems)

**Reference Type** Supported

**Category** Operating System Interface

**IA #** 10097

**Description** This API sets the VMS process name. It only has meaning on Caché/OpenVMS systems, otherwise it just quits. It is the parameter passing version of the SETENV^%ZOSV: Set VMS Process Name (Caché/OpenVMS Systems) API.

**Format** `SETNM^%ZOSV(name)`

**Input Parameter** name: (required) This is a 1-15 character name to be given to the process at the VMS level.

**Output** none

**Example**

# T0^%ZOSV: Start RT Measure (Obsolete)

**NOTE:** This API is obsolete.

**Reference Type** Supported

**Category** Operating System Interface

**IA #** 10097

**Description** This API starts RT Measure. The Kernel site parameter flag to enable RT logging must be set for the volume set. The setting of this flag defines the XRTL variable. The call to this API should, thus, include a check for the existence of XRTL, such as the following:

```
>D:$D(XRTL) T0^%ZOSV
```

This API should be placed just before a process that may take a few seconds before the system responds with another prompt. If the minimal pause is at least a half second, there is enough variability to notice changes as the load on the system is increased or decreased. There should be no terminal IOs between the T0 start point and the T1 stop point.

**REF:** For more information on RT measure, please refer to the Resource Usage Monitor (RUM) documentation, located on the VDL at the following Web address:

http://www.va.gov/vdl/Infrastructure.asp?appID=130

**Format** `T0^%ZOSV`

**Input Variables** none

**Output Variables** XRT0: Output variable (start time).

The T0 call sets the XRT0 variable to the start time. To discard a sample, the XRT0 variable should be KILLed. Such a KILL would be appropriate if there is an exit path between the T0 and T1 check points that is circuitous or otherwise irrelevant to the normal execution of the code in question.

**NOTE:** On Caché systems, it only records to the nearest second.

**Example**

## T1^%ZOSV: Stop RT Measure (Obsolete)

**NOTE:** This API is obsolete.

**Reference Type** Supported

**Category** Operating System Interface

**IA #** 10097

**Description** This API stops RT Measure. This API logs the elapsed time into the ^%ZRTL global. The API should include a check for the existence of the XRT0 variable to confirm that the start time is available.

**REF:** For more information on RT measure, please refer to the Resource Usage Monitor (RUM) documentation, located on the VDL at the following Web address:

http://www.va.gov/vdl/Infrastructure.asp?appID=130

**Format** `T1^%ZOSV`

**Input Variables** XRTN (required) Routine name.

The XRTN variable is normally set to the name of the routine being monitored via the command:

```
>S XRTN=$T(+0)
```

To log more than one stop point in the same routine, a number or other characters can be concatenated (e.g., XRTN_1) so that a separate entry is made in the ^%ZRTL global, since the global is subscripted by routine name:

```
>S:$D(XRT0) XRTN=$T(+0) D:$D(XRT0) T1^%ZOSV
```

**Output** returns: Logs elapsed time into the ^%ZRTL global.

**Example**

# $$VERSION^%ZOSV(): Get OS Version Number or Name

**Reference Type** Supported

**Category** Operating System Interface

**IA #**

**Description** This extrinsic function returns the operating system version number or name.

**Format** `$$VERSION^%ZOSV([flag])`

**Input Parameters**

flag: (optional) If you pass a value of 1, the operating system name is returned instead of the version number.

**NOTE:** The name is as defined by the vendor and doesn't necessarily correspond with the OS name stored in ^%ZOSF("OS").

**Output**

returns: Returns the operating system version number or name, depending on the (optional) flag input parameter.

### Example 1

```
>W $$VERSION^%ZOSV(1)

Cache for OpenVMS/ALPHA V7.x (Alpha)
```

### Example 2

```
>W $$VERSION^%ZOSV

4.1.16
```

# 18. Security Keys: 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 the SECURITY KEY file (#19.1).

## Person Lookup

If a key is flagged for Person Lookup, a cross-reference on the NEW PERSON 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 can 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
```

# Application Program Interface (API)

Several APIs are available for programmers to work with security keys. These APIs are described below.

## DEL^XPDKEY(): Delete Security Key

**Reference Type** Supported

**Category** Security Keys

**IA #** 1367

**Description** This API deletes a security key from the SECURITY KEY file (#19.1). All necessary indexing is performed to maintain the ^XUSEC global. The security key is removed from all holders in the NEW PERSON file (#200).

**Format** `DEL^XPDKEY(key_name)`

**Input Parameters** key_name: (required) The name of the security key to delete.

**Output** none

**Example**

```
>D DEL^XPDKEY(key_name)
```

## $$LKUP^XPDKEY(): Look Up Security Key Value

**Reference Type** Supported

**Category** Security Keys

**IA #** 1367

**Description** This extrinsic function looks up a security key by name or by Internal Entry Number (IEN) value. It returns the security key:

- Name—If called with a security key number.
- IEN—If called with a security key name.

**Format** `$$LKUP^XPDKEY(key_value)`

**Input Parameters** key_value: (required) The name or IEN of the security key in question.

**Output** returns: Returns the security key:

Name—If called with a security key number.

IEN—If called with a security key name.

**Example**

```
>S value=$$LKUP^XPDKEY(key_value)
```

## $$RENAME^XPDKEY(): Rename Security Key

**Reference Type** Supported

**Category** Security Keys

**IA #** 1367

**Description** This extrinsic function renames a security key. All necessary indexing is performed to maintain the ^XUSEC global.

**Format** `$$RENAME^XPDKEY(oldname,newname)`

**Input Parameters**

| | |
|---|---|
| oldname: | (required) Name of security key to be renamed. |
| newname: | (required) New name for security key. |

**Output**

| | |
|---|---|
| returns: | Returns:<br>1—Success.<br>0—Failure. |

**Example**

## OWNSKEY^XUSRB(): Verify Security Keys Assigned to a User

**Reference Type** Supported

**Category** Security Keys

**IA #** 3277

**Description** This API is used by the XUS KEY CHECK RPC. This API can be used to verify if a user has a specified security key assigned. The calling routine sends one or a reference to a subscripted array and the API returns a subscripted array with the following possible values:

- 1—User owns key.
- 0—Key not found.

The DUZ variable should be defined before calling this API.

(This was developed as a Broker RPC and all RPCs have as the first parameter the return/output parameter.)

**Format** `OWNSKEY^XUSRB(ret,list[,ien])`

**Input Parameters**

ret: (required) Name of the subscripted return array. In every API that is used as an RPC the first parameter is the return array.

list: (required) A single value or an input subscripted array of security keys to be evaluated.

ien: (optional) The DUZ of a user for whom you want to check if he/she holds security keys.

**Output Parameter**

ret(): Returns a subscripted output array of the input value/subscripted array (i.e. list) with the following possible values shown:

1—User owns key.

0—Key *not* found.

**Example 1**

In the following example, the return array is named "ZZ" and the single security key to be checked is the XUPROG security key:

```
>K ZZ D OWNSKEY^XUSRB(.ZZ,"XUPROG") ZW ZZ
ZZ(0)=1
```

### Example 2

In the following example, the return subscripted array is named "ZZ" and the input array of security keys to be checked is named "LST":

```
>K LST S LST(1)="XUPROG",LST(2)="XUMGR",LST(3)="ABC"
>K ZZ D OWNSKEY^XUSRB(.ZZ,.LST) ZW ZZ
ZZ(1)=1
ZZ(2)=1
ZZ(3)=0
```

# 19. Server Options: Programmer Tools

## Tools for Processing Server Requests

When a server option 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 option can call custom programs via the following fields:

- ENTRY ACTION
- HEADER
- ROUTINE
- EXIT ACTION

**REF:** For more information on server options, please refer to Chapter 11 in the *Kernel Systems Manual*.

**REF:** For more information on the programmer API for processing server requests, please refer to the *MailMan Programmer Manual*.

## Key Variables When a Server Option is Running

There are key variables that are set up when a server option is running. You can reference these key variables during any routine run by the server option's ENTRY ACTION, HEADER, ROUTINE, and EXIT ACTION fields. The key variables for server options 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 Request Bulletin or Mailman Reply

Server options 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.
```

**Figure 19-1. XQSERVER: Default bulletin**

You can use the same method to append text to MailMan messages.

## Customizing a Server Request Bulletin

Please note that the first six data elements in a server request 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 of the server request.
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:** XQSERVER has a line of text in it that says:

This is the server request bulletin XQSERVER

To avoid confusion, you should edit this line using the Bulletin Edit option to reflect the name of the new bulletin.

# 20. Signon/Security: Programmer Tools

Kernel's Signon/Security module sets up a standard VistA programming environment as a foundation for software applications. Once a signon session has been created, applications can assume that system-wide variables exist for common reference. For example, key variables defined via Signon/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 APIs and cannot be used in software application routines. These utilities do allow programmers to simulate ordinary user signon and yet work from programmer mode to test code and diagnose errors.

### ^XUP: Programmer Signon

The ^XUP routine can be called as a quick way to enter Kernel and set up a standard environment:

```
>D ^XUP: Programmer Signon
```

It does the following:

- Sets up DT.
- Calls ^%ZIS.
- Prompts for Access code if DUZ is zero or undefined.
- KILLs and rebuilds ^XUTL("XQ",$J).
- KILLs ^UTILITY($J).
- 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.

### ^XUS: User Signon: No Error Trapping

^XUS determines whether access to the computer is allowed, and then sets up the user with the proper environment:

```
>D ^XUS
```

This routine can be called to establish the signon environment. A recommended alternative for programmers is to call ^XUP, which establishes signon 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.

## H^XUS: Programmer Halt

The following is an obsolete utility:

```
>D H^XUS
```

It simply transfers control to ^XUSCLEAN.

## ^XUSCLEAN: Programmer Halt

Programmers are advised to call the ^XUSCLEAN routine when signing off:

```
>D ^XUSCLEAN
```

It is the same code that Kernel uses when a user signs off or restarts. It notes the signoff time in the SIGN-ON LOG file (#3.081) and KILLs the $J nodes in ^XUTL and ^UTILITY. It then performs a normal halt.

## ^ZU: User Signon

The ZU routine sets the error trap and then calls ^XUS:

```
>D ^ZU
```

User signons should be tied to ^ZU.

# XU USER SIGN-ON Option

Some software applications asked for the means to execute an action at user signon, but not through the alert system. Kernel provides the XU USER SIGN-ON option that software applications can attach to and perform software application-specific tasks on user signon.

## XU USER SIGN-ON: Package-specific Signon Actions

Kernel V. 8.0 introduced a method to support software application-specific signon actions. Kernel exports an extended-action option called XU USER SIGN-ON. Packages that want Kernel to execute a software application-specific user signon 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 software application-specific user signon 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 signon, Kernel executes the XU USER SIGN-ON option, which in turn executes any options that software applications 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 software application-specific signon actions have completed. Also, you should not perform any tasks requiring interaction in an action attached to the XU USER SIGN-ON option.

The DUZ variable will be defined at the time the signon actions are executed; DUZ is set as it normally is to the person's Internal Entry Number (IEN) in the NEW PERSON file (#200).

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

- Alert the user to a software application status.
- Issue a reminder.
- Notify the software application of the signon of a software application user.

**Example**

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

```
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
```

**Figure 20-1. ZZTALK: Protocol**

# XU USER TERMINATE Option

Kernel V. 8.0 introduced a method to support software application-specific user termination actions. Kernel V. 8.0 exports an extended-action option called XU USER TERMINATE. Packages that want Kernel to execute a software application-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 software applications to have Kernel execute a software application-specific routine when Kernel terminated a user. The method was for the software application to have a routine tag and name in fields 200.1 (USER TERMINATE TAG) and 200.2 (USER TERMINATE ROUTINE) of the software application's PACKAGE file (#9.4) entry. When Kernel V. 7.1 terminated a user, it executed the TAG^ROUTINE API stored in these fields, if any.

Kernel V. 8.0 continues to execute the API, if any, stored in a software application's PACKAGE file (#9.4) entry. However, Kernel V. 8.0 will be the last version to support that method of software application-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 software application-specific user termination routine. To attach it to the XU USER TERMINATE option, do the following:

1. Export your option with a KIDS action of SEND.
2. Export the XU USER TERMINATE option with a KIDS action of USE AS LINK FOR MENU ITEMS.

Kernel defines the XUIFN variable at the time your action executes; it is defined as the Internal Entry Number (IEN) in the NEW PERSON file (#200) 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 as follows:

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

# Application Program Interface (API)

Several APIs are available for programmers to work with signon/security. These APIs are described below.

## $$GET^XUPARAM(): Get Parameters

**Reference Type** Supported

**Category** Signon/Security

**IA #** 2542

**Description** This extrinsic function gets simple parameters from the KERNEL PARAMETERS file (#8989.2) that the site can edit.

**Format** `$$GET^XUPARAM(parameter_name[,style])`

**Input Parameters**

| | |
|---|---|
| parameter_name: | (required) This is the namespaced name of the parameter to look up in the KERNEL PARAMETERS file (#8989.2) and return the REPLACEMENT value or DEFAULT. |
| style: | (optional) This input parameter controls the return value if the REPLACEMENT value or DEFAULT are empty. |

**Output**

| | |
|---|---|
| returns: | Returns the REPLACEMENT value or DEFAULT. |

**Example**

# $$KSP^XUPARAM(): Return Kernel Site Parameter

**Reference Type** Supported

**Category** Signon/Security

**IA #** 2541

**Description** This extrinsic function retrieves a Kernel site parameter. The following parameters are currently supported:

- INST: Internal Entry Number (IEN) of the site's institution, in the site's INSTITUTION file (#4).
- SPOOL DOC: MAX SPOOL DOCUMENTS PER USER (internal value) from the site's KERNEL SYSTEM PARAMETERS file (#8989.3).
- SPOOL LIFE: MAX SPOOL DOCUMENT LIFE-SPAN (internal value) from the site's KERNEL SYSTEM PARAMETERS file (#8989.3).
- SPOOL LINE: MAX SPOOL LINES PER USER (internal value) from site's KERNEL SYSTEM PARAMETERS file (#8989.3).
- WHERE: Site's domain name (FREE TEXT value), from the site's DOMAIN file (#4.2).

**Format** `$$KSP^XUPARAM(param)`

**Input Parameters** param: (required) Site parameter to retrieve. Currently, the following values for param are supported:

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

**Output** returns: Returns the requested site parameter value.

### Example 1

```
>S A6ASITE=$$KSP^XUPARAM("WHERE")
```

### Example 2

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

## $$LKUP^XUPARAM(): Look Up Parameters

**Reference Type** Supported

**Category** Signon/Security

**IA #** 2542

**Description** This extrinsic function looks up simple parameters from the KERNEL PARAMETERS file (#8989.2) that the site can edit.

**Format** `$$LKUP^XUPARAM(parameter_name[,style])`

**Input Parameters**

| | |
|---|---|
| parameter_name: | (required) This is the namespaced name of the parameter to look up in the KERNEL PARAMETERS file (#8989.2) and return the REPLACEMENT value or DEFAULT. |
| style: | (optional) This input parameter controls the return value if the REPLACEMENT value or DEFAULT are empty. |

**Output** returns: Returns the REPLACEMENT value or DEFAULT.

**Example**

## SET^XUPARAM(): Set Parameters

**Reference Type** Supported

**Category** Signon/Security

**IA #** 2542

**Description** This API sets simple parameters in the KERNEL PARAMETERS file (#8989.2).

**Format** `SET^XUPARAM(parameter_name[,style])`

**Input Parameters**

| | |
|---|---|
| parameter_name: | (required) This is the namespaced name of the parameter to set in the KERNEL PARAMETERS file (#8989.2). |
| style: | (optional) This input parameter controls the return value if the REPLACEMENT value or DEFAULT are empty. |

**Output** none

**Example**

## $$PROD^XUPROD(): Production Vs. Test Account

**Reference Type** Supported

**Category** Signon/Security

**IA #**

**Description** This API was released with Kernel Patch XU*8.0*284. It is called by applications to check and see if the application is running in a Production or a Test account.

The Ask if Production Account option [XU SID ASK] on the Kernel Management Menu [XUKERNEL], asks if the current account is the Production account. It returns the following values:

- True (1 or non-zero)—If the answer is "YES," the account is the Production account, so the current system ID (SID) is set as the Production SID.
- False (zero)—If the answer is "NO," the account is *not* the Production account, so a fake value is stored.

The Startup PROD check option [XU SID STARTUP] can be scheduled for startup so that when TaskMan starts the SID is checked. The first check each day gets the current SID and compares it with the stored SID to see if they match.

**Format**

```
$$PROD^XUPROD([force])
```

**Input Parameters**

force: (optional) The parameter value of 1 allows an application to force a full test.

**Output**

returns: Returns a Boolean value:

True (1 or non-zero)—Production account, current SID is set as the Production SID.

False (zero)—Test account.

**Example**

## H^XUS: Programmer Halt

**Reference Type** Supported

**Category** Signon/Security

**IA #** 10044

**Description** This API is the programmer HALT.

**Format** `H^XUS`

**Input Parameters** none

**Output** none

**Example**

## SET^XUS1A(): Output Message During Signon

**Reference Type** Supported

**Category** Signon/Security

**IA #** 3057

**Description** This API is used to perform any output during a software application-specific actions executed at signon. This function should *only* be used by action-type options attached to and executed by Kernel's XU USER SIGN-ON extended action.

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

**Format** `SET^XUS1A(string)`

**Input Parameters** string: (required) 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

**Details**

As of Kernel V. 8.0, software applications 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 signon.

**REF:** For more information on software application-specific action executed at signon, please refer to the "XU USER SIGN-ON: Package-specific Signon Actions" topic in this chapter.

**Example**

## AVHLPTXT^XUS2: Get Help Text

| | | |
|---|---|---|
| **Reference Type** | Controlled Subscription | |
| **Category** | Signon/Security | |
| **IA #** | 4057 | |
| **Description** | This API retrieves help text to display to the user when they change their Verify code. | |
| **Format** | `AVHLPTXT^XUS2` | |
| **Input Parameters** | none | |
| **Output** | returns: | Returns the help text for a user to use when entering a new Verify code. |

**Example**

# $$CREATE^XUSAP: Create Application Proxy User

**Reference Type** Controlled Subscription

**Category** Signon/Security

**IA #** 4677

**Description** Released with Kernel Patch XU*8.0*361, this extrinsic function is a non-interactive API to create an Application Proxy User to support J2EE middle-tier applications. The Application Proxy User represents an application and *not* an end-user.

The Application Proxy User is a special category of user account that gets created in the NEW PERSON file (#200) and can execute authorized RPCs. The Application Proxy User created *must* adhere to the following criteria:

- The name added to the NEW PERSON file (#200) *must* be unique.
- It *must* have a user class of "Application Proxy," as defined in the USER CLASS file (#201) and pointed to by the USER CLASS field (#9.5) in the NEW PERSON file (#200).
- It *must not* have an Access or Verify code assigned to it.
- It *must not* have a Primary menu assigned to it.
- It *must* have one or more Secondary menu options assigned to it.
- The RPCs that the menu options reference *must* have the APP PROXY ALLOWED field (#.11) in the REMOTE PROCEDURE file (#8994) set to "YES."

**Format**

```
$$CREATE^XUSAP(proxyusername[,filemanaccesscode][,options])
```

**Input Parameters**

proxyusername: (required) This is the name of the Application Proxy User. This name *must* be unique and should be namespaced.

filemanaccesscode: (optional) This is the VA FileMan Access code. It *cannot* be an at-sign ("**@**").

**REF:** For more information, please refer to the *VA FileMan Advanced User Manual*.

options: (optional) This is the name of a single option name (e.g., XUS FATKAAT PROXY LOGON) or an array of options, such as XUOPT("XMUSER")=1. Applications can only access the Remote Procedure Calls (RPCs) contained in the options provided in this input parameter. RPCs are tied to "B"-type options.

**Output**

returns: Returns:

- IEN of entry created in NEW PERSON file (#200)—Successful; writes new Application Proxy User to the NEW PERSON file (#200).
- "0^Name In Use"—Unsuccessful; Application Proxy User of that name already exists in the the NEW PERSON file (#200).
- -1—Unsuccessful; could not create Application Proxy User OR error in call to UPDATE^DIE.

  **NOTE:** For more information on the UPDATE^DIE-related error , users should check ^TMP("DIERR",$J).

**Example**

The following example shows a *successful* creation of an Application Proxy User:

```
>IF $$CREATE^XUSAP("FATKAAT,PROXY","","XUS FATKAAT PROXY LOGON")>0 W !,"Proxy Created"

Proxy Created
```

# KILL^XUSCLEAN: Clear all but Kernel Variables

**Reference Type** Supported

**Category** Signon/Security

**IA #** 10052

**Description** This API clears the partition of all but key variables essential to 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 software-wide variables. For example, a user might enter the top-level menu for a software application, 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 software application, 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 software-wide variable.

KILL^XUSCLEAN now provides a way for sites and developers to work around this problem. For any menu-type option, the PROTECTED VARIABLES field in the OPTION file (#19) 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 descendent from the protected menu, the variables are protected until the menu subtree is exited.

So, for example, to protect a software-wide variable for an entire software application, you can enter that variable in the PROTECTED VARIABLES field for the top-level menu in the software application. As long as a user doesn't exit the top-level menu of the software application's menu tree, the software-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.

**Format** `KILL^XUSCLEAN`

**Input Parameters** none

**Output** none

**Example**

## $$ADD^XUSERNEW(): Add New Users

**Reference Type** Supported

**Category** Signon/Security

**IA #** 10053

**Description** This extrinsic function adds new entries to the NEW PERSON file (#200). It was modified with Kernel Patch XU*8.0*134. After prompting for the user's name, it parses the input into its component parts, and then prompts for each name component separately, presenting the parsed input as defaults. It then prompts for the default identifiers for the NEW PERSON file (#200) entry in the following order:

1. INITIAL (#1)
2. SSN (#9)
3. SEX (#4)

If the user of this function has the XUSPF200 security key, entry of the SSN is not required. The default identifiers can be locally modified by modifying the NEW PERSON IDENTIFIERS field in the KERNEL SYSTEM PARAMETERS file (#8989.3).

To prompt for additional fields during this call, you pass a DR string containing the field(s) for which you wish to prompt 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 (#200), and **-1** will be returned.

**Format**

```
$$ADD^XUSERNEW([dr_string][,keys])
```

**Input Parameters**

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.

**REF:** For information about DIC, please refer to the VA FileMan documentation.

keys: (optional) A comma-delimited string of keys to assign to the newly created user.

**Output** returns: Returns a value similar in format to the value of "Y" returned from a standard DIC call:

-1—User neither existed nor could be added.

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.

N^S^1—N and S are defined as above, and the 1 indicates the user has just been added to the file.

**REF:** For information about DIC, please refer to the VA FileMan documentation.

### Example 1

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

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

Enter NEW PERSON's name (Family,Given Middle Suffix): KRNUSER,TWO E
 Are you adding 'KRNUSER,TWO E' as a new NEW PERSON (the 1602ND)? No// Y <Enter>
(Yes)
Checking SOUNDEX for matches.
No matches found.
Name components.
FAMILY (LAST) NAME: KRNUSER// <Enter>
GIVEN (FIRST) NAME: TWO// <Enter>
MIDDLE NAME: E// <Enter>
SUFFIX: <Enter>
Now for the Identifiers.
INITIAL: TEK
SSN: 000222222
SEX: M <Enter>  MALE
>W X
1000118^KRNUSER,TWO E^1
>
```

**Figure 20-2. $$ADD^XUSERNEW: Example of adding a new user**

### Example 2

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

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

**Example 3**

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

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

## $$CHECKAV^XUSRB(): Check Access/Verify Codes

**Reference Type** Controlled Subscription

**Category** Signon/Security

**IA #** 2882

**Description** This extrinsic function checks an Access/Verify code pair (delimited by a semi-colon) and returns whether or not it is a valid pair.

**Format** `$$CHECKAV^XUSRB(access_verify)`

**Input Parameters** access_verify: (required) This is a string containing the Access and Verify code pair delimited by a semi-colon (i.e., Access code;Verify code).

**Output** returns: Returns:

Internal Entry Number (IEN)—Codes are OK.

Zero (0)—Codes are *not* OK.

**Example**

```
>S X=$CHECKAV^XUSRB(<string>)
```

String = Access code;Verify code

## CVC^XUSRB: VistALink—Change User's Verify Code

**Reference Type** Controlled Subscription

**Category** Signon/Security

**IA #** 4054

**Description** This API changes a VistALink user's Verify code.

**Format** `CVC^XUSRB`

**Input Parameters** none

**Output** duz: If DUZ is defined, we consider the "change verify code" operation to have been successful.

**Example**

## $$INHIBIT^XUSRB: Check if Logons Inhibited

**Reference Type** Supported

**Category** Signon/Security

**IA #** 3277

**Description** This extrinsic function is used to check if logons have been inhibited.

**Format** `$$INHIBIT^XUSRB`

**Input Parameters** none

**Output** none

**Example**

## INTRO^XUSRB: VistALink—Get Introductory Text

**Reference Type** Controlled Subscription

**Category** Signon/Security

**IA #** 4054

**Description** This API retrieve the introductory text from M to display in VistALink.

**Format** `INTRO^XUSRB`

**Input Parameters** none

**Output** returns: Returns each line in the introductory text as a value stored at the first subscript level node of the pass-by-reference first parameter to the method call. For example:

RETURN(0)=line 1 RETURN(1)=line 2 etc.

**Example**

## LOGOUT^XUSRB: VistALink—Log Out User From M

**Reference Type** Controlled Subscription

**Category** Signon/Security

**IA #** 4054

**Description** This API logs out a VistALink user from M.

**Format** `LOGOUT^XUSRB`

**Input Parameters** none

**Output** none

**Example**

## SETUP^XUSRB(): VistALink—Set Up User's Partition in M

**Reference Type** Controlled Subscription

**Category** Signon/Security

**IA #** 4054

**Description** This API sets up the envirnoment (i.e., RPC Broker orVistALink partition) in M prior to a GUI signon.

**Format** `SETUP^XUSRB(ret)`

**Input Parameters**

| | |
|---|---|
| ret: | (required) Name of the subscripted return array. In every API that is used as an RPC the first parameter is the return array. |

Make sure to perform the following steps before calling SETUP^XUSRB:

- NEW all of the following variables.
- Set the input variables you want to change.
- Call SETUP^XUSRB.

If you do not follow these steps, the input variables could unintentionally assume the value of the variables of the current running task.

**Input Variables**

| | |
|---|---|
| XWBTIP: | (required) The Internet Protocol (IP) address of the client workstation. |
| XWBCLMAN: | (optional) The client workstation name. |
| XWBVER: | (optional) This is the version of the RPC Broker software on the client workstation. |

**Output**

RET(): Returns a subscripted output array:

RET(0)—Server option name

RET(1)—Volume

RET(2)—UCI

RET(3)—Device

RET(4)—# Attempts

RET(5)—Skip signon-screen

RET(6)—Domain name

**Example**

## VALIDAV^XUSRB(): VistALink—Validate User Credentials

**Reference Type** Controlled Subscription

**Category** Signon/Security

**IA #** 4054

**Description** This API validates a VistALink user's credentials for signon to M.

**Format** `VALIDAV^XUSRB(credential)`

**Input Parameters** credential: (required) A credential (typically the encoded "Access code;Verify code" string) to use to attempt a signon for the current user.

**Output** returns: ;Return R(0)=DUZ, R(1)=(0=OK, 1,2...=Can't sign on for some reason)
; R(2)=verify needs changing, R(3)=Message, R(4)=0, R(5)=msg cnt, R(5+n)
; R(R(5)+6)=# div user must select from, R(R(5)+6+n)=div

**Example**

## $$DECRYP^XUSRB1(): Decrypt String

**Reference Type** Supported

**Category** Signon/Security

**IA #** 2241

**Description** This extrinsic function decrypts a string that was encrypted on a Client system. This function decrypts a string that has been encrypted using the Encrypt Delphi function supplied by the RPC Broker, returning the decrypted string.

**Format** `$$DECRYP^XUSRB1(encrypted_string)`

**Input Parameters** encrypted_string: (required) Encrypted string to be decrypted.

**Output** returns: Returns the decrypted string.

**Example**

## $$ENCRYP^XUSRB1(): Encrypt String

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Signon/Security | |
| **IA #** | 2240 | |
| **Description** | This extrinsic function encrypts a string before transport to a Client system, where it will be decrypted. This function performs encryption on the input string, returning the encrypted string. | |
| **Format** | `$$ENCRYP^XUSRB1(string)` | |
| **Input Parameters** | string: | (required) The input string to be encrypted. |
| **Output** | returns: | Returns the encrypted string. |

**Example**

# $$HANDLE^XUSRB4(): Return Unique Session ID String

**Reference Type** Supported

**Category** Signon/Security

**IA #** 4770

**Description** (This API is made available with Kernel Patch XU*8.0*395.) This extrinsic function returns a unique Caché cluster string for a VistA system for use by Health*e*Vet Desktop applications.

**Format** `$$HANDLE^XUSRB4("namespace"[,timetolive])`

**Input Parameters**

"namespace": (required) This input parameter should start with the VistA software namespace. In addition, users can add any additional application/software identifiers.

timetolive: (optional) This input parameter indicates the number of days that this handle will be available for use. Possible values range from 1 to 7. The default is 1. The ^XTMP global requires that the zero node hold the save through date. This value gets cleaned up via the XQ82 routine (i.e., Clean old Job Nodes in XUTL option [XQ XUTL $J NODES]).

**Output**

returns: Returns the unique Vista system Caché cluster string. The value generated includes the data entered in the namespace input parameter and $J and $H. If this value is already defined, a new value is generated.

### Example

In this example, we are creating a unique session ID for the RPC Broker namespace (i.e., "XWB"):

```
>S HDL=$$HANDLE^XUSRB4("XWB-CCOW")

>W HDL
XWB-CCOW928-57785_0
```

When checking the ^XTMP temporary global you would see:

```
^XTMP("XWB-CCOW928-57785_0",0) = 3050805^3050804
```

## ^XUVERIFY(): Verify Access and Verify Codes

The ^XUVERIFY API 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.

**Reference Type** Supported

**Category** Signon/Security

**IA #** 10051

**Description** This API can be used anytime in an application to check if the current user can enter the Access or Verify code that matches the currently signed-in user.

**Format** `^XUVERIFY(%,%duz)`

**Input Parameters**

%: (required) If % equals:

"A"—Check the Access code.

"V"—C heck the Verify code.

"AV"—Check both the Access and Verify code.

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

**Output**

returns: Returns the following values:

2—Failure (the incorrect code was entered).

1—Success (the correct code was entered).

0—A question mark was entered.

-1—An up-arrow was entered.

**Example**

## $$CHECKAV^XUVERIFY(): Check Access/Verify Codes

**Reference Type** Supported

**Category** Signon/Security

**IA #** 10051

**Description** This extrinsic function checks an Access/Verify code pair entered by the user (delimited by a semi-colon) and returns whether or not it is a valid pair.

**Format** `$$CHECKAV^XUVERIFY(access_verify)`

**Input Parameters** access_verify: (required) This is a string containing the Access and Verify code pair delimited by a semi-colon (i.e., Access code;Verify code).

**Output** returns: Returns:

Internal Entry Number (IEN)—Codes are OK.

Zero (0)—Codes are *not* OK.

**Example**

```
>S X=$CHECKAV^XUVERIFY(<string>)
```

String = Access code;Verify code

## WITNESS^XUVERIFY(): Return IEN of Users with A/V Codes & Security Keys

**Reference Type** Controlled Subscription

**Category** Signon/Security

**IA #**

**Description** This API returns the IEN of a user if he/she has an Access code, Verify code, and security keys.

**Format** `WITNESS^XUVERIFY(prefix,keys)`

**Input Parameters** prefix: (?????)

keys: (?????)

**Output** returns: Returns:

- IEN (successful)—The user has an Access code, Verify code, and security keys.
- 0 (failure)—The user does *not* have an Access code, Verify code, and security keys.

**Example**

## GETPEER^%ZOSV: VistALink—Get IP Address for Current Session

**Reference Type** Controlled Subscription

**Category** Signon/Security

**IA #** 4056

**Description** This API retrieves an IP address value for the current session, which is required as input (i.e., XWBTIP input variable) for the SETUP^XUSRB(): VistALink—Set Up User's Partition in M API. The VistALink security module calls this API.

**Format** `GETPEER^%ZOSV`

**Input Parameters** none

**Output** returns: Returns the Internet Protocol (IP) address of the current connected session to M.

**Example**

# 21. Spooling: 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 ztio input parameter 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.

**REF:** For the details of issuing form feeds, please refer to the "Form Feeds" topic in the "Special Device Issues" chapter ion this manual.

```
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
```

**Figure 21-1. Spooling: Sending output to the spooler (and pre-defining ZTIO)**

```
SAMPLE  ;SAMPLE ROUTINE
        ;
        S %ZIS="QM" D ^%ZIS G EXIT:POP
        I $D(IO("Q")) D  Q
        .S ZTRTN="DQ^SAMPLE",ZTDESC="Sample Test routine"
        .D ^%ZTLOAD D HOME^%ZIS K IO("Q") Q
DQ      U IO W !,"THIS IS YOUR REPORT"
        W !,"LINE 2"
        W !,"LINE 3"
        D ^%ZISC
EXIT    S:$D(ZTQUEUED) ZTREQ="@" K VAR1,VAR2,VAR3 Q
```

**Figure 21-2. Spooling: Allowing output to go the spooler (*without* pre-defining ZTIO)**

# Application Program Interface (API)

Several APIs are available for programmers to work with spooling. These APIs are described below.

## DSD^ZISPL: Delete Spool Data File Entry

**Reference Type** Controlled Subscription

**Category** Spooling

**IA #** 1092

**Description** This API Deletes SPOOL DATA file (#3.519) entry following transfer of data, to minimize consumption of data.

**Format** `DSD^ZISPL`

**Input Parameters** none

**Output** none

**Example**

## DSDOC^ZISPL: Delete Spool Document File Entry

**Reference Type** Controlled Subscription

**Category** Spooling

**IA #** 1092

**Description** This API deletes the SPOOL DOCUMENT file (#3.51) entry following transfer of data, to minimize consumption of disk space.

**Format** `DSDOC^ZISPL`

**Input Parameters** none

**Output** none

**Example**

# 22. TaskMan: 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.

**REF:** For more information on why and when to use TaskMan to perform queuing, please refer to the "TaskMan System Management: Overview" chapter in the *Kernel Systems Manual*.

## 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 (#19.2); TaskMan 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.
- Options that are scheduled (in which case, TaskMan itself acts as the queuer).

#### Calling ^%ZTLOAD to Create Tasks

One common way to create tasks is to call TaskMan's main API, ^%ZTLOAD. You can use ^%ZTLOAD interactively, or non-interactively.

**REF:** For more information on queuing tasks with ^%ZTLOAD, please refer to the ^%ZTLOAD: Queue a Task topic in this chapter.

### Calling EN^XUTMDEVQ to Create Tasks

The EN^XUTMDEVQ API 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, TaskMan itself (rather than application code) acts as the queuer. Site managers use TaskMan 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 TaskMan 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 TaskMan, 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 can 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 (#19) 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 (#19.2) 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 software application installation guide for all the queuable options, since options are usually set on their schedules shortly after installation.

## Tasks

This topic describes information about Tasks. It applies whether the queuer that queued the task was a call to ^%ZTLOAD, or TaskMan itself was running the task because it was scheduled in the OPTION SCHEDULING file (#19.2).

When you write a task, you create an API that TaskMan can call to perform the work. The Submanager calls the API 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 API, 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.
- The responsibilities of the tasks themselves.

### Key Variables and Environment When Task is Running

All VistA 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)
  ```

- **ZTRTN:** This variable is the API that TaskMan will DO to start the task.
- **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 API. 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 TaskMan's error log, and in the entry for your task in the TASKS file (#14.4).
- **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
- Device Handler
- Resource devices
- 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 can 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 can 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 (#14.4) 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 (#14.4) entries:

- ZTREQ output variable
- KILL^%ZTLOAD API

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:** 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, ensuring 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 API 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 |

**Table 22-1. TaskMan: ZTREQ piece and equivalent REQ^ZTLOAD variable**

All of these ^-pieces in ZTREQ are optional; only set the pieces that affect parameters you want to change. 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 can 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 API below. Remember, however, to include at least one caret ("^") 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 API (^%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 APIs (OPEN^%ZISUTL, USE^%ZISUTL, and CLOSE^%ZISUTL).

### Writing Two-step Tasks

A situation you should always consider is how to deal with jobs that will take a long time to gather data and then print a report of that data. If you write this as a *single* job that *both* gathers and prints data, any requested IO device that will eventually be used to print that data will sit idle for a long period of time. Thus, the IO device is unused and unavailable to any other tasks during that entire period of time it takes to gather the data for your report.

If you write the task to start without a device, and to call the ^%ZIS: Standard Device Call API to open the device when the report is ready, two different problems occur:

1. 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.
2. 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. However, some system managers discourage use of spoolers, because of the possibility for disk crashes resulting from users who send excessively large reports to the spooler.

Therefore, the best solution to this problem involves splitting the job into two separate tasks:

1. Gather—The first task runs without a device, gathers and generates the report data in the ^XTMP global, and schedules the second task (Print).
2. Print—The second task runs with the IO device and prints the report data generated by the first task (Gather).

In order to perform these two separate but associated tasks, Kernel provides the following APIs:

1. $$QQ^XUTMDEVQ(): Double Queue—Direct Queuing in a Single Call—This API creates the Gather and the Print tasks. The gather task is scheduled to run, while the print task is not scheduled.
2. $$REQQ^XUTMDEVQ(): Schedule Second Part of a Task—At the end of the Gather task, it invokes the $$REQQ^XUTMDEVQ API to schedule the Print task.

**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. TaskMan 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 (#14.8).
3. If the SYNC FLAG pair does not exist in the TASK SYNC FLAG file (#14.8), TaskMan creates an entry for the SYNC FLAG pair in the TASK SYNC FLAG file (#14.8) and starts the task.

   If, on the other hand, the SYNC FLAG pair already exists in the TASK SYNC FLAG file (#14.8), then any task requiring the same SYNC FLAG has to wait until the corresponding entry in the TASK SYNC FLAG file (#14.8) 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 zero (0) or not defined, TaskMan deletes the SYNC FLAG pair entry in the TASK SYNC FLAG file (#14.8). 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 (#14.8) 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 (#14.8), 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.

# Application Program Interface (API)

Several APIs are available for programmers to work with TaskMan. These APIs are described below.

## $$DEV^XUTMDEVQ(): Force Queueing—Ask for Device

**Reference Type** Supported

**Category** TaskMan

**IA #** 1519

**Description** (Added with Kernel Patch XU*8.0*275.) This extrinsic function encapsulates the logic to handle direct (forced) queuing in a single call and ask users for a device.

**Format**
```
$$DEV^XUTMDEVQ(ztrtn[,ztdesc][,%var][,%voth][,%zis][,iop]
[,%wr])
```

**Input Parameters**

| | |
|---|---|
| ztrtn: | (required) The API that TaskMan will DO to start the task (job). You can specify it as "LABEL^ROUTINE" or "^ROUTINE" or "ROUTINE". |
| ztdesc: | (optional) Task description, up to 200 characters describing the task, with the software application name at the front. Default to name of [tag]^routine. |
| %var: | (optional) ZTSAVE values for the task. Single value or passed by reference, this will be used to S ZTSAVE(). It can be a string of variable names separated by "**;**". Each **;**-piece will be used as a subscript in ZTSAVE. |
| %voth: | (optional) Passed by reference, %voth(sub)="" or explicit value sub—this is any other %ZTLOAD variable besides ZTRTN, ZTDESC, ZTIO, ZTSAVE. For example: `%VOTH("ZTDTH")=$H` |
| %zis: | (optional) Default value "MQ". Passed by reference, standard %zis variable array for calling the Device Handler. |
| iop: | (optional) The IOP variable as defined in Kernel's Device Handler. |
| %wr: | (optional) If %WR>0 then write text to the screen as to whether or not the queueing was successful. |

**Output** returns

Returns:

- 0—If run ztrtn without queuing.
- -1—If unsuccessful device call or failed the %ztload call.

### Example

This example is a job that consists of gathering information and then printing it. Assume that the gathering takes a few hours. We do not want the device that the user selects to be tied up for that time, so we divide the job into two tasks. The first task gathers the information, and the second task prints it. We use the $$DEV^XUTMDEVQ API to select the device and queue up the print task, and the $$NODEV^XUTMDEVQ(): Force Queueing—No Device Selection API to schedule the gather task. We use the REQ^%ZTLOAD: Requeue a Task API to schedule the print task when the gather task finishes.

**NOTE:** You could also use the $$REQQ^XUTMDEVQ(): Schedule Second Part of a Task API to schedule the print task.

```
ARHBQQ   ;SFVAMC/GB - Demo of 'gather' and 'print' in 2 tasks ;1/19/06  08:31
         ;;1.1
DEV      ;
         N ARH,ARHZTSK,X
         ;The user doesn't know it, but he's actually queuing the second task,
         ;the "print" portion of the job.  The only question the user will be
         ;asked is to select the device.
         S ARH("ZTDTH")="@" ;Don't schedule the task to run, we'll do it later.
         ;In the following, the "Q" sets IOP=Q, which forces queueing.
         S X=$$DEV^XUTMDEVQ("PRINT^ARHBQQ","ARHB Print",,.ARH,,"Q",1)
         W !,"X=",X
         Q:X<1
         N ARH
         ;Now queue the first task, the "gather" portion of the job.  The user
         ;won't be asked any questions.
         S ARHZTSK=X ; Save the ZTSK number of the "print" task.
         S ARH("ZTDTH")=$H ; Force the task to start now.
         ;To ask the user the start time, comment out the above line.
         S X=$$NODEV^XUTMDEVQ("GATHER^ARHBQQ","ARHB Gather","ARHZTSK",.ARH,1)
         W !,"X=",X
         Q
```

**Figure 22-1. $$DEV^XUTMDEVQ example—Sample code**

## EN^XUTMDEVQ(): Run a Task (Directly or Queued)

**Reference Type** Supported

**Category** TaskMan

**IA #** 1519

**Description** This API 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. Thus, EN^XUTMDEVQ provides a simple way to facilitate both queuing and running a job directly.

If the IOP parameter 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).

**Format** `EN^XUTMDEVQ(ztrtn,ztdesc,.ztsave[,.%zis])`

**Input Parameters**

ztrtn: (required) The API that TaskMan will DO to start the task. You can specify it as "LABEL^ROUTINE" or "^ROUTINE" or "ROUTINE".

ztdesc: (required) Task description, up to 200 characters describing the task, with the software application name at the front.

.ztsave: (required) Pass by reference. Set up this array in the same format as the ztsave input array is set up for the ^%ZTLOAD TaskMan API. The array you set up in ztsave is passed directly as ztsave to TaskMan 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 API. The array you set up in the %zis input 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.

retztsk: (optional) This is the return task number (i.e., ztsk). Put a number in this parameter, such that $G(retztsk), then ztsk will exist as an output variable. Otherwise, ztsk will *not* exist as an output variable.

**Output**

ztsk: If a number is entered in the retztsk input parameter, the task number assigned to a task is returned.

**Example**

```
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
```

**Figure 22-2. EN^XUTMDEVQ: Sample report**

# $$NODEV^XUTMDEVQ(): Force Queueing—No Device Selection

**Reference Type** Supported

**Category** TaskMan

**IA #** 1519

**Description** (Added with Kernel Patch XU*8.0*275.) This extrinsic function encapsulates the logic to handle direct (forced) queuing in a single call and does *not* ask users for a device.

**Format** `$$NODEV^XUTMDEVQ(ztrtn[,ztdesc][,%var][,%voth][,%zis][,iop][,%wr])`

**Input Parameters**

| | |
|---|---|
| ztrtn: | (required) The API that TaskMan will DO to start the task (job). You can specify it as "LABEL^ROUTINE" or "^ROUTINE" or "ROUTINE". |
| ztdesc: | (optional) Task description, up to 200 characters describing the task, with the software application name at the front. Default to name of [tag]^routine. |
| %var: | (optional) ZTSAVE values for the task. Single value or passed by reference, this will be used to S ZTSAVE(). It can be a string of variable names separated by "**;**". Each **;**-piece will be used as a subscript in ZTSAVE. |
| %voth: | (optional) Passed by reference, %voth(sub)="" or explicit value sub—this is any other %ZTLOAD variable besides ZTRTN, ZTDESC, ZTIO, ZTSAVE. For example: `%VOTH("ZTDTH")=$H` |
| %zis: | (optional) Default value "MQ". Passed by reference, standard %zis variable array for calling the Device Handler. |
| iop: | (optional) The IOP variable as defined in Kernel's Device Handler. |
| %wr: | (optional) If %WR>0 then write text to the screen as to whether or not the queueing was successful. |

**Output** returns

Returns:

- 0—If run ztrtn without queuing.
- -1—If unsuccessful device call or failed the %ztload call.

### Example

This example is a job that consists of gathering information and then printing it. Assume that the gathering takes a few hours. We do not want the device that the user selects to be tied up for that time, so we divide the job into two tasks. The first task gathers the information, and the second task prints it. We use the $$DEV^XUTMDEVQ(): Force Queueing—Ask for Device API to select the device and queue up

the print task, and the $$NODEV^XUTMDEVQ API to schedule the gather task. We use the REQ^%ZTLOAD: Requeue a Task API to schedule the print task when the gather task finishes.

**NOTE:** You could also use the $$REQQ^XUTMDEVQ(): Schedule Second Part of a Task API to schedule the print task.

```
ARHBQQ   ;SFVAMC/GB - Demo of 'gather' and 'print' in 2 tasks ;1/19/06  08:31
         ;;1.1
DEV      ;
         N ARH,ARHZTSK,X
         ;The user doesn't know it, but he's actually queuing the second task,
         ;the "print" portion of the job.  The only question the user will be
         ;asked is to select the device.
         S ARH("ZTDTH")="@" ;Don't schedule the task to run, we'll do it later.
         ;In the following, the "Q" sets IOP=Q, which forces queueing.
         S X=$$DEV^XUTMDEVQ("PRINT^ARHBQQ","ARHB Print",,.ARH,,"Q",1)
         W !,"X=",X
         Q:X<1
         N ARH
         ;Now queue the first task, the "gather" portion of the job.  The user
         ;won't be asked any questions.
         S ARHZTSK=X ; Save the ZTSK number of the "print" task.
         S ARH("ZTDTH")=$H ; Force the task to start now.
         ;To ask the user the start time, comment out the above line.
         S X=$$NODEV^XUTMDEVQ("GATHER^ARHBQQ","ARHB Gather","ARHZTSK",.ARH,1)
         W !,"X=",X
         Q
```

**Figure 22-3. $$NODEV^XUTMDEVQ example—Sample code**

# $$QQ^XUTMDEVQ(): Double Queue—Direct Queuing in a Single Call

**Reference Type** Supported

**Category** TaskMan

**IA #** 1519

**Description** (Added with Kernel Patch XU*8.0*275 and updated with Kernel Patch XU*8.0*389.) This extrinsic function encapsulates the logic to handle direct queuing in a single call.This extrinsic function does a double queuing:

- Queue up the second task to a device, but do *not* schedule the task in TaskMan.
- Queue up the first task to ZTIO="" and schedule it.

If it will take a long time to gather and print data, users should split the job into two tasks:

1. Gather Data—The first task gathers the data.
2. Print Data—The second task prints the data.

Separating the data gathering task from the data print task helps avoid unnecessarily tying up a printer while large amounts of data are gathered.

The task number of the second task (i.e., print data) is added to the saved variables with the name XUTMQQ. This makes it easier to schedule the second task when the first task (i.e., gather data) has finished.

To schedule the second task to run at the end of the first task, you must call the $$REQQ^XUTMDEVQ(): Schedule Second Part of a Task API.

**Format**

```
$$QQ^XUTMDEVQ(%rtn[,%desc][,%var1][,%voth1][,%zis][,iop][,%wr],
%rtn2[,%desc2][,%var2][,%voth2])
```

**Input Parameters**

%rtn: (required) First task that TaskMan will run, usually a search and build sorted data type process (i.e., gather data). The API that TaskMan will DO to start the task. You can specify it as "LABEL^ROUTINE" or "^ROUTINE" or "ROUTINE". [tag]^routine that TaskMan will run.

%desc: (optional) First task description, up to 200 characters describing the task, with the software application name at the front. Defaults to name of [tag]^routine.

%var1: (optional) ZTSAVE values for the first task. Single value or passed by reference, this will be used to SET ZTSAVE(). It can be a string of variable names separated by "**;**". Each **;**-piece will be used as a subscript in ZTSAVE.

%voth1: (optional) First task other parameter. Passed by reference, %voth(sub)="" or explicit value sub—this is any other %ZTLOAD variable besides ZTRTN, ZTDESC, ZTIO, ZTSAVE. For example:

```
%VOTH("ZTDTH")=$H
```

%zis: (optional) Default value "MQ". Passed by reference, standard %ZIS variable array for calling the Device Handler. Except for one difference, the second task of the job will be tasked to this device call.

Exception:

- IF $D(%ZIS)=0 then default value is "MQ" and call the Device Handler.
- IF $D(%ZIS)=1,%ZIS="" then queue the second task also with ZTIO="" (i.e., do not do the Device Handler call).

iop: (optional) The IOP variable as defined in Kernel's Device Handler. Default value "Q"—if IOP is passed and IOP does not start with "Q;" then "Q;" will be added.

%wr: (optional) If %WR>0 then write text to the screen as to whether or not the queueing was successful.

%rtn2: (required) Second task that TaskMan will run, usually a print process (i.e., print data).The API that TaskMan will DO to start the task. You can specify it as "LABEL^ROUTINE" or "^ROUTINE" or "ROUTINE".

%desc2: (optional) Second task description, up to 200 characters describing the task, with the software application name at the front. Default to name of [tag]^routine.

%var2: (optional) ZTSAVE values for the second task. Single value or passed by reference, this will be used to S ZTSAVE(). It can be a string of variable names separated by "**;**". Each **;**-piece will be used as a subscript in ZTSAVE.

- If %var1 is *not* passed and $D(%VAR), then also send %VAR data to the second task.
- If $D(%VAR1), then do *not* send %VAR data to the second task.

%voth2: (optional) Second task other parameter, usually not needed. Passed by reference, %voth(sub)="" or explicit value sub—this is any other %ZTLOAD variable besides ZTRTN, ZTDESC, ZTIO, ZTSAVE. For example:

```
%VOTH("ZTDTH")=$H
```

**NOTE:** If %voth1("ZTDTH") is passed, it will be ignored as it is necessary to S ZTDTH="**@**" for the second task—this creates the task but does not schedule it.

**Output** ztsk1^ztsk2: Returns:

- ztsk1^ztsk2—If successfully queued:
  - ztsk1 = ZTSK value of first task.
  - ztsk2 = ZTSK value of second task.
- -1—If unsuccessful device call or failed %ZTLOAD call.

**Example**

This example is a job that consists of gathering information and then printing it. Assume that the gathering takes a few hours. We do not want the device that the user selects to be tied up for that time, so we divide the job into two tasks. The first task gathers the information, and the second task prints it. We use the $$QQ^XUTMDEVQ API to select the device, schedule the gather task, and queue the print task. We use the $$REQQ^XUTMDEVQ(): Schedule Second Part of a Task API to schedule the print task when the gather task finishes.

**NOTE:** This is the easiest way to divide a job into two tasks.

```
ARHBQQ   ;SFVAMC/GB - Demo of 'gather' and 'print' in 2 tasks ;1/19/06  08:31
         ;;1.1
QQ       ;
         N X
         S X=$$QQ^XUTMDEVQ("GATHERQ^ARHBQQ","ARHB
Gather",,,,,1,"PRINTQ^ARHBQQ","ARHB Print")
         W !,"X=",X
         Q
GATHERQ  ;
         N ARHJ,X
         S ZTREQ="@"
         S ARHJ="ARHB-QQ"_"-"_$J_"-"_$H ; namespace + unique ID
         K ^XTMP(ARHJ) ; Use ^XTMP to pass a lot of data between tasks.
         S ^XTMP(ARHJ,0)=$$FMADD^XLFDT(DT,1)_U_DT ; Save-thru and create dates.
         S ^XTMP(ARHJ)="HI MOM!" ; Pretend this is a lot of data!
         ; XUTMQQ holds the ZTSK of the print task
         S X=$$REQQ^XUTMDEVQ(XUTMQQ,$H,"ARHJ") ; Schedule print task to start
         Q
PRINTQ   ;
         S ZTREQ="@"
         ;U IO ; Don't need this if invoked using a ^XUTMDEVQ API.
         W !,"The secret message is: '",$G(^XTMP(ARHJ)),"'"
         K ^XTMP(ARHJ)
         Q
```

**Figure 22-4. $$QQ^XUTMDEVQ example—Sample code**

# $$REQQ^XUTMDEVQ(): Schedule Second Part of a Task

**Reference Type** Supported

**Category** TaskMan

**IA #** 1519

**Description** (Added with Kernel Patch XU*8.0*389.) This extrinsic function schedules the second task (i.e., print data) from the $$QQ^XUTMDEVQ(): Double Queue—Direct Queuing in a Single Call API.

If it will take a long time to gather and print data, users should split the job into two tasks:

1. Gather Data—The first task gathers the data.
2. Print Data—The second task prints the data.

Separating the data gathering task from the data print task helps avoid unnecessarily tying up a printer while large amounts of data are gathered.

This API makes sure that only the scheduled time and any variables in %VAR are passed to the REQ^%ZTLOAD: Requeue a Task.

**Format**

```
$$REQQ^XUTMDEVQ(xutsk,xudth[,[.]%var])
```

**Input Parameters**

| | |
|---|---|
| xutsk: | (required) This input parameter is the TaskMan task to schedule the second task from the $$QQ^XUTMDEVQ(): Double Queue—Direct Queuing in a Single Call API and should be in the XUTMQQ variable. |
| xudth: | (required) This input parameter is the new scheduled run time. |
| [.]%var: | (optional) This input parameter is converted to the ZTSAVE variable; it is the same as the %var input parameter for the $$DEV^XUTMDEVQ(): Force Queueing—Ask for Device API. |

**Output**

returns: Returns:

- 1—Successful
- 0—Unsuccessful

### Example

This example is a job that consists of gathering information and then printing it. Assume that the gathering takes a few hours. We do not want the device that the user selects to be tied up for that time, so we divide the job into two tasks. The first task gathers the information, and the second task prints it. We use the $$QQ^XUTMDEVQ(): Double Queue—Direct Queuing in a Single Call API to select the device, schedule the gather task, and queue the print task. We use the $$REQQ^XUTMDEVQ API to schedule the print task when the gather task finishes.

**NOTE:** This is the easiest way to divide a job into two tasks.

```
ARHBQQ   ;SFVAMC/GB - Demo of 'gather' and 'print' in 2 tasks ;1/19/06  08:31
         ;;1.1
QQ       ;
         N X
         S X=$$QQ^XUTMDEVQ("GATHERQ^ARHBQQ","ARHB
Gather",,,,,1,"PRINTQ^ARHBQQ","ARHB Print")
         W !,"X=",X
         Q
GATHERQ  ;
         N ARHJ,X
         S ZTREQ="@"
         S ARHJ="ARHB-QQ"_"-"_$J_"-"_$H ; namespace + unique ID
         K ^XTMP(ARHJ) ; Use ^XTMP to pass a lot of data between tasks.
         S ^XTMP(ARHJ,0)=$$FMADD^XLFDT(DT,1)_U_DT ; Save-thru and create dates.
         S ^XTMP(ARHJ)="HI MOM!" ; Pretend this is a lot of data!
         ; XUTMQQ holds the ZTSK of the print task
         S X=$$REQQ^XUTMDEVQ(XUTMQQ,$H,"ARHJ") ; Schedule print task to start
         Q
PRINTQ   ;
         S ZTREQ="@"
         ;U IO ; Don't need this if invoked using a ^XUTMDEVQ API.
         W !,"The secret message is: '",$G(^XTMP(ARHJ)),"'"
         K ^XTMP(ARHJ)
         Q
```

**Figure 22-5. $$REQQ^XUTMDEVQ example—Sample code**

## DISP^XUTMOPT(): Display Option Schedule

**Reference Type** Supported

**Category** TaskMan

**IA #** 1472

**Description** This API is used to display the schedule for an option.

**Format** `DISP^XUTMOPT(option_name)`

**Input Parameters** option_name: (required) The name of the option from the OPTION file (#19) for which the TaskMan schedule is to be displayed.

**Output** returns: Returns the TaskMan option schedule.

**Example**

```
>D DISP^XUTMOPT(option_name)
```

## EDIT^XUTMOPT(): Edit an Option's Scheduling

**Reference Type** Supported

**Category** TaskMan

**IA #** 1472

**Description** This API allows users to edit an option's scheduling in the OPTION SCHEDULING file (#19.2).

**Format** `EDIT^XUTMOPT(option_name)`

**Input Parameters** option_name: (required) The name of the option from the OPTION file (#19) whose schedule the user is to be allowed to edit.

**Output** returns: Returns the requested option in order to edit the schedule.

**Example**

# OPTSTAT^XUTMOPT(): Obtain Option Schedule

**Reference Type** Supported

**Category** TaskMan

**IA #** 1472

**Description** This API allows an application to find out when a option is scheduled and get other data.

**Format** `OPTSTAT^XUTMOPT(option_name,.root)`

**Input Parameters**

| | |
|---|---|
| option_name: | (required) The name of the option from the OPTION file (#19) upon which to return data. |
| .root: | (required) This variable is passed by reference. This is an array because the same task can be scheduled more than once. |

**Output Parameters**

| | |
|---|---|
| .root: | Returns an array of data about the option in question. |

**Example**

```
>D OPTSTAT^XUTMOPT("OPTION NAME",.ROOT)
```

Returns an array of data in ROOT (pass by ref) in the form:

```
ROOT=count ROOT(1)=task number^scheduled time^reschedule freq^special queueing flag
```

# RESCH^XUTMOPT(): Set Up Option Schedule

**Reference Type** Supported

**Category** TaskMan

**IA #** 1472

**Description** This API allows an application to set up the schedule for an option.

**Format**

```
RESCH^XUTMOPT(option_name[,when_to_run][,device_to_use]
[,reschedule_freq][,flags][,.error_array])
```

**Input Parameters**

| | |
|---|---|
| option_name: | (required) The name of the option from the OPTION file (#19) to be rescheduled. |
| when_to_run: | (optional) The new scheduled time for the option to run. |
| device_to_use: | (optional) The device to use for the rescheduled option. |
| reschedule_freq: | (optional) The frequency to run the rescheduled option. |
| flags: | (optional) If the flag is set to an "L" LAYGO a new entry if needed. |
| .error_array: | (optional) Passed by reference. |

**Output Parameters** .error_array: (optional) This will be set to -1 if the option was not found.

**Example**

# EN^XUTMTP(): Display HL7 Task Information

**Reference Type** Controlled Subscription

**Category** TaskMan

**IA #** 3521

**Description** This API is called to display the Health Level Seven (HL7)-related task information. First, the currently running tasks are examined in the SCHEDULE file. For each task found, examine the ROUTINE field. If the ROUTINE field contains "HL", it is a Health Level Seven-related task.

**Format**

```
EN^XUTMTP(task[,ztenv,ztkey,ztname,ztflag,xutmuci])
```

**Input Parameters**

| | |
|---|---|
| task: | (required) TaskMan's task ID. |
| ztenv: | (optional) Set = 1. |
| ztkey: | (optional) Set = 0. |
| ztname: | (optional) Set = ,User name. |
| ztflag: | (optional) Set = 1. |
| xutmuci: | (optional) X ^%ZOSF("UCI") S XUTMUCI=Y |

**Output** returns: Returns the HL7-related task information. The following is an example of the information displayed by this API:

```
261181: EN^HLCSLM, HL7 Link Manager. No device.
DEV,MOU.
From 12/31/2001 at 14:17,  By KRNUSER,THIRTY.
Started running 12/31/2001 at 14:17.  Job #:
562039155
```

**Example**

## ^%ZTLOAD: Queue a Task

^%ZTLOAD is the main API used to create and schedule tasks (commonly referred to as "queuing"). Queuing tells TaskMan to use a background partition to DO a certain API at a certain time, with certain other conditions established as described by the input variables.

| | |
|---|---|
| **Reference Type** | Supported |
| **Category** | TaskMan |
| **IA #** | 10063 |
| **Description** | This API, 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. |
| **Format** | `^%ZTLOAD` |

Make sure to perform the following steps before calling ^%ZTLOAD:

- NEW all of the following variables.
- Set the input variables you want changed.
- Call ^%ZTLOAD.

If you do not follow these steps, the input variables could unintentionally assume the values of the variables of the current running task.

**Input Variabless**

ZTRTN: (required) The API that TaskMan will DO to start the task. You can specify it as "LABEL^ROUTINE" or "^ROUTINE" or "ROUTINE". If it is not passed, the original API is used.

ZTDESC: (required) Task description, up to 200 characters describing the task, with the software application 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="". The ztio variable accepts the same I/O formatting string as IOP.

**REF:** For more information, please refer to the "Device Handler" section in this manual.

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; 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 variable array. An array whose nodes specify input variables 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 variables 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 TaskMan 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 variable).

**REF:** For complete information on how to use SYNC FLAGs, please refer to the "Using SYNC FLAGs to Control Sequences of Tasks" topic in this chapter.

**Output Variables**

ZTSK: (Usually returned) The task number assigned to a task, returned whenever a task is successfully created. It can be used as an input variable to the other TaskMan application mode APIs.

**NOTE:** 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.

### Interactive Use of ^%ZTLOAD

The VistA 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 variables 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
```

**Figure 22-6. TaskMan: Print queuer sample code**

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:** 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:

- Concurrent—Those that can run concurrently.
- Sequential—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 software application 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.

**REF:** For more information on SYNC FLAGs, please refer to the "Using SYNC FLAGs to Control Sequences of Tasks" topic in this chapter.

### Example

This example is a job that consists of gathering information and then printing it. Assume that the gathering takes a few hours. We do not want the device that the user selects to be tied up for that time, so we divide the job into two tasks. The first task gathers the information and the second task prints it. We use the ^%ZIS: Standard Device Call API to select the device, the ^%ZTLOAD API to queue the print task, and the ^%ZTLOAD API to schedule the gather task. We use the REQ^%ZTLOAD: Requeue a Task API to schedule the print task when the gather task finishes.

**NOTE:** This process is made easier by using the $$QQ^XUTMDEVQ(): Double Queue—Direct Queuing in a Single Call and $$REQQ^XUTMDEVQ(): Schedule Second Part of a Task APIs.

Sample Code:

```
ARHBQQ   ;SFVAMC/GB - Demo of 'gather' and 'print' in 2 tasks ;1/19/06  08:31
         ;;1.1
ZTLOAD   ;
         N ARH,ARHZTSK,X,ZTSAVE,%ZIS,ZTSK,ZTDTH,ZTRTN,ZTDESC,ZTIO,POP
         W !,"Queue the second task (the print task) first.",!
         ;Let's deal with the second task first.
         ;The user doesn't know it, but he's actually queuing the second task,
         ;the "print" portion of the job.  The only question the user will be
         ;asked is to select the device.
         ;
         S %ZIS="QM"
         S IOP="Q" ;Force queuing.
         D ^%ZIS Q:POP  ; Select Device
         W !,"Finished with %ZIS."
         ;
         S ZTDTH="@" ;Don't schedule the task to run, we'll do it later
         ;If we didn't need to set ZTDTH, we could use EN^XUTMDEVQ, but that
         ;API 'new's ZTDTH, so we can't set it.
         ;
         ;BTW, Did you know that there's a 5th parameter in EN^XUTMDEVQ?
         ;Usually, EN^XUTMDEVQ will 'new' ZTSK, so you can't get to it.
         ;If you put "1" as the 5th parameter, ZTSK will exist when EN returns.
         ;D EN^XUTMDEVQ("PRINT^ARHBQQ","ARHB Print",.ZTSAVE,.%ZIS,1)
         ;
         S ZTRTN="PRINT^ARHBQQ"
         S ZTDESC="ARHB Print"
         D ^%ZTLOAD
         D HOME^%ZIS
         W !,"ZTSK=",$G(ZTSK)
         Q:'$D(ZTSK)
         S ARHZTSK=ZTSK
         ;
         N ZTSAVE,%ZIS,ZTSK,ZTDTH,ZTRTN,ZTDESC,ZTIO,IOP
         W !,"Now queue the first task (the gather task).",!
         ;Now queue the first task, the "gather" portion of the job.
         ;Since we don't need a device,
         ;the user will only be asked when to start the task.
         ;(I wasn't able to get EN^XUTMDEVQ to work for me.  I tried setting
         ;IOP="Q;" to let it know that it should be queued and it didn't need
         ;a device, but it did nothing, and returned a null ZTSK.)
         F I="ARHZTSK" S ZTSAVE(I)="" ; Save the ZTSK of the "print" task.
         S ZTIO="" ; We don't need a device.
         S IOP="Q" ; Force queuing.
         S ZTRTN="GATHER^ARHBQQ"
         S ZTDESC="ARHB Gather"
         D ^%ZTLOAD
         D HOME^%ZIS
         W !,"ZTSK=",$G(ZTSK)
         Q
GATHER   ;
         N ARHJ
         S ZTREQ="@"
         S ARHJ="ARHB-QQ"_"-"_$J_"-"_$H ; namespace + unique ID
         K ^XTMP(ARHJ) ; Use ^XTMP to pass a lot of data between tasks.
         S ^XTMP(ARHJ,0)=$$FMADD^XLFDT(DT,1)_U_DT ; Save-thru and create dates.
         S ^XTMP(ARHJ)="HI MOM!" ; Pretend this is a lot of data.
         D SPRINT
         Q
SPRINT   ; Now schedule the "print" task to run.
```

```
         N ZTSK,ZTDTH,I,ZTRTN,ZTDESC,ZTIO,ZTSAVE ; Very important to NEW the
         ; input variables to REQ^%ZTLOAD, otherwise they retain the values of
         ; the currently running task, and you could unintentionally change the
         ; "print" task to rerun the "gather" task.
         F I="ARHJ" S ZTSAVE(I)="" ; Let the "print" task know the "$J" value.
         S ZTSK=ARHZTSK
         S ZTDTH=$H
         D REQ^%ZTLOAD
         ;Instead of the above 8 lines we could have simply:
         ;S X=$$REQQ^XUTMDEVQ(ARHZTSK,$H,"ARHJ")
         Q
PRINT    ;
         S ZTREQ="@"
         U IO ; Don't need this if invoked using a ^XUTMDEVQ API.
         W !,"The secret message is: '",$G(^XTMP(ARHJ)),"'"
         K ^XTMP(ARHJ)
         Q
```

**Figure 22-7. ^%ZTLOAD example—Sample code**

Code Execution:

```
VAH>D ZTLOAD^ARHBQQ

Queue the second task (the print task) first.
QUEUE TO PRINT ON
DEVICE: HOME// P-MESS

 1 P-MESSAGE-ENGWO-HFS-VXD   HFS FILE ==> MAILMESSAGE
 2 P-MESSAGE-HFS-VXD   HFS FILE ==> MAILMESSAGE
Choose 1-2> 2 <Enter>  P-MESSAGE-HFS-VXD  HFS FILE ==> MAILMESSAGE

Subject: MY PRINT

     Select one of the following:

          M         Me
          P         Postmaster

From whom: Postmaster// <Enter>
Send mail to: KRNUSER,ONE// <Enter>  KRNUSER,ONE
Select basket to send to: IN// <Enter>
And Send to: <Enter>
Finished with %ZIS.
ZTSK=2921497
Now queue the first task (the gather task).

Requested Start Time: NOW// <Enter>  (JAN 25, 2005@11:30:35)
ZTSK=2921499
```

**Figure 22-8. ^%ZTLOAD example—Sample code execution**

Output:

```
Subj: MY PRINT  [#28881111] 01/25/05@11:30  2 lines
From: POSTMASTER (Sender: BEUSCHEL,GARY - COMPUTER SPECIALIST)  In 'IN'
basket.
Page 1  *New*
-------------------------------------------------------------------------------

The secret message is: 'HI MOM!'

Enter message action (in IN basket): Ignore//
```

**Figure 22-9. ^%ZTLOAD example—Sample output**

## $$ASKSTOP^%ZTLOAD: Stop TaskMan Task

**Reference Type** Supported

**Category** TaskMan

**IA #** 10063

**Description** This extrinsic function asks TaskMan to stop running a specified task.

**Format** `$$ASKSTOP^%ZTLOAD(ztsk)`

**Input Parameters** ztsk: (required) Task number of the TaskMan task to be stopped.

**Output** returns: Returns:

1—Task stopped.

0—Task was *not* found.

**Example**

## DESC^%ZTLOAD(): Find Tasks with a Description

**Reference Type** Supported

**Category** TaskMan

**IA #** 10063

**Description** This API finds tasks with a specific description.

**Format** `DESC^%ZTLOAD(description,list)`

**Input Parameters** description: (required) The TaskMan task description.

**Output Parameters** list: Returns a list of task(s) with the specified description.

**Example**

## DQ^%ZTLOAD: Unschedule a Task

**Reference Type** Supported

**Category** TaskMan

**IA #** 10063

**Description** This API unschedules 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 (#14.4). However, unscheduling a task that has already started running does *not* stop the task in any way.

**Format** `DQ^%ZTLOAD`

Make sure to perform the following steps before calling ^%ZTLOAD:

- NEW all of the following input variables.
- Set the input variables you want changed.
- Call DQ^%ZTLOAD.

If you do not follow these steps, the input variables could unintentionally assume the values of the variables of the current running task.

| | | |
|---|---|---|
| **Input Variables** | ZTSK: | (required) The number of the task to unschedule. This task must currently be defined in the TASKS file (#14.4) or the call will fail. |
| **Output Variables** | ZTSK(0): | Returns:<br>1—Task was unscheduled successfully.<br>0—Task was *not* unscheduled successfully. |

**Example**

## ISQED^%ZTLOAD: Return Task Status

**Reference Type** Supported

**Category** TaskMan

**IA #** 10063

**Description** This API returns whether a task is currently pending. Pending means that the task is 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.

**Format**

```
ISQED^%ZTLOAD
```

Make sure to perform the following steps before calling ^%ZTLOAD:

- NEW all of the following input variables.
- Set the input variables you want changed.
- Call ISQED^%ZTLOAD.

If you do not follow these steps, the input variables could unintentionally assume the values of the variables of the current running task.

**Input Variables**

ZTSK: (required) Task number of the task to look up. The task must be currently defined on the volume set to be searched, or the lookup 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 Variables**

ZTSK(0): ZTSK(0) is returned as follows:

1—Task ZTSK is currently scheduled or waiting on volume set ZTCPU.

0—Task ZTSK is *not* currently scheduled or waiting on volume set ZTCPU.

NULL ("")—The lookup was unsuccessful.

ZTSK("E"): (sometimes returned) The error code, returned when some error condition prevented a successful lookup. 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 (#14.5).

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 zero (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 zero (0) or 1.

**Example**

## $$JOB^%ZTLOAD(): Return a Job Number for a Task

**Reference Type** Supported

**Category** TaskMan

**IA #** 10063

**Description** This extrinsic function was released with Kernel Patch XU*8.0*339. It returns the job number for a running TaskMan task.

**Format** `JOB^%ZTLOAD(ztsk)`

**Input Parameters**

| | |
|---|---|
| ztsk: | (required) Task number of the running TaskMan task. If the specified task is *not* running, it returns null. |

**Output**

| | |
|---|---|
| returns: | Returns the job number for the specified running TaskMan task. |

**Example**

# KILL^%ZTLOAD: Delete a Task

**Reference Type** Supported

**Category** TaskMan

**IA #** 10063

**Description** This API deletes 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:** Tasks can delete their own records through the use of the ZTREQ output variable.

**Format**

```
KILL^%ZTLOAD
```

Make sure to perform the following steps before calling ^%ZTLOAD:

- NEW all of the following input variables.
- Set the input variables you want changed.
- Call KILL^%ZTLOAD.

If you do not follow these steps, the input variables could unintentionally assume the values of the variables of the current running task.

**Input Variables** ZTSK: (required) Task number of the TaskMan task to delete.

**Output Variables** ZTSK(0): Returns:

1—Successful deletion of the task.

0—Requested task number is invalid.

**Example**

## OPTION^%ZTLOAD(): Find Tasks for an Option

**Reference Type** Supported

**Category** TaskMan

**IA #** 10063

**Description** This API finds TaskMan tasks for a specific option.

**Format** `OPTION^%ZTLOAD(option,list)`

**Input Parameters** option: (required) The name of the specific option.

**Output Parameters** list: Returns a list of TaskMan task(s) for the specified option.

**Example**

## PCLEAR^%ZTLOAD(): Clear Persistent Flag for a Task

**Reference Type** Supported

**Category** TaskMan

**IA #** 10063

**Description** This API clears the persistent flag for a TaskMan task (clears the persistent node).

**Format** `PCLEAR^%ZTLOAD(ztsk)`

**Input Parameters** ztsk: (required) The TaskMan task number.

**Output** none

**Example**

# $$PSET^%ZTLOAD(): Set Task as Persistent

**Reference Type** Supported

**Category** TaskMan

**IA #** 10063

**Description** This extrinsic function sets a TaskMan task as persistent (sets the persistent node). A task that is marked as persistent is restarted if TaskMan finds that the lock on ^%ZTSCH("TASK",tasknumber) has been removed. This adds the requirement that the task only use incremental locks, that the entry in ^%ZTSK(task... be left in place as this is used to restart the task, and that the task can be restarted from the data that is in the ^%ZTSK(task,... global.

**Format**

```
$$PSET^%ZTLOAD(ztsk)
```

**Input Parameters** ztsk: (required) The TaskMan task number.

**Output** returns: Returns:

1—Flag was set.

0—Flag was *not* set.

**Example**

# REQ^%ZTLOAD: Requeue a Task

**Reference Type** Supported

**Category** TaskMan

**IA #** 10063

**Description** This API unschedules, edits, and reschedules a task. Unscheduling ensures the task is not pending but does not stop it from running. Editing is limited to the API, 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.

**CAUTION: Because requeuing 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; thus, programmers should use requeuing very carefully. Queuing a new task is usually a better way to accomplish the same goals.**

**NOTE:** Tasks can reschedule themselves through use of the ZTREQ output variable.

**Format**

```
REQ^%ZTLOAD
```

Make sure to perform the following steps before calling ^%ZTLOAD:

- NEW all of the following input variables.
- Set the input variables you want changed.
- Call REQ^%ZTLOAD.

If you do not follow these steps, the input variables could unintentionally assume the values of the variables of the current running task.

**Input Variables**

ZTSK: (required) The TaskMan 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 software application 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.

The ztdth input variable can 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 60 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) The API TaskMan will DO to start the task. You can specify it as "LABEL^ROUTINE" or "^ROUTINE" or "ROUTINE". If it is not passed, the original API is used.

ZTSAVE: (optional) Input variable array. An array whose nodes specify input variables to the task beyond the usual set all tasks receive. It is set up in the same format as the ZTSAVE input variable for the ^%ZTLOAD API.

**Output Variables**

ZTSK(0): Returns:

1—Task is defined.

0—Task is *not* defined or ZTDTH was passed in a bad format.

**Example**

This example is a job that consists of gathering information and then printing it. Assume that the gathering takes a few hours. We do not want the device that the user selects to be tied up for that time, so we divide the job into two tasks. The first task gathers the information and the second task prints it. We use the ^%ZIS: Standard Device Call API to select the device, the ^%ZTLOAD: Queue a Task API to queue the print task, and the ^%ZTLOAD: Queue a Task API to schedule the gather task. We use the REQ^%ZTLOAD API to schedule the print task when the gather task finishes.

**NOTE:** This process is made easier by using the $$QQ^XUTMDEVQ(): Double Queue—Direct Queuing in a Single Call and $$REQQ^XUTMDEVQ(): Schedule Second Part of a Task APIs.

Sample Code:

```
ARHBQQ   ;SFVAMC/GB - Demo of 'gather' and 'print' in 2 tasks ;1/19/06  08:31
         ;;1.1
ZTLOAD   ;
         N ARH,ARHZTSK,X,ZTSAVE,%ZIS,ZTSK,ZTDTH,ZTRTN,ZTDESC,ZTIO,POP
         W !,"Queue the second task (the print task) first.",!
         ;Let's deal with the second task first.
         ;The user doesn't know it, but he's actually queuing the second task,
         ;the "print" portion of the job.  The only question the user will be
         ;asked is to select the device.
         ;
         S %ZIS="QM"
         S IOP="Q" ;Force queuing.
         D ^%ZIS Q:POP  ; Select Device
         W !,"Finished with %ZIS."
         ;
         S ZTDTH="@" ;Don't schedule the task to run, we'll do it later
         ;If we didn't need to set ZTDTH, we could use EN^XUTMDEVQ, but that
         ;API 'new's ZTDTH, so we can't set it.
         ;
         ;BTW, Did you know that there's a 5th parameter in EN^XUTMDEVQ?
         ;Usually, EN^XUTMDEVQ will 'new' ZTSK, so you can't get to it.
         ;If you put "1" as the 5th parameter, ZTSK will exist when EN returns.
         ;D EN^XUTMDEVQ("PRINT^ARHBQQ","ARHB Print",.ZTSAVE,.%ZIS,1)
         ;
         S ZTRTN="PRINT^ARHBQQ"
         S ZTDESC="ARHB Print"
         D ^%ZTLOAD
         D HOME^%ZIS
         W !,"ZTSK=",$G(ZTSK)
         Q:'$D(ZTSK)
         S ARHZTSK=ZTSK
         ;
         N ZTSAVE,%ZIS,ZTSK,ZTDTH,ZTRTN,ZTDESC,ZTIO,IOP
         W !,"Now queue the first task (the gather task).",!
         ;Now queue the first task, the "gather" portion of the job.
         ;Since we don't need a device,
         ;the user will only be asked when to start the task.
         ;(I wasn't able to get EN^XUTMDEVQ to work for me.  I tried setting
         ;IOP="Q;" to let it know that it should be queued and it didn't need
         ;a device, but it did nothing, and returned a null ZTSK.)
         F I="ARHZTSK" S ZTSAVE(I)="" ; Save the ZTSK of the "print" task.
         S ZTIO="" ; We don't need a device.
         S IOP="Q" ; Force queuing.
         S ZTRTN="GATHER^ARHBQQ"
         S ZTDESC="ARHB Gather"
         D ^%ZTLOAD
         D HOME^%ZIS
         W !,"ZTSK=",$G(ZTSK)
         Q
GATHER   ;
         N ARHJ
         S ZTREQ="@"
         S ARHJ="ARHB-QQ"_"-"_$J_"-"_$H ; namespace + unique ID
         K ^XTMP(ARHJ) ; Use ^XTMP to pass a lot of data between tasks.
         S ^XTMP(ARHJ,0)=$$FMADD^XLFDT(DT,1)_U_DT ; Save-thru and create dates.
         S ^XTMP(ARHJ)="HI MOM!" ; Pretend this is a lot of data.
         D SPRINT
         Q
SPRINT   ; Now schedule the "print" task to run.
```

```
         N ZTSK,ZTDTH,I,ZTRTN,ZTDESC,ZTIO,ZTSAVE ; Very important to NEW the
         ; input variables to REQ^%ZTLOAD, otherwise they retain the values of
         ; the currently running task, and you could unintentionally change the
         ; "print" task to rerun the "gather" task.
         F I="ARHJ" S ZTSAVE(I)="" ; Let the "print" task know the "$J" value.
         S ZTSK=ARHZTSK
         S ZTDTH=$H
         D REQ^%ZTLOAD
         ;Instead of the above 8 lines we could have simply:
         ;S X=$$REQQ^XUTMDEVQ(ARHZTSK,$H,"ARHJ")
         Q
PRINT    ;
         S ZTREQ="@"
         U IO ; Don't need this if invoked using a ^XUTMDEVQ API.
         W !,"The secret message is: '",$G(^XTMP(ARHJ)),"'"
         K ^XTMP(ARHJ)
         Q
```

**Figure 22-10. REQ^%ZTLOAD example—Sample code**

Code Execution:

```
VAH>D ZTLOAD^ARHBQQ

Queue the second task (the print task) first.
QUEUE TO PRINT ON
DEVICE: HOME// P-MESS

 1 P-MESSAGE-ENGWO-HFS-VXD   HFS FILE ==> MAILMESSAGE
 2 P-MESSAGE-HFS-VXD   HFS FILE ==> MAILMESSAGE
Choose 1-2> 2 <Enter>  P-MESSAGE-HFS-VXD  HFS FILE ==> MAILMESSAGE

Subject: MY PRINT

     Select one of the following:

          M         Me
          P         Postmaster

From whom: Postmaster// <Enter>
Send mail to: KRNUSER,ONE// <Enter>  KRNUSER,ONE
Select basket to send to: IN// <Enter>
And Send to: <Enter>
Finished with %ZIS.
ZTSK=2921497
Now queue the first task (the gather task).

Requested Start Time: NOW// <Enter>  (JAN 25, 2005@11:30:35)
ZTSK=2921499
```

**Figure 22-11. ^%ZTLOAD example—Sample code execution**

Output:

```
Subj: MY PRINT  [#28881111] 01/25/05@11:30  2 lines
From: POSTMASTER (Sender: BEUSCHEL,GARY - COMPUTER SPECIALIST)  In 'IN'
basket.
Page 1  *New*
-------------------------------------------------------------------------------

The secret message is: 'HI MOM!'

Enter message action (in IN basket): Ignore//
```

**Figure 22-12. ^%ZTLOAD example—Sample output**

## RTN^%ZTLOAD(): Find Tasks that Call a Routine

**Reference Type** Supported

**Category** TaskMan

**IA #** 10063

**Description** This API finds TaskMan tasks that call a specific routine.

**Format** `RTN^%ZTLOAD(routine,list)`

**Input Parameters** routine: (required) The name of the specific routine called.

**Output Parameters** list: Returns a list of TaskMan task(s) that call the specified routine.

**Example**

# $$S^%ZTLOAD(): Check for Task Stop Request

**Reference Type** Supported

**Category** TaskMan

**IA #** 10063

**Description** This extrinsic function is used 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.

**Format**

```
$$S^%ZTLOAD([message])
```

**Input Parameters**

| | |
|---|---|
| message: | (optional) Allows you to leave a message for the creator of the TaskMan task. |

**Output**

| | |
|---|---|
| returns: | Returns:<br>1—Creator of the task that has asked the task to stop.<br>0—For all other cases. |

**Example**

# STAT^%ZTLOAD: Task Status

**Reference Type** Supported

**Category** TaskMan

**IA #** 10063

**Description** This API looks up tasks and retrieves 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 either is 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. 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.

**Format** `STAT^%ZTLOAD`

Make sure to perform the following steps before calling ^%ZTLOAD:

- NEW all of the following input variables.
- Set the input variables you want changed.
- Call STAT^%ZTLOAD.

If you do not follow these steps, the input variables could unintentionally assume the values of the variables of the current running task.

**Input Variables** ZTSK: (required) The TaskMan task number to look up. It must be defined on the current volume set.

**Output Variables** ZTSK(0): Returns:

1—Task is defined.

0—Task is *not* defined.

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 can include bad data, user intervention, hard error, and many other possibilities.

**Example**

## $$TM^%ZTLOAD: Check if TaskMan is Running

**Reference Type** Supported

**Category** TaskMan

**IA #** 10063

**Description** This extrinsic function determines if TaskMan is running. Use this function if you need to know the status of TaskMan.

**Format** `$$TM^%ZTLOAD`

**Input Parameters** none

**Output** returns: Returns:

1—TaskMan is running on the current volume set.

0—TaskMan is *not* running on the current volume set.

**Example**

## ZTSAVE^%ZTLOAD(): Build ZTSAVE Array

**Reference Type** Supported

**Category** TaskMan

**IA #** 10063

**Description** This API stores a string of variables in the ZTSAVE array.

**Format** `ZTSAVE^%ZTLOAD(string_of_variables[,kill_ztsave_flag])`

**Input Parameters**

| | |
|---|---|
| string_of_variables: | (required) Sting of variable names to be stored in the ZTSAVE array. |
| kill_ztsave_flag | (optional) Any positive value will first KILL the ZTSAVE array. |

**Output** returns: Stores the string of input variables in the ZTSAVE array.

**Example**

# 23. Unwinder: Programmer Tools

## Application Program Interface (API)

Several APIs are available for programmers to work with Kernel Unwinder. These APIs are described below.

### EN^XQOR(): Navigating Protocols

**Reference Type** Supported

**Category** Unwinder

**IA #** 10101

**Description** This API is the main routine for navigating protocols. The routine processes the initial protocol and the subordinate protocols. This processing of subordinate protocols happens according to the type of protocol and the navigation variables that get set along the way.

**Format** `EN^XQOR(x)`

**Input Parameters**

x: (required) Identifies the initial protocol that EN^XQOR should process. The "x" input parameter should be in variable pointer format. For example:

x="1234;ORD(101,"

would cause the processing to start with the protocol that has an internal entry number (IEN) of 1234.

An alternative to using variable pointer format is to set x equal to the name or number of the protocol and DIC equal to the number or global reference of the file you are working in (generally the PROTOCOL file [#101]).

**Output** none

**Example**

## EN1^XQOR(): Navigating Protocols

**Reference Type** Supported

**Category** Unwinder

**IA #** 10101

**Description** This API is identical to the EN^XQOR(): Navigating Protocols API, except that the entry and exit actions of the initial protocol are not executed. This API provides backwards compatibility with the way Kernel V. 6 processed protocols that were defined in the OPTION file (#19).

**Format** `EN1^XQOR(x)`

**Input Parameters**

x: (required) Identifies the initial protocol that EN^XQOR should process. The "x" input parameter should be in variable pointer format. For example:

x="1234;ORD(101,"

would cause the processing to start with the protocol that has an internal entry number (IEN) of 1234.

An alternative to using variable pointer format is to set x equal to the name or number of the protocol and DIC equal to the number or global reference of the file you are working in (generally the PROTOCOL file [#101]).

**Output** none

**Example**

## MSG^XQOR(): Enable HL7 Messaging

**Reference Type** Supported

**Category** Unwinder

**IA #** 10101

**Description** This API is used to enable Health Level Seven (HL7) messaging through the XQOR Unwinder.

**Format** `MSG^XQOR(protocol,.msgtext)`

**Input Parameters**

| | |
|---|---|
| protocol: | (required) The name of the protocol with which the HL7 message will be associated. |
| .msgtext | (required) The array containing the HL7 message. |

**Output** none

**Example**

## EN^XQORM(): Menu Item Display and Selection

**Reference Type** Supported

**Category** Unwinder

**IA #** 10140

**Description** This API handles the display of and selection from a menu; this routine processes a single menu only. This is the call that the EN^XQOR(): Navigating Protocols API uses to obtain menu selections. The caller is responsible to handle any selections from the menu that are returned in the y array. If you want navigation to the selected items handled for you, use the EN^XQOR(): Navigating Protocols API. The menus handled by this routine are the multiple selection, multiple column menus that are typical in Order Entry/Results Reporting (OE/RR).

**Format**

```
EN^XQORM(xqorm,xqorm(0))
```

**Input Parameters**

xqorm: (required) A variable pointer to the menu that should be displayed (e.g. XQORM="1234;ORD(101,").

xqorm(0) (required) A string of flags that control the display and prompting of the menu:

Numeric—Maximum number of selections allowed.

A—Prompt for a selection from the menu.

D—Display the menu.

**Output Parameters**

y(): This array contains the items that the user selected from the menu.

**Example**

## XREF^XQORM(): Force Menu Recompile

**Reference Type** Supported

**Category** Unwinder

**IA #** 10140

**Description** This API forces a menu to recompile. Menus are compiled into the XUTL global. This should happen automatically. However, you can use this API to force a menu to recompile.

**Format** `XREF^XQORM(xqorm)`

**Input Parameters** xqorm: (required) Variable pointer to the protocol that should be recompiled.

**Output** returns: Returns recompiled menu.

**Example**

## DISP^XQORM1(): Display Menu Selections From Help Code

**Reference Type** Supported

**Category** Unwinder

**IA #** 10102

**Description** This API displays menu selections from help code, if you have replaced the standard help by setting XQORM("**??**"). This API should only be called from within the code used by XQORM("**??**").

**Format** `DISP^XQORM1(x)`

**Input Parameters** x: (required) Must be "**?**".

**Output** returns: Returns menu selections.

**Example**

# 24. User: Programmer Tools

## Application Program Interface (API)

Several APIs are available for programmers to work with the user. These APIs are described below.

### $$CODE2TXT^XUA4A72(): Get HCFA Text

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | User | |
| **IA #** | 1625 | |
| **Description** | This extrinsic function returns the three parts of the Health Care Financing Administration (HCFA) text from the PERSON CLASS file (#8932.1) based on passing in the Internal Entry Number (IEN) or the VA's Vcode. | |
| **Format** | `$$CODE2RXT^XUA4A72(ien_or_vcode)` | |
| **Input Parameters** | ien_or_vcode: | (required) Pass in either the Internal Entry Number (IEN) or the VA Vcode for the text that should be returned. |
| **Output** | returns: | Returns HCFA text. |

**Example**

## $$GET^XUA4A72(): Get Specialty and Subspecialty for a User

**Reference Type** Supported

**Category** User

**IA #** 1625

**Description** This extrinsic function returns the "IEN^Profession^Specialty^Sub-specialty^Effect date^Expired date^VA code" for the person identified by the DUZ in effect on the date passed in, in internal VA FileMan format (TODAY if no date passed in). It returns:

-1—If DUZ does not point to a valid user or user has never had a Person Class assigned.

-2—If no active Person Class on that date.

**Format**

```
$$GET^XUA4A72(duz[,date])
```

**Input Parameters**

duz: (required) Internal Entry Number (IEN) for the person being checked in the NEW PERSON file (#200).

date: (optional) Date in internal VA FileMan format, to indicate effective date for determination.

**Output**

returns: Returns:

-1—If DUZ does not point to a valid user or user has never had a Person Class assigned.

-2—If no active Person Class on that date.

**Example**

## $$IEN2CODE^XUA4A72(): Get VA Code

**Reference Type** Supported

**Category** User

**IA #** 1625

**Description** This extrinsic function returns the VA CODE from the PERSON CLASS file (#8932.1) that corresponds to the Internal Entry Number (IEN) passed in. If the IEN passed in does *not* match a valid entry in the PERSON CLASS file (#8932.1), an empty string is returned.

**Format**

```
$$IEN2CODE^XUA4A72(ien)
```

**Input Parameters**

| | |
|---|---|
| ien: | (required) Internal Entry Number (IEN) in the PERSON CLASS file (#8932.1). |

**Output**

| | |
|---|---|
| returns: | Returns the VA CODE. |

**Example**

# $$DTIME^XUP(): Reset DTIME for USER

**Reference Type** Supported

**Category** User

**IA #** 4409

**Description** This extrinsic function resets the DTIME variable for the user identified by the first parameter "DUZ" of this function. This extrinsic function accepts two parameters:

1. IEN or DUZ of the user in the NEW PERSON file (#200).
2. IEN of the device in the DEVICE file (#3.5).

The return value should be assigned to the variable DTIME as shown in the examples. This DTIME variable is used on all timed READS where interactive responses are required for a given user.

**Format** `$$DTIME^XUP([duz][,ios])`

**Input Parameters**

duz: (optional) The Internal Entry Number (IEN) or DUZ of the user in the NEW PERSON file (#200).

ios: (optional) The IEN of the device in the DEVICE file (#3.5). This IEN should be the same value of IOS if present, and should reflect the current sign-on device of the user.

**Output**

returns: The return value will be based on the first available data found in the following fields/files (listed in search order):

1. TIMED READ (# OF SECONDS) field (#200.1) of the NEW PERSON file (#200).
2. TIMED READ (# OF SECONDS) field (#51.1) of the DEVICE file (#3.5).
3. DEFAULT TIMED READ (SECONDS) field (#210) of the KERNEL SYSTEM PARAMETERS file (#8989.3).
4. (default) If *no* data is available in any of the three fields above, then the return value defaults to 300 seconds.

**Example 1**

Sending DUZ only, returns the value in Field #200.1, TIMED READ (# OF SECONDS), of the NEW PERSON file (#200):

```
>S DTIME=$$DTIME^XUP(DUZ)

>W DTIME
1800
```

**Example 2**

Sending DUZ and IOS, returns the value in Field #200.1, TIMED READ (# OF SECONDS), of the NEW PERSON file (#200):

```
>S DTIME=$$DTIME^XUP(DUZ,IOS)

>W DTIME
1800
```

**Example 3**

Sending IOS only, returns the value in Field #51.1, TIMED READ (# OF SECONDS), of the DEVICE file (#3.5):

```
>S DTIME=$$DTIME^XUP(,IOS)

>W DTIME
500
```

**Example 4**

*Not* Sending DUZ or IOS, returns the value in Field #210, DEFAULT TIMED READ (SECONDS), of the KERNEL SYSTEM PARAMETERS file (#8989.3):

```
>S DTIME=$$DTIME^XUP(,)

>W DTIME
400
```

or

```
>S DTIME=$$DTIME^XUP()

>W DTIME
400
```

**Example 5**

*Not* Sending DUZ or IOS *and* no value is in Field #210, DEFAULT TIMED READ (SECONDS), of the KERNEL SYSTEM PARAMETERS file (#8989.3):

```
>S DTIME=$$DTIME^XUP()

>W DTIME
300
```

# $$ACTIVE^XUSER(): Status Indicator

**Reference Type** Supported

**Category** User

**IA #** 2343

**Description** This extrinsic function returns the active status indicator and latest signon information of a user in the NEW PERSON file (#200).

**Format** `$$ACTIVE^XUSER(ien)`

**Input Parameters**

ien: (required) Internal Entry Number (IEN) of the user to be checked in the NEW PERSON file (#200).

**Output**

returns: Returns any of the following codes:

""—Null, no user record found.

0—User *can't* sign on.

0^DISUSER—User can't sign on because of DISUSER flag.

0^TERMINATED^FMDATE—User terminated on date indicated.

1^NEW—A new user, can sign on.

1^ACTIVE^FMDATE—An active user, last signon date.

### Example 1

This is an example of an Active User in the NEW PERSON file (#200):

```
>S X=$$ACTIVE^XUSER(1529)

>WRITE X
1^ACTIVE^3030321.093756
```

### Example 2

This is an example of a Terminated User in the NEW PERSON file (#200):

```
>S X=$$ACTIVE^XUSER(957)

>WRITE X
0^TERMINATED^2980504
```

### Example 3

This is an example of a User with no record in the NEW PERSON file (#200), returns a null string:

```
>S X=$$ACTIVE^XUSER(999999999)

>W X

>
```

### Example 4

This is an example of a User in the NEW PERSON file (#200) with the DISUSER flag set:

```
>S X=$$ACTIVE^XUSER(111)

>W X
0^DISUSER
```

## $$DEA^XUSER(): Get DEA Number

**Reference Type** Supported

**Category** Public Key Infrastructure (PKI)

**IA #** 2343

**Description** This extrinsic function returns a user's Drug Enforcement Agency (DEA) number, if it exists in the DEA# field (#53.2) of the NEW PERSON file (#200). If the DEA# field value is null, the value returned depends on the optional FLAG input parameter (see below). This API was requested as part of the Public Key Infrastructure (PKI) Project.

**Format**

```
$$DEA^XUSER([flag][,userien])
```

**Input Parameters**

flag: (optional) This flag controls what is returned when the user does not have a value in the DEA# field (#53.2) of the NEW PERSON file (#200).

1—This routine will check to see if the user has a value in the VA# field (#53.3) of the NEW PERSON file (#200). If a value is found in that field, this routine will return that field value. Otherwise, this routine returns an empty string.

NULL or 0—This routine will check to see if the user has values in the VA# field (#53.3) of the NEW PERSON file (#200) and the (new) FACILITY DEA NUMBER field (#52) of the INSTITUTION file (#4). If values are found in both of those fields, this routine will return the following:

```
FACILITY DEA NUMBER field (#52)_"-
"_VA# field(#53.3)
```

userien: (optional) This value can be used to get the DEA# of some user besides the one that signed in. In CPRS, to check that a student's teacher has the required DEA#.

**Output**

returns: Returns the DEA# field (#53.2) value or the value returned based on the (optional) flag input parameter.

### Example 1

The following are the data values for this example:

- IEN = "1000118"
- DEA# (#53.2) field = "AB1234567"
- FACILITY DEA NUMBER field (#52) = "VA7654321"

- VA# field (#53.3) = "789"

If the FLAG input parameter is null or "0", this API would return "AB1234567," since the user has a DEA#:

```
>S X=$$DEA^XUSER(0,1000118)

>W X
AB1234567
```

If the FLAG input parameter is "1", this API would return "AB1234567," since the user has a DEA#:

```
>S X=$$DEA^XUSER(1,1000118)

>W X
AB1234567
```

**Example 2**

The following are the data values for this example:

- DEA# (#53.2) field = NULL
- FACILITY DEA NUMBER field (#52) = "VA7654321"
- VA# field (#53.3) = "789"

If the FLAG input parameter is NULL or "0", this API would return "VA7654321-789":

```
>S X=$$DEA^XUSER(0,)

>W X
VA7654321-789
```

If the FLAG input parameter is "1", this API would return "789":

```
>S X=$$DEA^XUSER(1,)

>W X
789
```

**Example 3**

The following are the data values for this example:

- DEA# (#53.2) field = NULL
- FACILITY DEA NUMBER field (#52) = "VA7654321"
- VA# field (#53.3) = NULL

If the FLAG input parameter is NULL or "0", this API would return "":

```
>S X=$$DEA^XUSER(0,)

>W X
```

If the FLAG input parameter is "1", this API would return "":

```
>S X=$$DEA^XUSER(1,)

>W X
```

In both cases, it returns an empty string.

## DIV4^XUSER(): Get User Divisions

**Reference Type** Controlled Subscription

**Category** User

**IA #** 2533

**Description** This API returns all divisions for a user. It returns:

1—If the user has a Division entry in the NEW PERSON file (#200). It indicates that the array of pointers to the Institution file has been defined.

0—The array of pointers to the NSTITUTION file (#4) has *not* been defined.

**Format**

```
DIV4^XUSER(.array[,duz])
```

**Input Parameters**

.array: (required) This parameter is a local variable (i.e., array name) passed by reference.

duz: (optional) The Internal Entry Number (IEN) of the user in the NEW PERSON file (#200). If DUZ is not passed as a parameter, the function defaults to the value of DUZ in the application's partition.

**Output Parameters**

.array: Returns:

1—If the user has a Division entry in the NEW PERSON file (#200). It indicates that the array of pointers to the Institution file has been defined.

The array includes all IENs for the INSTITUTION file (#4) that have been assigned to the user.

The array is defined and left in the application's partition, if the user indicated by the value of the DUZ input parameter has divisions defined in the respective NEW PERSON file (#200) entry. The format is:

ARRAY([^DIC(4 IEN])

0—The array of pointers to the NSTITUTION file (#4) has *not* been defined.

**Example**

```
>S X=$$DIV4^XUSER(.ZZ,duz)
```

## $$LOOKUP^XUSER(): New Person File Lookup

**Reference Type** Supported

**Category** User

**IA #** 2343

**Description** This extrinsic function does a user lookup on the NEW PERSON file (#200) screening out users that are terminated. You are first asked to enter a name of a user in the NEW PERSON file (#200). By default, the function then asks if the correct user name was selected. For example:

```
Select NEW PERSON NAME: KRNUSER,THREE
Is KRNUSER,THREE the one you want? YES//
```

If the optional input parameter is set to "Q" then the second, confirmation prompt is suppressed. The return is in the same format as a call to DIC (i.e., IEN^NAME). Adding new entries is <I>not</I> allowed.

**Format** `$$LOOKUP^XUSER([""])`

**Input Parameters** "": (optional) This optional input parameter does the following:

null—(default)Do *not* suppress the NEW PERSON file (#200) name confirmation prompt for each entry selected.

A—Screen out terminated users.

Q—Suppress the NEW PERSON file (#200) name confirmation prompt for each entry selected.

AQ—Screen out terminated users and suppress the NEW PERSON file (#200) name confirmation prompt for each entry selected.

**Output** returns: Returns the Internal Entry Number (IEN) and NAME of the user in the NEW PERSON file (#200) entered after the "Select NEW PERSON NAME:" prompt (IEN^NAME).

### Example 1

This is an example of a lookup of an active user when *not* passing in the optional "Q" parameter:

```
>S LRDOC=$$LOOKUP^XUSER("")

Select NEW PERSON NAME: ?
 Answer with NEW PERSON NAME, or INITIAL, or SSN, or VERIFY CODE, or
 NICK NAME, or SERVICE/SECTION, or DEA#, or ALIAS
 Do you want the entire 1601-Entry NEW PERSON List? N <Enter> (No)
Select NEW PERSON NAME: KRNUSER,TWO E <Enter>       TK         COMPUTER SPECIALIST
Is KRNUSER,TWO E the one you want? YES// <Enter>

>W LRDOC
1529^KRNUSER,TWO E
```

**Figure 24-1. $$LOOKUP^XUSER: Example showing confirmation prompt**

### Example 2

This is an example of a lookup of an active user when passing in the optional "Q" parameter:

```
>S LRDOC=$$LOOKUP^XUSER("Q")

Select NEW PERSON NAME: KRNUSER,TWO E <Enter>      TK          COMPUTER SPECIALIST

>W LRDOC
1529^KRNUSER,TWO E
```

**Figure 24-2. $$LOOKUP^XUSER: Example suppressing confirmation prompt**

### Example 3

This is an example of a lookup of a terminated user when passing in the optional "A" parameter:

```
>S LRDOC=$$LOOKUP^XUSER("A")

Select NEW PERSON NAME: KRNUSER,EIGHT <Enter>       EK
                 This user was terminated on May 04, 1998
Select NEW PERSON NAME:
```

**Figure 24-3. $$LOOKUP^XUSER: Example of a terminated user**

## $$NAME^XUSER(): Get Name of User

**Reference Type** Supported

**Category** User

**IA #** 2343

**Description** This extrinsic function returns the full name of the specified user in a mixed case displayable format. The user's given name (i.e., First Last) is returned unless a second parameter of "F" is passed in to get the Family name (i.e., Last,First).

**Format** `$$NAME^XUSER(ien[,format])`

**Input Parameters**

ien: (required) Internal Entry Number (IEN) of the provider to be checked in the NEW PERSON file (#200).

format: (optional) This parameter indicates if the user's name should be returned formatted by Family or Given name, respectively. Possible values are:

F—Family (e.g., "Krnuser,Two").

G (default)—Given (e.g., "Two Krnuser").

**Output**

returns: Returns user's family or given name.

### Example 1

Retrieving the user name in Given format:

```
>S X=$$NAME^XUSER(1529)

>W X
Two E Krnuser
```

### Example 2

Retrieving the user name in Family format:

```
>S X=$$NAME^XUSER(1529,"F")

>W X
Krnuser,Two E.
```

## $$PROVIDER^XUSER(): Providers in New Person File

**Reference Type** Supported

**Category** User

**IA #** 2343

**Description** This extrinsic function was requested to be added by the Computerized Patient Record System (CPRS) Development Team. It is used to indicate any provider in the NEW PERSON file (#200). The definition of a provider is any entry in the NEW PERSON file (#200) that doesn't have a termination date. Additional parameters may be added in the future in order to perform other tests/checks.

**Format** `$$PROVIDER^XUSER(ien)`

**Input Parameters** ien: (required) Internal Entry Number (IEN) of the provider to be checked in the NEW PERSON file (#200).

**Output** returns: Returns any of the following codes:

1—Provider has a record and no termination date.

0^TERMINATED^FMDATE—Provider terminated on date indicated.

""—Null, no provider record found.

### Example 1

This is an example of an Active Provider in the NEW PERSON file (#200):

```
>S X=$$PROVIDER^XUSER(1529)

>WRITE X
1
```

### Example 2

This is an example of a Terminated Provider in the NEW PERSON file (#200):

```
>S X=$$PROVIDER^XUSER(957)

>W X
0^TERMINATED^2980504
```

**Example 3**

This is an example of a Provider with no record in the NEW PERSON file (#200), returns a null string:

```
>S X=$$PROVIDER^XUSER(000999999)

>W X

>
```

## $$KCHK^XUSRB(): Check If User Holds Security Key

| | | |
|---|---|---|
| **Reference Type** | Controlled Subscription | |
| **Category** | User | |
| **IA #** | 2120 | |
| **Description** | This extrinsic function checks to see if a user holds a given security key. | |
| **Format** | `$$KCHK^XUSRB(key)` | |
| **Input Parameters** | key: | (required) The name of the security key to be checked. |
| **Output** | returns: | Returns:<br>1—User holds security key.<br>0—User does *not* hold security key. |

**Example 1**

The following example illustrates the results when a user holds the security key input:

```
>S X=$$KCHK^XUSRB("XUPROGMODE")

>W X
1
```

**Example 2**

The following example illustrates the results when a user does *not* hold the security key input:

```
>S X=$$KCHK^XUSRB("XUMGR")

>W X
0
```

## DIVGET^XUSRB2(): Get Divisions for Current User

**Reference Type** Controlled Subscription

**Category** User

**IA #** 4055

**Description** This API retrieves the list of divisions for the current user.

(This was developed as a Broker RPC and all RPCs have as the first parameter the return/output parameter.)

**Format**

```
DIVGET^XUSRB2(ret,ien)
```

**Input Parameters**

ret: (required) Name of the subscripted return array. In every API that is used as an RPC the first parameter is the return array.

ien: (required) The DUZ or user name of the user for whom you are getting the division list.

**Output Parameters**

ret(): Returns a subscripted output array. If + of the value at the first-level 0 subscript of the return value is false, then the user does *not* have any divisions from which to select.

Otherwise, for each division that a user has, a node will be present in the return value, at the first subscript level, starting at zero (0) and incrementing from there. The value of the node is three pieces:

```
ien^division name^station #
```

**Example**

## DIVSET^XUSRB2(): Set Division for Current User

**Reference Type** Controlled Subscription

**Category** User

**IA #** 4055

**Description** This API sets the division for the current user.

(This was developed as a Broker RPC and all RPCs have as the first parameter the return/output parameter.)

**Format**

```
DIVSET^XUSRB2(ret,div)
```

**Input Parameters**

ret: (required) Name of the subscripted return array. In every API that is used as an RPC the first parameter is the return array.

div: (required) This is the division to select. If passed with a leading ` an Internal Entry Number (IEN) is being passed and will be processed as such.

**Output Parameters**

ret(): Returns a Boolean value in the subscripted output array:

True (non-zero)—Division selection is considered successful.

False (zero)—Division selection failed.

**Example**

## USERINFO^XUSRB2(): Get Demographics for Current User

**Reference Type** Controlled Subscription

**Category** User

**IA #** 4055

**Description** This API retrieves various user demographic information for the current user.

(This was developed as a Broker/VistALink RPC and all RPCs have as the first parameter the return/output parameter.)

**Format**

```
USERINFO^XUSRB2(ret)
```

**Input Parameters**

ret: (required) Name of the subscripted return array. In every API that is used as an RPC the first parameter is the return array.

**Output Parameters**

ret(): Returns a subscripted output array:

RET(1)—User's name from the .01 field of the NEW PERSON file (#200).

RET(2)—Concatenated user name from the NAME COMPONENTS file (#20).

RE(3)—Logged on division:

```
ien^name^number
```

RET(4)—User's title from the NEW PERSON file (#200).

RET(5)—User's service section from NEW PERSON file (#200, external format).

RET(6)—User's language from the NEW PERSON file (#200).

RET(7)—User's timeout.

**Example**

# 25. XGF Function Library: Programmer Tools

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 such as: **<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.

## Direct Mode Utilities

### ^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
```

| Demo Function | Associated Direct Mode Utility |
|---|---|
| Cursor/Text Output | IOXY^XGF, SAY^XGF, SAYU^XGF |
| Video Attributes | CHGA^XGF, SETA^XGF |
| Text Windows | CLEAR^XGF, FRAME^XGF, RESTORE^XGF, SAVE^XGF, WIN^XGF |
| Keyboard Reader | $$READ^XGF |
| Setup/Cleanup | CLEAN^XGF, INITKB^XGF, PREP^XGF, RESETKB^XGF |

**Table 25-1. XGF Function Library: Demo functional division**

# Application Program Interface (API)

Several APIs are available for programmers to work with the XGF Function Library. These APIs are described below.

## CHGA^XGF(): Screen Change Attributes

**Reference Type** Supported

**Category** XGF Function Library

**IA #** 3173

**Description** This API changes individual video attributes for subsequent screen WRITEs.

Use this API to change individual video attributes for subsequent output. This API is different from SETA^XGF in that individual video attributes can be set without affecting all video attributes at once.

A call to the PREP^XGF(): Screen/Keyboard Setup API 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 API. 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(): Screen/Keyboard Setup, or SETA^XGF(): Screen Video Attributes API call is made. If you want only a temporary change in attribute, SAY^XGF may be a better function to use.

**Format**

```
CHGA^XGF(atr_codes)
```

**Input Parameters** atr_codes: (required) Codes are as follows:

B1—Blink on
B0—Blink off

E1—Turn all off

G1—Graphics on
G0—Graphics off

I1—Intensity high
I0—Intensity normal

R1—Reverse video on
R0—Reverse video off

U1—Underline on
U0—Underline off

**Output Parameters** 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.

**REF:** See also: SETA^XGF(): Screen Video Attributes API.

**Example 1**

To clear the screen in blinking, reverse video and high intensity, do the following:

```
>D CHGA^XGF("R1B1I1"),CLEAR^XGF(0,0,23,79)
```

**Example 2**

To print Hello World, do the following:

```
>D CHGA^XGF("I1"),SAY^XGF(,,"Hello ")
>D CHGA^XGF("U1"),SAY^XGF(,,"World")
```

**Example 3**

To draw the bottom of a small box, do the following:

```
>D CHGA^XGF("G1")
>D SAY^XGF(,,IOBLC_IOHL_IOHL_IOBRC)
>D CHGA^XGF("G0")
```

## CLEAN^XGF: Screen/Keyboard Exit and Cleanup

**Reference Type** Supported

**Category** XGF Function Library

**IA #** 3173

**Description** This API exits 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: Exit XGF Keyboard API 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 the RESETKB^XGF: Exit XGF Keyboard API is *not* necessary.

**Format** `CLEAN^XGF`

**Input Parameters** none

**Output** none

**REF:** See also: PREP^XGF(): Screen/Keyboard Setup API.

**Example**

# CLEAR^XGF(): Screen Clear Region

**Reference Type** Supported

**Category** XGF Function Library

**IA #** 3173

**Description** This API clears a rectangular region of the screen. It 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 is used, the rectangular region is cleared in the new attribute.

A call to the PREP^XGF(): Screen/Keyboard Setup API 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.

**Format** `CLEAR^XGF(top,left,bottom,right)`

**Input Parameters**

| | |
|---|---|
| top: | (required) Top screen coordinate for box. |
| left: | (required) Left screen coordinate for box. |
| bottom: | (required) Bottom screen coordinate for box. |
| right: | (required) Right screen coordinate for box. |

**Output Parameters**

| | |
|---|---|
| $x and $y: | Set to the right and bottom specified as parameters. |

**REF:** See also: RESTORE^XGF(): Screen Restore, SAVE^XGF(): Screen Save, and WIN^XGF(): Screen Text Window APIs.

### Example 1

For example, to clear the entire screen, do the following:

```
>D CLEAR^XGF(0,0,23,79)
```

### Example 2

To clear a rectangular region in the center of the screen, do the following:

```
>D CLEAR^XGF(5,20,15,60)
```

## FRAME^XGF(): Screen Frame

**Reference Type** Supported

**Category** XGF Function Library

**IA #** 3173

**Description** This API draws a box frame on the screen. It displays 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 the WIN^XGF(): Screen Text Window API instead.

A call to the PREP^XGF(): Screen/Keyboard Setup API 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.

**Format** `FRAME^XGF(top,left,bottom,right)`

**Input Parameters**

| | |
|---|---|
| top: | (required) Top screen coordinate for box. |
| left: | (required) Left screen coordinate for box. |
| bottom: | (required) Bottom screen coordinate for box. |
| right: | (required) Right screen coordinate for box. |

**Output Parameters**

| | |
|---|---|
| $x and $y: | Set to the right and bottom specified as parameters. |

**REF:** See also: RESTORE^XGF(): Screen Restore and WIN^XGF(): Screen Text Window APIs.

### Example

For example, to draw a box in the center of the screen, do the following:

```
>D FRAME^XGF(5,20,15,60)
```

# INITKB^XGF(): Keyboard Setup Only

**Reference Type** Supported

**Category** XGF Function Library

**IA #** 3173

**Description** This API sets up the XGF keyboard environment only. You should call INITKB^XGF once, before you start making calls to the $$READ^XGF function. This API turns on escape processing and any terminators that are passed.

Use this API only if you are using XGF's Keyboard Reader independently from XGF's screen functions. Otherwise, a call to the PREP^XGF(): Screen/Keyboard Setup API does everything to set up keyboard processing that INITKB^XGF does, and a separate call to INITKB^XGF is not necessary.

Unlike the PREP^XGF(): Screen/Keyboard Setup API, 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 the ENS^%ZISS: Set Up Screen-handling Variables API.

**Format**

```
INITKB^XGF([term_str])
```

**Input Parameters**

term_str: (optional) String of characters that should terminate the READ.

This parameter can be one of two forms:

- A single asterisk ("*****") character turns on all terminators.
- The string of terminating characters, such as $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

**REF:** See also: RESETKB^XGF: Exit XGF Keyboard API.

**Example**

# IOXY^XGF(): Screen Cursor Placement

**Reference Type** Supported

**Category** XGF Function Library

**IA #** 3173

**Description** This API positions the cursor on the screen at a screen coordinate. This API 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 the PREP^XGF(): Screen/Keyboard Setup API 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:** You must use quotes to pass a "+" or "-". Otherwise, to specify exact locations for row and column, pass numbers.

**Format** `IOXY^XGF(row,col)`

**Input Parameters**

| | |
|---|---|
| row: | (required) Row position to which the cursor is moved. |
| col: | (required) Column position to which the cursor is moved. |

**Output Parameters**

| | |
|---|---|
| $x and $y: | Set to the row and column specified as parameters. |

**REF:** See also: SAY^XGF(): Screen String and SAYU^XGF(): Screen String with Attributes APIs.

### Example

For example, to position the cursor at row 12, column 39, do the following:

```
>D IOXY^XGF(12,39)
```

## PREP^XGF(): Screen/Keyboard Setup

**Reference Type** Supported

**Category** XGF Function Library

**IA #** 3173

**Description** This API sets up the XGF screen and keyboard environments.

Before using any XGF screen functions, you must call the PREP^XGF API. 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 the INITKB^XGF(): Keyboard Setup Only API does to set up the XGF keyboard environment, including turning escape processing and terminators on. If you call PREP^XGF, a call to the INITKB^XGF(): Keyboard Setup Only API would be redundant.

**Format** `PREP^XGF(xgcuratr)`

**Input Parameters** none

**Output Parameter** xgcuratr: One-character parameter containing the state of the current video attribute.

Also, the GSET^%ZISS: Set Up Graphic Variables API is called, so all output variables for screen graphics from GSET^%ZISS are defined.

**REF:** See also: CLEAN^XGF: Screen/Keyboard Exit and Cleanup API.

**Example**

# $$READ^XGF(): Read Using Escape Processing

**Reference Type** Supported

**Category** XGF Function Library

**IA #** 3173

**Description** This extrinsic function 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, caret ("^") 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 |

**Table 25-2. XGF Function Library: Mnemonics for keys that terminate READs**

**Format** `$$READ^XGF([no_of_char][,timeout])`

**Input Parameters** no_of_char: (optional) Maximum number of characters to READ.

| | | |
|---|---|---|
| | timeout: | (optional) Maximum duration of READ, in seconds. |
| **Output Parameters** | returns: | Returns the string READ from the user. |
| | XGRT: | Set to the mnemonic of the key that terminated the READ. |
| | | **REF:** For a list of possible values, please refer to the list below or the table in routine XGKB. |
| | DTOUT: | If defined, signifies that the READ timed out. |

**Example 1**

To READ a name (with a maximum length of 30) from input and display that name on the screen, do the following:

```
D INITKB^XGF("*")
W "Name: " S NM=$$READ^XGF(30)
D SAY^XGF(10,20,"Hello "_NM)
```

**Figure 25-1. SAY^XGF: Example to READ a name**

**Example 2**

To accept only **<Up-Arrow>** or **<Down-Arrow>** keys to exit a routine, do the following:

```
;Only accept UP or DOWN arrow keys
F  S %=$$READ^XGF(1) Q:XGRT="UP"!(XGRT="DOWN")
```

**Figure 25-2. $$READ^XGF: Example to accept only Up-Arrow and Down-Arrow keys**

**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: Exit XGF Keyboard

**Reference Type** Supported

**Category** XGF Function Library

**IA #** 3173

**Description** This API exits the XGF keyboard environment. You should use the RESETKB^XGF call once you finish making calls to the $$READ^XGF(): Read Using Escape Processing function. The RESETKB^XGF API turns terminators and escape processing off and removes any XGF keyboard environment variables. Subsequent READs are processed normally.

Use this API only if you are using XGF's Keyboard Reader independently from XGF's screen functions. Otherwise, a call to the CLEAN^XGF: Screen/Keyboard Exit and Cleanup API does everything to clean up keyboard processing that the RESETKB^XGF API does, and a separate call to the RESETKB^XGF API is *not* necessary.

Unlike the CLEAN^XGF: Screen/Keyboard Exit and Cleanup API, the RESETKB^XGF API *does not set* the keypad to numeric mode.

**Format** `RESETKB^XGF`

**Input Parameters** none

**Output** none

**REF:** See also: INITKB^XGF(): Keyboard Setup Only API.

**Example**

# RESTORE^XGF(): Screen Restore

**Reference Type** Supported

**Category** XGF Function Library

**IA #** 3173

**Description** This API restores a previously saved screen region. You can save screen regions using the WIN^XGF(): Screen Text Window and SAVE^XGF(): Screen Save APIs. RESTORE^XGF restores the saved screen region in the same screen position as the screen region was saved from.

A call to the PREP^XGF(): Screen/Keyboard Setup API 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 (i.e., 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.

**Format** `RESTORE^XGF(save_root)`

**Input Parameters** save_root: (required) Global/local array node, closed root form.

**Output Parameters** $x and $y: Set to the bottom right coordinate of the restored window.

**REF:** See also: CLEAR^XGF(): Screen Clear Region, SAVE^XGF(): Screen Save, and WIN^XGF(): Screen Text Window APIs.

### Example

To restore the screen contents saved to the local array SELECT to their original position, do the following:

```
>D RESTORE^XGF("SELECT")
```

## SAVE^XGF(): Screen Save

**Reference Type** Supported

**Category** XGF Function Library

**IA #** 3173

**Description** This API saves 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 the PREP^XGF(): Screen/Keyboard Setup API 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 (i.e., 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.

**Format** `SAVE^XGF(top,left,bottom,right,save_root)`

**Input Parameters**

| | |
|---|---|
| top: | (required) Top screen coordinate for box. |
| left: | (required) Left screen coordinate for box. |
| bottom: | (required) Bottom screen coordinate for box. |
| right: | (required) Right screen coordinate for box. |
| save_root: | (required) Global/local array node, closed root form. |

**Output Parameter** $x and $y: Left unchanged.

**REF:** See also: CLEAR^XGF(): Screen Clear Region, RESTORE^XGF(): Screen Restore, and WIN^XGF(): Screen Text Window APIs.

### Example

For example, to save the screen contents between rows 5 and 15 and columns 20 and 60 in the SELECT local array, do the following:

```
>D SAVE^XGF(5,20,15,60,"SELECT")
```

## SAY^XGF(): Screen String

**Reference Type** Supported

**Category** XGF Function Library

**IA #** 3173

**Description** This API outputs a string to the screen (with optional positioning and attribute control).

Use this API 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 the PREP^XGF(): Screen/Keyboard Setup API 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 (e.g., "-5" or "+10").

**NOTE:** 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.

**REF:** For a discussion of valid video attribute codes for the video attribute parameter, please refer to the SETA^XGF(): Screen Video Attributes API.

**Format**

```
SAY^XGF([row][,col,]str[,atr])
```

**Input Parameters**

| | |
|---|---|
| row: | (optional) Row position to start WRITE. |
| col: | (optional) Column position to start WRITE. |
| str: | (required) String to WRITE. |

| | | |
|---|---|---|
| | atr: | (optional) Video attribute with which to WRITE string. |
| | | **REF:** For description of atr codes, please refer to the $$READ^XGF(): Read Using Escape Processing API. |
| **Output Parameters** | $x and $y: | Set to position of the last character output. |

**REF:** See also: IOXY^XGF(): Screen Cursor Placement and SAYU^XGF(): Screen String with Attributes APIs.

**Example 1**

For example, to print "Hello, World" in the center of the screen, in the current video attribute, do the following:

```
>D SAY^XGF(11,35,"Hello World")
```

**Example 2**

To print "ERROR!" at (row,col) position ($X+1,$Y+5), in reverse and bold video attributes, do the following:

```
>D SAY^XGF("+","+5",0,"ERROR!","R1B1")
```

**Example 3**

To print "..." at the current cursor position, in the current video attribute, do the following:

```
>D SAY^XGF(,,"...")
```

## SAYU^XGF(): Screen String with Attributes

**Reference Type** Supported

**Category** XGF Function Library

**IA #** 3173

**Description** This API outputs a string to the screen (with optional position and attribute control), including the ability to underline an individual character.

This API 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 the PREP^XGF(): Screen/Keyboard Setup API 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 (e.g., "-5" or "+10").

**NOTE:** 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.

**Format**

```
SAYU^XGF([row][,col,]str[,atr])
```

**Input Parameters**

| | |
|---|---|
| row: | (optional) Row position to start WRITE. |
| col: | (optional) Column position to start WRITE. |
| str: | (required) String to WRITE ("**&**" underlines next character). |
| atr: | (optional) Video attribute with which to WRITE a string. |

**REF:** For a description of atr codes, please refer to the $$READ^XGF(): Read Using Escape Processing API.

**Output Parameters**

| | |
|---|---|
| $x,$y: | Set to the position of the last character output. |

**REF:** See also: IOXY^XGF(): Screen Cursor Placement and SAY^XGF(): Screen String APIs.

### Example

For example, to print Save at row 5, column 10, do the following:

```
>D SAYU^XGF(5,10,"&Save")
```

## SETA^XGF(): Screen Video Attributes

**Reference Type** Supported

**Category** XGF Function Library

**IA #** 3173

**Description** This API sets all video attribute simultaneously, for subsequent screen output. This API is different from the $$READ^XGF(): Read Using Escape Processing API in that it takes a different form of the attribute argument, and, unlike the CHGA^XGF(): Screen Change Attributes API, it sets all attributes. The change in attribute remains in effect until you make another CHGA^XGF(): Screen Change Attributes, CLEAN^XGF: Screen/Keyboard Exit and Cleanup, or SETA^XGF API call. If you want only a temporary change in attribute, the SAY^XGF(): Screen String API might be a better function to use.

A call to the PREP^XGF(): Screen/Keyboard Setup API must be made at some point prior to calling the SETA^XGF API.

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 xgcuratr parameter, however. Rather than trying to use the SETA^XGF API 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.

**Format**

```
SETA^XGF(atr_code)
```

**Input Parameters**

| | |
|---|---|
| atr_code: | (required) Single character containing the states of all video attributes as the bit values. This argument itself should be derived from a previous call to the PREP^XGF(): Screen/Keyboard Setup, CHGA^XGF(): Screen Change Attributes, or SETA^XGF APIs. |

**Output Parameters**

| | |
|---|---|
| xgcuratr: | This variable always holds the current screen attribute coded as a single character, and is updated when you call SETA^XGF. |
| $x and $y: | Left unchanged. |

**REF:** See also: $$READ^XGF(): Read Using Escape Processing API.

### 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 the following:

```
>D PREP^XGF S SAVEATR=XGCURATR
>D SOME^THING
>D SETA^XGF(SAVEATR)
```

## WIN^XGF(): Screen Text Window

**Reference Type** Supported

**Category** XGF Function Library

**IA #** 3173

**Description** This API opens 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 the PREP^XGF(): Screen/Keyboard Setup API 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 (i.e., closed right parenthesis and with variable references such as $J fully resolved). Using the 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(): Screen Restore API.

**Format**

```
WIN^XGF(top,left,bottom,right[,save_root])
```

**Input Parameters**

| | |
|---|---|
| top: | (required) Top screen coordinate for box. |
| left: | (required) Left screen coordinate for box. |
| bottom: | (required) Bottom screen coordinate for box. |
| right: | (required) Right screen coordinate for box. |
| save_root: | (optional) Global/local array node, closed root form. |

**Output Parameters**

| | |
|---|---|
| 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. |

**REF:** See also: CLEAR^XGF(): Screen Clear Region, FRAME^XGF(): Screen Frame, RESTORE^XGF(): Screen Restore, and SAVE^XGF(): Screen Save APIs.

**Example 1**

To draw an empty box in the center of the screen (and save the underlying screen region under array SELECT), do the following:

```
>D WIN^XGF(5,20,15,60,"SELECT")
```

**Example 2**

To save the same window to a global array (to illustrate the use of $NAME to specify a fully resolved root), do the following:

```
>D WIN^XGF(5,20,15,60,$NA(^TMP($J)))
```

# 26. XLF Function Library: Programmer Tools

## Application Program Interface (API)

Several APIs are available for programmers to work with the XLF Function Library. These APIs are described below.

The XLF Function Library provides the following functions:

- Date
- Hyperbolic Trigonometric
- Mathematical
- Measurement
- String Computations
- Utilities

## CRC Functions—XLFCRC

These functions are provided to help process strings.

### $$CRC16^XLFCRC(): Cyclic Redundancy Code 16

**Reference Type** Supported

**Category** CRC Functions

**IA #** 3156

**Description** This extrinsic function computes a Cyclic Redundancy Code (CRC) of the 8-bit character string, using X^16 + X^15 + X^2 + 1 as the polynomial. The optional parameter "seed" may supply an initial value, which allows for running CRC calculations on multiple strings. If the parameter "seed" is *not* specified, a default value of zero (0) is assumed. The value of "seed" is limited to 0 <= seed <= 2^16. The function value will be between 0 and 2^16.

**Format** `$$CRC16^XLFCRC(string[,seed])`

**Input Parameters**

| | |
|---|---|
| string: | (required) String upon which to compute the CRC16. |
| seed: | (optional) Seed value. Needed to compute the CRC16 over multiple strings. |

**Output**

| | |
|---|---|
| returns: | Returns the Cyclic Redundancy Code (CRC) 16 value. |

### Example

```
SET CRC=$$CRC16^XLFCRC(string)
```

A checksum can also be calculated over multiple strings.

```
SET (I,C)=0
FOR  SET I=$ORDER(X(I)) QUIT:'I  DO
. SET C=$$CRC16^XLFCRC(X(I),C)
```

Or

```
SET I=0,C=4294967295
FOR  SET I=$ORDER(X(I)) QUIT:'I  DO
. SET C=$$CRC16^XLFCRC(X(I),C)
```

As long as the save method is used all the time.

### Example 2

```
CRC162 ;Test call CRC16^XLFCRC multiple times
S TEXT="Now is the time for all good children",TEXT2="to come to the aid of their
country."
S CRC=0,CRC=$$CRC16^XLFCRC(TEXT,CRC)
If 23166=$$CRC16^XLFCRC(TEXT2,CRC) WRITE !,"CRC16 OK"
Q
```

**NOTE:** These have been approved for inclusion in a future ANSI M language standard as part of the library.

## $$CRC32^XLFCRC(): Cyclic Redundancy Code 32

**Reference Type** Supported

**Category** CRC Functions

**IA #** 3156

**Description** This extrinsic function computes a Cyclic Redundancy Code (CRC) of the 8-bit character string, using X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1 as the polynomial. The optional parameter "seed" may supply an initial value, which allows for running CRC calculations on multiple strings. If the parameter "seed" is *not* specified, a default value of 4,294,967,295 (2^32-1) is assumed. The value of "seed" is limited to 0 <= seed <= 2^32. The function value will be between 0 and 2^32.

**Format** `$$CRC32^XLFCRC(string[,seed])`

**Input Parameters**

| | |
|---|---|
| string: | (required) String upon which to compute the CRC32. |
| seed: | (optional) Seed value. Needed to compute the CRC32 over multiple strings. |

**Output**

| | |
|---|---|
| returns: | Returns the Cyclic Redundancy Code (CRC) 32 value. |

**Example**

```
SET CRC=$$CRC32^XLFCRC(string)
```

A checksum can also be calculated over multiple strings.

```
SET (I,C)=0
FOR  SET I=$ORDER(X(I)) QUIT:'I  DO
. SET C=$$CRC32^XLFCRC(X(I),C)
```

Or

```
SET I=0,C=4294967295
FOR  SET I=$ORDER(X(I)) QUIT:'I  DO
. SET C=$$CRC32^XLFCRC(X(I),C)
```

As long as the save method is used all the time.

**Example 2**

```
CRC322 ;Test call CRC32^XLFCRC multiple times
S TEXT="Now is the time for all good children",TEXT2="to come to the aid of their
country."
S CRC=0,CRC=$$CRC32^XLFCRC(TEXT,CRC)
If 715820230=$$CRC32^XLFCRC(TEXT2,CRC) WRITE !,"CRC32 OK"
Q
```

**NOTE:** These have been approved for inclusion in a future ANSI M language standard as part of the library.

# Date Functions—XLFDT

## $$%H^XLFDT(): Convert Seconds to $H

**Reference Type** Supported

**Category** Date Functions

**IA #** 10103

**Description** This extrinsic function converts the number of seconds input to a $H formatted date. It converts the output of the $$SEC^XLFDT(): Convert $H/VA FileMan date to Seconds API back to a $H value.

**Format** `$$%H^XLFD(seconds)`

**Input Parameters** seconds: (required) Input seconds.

**Output** returns: Returns seconds in $H date format.

**Example**

```
>S X=$$%H^XLFDT(5108536020)

>W X
59126,49620
```

## $$DOW^XLFDT(): Day of Week

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Date Functions | |
| **IA #** | 10103 | |
| **Description** | This extrinsic function returns the corresponding day of the week from a date in VA FileMan format. | |
| **Format** | `$$DOW^XLFD(x[,y])` | |
| **Input Parameters** | x: | (required) VA FileMan date. |
| | y: | (optional) 1 to return a day-of-week number. |
| **Output** | returns: | Returns the day of the week. |

### Example 1

```
>S X=$$DOW^XLFDT(2901231.111523)

>W X
Monday
```

### Example 2

```
>S X=$$DOW^XLFDT(2901231.111523,1)

>W X
1
```

## $$DT^XLFDT: Current Date (VA FileMan Date Format)

**Reference Type** Supported

**Category** Date Functions

**IA #** 10103

**Description** This extrinsic function returns the current date in VA FileMan format.

**Format** `$$DT^XLFDT`

**Input Parameters** none

**Output** returns: Returns the current date in VA FileMan format.

**Example**

```
>S X=$$DT^XLFDT

>W X
3040126
```

## $$FMADD^XLFDT(): VA FileMan Date Add

**Reference Type** Supported

**Category** Date Functions

**IA #** 10103

**Description** This extrinsic function returns the result of adding days, hours, minutes, and seconds to a date in VA FileMan format.

**Format** `$$FMADD^XLFDT(x,d,h,m,s)`

**Input Parameters**

| | |
|---|---|
| x: | (required) VA FileMan date (in quotes). |
| d: | (required) Days. |
| h: | (required) Hours. |
| m: | (required) Minutes. |
| s: | (required) Seconds. |

**Output** returns: Returns the updated date and time in VA FileMan format.

**Example**

```
>S X=$$FMADD^XLFDT(2901231.01,2,2,20,15)

>W X
2910102.032015
```

## $$FMDIFF^XLFDT(): VA FileMan Date Difference

**Reference Type** Supported

**Category** Date Functions

**IA #** 10103

**Description** This extrinsic function returns the difference between two VA FileMan format dates.

**Format** `$$FMDIFF^XLFDT(x1,x2[,x3])`

**Input Parameters**

x1: (required) VA FileMan date.

x2: (required) VA FileMan date, to subtract from the x1 date.

x3: (optional) If null ('$D(x3), return the difference in days. Otherwise:

- If x3 = 1, return the difference in days.
- If x3 = 2, return the difference in seconds.
- If x3 = 3, return the difference in days hours:minutes:seconds format (DD HH:MM:SS).

**Output** returns: Returns the date and/or time difference.

**Example 1**

The following example returns the difference between two dates/times in days (x3 = null or 1). In this example, the first date is 2 days less than the second date:

```
>S X=$$FMDIFF^XLFDT(2901229,2901231.111523)

>W X
-2

>S X=$$FMDIFF^XLFDT(2901229,2901231.111523,1)

>W X
-2
```

### Example 2

The following example returns the difference between two dates/times in seconds (x3 = 2). In this example, the first date is 150,079 seconds greater than the second date:

```
>S X=$$FMDIFF^XLFDT(2901231.111523,2901229.173404,2)

>W X
150079
```

### Example 3

The following example returns the difference between two dates/times in DD HH:MM:SS (x3 = 3). In this example, the first date is 1 day, 1 hour, 24 minutes, and 2 seconds greater than the second date:

```
>S X=$$FMDIFF^XLFDT(2901231.024703,2901230.012301,3)

>W X
1 1:24:2
```

## $$FMTE^XLFDT(): Convert VA FileMan Date to External Format

**Reference Type** Supported

**Category** Date Functions

**IA #** 10103

**Description** This extrinsic function converts a VA FileMan formatted input date to an external formatted date.

**Format** `$$FMTE^XLFDT(x[,y])`

**Input Parameters**

x: (required) 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 = 5 then return MM/DD/YYYY@HH:MM:SS format.
- If +y = 6 then return DD/MM/YYYY@HH:MM:SS format.
- If +y = 7 then return YYYY/MM/DD@HH:MM:SS format.
- If y contains a "D" then date only.
- If y contains an "F" then output date with leading spaces.
- If y contains an "M" then only output "HH:MM".
- If y contains a "P" then output "HH:MM:SS am/pm".
- If y contains an "S" then force seconds in the output.
- If y contains a "Z" then output date with leading zeroes.

**Output**

returns: Returns the external formatted date.

### Example 1

Return the date in the following format: Standard VA FileMan date format.

```
>S X=$$FMTE^XLFDT(2940629.105744,1)

>W X
Jun 29, 1994@10:57:44
```

### Example 2

Return the date in the following format: Standard VA FileMan date format and include am/pm.

```
>S X=$$FMTE^XLFDT(2940629.1057,"1P")

>W X
Jun 29, 1994 10:57 am
```

### Example 3

Return the date in the following format: MM/DD/YY@HH:MM:SS.

```
>S X=$$FMTE^XLFDT(2940629.105744,2)

>W X
6/29/94@10:57:44
```

### Example 4

Return the date in the following format: MM/DD/YY@HH:MM.

```
>S X=$$FMTE^XLFDT(2940629.105744,"2M")

>W X
6/29/94@10:57
```

### Example 5

Return the date in the following format: MM/DD/YY@HH:MM:SS and include am/pm.

```
>S X=$$FMTE^XLFDT(2940629.105744,"2P")

>W X
6/29/94 10:57:44 am
```

**Example 6**

Return the date in the following format: MM/DD/YY@HH:MM:SS, forcing seconds to display when no seconds were included in the input parameter.

```
>S X=$$FMTE^XLFDT(2940629.1057,"2S")

>W X
6/29/94@10:57:00
```

**Example 7**

Return the date in the following format: MM/DD/YY@HH:MM:SS, forcing seconds to display when no seconds were included in the input parameter, and include leading spaces.

```
>S X=$$FMTE^XLFDT(2940629.1057,"2SF")

>W X
 6/29/94@10:57:00
```

**Example 8**

Return the date in the following format: DD/MM/YY@HH:MM:SS and include leading spaces.

```
>S X=$$FMTE^XLFDT(2940629.105744,"3F")

>W X
29/ 6/94@10:57:44
```

**Example 9**

Return the date in the following format: YY/MM/DD, ignore the time values entered and only display the date.

```
>S X=$$FMTE^XLFDT(2940629.1057,"4D")

>W X
94/6/29
```

**Example 10**

To output a really short date/time try the following, convert space to zero and remove slash, as shown below:

```
>S X=$TR($$FMTE^XLFDT(2940629.1057,"4F")," /","0")

>W X
940629@10:57
```

### Example 11

Return the date in the following format: MM/DD/YYYY@HH:MM:SS.

```
>S X=$$FMTE^XLFDT(3000229.110520,5)

>W X
2/29/2000@11:05:20
```

### Example 12

Return the date in the following format: MM/DD/YYYY@HH:MM:SS and include leading spaces.

```
>S X=$$FMTE^XLFDT(3000229.110520,"5F")

>W X
 2/29/2000@11:05:20
```

### Example 13

Return the date in the following format: MM/DD/YYYY@HH:MM:SS, forcing seconds.

```
>S X=$$FMTE^XLFDT(3000229.1105,"5S")

>W X
2/29/2000@11:05:00
```

### Example 14

Return the date in the following format: MM/DD/YYYY HH:MM:SS, include leading zeroes, and include am/pm.

```
>S X=$$FMTE^XLFDT(3000229.110520,"5ZP")

>W X
02/29/2000 11:05:20 am
```

### Example 15

Return the date in the following format: DD/MM/YYYY@HH:MM:SS, with leading spaces.

```
>S X=$$FMTE^XLFDT(3000229.110520,"6F")

>W X
29/ 2/2000@11:05:20
```

**Example 16**

Return the date in the following format: DD/MM/YYYY@HH:MM:SS, with leading zeroes.

```
>S X=$$FMTE^XLFDT(3000229.1105,"6Z")

>W X
29/02/2000@11:05
```

**Example 17**

Return the date in the following format: YYYY/MM/DD@HH:MM:SS.

```
>S X=$$FMTE^XLFDT(3000301.1105,7)

>W X
2000/3/1@11:05
```

**Example 18**

Return the date in the following format: YYYY/MM/DD, ignore the time values entered and only display the date.

```
>S X=$$FMTE^XLFDT(3000301.1105,"7D")

>W X
2000/3/1
```

## $$FMTH^XLFDT(): Convert VA FileMan Date to $H

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Date Functions | |
| **IA #** | 10103 | |
| **Description** | This extrinsic function converts a VA FileMan formatted input date to a $H formatted date. | |
| **Format** | `$$FMTH^XLFDT(x[,y])` | |
| **Input Parameters** | x: | (required) VA FileMan date. |
| | y: | (optional) 1 to return the date portion only (no seconds). |
| **Output** | returns: | Returns the converted date in $H format. |

**Example 1**

```
>S X=$$FMTH^XLFDT(2901231.111523)

>W X
54786,40523
```

**Example 2**

```
>S X=$$FMTH^XLFDT(2901231.111523,1)

>W X
54786
```

## $$FMTHL7^XLFDT(): Convert VA FileMan Date to HL7 Date

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Date Functions | |
| **IA #** | 10103 | |
| **Description** | This extrinsic function converts a VA FileMan formatted input date/time into an HL7 formatted date, including the time offset. | |
| **Format** | `$$FMTHL7^XLFDT(fm_date_time)` | |
| **Input Parameters** | fm_date_time: | (required) VA FileMan date. |
| **Output** | returns: | Returns the converted date in HL7 format. |

**Example**

```
>S X=$$FMTHL7^XLFDT(3001127.1525)

>W X
200011271525-0800
```

## $$HADD^XLFDT(): $H Add

**Reference Type** Supported

**Category** Date Functions

**IA #** 10103

**Description** This extrinsic function returns the result of adding days, hours, minutes, and seconds to a date in $H format.

**Format** `$$HADD^XLFDT(x,d,h,m,s)`

**Input Parameters**

| | |
|---|---|
| x: | (required) $H date (in quotes). |
| d: | (required) Days. |
| h: | (required) Hours. |
| m: | (required) Minutes. |
| s: | (required) Seconds. |

**Output**

| | |
|---|---|
| returns: | Returns the resultant date in $H format. |

**Example**

```
>S X=$$HADD^XLFDT("54786,3600",2,2,20,15)

>W X
54788,12015
```

# $$HDIFF^XLFDT(): $H Difference

**Reference Type** Supported

**Category** Date Functions

**IA #** 10103

**Description** This extrinsic function returns the difference between two $H formatted dates.

**Format** `$$HDIFF^XLFDT(x1,x2[,x3])`

**Input Parameters**

x1: (required) $H date (in quotes).

x2: (required) $H date (in quotes) to subtract from the x1 date.

x3: (optional) If null ('$D(x3), return the difference in days. Otherwise:

- If x3 = 1, return the difference in days.
- If x3 = 2, return the difference in seconds.
- If x3 = 3, return the difference in days hours:minutes:seconds format (DD HH:MM:SS).

**Output** returns: Returns the $H difference.

### Example 1

Return the &H difference in days.

```
>S X=$$HDIFF^XLFDT("54789,40523","54786,25983",1)

>W X
3
```

### Example 2

Return the &H difference in seconds.

```
>S X=$$HDIFF^XLFDT("54789,40523","54786,25983",2)

>W X
273740
```

**Example 3**

Return the &H difference in days hours:minutes:seconds format (DD HH:MM:SS).

```
>S X=$$HDIFF^XLFDT("54789,40523","54786,25983",3)

>W X
3 4:02:20
```

## $$HL7TFM^XLFDT(): Convert HL7 Date to VA FileMan Date

**Reference Type** Supported

**Category** Date Functions

**IA #** 10103

**Description** This extrinsic function converts an HL7 formatted input date/time into a VA FileMan formatted date/time.

**Format** `$$HL7TFM^XLFDT(hl7_date_time[,local_uct][,time_flag])`

**Input Parameters**

hl7_date_time: (required) HL7 formatted date and time.

local_uct: (optional) This parameter controls if any time offset is applied to the time. If a time offset is included, then time offset can be applied to give Local time or Coordinated Universal Time (UTC, a.k.a. GMT, or Greenwich Mean Time) time offset from the MAILMAN TIME ZONE file (#4.4). The default is to return Local time. Valid values are:

- L (default)—Local time.
- U—UTC time.

time_flag: (optional) This parameter is set to 1 if the value in the hl7_date_time input parameter is just a time value. The default assumes that the hl7_date_time input parameter is a date and time value.

**Output** returns: Returns the converted date in VA FileMan format.

**Example 1**

To get date with no offset:

```
>S X=$$HL7TFM^XLFDT("200011271525-0700")

>W X
3001127.1525
```

### Example 2

To get UCT time offset:

```
>S X=$$HL7TFM^XLFDT("200011271525-0700","U")

>W X
3001127.2225
```

### Example 3

To get Local time in PST offset:

```
>S X=$$HL7TFM^XLFDT("200011271525-0700","L")

>W X
3001127.1425
```

### Example 4

To get Local time when only providing a time (no date) as the input parameter:

```
>S X=$$HL7TFM^XLFDT("1525-0700","L",1)

>W X
.1525
```

## $$HTE^XLFDT(): Convert $H to External Format

**Reference Type** Supported

**Category** Date Functions

**IA #** 10103

**Description** This extrinsic function converts a $H formatted input date to an external formatted date.

**Format** `$$HTE^XLFDT(x[,y])`

**Input Parameters**

x: (required) $H date (in quotes).

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 = 5 then return MM/DD/YYYY@HH:MM:SS format.
- If +y = 6 then return DD/MM/YYYY@HH:MM:SS format.
- If +y = 7 then return YYYY/MM/DD@HH:MM:SS format.
- If y contains a "D" then date only.
- If y contains a "F" then output date with leading blanks.
- If y contains a "M" then output "HH:MM" only.
- If y contains a "P" then output "HH:MM:SS am/pm".
- If y contains a "S" then force seconds in the output.
- If y contains a "Z" then output date with leading zeroes.

**Output**

returns: Returns the external format of a $H date.

### Example 1

Return the date in the following format: Standard external format.

```
>S X=$$HTE^XLFDT("54786,40523")

>W X
Dec 31, 1990@11:15:23
```

### Example 2

Return the date in the following format: MM/DD/YY@HH:MM:SS.

```
>S X=$$HTE^XLFDT("54786,40523",2)

>W X
12/31/90@11:15:23
```

### Example 3

Return the date in the following format: MM/DD/YY@HH:MM:SS, omitting the seconds.

```
>S X=$$HTE^XLFDT("57386,33723","2M")

>W X
2/12/98@09:22
```

### Example 4

Return the date in the following format: MM/DD/YYYY@HH:MM:SS.

```
>S X=$$HTE^XLFDT("57351,27199",5)

>W X
1/8/1998@07:33:19
```

### Example 5

Return the date in the following format: DD/MM/YYYY@HH:MM:SS.

```
>S X=$$HTE^XLFDT("57351,27199",6)

>W X
8/1/1998@07:33:19
```

**Example 6**

Return the date in the following format: YYYY/MM/DD@HH:MM:SS.

```
>S X=$$HTE^XLFDT("57351,27199",7)

>W X
1998/1/8@07:33:19
```

## $$HTFM^XLFDT(): Convert $H to VA FileMan Date Format

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Date Functions | |
| **IA #** | 10103 | |
| **Description** | This extrinsic function converts a $H formatted input date to a VA FileMan formatted date. | |
| **Format** | `$$HTFM^XLFDT(x[,y])` | |
| **Input Parameters** | x: | (required) $H date (in quotes). |
| | y: | (optional) 1 to return the date portion only (no seconds). |
| **Output** | returns: | Returns the converted $H date in VA FileMan format. |

**Example 1**

```
>S X=$$HTFM^XLFDT("54786,40523")

>W X
2901231.111523
```

**Example 2**

```
>S X=$$HTFM^XLFDT("54786,40523",1)

>W X
2901231
```

# $$NOW^XLFDT: Current Date and Time (VA FileMan Format)

**Reference Type** Supported

**Category** Date Functions

**IA #** 10103

**Description** This extrinsic function returns the current date and time in VA FileMan format.

**Format** `$$NOW^XLFDT`

**Input Parameters** none

**Output** returns: Returns the current date and time in VA FileMan format.

**Example**

```
>S X=$$NOW^XLFDT

>W X
3040126.103044
```

## $$SCH^XLFDT(): Next Scheduled Runtime

**Reference Type** Supported

**Category** Date Functions

**IA #** 10103

**Description** This extrinsic function returns the next run-time based on Schedule code.

**Format** `$$SCH^XLFDT(schedule_string,base_date[,force_future_flag])`

**Input Parameters** schedule_string: (required) 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.
- nM(list)—Complex month increment. For example: 1M(15,L), which means schedule it to run every month (1M) on the 15 and last day of the month (15,L).
  - dd[@time]—Day of month (e.g., 12).
  - nDay[@time]—day of week in month (e.g., 1M, first Monday); (see "Day Code" list that follows).
  - Day.
  - L—Last day of month.
  - LDay—Last specific day in month (e.g., LM [last Monday],LT [last Tuesday],LW [last Wednesday]...).
- Day[@time]—Day of week (see "Day Code" list that follows).
  - Day.
  - D—Every weekday.
  - E—Every weekend day (Saturday, Sunday).

**Day Code (used in schedule codes above)**

- M—Monday
- T—Tuesday
- W—Wednesday
- R—Thursday
- F—Friday
- S—Saturday
- U—Sunday

base_date: (required) VA FileMan date to which the interval is added.

force_future_flag: (optional) If passed with a 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.

**Output** returns: Returns the next run-time.

**Example 1**

To schedule something to run every month on the 15th of the month at 2:00 p.m. and on the last day of every month at 6:00 p.m., you would enter the following:

```
>S X=$$SCH^XLFDT("1M(15@2PM,L@6PM)",2931003)

>W X
2931015.14
```

**Example 2**

To schedule something to run every month on the 15th of the month at 2:00 p.m. and on the last day of every month at 6:00 p.m., you would enter the following:

```
>S X=$$SCH^XLFDT("1M(15@2PM,L@6PM)",2931028)

>W X
2931031.18
```

**Example 3**

To schedule something to run every 3 months on the last day of every month at 6:00 p.m., you would enter the following:

```
>S X=$$SCH^XLFDT("3M(L@6PM)",2930931)

>W X
2931231.18
```

## $$SEC^XLFDT(): Convert $H/VA FileMan date to Seconds

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Date Functions | |
| **IA #** | 10103 | |
| **Description** | This extrinsic function converts a $H or VA FileMan formatted input date to the number of seconds. The input date can be entered as either a VA FileMan date or a $H date. If entered as a VA FileMan date, the date is first converted to $H via the $$FMTH^XLFDT(): Convert VA FileMan Date to $H API. | |
| **Format** | `$$SEC^XLFDT(x)` | |
| **Input Parameters** | x: | (required) VA FileMan or $H date. |
| **Output** | returns: | Returns the $H date in seconds. |

**Example 1**

Inputting a VA FileMan date/time:

```
>S X=$$SEC^XLFDT(3021118.1347)

>W X
5108536020
```

**Example 2**

Inputting a $H date:

```
>S X=$$SEC^XLFDT($H)

>W X
5146022146
```

# $$TZ^XLFDT: Time Zone Offset (GMT)

**Reference Type** Supported

**Category** Date Functions

**IA #** 10103

**Description** This extrinsic function returns the Time Zone offset from Greenwich mean time (GMT) based on a pointer from the TIME ZONE field (#1) in the MAILMAN SITE PARAMETERS file (#4.3) to the MAILMAN TIME ZONE file (#4.4).

The accuracy of this value is dependent on IRM updating the TIME ZONE field (#1) in the MAILMAN SITE PARAMETERS file (#4.3) to accurately point to the site's correct time zone, including whether it is standard time (ST) or daylight savings time (DST).

**Format** `$$TZ^XLFDT`

**Input Parameters** none

**Output** returns: Returns the Time Zone offset from GMT.

**Example**

For Pacific Daylight Savings Time (PDT), the offset from GMT is:

```
>S X = $$TZ^XLFDT

>W X
-0700
```

## $$WITHIN^XLFDT(): Checks Dates/Times Within Schedule

**Reference Type** Supported

**Category** Date Functions

**IA #**

**Description** This extrinsic function returns whether or not a date/time is within a specified schedule string.

**Format** `$$WITHIN^XLFDT(schedule_string,base_date)`

**Input Parameters**

| | |
|---|---|
| schedule_string: | (required) Interval to add to base_date, see the $$SCH^XLFDT(): Next Scheduled Runtime API for alternate values. |
| base_date: | (required) VA FileMan date checked to determine if it is within the input schedule string. |

**Output** returns: Returns

**Example**

# Hyperbolic Trigonometric Functions—XLFHYPER

The following hyperbolic trigonometric functions provide an additional set of mathematical operations beyond the math functions in XLFMTH.

**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^XLFHYPER(): Hyperbolic Arc-cosine

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Hyperbolic Trigonometric Functions | |
| **IA #** | 10144 | |
| **Description** | This extrinsic function returns the hyperbolic arc-cosine, with radians output. | |
| **Format** | `$$ACOSH^XLFHYPER(x[,n])` | |
| **Input Parameters** | x: | (required) Number for which you want the hyperbolic arc-cosine. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the hyperbolic arc-cosine. |

**Example**

```
>S X=$$ACOSH^XLFHYPER(3,12)

>W X
1.762747174
```

## $$ACOTH^XLFHYPER(): Hyperbolic Arc-cotangent

**Reference Type** Supported

**Category** Hyperbolic Trigonometric Functions

**IA #** 10144

**Description** This extrinsic function returns the hyperbolic arc-cotangent, with radians output.

**Format** `$$ACOTH^XLFHYPER(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the hyperbolic arc-cotangent. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the hyperbolic arc-cotangent. |

**Example**

```
>S X=$$ACOTH^XLFHYPER(3,12)

>W X
.34657359025
```

## $$ACSCH^XLFHYPER(): Hyperbolic Arc-cosecant

**Reference Type** Supported

**Category** Hyperbolic Trigonometric Functions

**IA #** 10144

**Description** This extrinsic function returns the hyperbolic arc-cosecant, with radians output.

**Format** `$$ACSCH^XLFHYPER(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the hyperbolic arc-cosecant. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the hyperbolic arc-cosecant. |

**Example**

```
>S X=$$ACSCH^XLFHYPER(3,12)

>W X
.3274501502
```

## $$ASECH^XLFHYPER(): Hyperbolic Arc-secant

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Hyperbolic Trigonometric Functions | |
| **IA #** | 10144 | |
| **Description** | This extrinsic function returns the hyperbolic arc-secant, with radians output. | |
| **Format** | `$$ASECH^XLFHYPER(x[,n])` | |
| **Input Parameters** | x: | (required) Number for which you want the hyperbolic arc-secant. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the hyperbolic arc-secant. |

**Example**

```
>S X=$$ASECH^XLFHYPER(.3,12)

>W X
1.8738202425
```

## $$ASINH^XLFHYPER(): Hyperbolic Arc-sine

**Reference Type** Supported

**Category** Hyperbolic Trigonometric Functions

**IA #** 10144

**Description** This extrinsic function returns the hyperbolic arc-sine, with radians output.

**Format** `$$SINH^XLFHYPER(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the hyperbolic arc-sine. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns | Returns the hyperbolic arc-sine. |

**Example**

```
>S X=$$SINH^XLFHYPER(3,12)

>W X
10.0178749273
```

## $$ATANH^XLFHYPER(): Hyperbolic Arc-tangent

**Reference Type** Supported

**Category** Hyperbolic Trigonometric Functions

**IA #** 10144

**Description** This extrinsic function returns the hyperbolic arc-tangent, with radians output.

**Format** `$$ATANH^XLFHYPER(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the hyperbolic arc-tangent. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the hyperbolic arc-tangent. |

**Example**

```
>S X=$$ATANH^XLFHYPER(.3,12)

>W X
.3095196042
```

## $$COSH^XLFHYPER(): Hyperbolic Cosine

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Hyperbolic Trigonometric Functions | |
| **IA #** | 10144 | |
| **Description** | This extrinsic function returns the hyperbolic arc-cosine, with radians output. | |
| **Format** | `$$COSH^XLFHYPER(x[,n])` | |
| **Input Parameters** | x: | (required) Number for which you want the hyperbolic cosine. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the hyperbolic cosine. |

**Example**

```
>S X=$$COSH^XLFHYPER(3,12)

>W X
10.0676619957
```

## $$COTH^XLFHYPER(): Hyperbolic Cotangent

**Reference Type** Supported

**Category** Hyperbolic Trigonometric Functions

**IA #** 10144

**Description** This extrinsic function returns the hyperbolic cotangent, with radians output.

**Format** `$$COTH^XLFHYPER(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the hyperbolic cotangent. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the hyperbolic cotangent. |

**Example**

```
>S X=$$COTH^XLFHYPER(3,12)

>W X
1.00496982332
```

## $$CSCH^XLFHYPER(): Hyperbolic Cosecant

**Reference Type** Supported

**Category** Hyperbolic Trigonometric Functions

**IA #** 10144

**Description** This extrinsic function returns the hyperbolic cosecant, with radians output.

**Format** `$$CSCH^XLFHYPER(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the hyperbolic cosecant. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the hyperbolic cosecant. |

**Example**

```
>S X=$$CSCH^XLFHYPER(3,12)

>W X
.09982156967
```

## $$SECH^XLFHYPER(): Hyperbolic Secant

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Hyperbolic Trigonometric Functions | |
| **IA #** | 10144 | |
| **Description** | This extrinsic function returns the hyperbolic secant, with radians output. | |
| **Format** | `$$SECH^XLFHYPER(x[,n])` | |
| **Input Parameters** | x: | (required) Number for which you want the hyperbolic secant. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the hyperbolic secant. |

**Example**

```
>S X=$$SECH^XLFHYPER(3,12)

>W X
.09932792742
```

# $$SINH^XLFHYPER(): Hyperbolic Sine

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Hyperbolic Trigonometric Functions | |
| **IA #** | 10144 | |
| **Description** | This extrinsic function returns the hyperbolic sine, with radians output. | |
| **Format** | `$$SINH^XLFHYPER(x[,n])` | |
| **Input Parameters** | x: | (required) Number for which you want the hyperbolic sine. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the hyperbolic sine. |

### Example 1

```
>S X=$$SINH^XLFHYPER(.707)

>W X
.767388542
```

### Example 2

```
>S X=$$SINH^XLFHYPER(.3,12)

>W X
.30452029345
```

# $$TANH^XLFHYPER(): Hyperbolic Tangent

**Reference Type** Supported

**Category** Hyperbolic Trigonometric Functions

**IA #** 10144

**Description** This extrinsic function returns the hyperbolic tangent of **x** (tan **x** = sin **x**/cos **x**), with radians output.

**Format** `$$TANH^XLFHYPER(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the hyperbolic tangent. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the hyperbolic tangent. |

**Example**

```
>S X=$$TANH^XLFHYPER(3,12)

>W X
.99505475368
```

# Mathematical 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.

**NOTE:** Each optional parameter in brackets **[ ]** denotes the maximum and default precision for the function. Precision means the detail of the result, in terms of number of digits.

## $$ABS^XLFMTH(): Absolute Value

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Math Functions | |
| **IA #** | 10105 | |
| **Description** | This extrinsic function returns the absolute value of the number in x. | |
| **Format** | `$$ABS^XLFMTH(x)` | |
| **Input Parameters** | x: | (required) Number for which you want the absolute value. |
| **Output** | returns: | Returns the absolute value of a number. |

**Example**

```
>S X=$$ABS^XLFMTH(-42.45)

>W X
42.45
```

## $$ACOS^XLFMTH(): Arc-cosine (Radians)

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the arc-cosine, with radians output.

**Format** `$$ACOS^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the arc-cosine in radians. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the arc-cosine of a number output in radians. |

**Example**

```
>S X=$$ACOS^XLFMTH(.5)

>W X
1.047197551
```

## $$ACOSDEG^XLFMTH(): Arc-cosine (Degrees)

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the arc-cosine, with degrees output.

**Format** `$$ACOSDEG^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the arc-cosine in degrees. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the arc-cosine of a number output in degrees. |

**Example**

```
>S X=$$ACOSDEG^XLFMTH(.5)

>W X
60
```

## $$ACOT^XLFMTH(): Arc-cotangent (Radians)

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Math Functions | |
| **IA #** | 10105 | |
| **Description** | This extrinsic function returns the arc-cotangent, with radians output. | |
| **Format** | `$$ACOT^XLFMTH(x[,n])` | |
| **Input Parameters** | x: | (required) Number for which you want the arc-cotangent in radians. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the arc-cotangent of a number output in radians. |

**Example**

```
>S X=$$ACOT^XLFMTH(.5)

>W X
1.107148718
```

## $$ACOTDEG^XLFMTH(): Arc-cotangent (Degrees)

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the arc-cotangent, with degrees output.

**Format** `$$ACOTDEG^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the arc-cotangent in degrees. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the arc-cotangent of a number output in degrees. |

**Example**

```
>S X=$$ACOTDEG^XLFMTH(.5)

>W X
63.43494882
```

## $$ACSC^XLFMTH(): Arc-cosecant (Radians)

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the arc-cosecant, with radians output.

**Format** `$$ACSC^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the arc-cosecant in radians. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the arc-cosecant of a number output in radians. |

**Example**

```
>S X=$$ACSC^XLFMTH(1.5)

>W X
.729727656
```

## $$ACSCDEG^XLFMTH(): Arc-cosecant (Degrees)

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Math Functions | |
| **IA #** | 10105 | |
| **Description** | This extrinsic function returns the arc-cosecant, with degrees output. | |
| **Format** | `$$ACSCDEG^XLFMTH(x[,n])` | |
| **Input Parameters** | x: | (required) Number for which you want the arc-cosecant in degrees. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the arc-cosecant of a number output in degrees. |

**Example**

```
>S X=$$ACSCDEG^XLFMTH(1.5)

>W X
41.8103149
```

## $$ASEC^XLFMTH(): Arc-secant (Radians)

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the arc-secant, with radians output.

**Format** `$$ASEC^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the arc-secant in radians. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the arc-secant of a number output in radians. |

**Example**

```
>S X=$$ASEC^XLFMTH(1.5)

>W X
.841068671
```

## $$ASECDEG^XLFMTH(): Arc-secant (Degrees)

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the arc-secant, with degrees output.

**Format** `$$ASECDEG^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the arc-secant in degrees. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the arc-secant of a number output in degrees. |

**Example**

```
>S X=$$ASECDEG^XLFMTH(1.5)

>W X
48.1896851
```

## $$ASIN^XLFMTH(): Arc-sine (Radians)

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Math Functions | |
| **IA #** | 10105 | |
| **Description** | This extrinsic function returns the arc-sine, with radians output. | |
| **Format** | `$$ASIN^XLFMTH(x[,n])` | |
| **Input Parameters** | x: | (required) Number for which you want the arc-sine in radians. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the arc-sine of a number output in radians. |

**Example**

```
>S X=$$ASIN^XLFMTH(.5)

>W X
.523598776
```

## $$ASINDEG^XLFMTH(): Arc-sine (Degrees)

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the arc-sine, with degrees output.

**Format** `$$ASINDEG^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the arc-sine in degrees. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the arc-sine of a number output in degrees. |

**Example**

```
>S X=$$ASINDEG^XLFMTH(.5)

>W X
30
```

## $$ATAN^XLFMTH(): Arc-tangent (Radians)

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the arc-tangent, with radians output.

**Format** `$$ATAN^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the arc-tangent in radians. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the arc-tangent of a number output in radians. |

**Example**

```
>S X=$$ATAN^XLFMTH(.5)

>W X
.463647609
```

## $$ATANDEG^XLFMTH(): Arc-tangent (Degrees)

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Math Functions | |
| **IA #** | 10105 | |
| **Description** | This extrinsic function returns the arc-tangent, with degrees output. | |
| **Format** | `$$ATANDEG^XLFMTH(x[,n])` | |
| **Input Parameters** | x: | (required) Number for which you want the arc-tangent in degrees. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the arc-tangent of a number output in degrees. |

**Example**

```
>S X=$$ATANDEG^XLFMTH(.5)

>W X
26.56505118
```

## $$COS^XLFMTH(): Cosine (Radians)

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the cosine, with radians input.

**Format** `$$COS^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Radians input number for which you want the cosine. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the cosine of radians input number. |

**Example**

```
>S X=$$COS^XLFMTH(1.5)

>W X
.070737202
```

## $$COSDEG^XLFMTH(): Cosine (Degrees)

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the cosine, with degrees input.

**Format** `$$COSDEG^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Degrees input number for which you want the cosine. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the cosine of degrees input number. |

**Example**

```
>S X=$$COSDEG^XLFMTH(45)

>W X
.707106781
```

## $$COT^XLFMTH(): Cotangent (Radians)

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Math Functions | |
| **IA #** | 10105 | |
| **Description** | This extrinsic function returns the cotangent, with radians input. | |
| **Format** | `$$COT^XLFMTH(x[,n])` | |
| **Input Parameters** | x: | (required) Radians input number for which you want the cotangent. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the cotangent of radians input number. |

**Example**

```
>S X=$$COT^XLFMTH(1.5)

>W X
.070914844
```

## $$COTDEG^XLFMTH(): Cotangent (Degrees)

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the cotangent, with degrees input.

**Format** `$$COTDEG^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Degrees input number for which you want the cotangent. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the cotangent of degrees input number. |

**Example**

```
>S X=$$COTDEG^XLFMTH(45)

>W X
1
```

## $$CSC^XLFMTH(): Cosecant (Radians)

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the cosecant, with radians input.

**Format** `$$CSC^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Radians input number for which you want the cosecant. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the cosecant of radians input number. |

**Example**

```
>S X=$$CSC^XLFMTH(1.5)

>W X
1.002511304
```

## $$CSCDEG^XLFMTH(): Cosecant (Degrees)

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Math Functions | |
| **IA #** | 10105 | |
| **Description** | This extrinsic function returns the cosecant, with degrees input. | |
| **Format** | `$$CSCDEG^XLFMTH(x[,n])` | |
| **Input Parameters** | x: | (required) Degrees input number for which you want the cosecant. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the cosecant of degrees input number. |

**Example**

```
>S X=$$CSCDEG^XLFMTH(45)

>W X
1.414213562
```

# $$DECDMS^XLFMTH(): Convert Decimals to Degrees:Minutes:Seconds

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Math Functions | |
| **IA #** | 10105 | |
| **Description** | This extrinsic function converts a number from decimal to degrees:minutes:seconds. | |
| **Format** | `$$DECDMS^XLFMTH(x[,n])` | |
| **Input Parameters** | x: | (required) Decimal number to be converted to degree:minutes:second. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the converted decimal input number to degrees:minutes:seconds. |

**Example**

```
>S X=$$DECDMS^XLFMTH(30.7)

>W X
30:42:0
```

## $$DMSDEC^XLFMTH(): Convert Degrees:Minutes:Seconds to Decimal

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function converts a number from degrees:minutes:seconds to a decimal.

**Format** `$$DMSDEC^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Degrees:minutes:seconds input number to be converted to decimal. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the converted degrees:minutes:seconds input number to decimal. |

**Example**

```
>S X=$$DMSDEC^XLFMTH("30:42:0")

>W X
30.7
```

## $$DTR^XLFMTH(): Convert Degrees to Radians

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function converts degrees to radians.

**Format** `$$DTR^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Degrees input number to be converted to radians. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the converted degrees input number to radians. |

**Example**

```
>S X=$$DTR^XLFMTH(45)

>W X
.7853981634
```

## $$E^XLFMTH(): e—Natural Logarithm

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Math Functions | |
| **IA #** | 10105 | |
| **Description** | This extrinsic function returns **e** (natural logarithm). | |
| **Format** | `$$E^XLFMTH([n])` | |
| **Input Parameters** | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns **e**, natural logarithm. |

**Example**

```
>S X=$$E^XLFMTH(12)

>W X
2.71828182846
```

## $$EXP^XLFMTH(): e—Natural Logarithm to the Nth Power

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns **e** (natural logarithm) to the **x** power (exponent).

**Format** `$$EXP^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) The power to which you want **e** raised. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the value of **e** to the specified power. |

**Example**

```
>S X=$$EXP^XLFMTH(1.532)

>W X
4.6274224185
```

## $$LN^XLFMTH(): Natural Log (Base e)

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the natural log of **x** (Base e).

**Format** `$$LN^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the natural log. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the natural log of a number. |

**Example**

```
>S X=$$LN^XLFMTH(4.627426)

>W X
1.532000774
```

## $$LOG^XLFMTH(): Logarithm (Base 10)

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Math Functions | |
| **IA #** | 10105 | |
| **Description** | This extrinsic function returns the logarithm (Base 10) of **x**. | |
| **Format** | `$$LOG^XLFMTH(x[,n])` | |
| **Input Parameters** | x: | (required) Number for which you want the logarithm. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the logarithm (Base 10) of input number. |

**Example**

```
>S X=$$LOG^XLFMTH(3.1415)

>W X
.4971370641
```

# $$MAX^XLFMTH(): Maximum of Two Numbers

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the maximum value by comparing the number in **x** with the number in **y**.

**Format** `$$MAX^XLFMTH(x,y)`

**Input Parameters**

| | | |
|---|---|---|
| | x: | (required) First number to compare with second number in **y** to determine which is higher in value. |
| | y | (required) Second number to compare with first number in **x** to determine which is higher in value. |

**Output**

| | | |
|---|---|---|
| | returns: | Returns the highest number. |

**Example**

```
>S X=$$MAX^XLFMTH(53,24)

>W X
53
```

## $$MIN^XLFMTH(): Minimum of Two Numbers

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the minimum value by comparing the number in **x** with the number in **y**.

**Format** `$$MIN^XLFMTH(x,y)`

**Input Parameters**

| | |
|---|---|
| x: | (required) First number to compare with second number in **y** to determine which is lower in value. |
| y | (required) Second number to compare with first number in **x** to determine which is lower in value. |

**Output**

| | |
|---|---|
| returns: | Returns the lowest number. |

**Example**

```
>S X=$$MIN^XLFMTH(53,24)

>W X
24
```

## $$PI^XLFMTH(): PI

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns pi.

**Format** `$$PI^XLFMTH([n])`

**Input Parameters**

| | |
|---|---|
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns pi. |

**Example**

```
>S X=$$PI^XLFMTH(12)

>W X
3.14159265359
```

## $$PWR^XLFMTH(): X to the Y Power

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Math Functions | |
| **IA #** | 10105 | |
| **Description** | This extrinsic function returns **x** to the **y** power. This function makes use of LN and EXP. | |
| **Format** | `$$PWR^XLFMTH(x,y[,n])` | |
| **Input Parameters** | x: | (required) Number for which you want the exponent value. |
| | y: | (required) The exponent to which the input number (**x**) should be raised. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the exponent value. |

**Example**

```
>S X=$$PWR^XLFMTH(3.2,1.5)

>W X
5.7243340224
```

# $$RTD^XLFMTH(): Convert Radians to Degrees

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Math Functions | |
| **IA #** | 10105 | |
| **Description** | This extrinsic function converts radians to degrees. | |
| **Format** | `$$RTD^XLFMTH(x[,n])` | |
| **Input Parameters** | x: | (required) Radians input number to be converted to degrees. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the converted radians input number to degrees. |

**Example**

```
>S X=$$RTD^XLFMTH(1.5,12)

>W X
85.9436692696
```

# $$SD^XLFMTH(): Standard Deviation

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the standard deviation. Standard deviation is defined as:

> "a measure of variability equal to the square root of the arithmetic average of the squares of the deviations from the mean in a frequency distribution."[1]

**Format** `$$SD^XLFMTH(%s1,%s2,%n)`

**Input Parameters**

| | |
|---|---|
| `%s1`: | (required) Sum. |
| `%s2` | (required) Sum of squares. |
| `%n` | (required) Count. |

**Output**

| | |
|---|---|
| returns: | Returns the standard deviation. |

**Example**

```
>S X=$$SD^XLFMTH(5,25,2)

>W X
3.53553390593
```

[1] Definition as taken from: *Webster's New World College Dictionary*, Fourth Edition; Michael Agnes, Editor in Chief; David B. Guralink, Editor in Chief Emeritus; Copyright 2001, 2000, 1999 by IDG Books Worldwide, Inc.; ISBN 0-02-863118-8.

## $$SEC^XLFMTH(): Secant (Radians)

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the secant of a number, with radians input.

**Format** `$$SEC^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number in radians for which you want the secant. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the secant of radians input number. |

**Example**

```
>S X=$$SEC^XLFMTH(1.5)

>W X
14.1368329
```

## $$SECDEG^XLFMTH(): Secant (Degrees)

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the secant of a number, with degrees input.

**Format** `$$SECDEG^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number in degrees for which you want the secant. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the secant of degrees input number. |

**Example**

```
>S X=$$SECDEG^XLFMTH(45)

>W X
1.414213562
```

## $$SIN^XLFMTH(): Sine (Radians)

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Math Functions | |
| **IA #** | 10105 | |
| **Description** | This extrinsic function returns the sine of a number, with radians input. | |
| **Format** | `$$SIN^XLFMTH(x[,n])` | |
| **Input Parameters** | x: | (required) Number in radians for which you want the sine. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the sine of radians input number. |

**Example**

```
>S X=$$SIN^XLFMTH(.7853982)

>W X
.707106807
```

## $$SINDEG^XLFMTH(): Sine (Degrees)

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the sine of a number, with degrees input.

**Format** `$$SINDEG^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number in degrees for which you want the sine. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the sine of degrees input number. |

**Example**

```
>S X=$$SINDEG^XLFMTH(45)

>W X
.707106781
```

## $$SQRT^XLFMTH(): Square Root

**Reference Type** Supported

**Category** Math Functions

**IA #** 10105

**Description** This extrinsic function returns the square root of a number.

**Format** `$$SQRT^XLFMTH(x[,n])`

**Input Parameters**

| | |
|---|---|
| x: | (required) Number for which you want the square root. |
| n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |

**Output**

| | |
|---|---|
| returns: | Returns the square root of input number. |

**Example**

```
>S X=$$SQRT^XLFMTH(153)

>W X
12.3693168769
```

## $$TAN^XLFMTH(): Tangent (Radians)

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Math Functions | |
| **IA #** | 10105 | |
| **Description** | This extrinsic function returns the tangent of a number (tan **x** = sin **x**/cos **x**), with radians input. | |
| **Format** | `$$TAN^XLFMTH(x[,n])` | |
| **Input Parameters** | x: | (required) Number in radians for which you want the tangent. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the tangent of radians input number. |

**Example**

```
>S X=$$TAN^XLFMTH(.7853982)

>W X
1.000000073
```

# $$TANDEG^XLFMTH(): Tangent (Degrees)

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Math Functions | |
| **IA #** | 10105 | |
| **Description** | This extrinsic function returns the tangent of a number, with degrees input. | |
| **Format** | `$$TANDEG^XLFMTH(x[,n])` | |
| **Input Parameters** | x: | (required) Number in degrees for which you want the tangent. |
| | n: | (optional) The precision for the function. Precision means the detail of the result, in terms of number of digits. |
| **Output** | returns: | Returns the tangent of degrees input number. |

**Example**

```
>S X=$$TANDEG^XLFMTH(45)

>W X
1
```

# Measurement Functions—XLFMSMT

This routine contains APIs to allow conversion between U.S. (English) and Metric units.

## $$BSA^XLFMSMT(): Body Surface Area Measurement

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Measurement Functions | |
| **IA #** | 3175 | |
| **Description** | This extrinsic function returns the body surface area. | |
| **Format** | `$$BSA^XLFMSMT(ht,wt)` | |
| **Input Parameters** | ht: | (required) Height in centimeters. |
| | wt: | (required) Weight in kilograms. |
| **Output** | returns: | Returns the body surface area measurement. |

### Example 1

```
>S X=$$BSA^XLFMSMT(175,86)

>W X
2.02
```

### Example 2

```
>S X=$$BSA^XLFMSMT($$LENGTH^XLFMSMT(69,"IN","CM"),$$WEIGHT^XLFMSMT(180,"LB","KG"))

>W X
1.98
```

# $$LENGTH^XLFMSMT(): Convert Length Measurement

**Reference Type** Supported

**Category** Measurement Functions

**IA #** 3175

**Description** This extrinsic function converts U.S. length to Metric length and vice versa. It returns the equivalent value with units.

**Format** `$$LENGTH^XLFMSMT(value,from,to)`

**Input Parameters**

value: (required) A positive numeric value.

from: (required) Unit of measure of the value input parameter (see Table 26-1).

to: (required) Unit of measure to which the value input parameter is converted (see Table 26-1).

Valid units in either uppercase or lowercase are:

| Metric | US |
|---|---|
| km—kilometers | mi—miles |
| m—meters | yd—yards |
| mm—millimeters | ft—feet |
| cm—centimeters | in—inches |

**Table 26-1. $$LENGTH^XLFMSMT: Valid units**

**Output** returns: Returns the length measurement.

**Example 1**

Converting U.S. length to Metric length:

```
>S X=$$LENGTH^XLFMSMT(12,"IN","CM")

>W X
30.48 CM
```

**Example 2**

Converting Metric length to U.S. length:

```
>S X=$$LENGTH^XLFMSMT(30.48,"cm","in")

>W X
12 IN
```

## $$TEMP^XLFMSMT(): Convert Temperature Measurement

**Reference Type** Supported

**Category** Measurement Functions

**IA #** 3175

**Description** This extrinsic function converts U.S. temperature to Metric temperature and vice versa. It returns the equivalent value with units.

**Format** `$$TEMP^XLFMSMT(value,from,to)`

**Input Parameters**

value: (required) A positive numeric value.

from: (required) Unit of measure of the value input parameter (see Table 26-2).

to: (required) Unit of measure to which the value input parameter is converted (see Table 26-2).

Valid units in either uppercase or lowercase are:

| Metric | US |
|---|---|
| C—Celsius | F—Fahrenheit |

**Table 26-2: $$TEMP^XLFMSMT: Valid units**

**Output** returns: Returns the temperature measurement.

**Example 1**

Converting Fahrenheit to Celsius:

```
>S X=$$TEMP^XLFMSMT(72,"F","C")

>W X
22.222 C
```

**Example 2**

Converting Celsius to Fahrenheit:

```
>S X=$$TEMP^XLFMSMT(0,"c","f")

>W X
32 F
```

## $$VOLUME^XLFMSMT(): Convert Volume Measurement

**Reference Type** Supported

**Category** Measurement Functions

**IA #** 3175

**Description** This extrinsic function converts U.S. volume to Metric volume and vice versa. Converts milliliters to cubic inches or quarts or ounces. It returns the equivalent value with units.

**Format** `$$VOLUME^XLFMSMT(value,from,to)`

**Input Parameters**

value: (required) A positive numeric value.

from: (required) Unit of measure of the value input parameter (see Table 26-3).

to: (required) Unit of measure to which the value input parameter is converted (see Table 26-3).

Valid units in either uppercase or lowercase are:

| Metric | US |
|---|---|
| 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 |

**Table 26-3. $$VOLUME^XLFMSMT: Valid units**

**Output** returns: Returns the volume measurement.

**Example 1**

Converting U.S. volume to Metric volume:

```
>S X=$$VOLUME^XLFMSMT(12,"CF","ML")

>W X
339800.832 ML
```

**Example 2**

Converting Metric volume to U.S. volume:

```
>S X=$$VOLUME^XLFMSMT(339800.832,"ml","cf")

>W X
11.998 CF
```

# $$WEIGHT^XLFMSMT(): Convert Weight Measurement

**Reference Type** Supported

**Category** Measurement Functions

**IA #** 3175

**Description** This extrinsic function converts U.S. weights to approximate Metric weights and vice versa. It returns the equivalent value with units.

**Format** `$$WEIGHT^XLFMSMT(value,from,to)`

**Input Parameters**

value: (required) A positive numeric value.

from: (required) Unit of measure of the value input parameter (see Table 26-4).

to: (required) Unit of measure to which the value input parameter is converted (see Table 26-4).

Valid units in either uppercase or lowercase are:

| Metric | US |
|---|---|
| t—metric tons | tn— tons |
| kg—kilograms | lb—pounds |
| g—grams | oz—ounces |
| mg—milligram | gr—grain |

**Table 26-4. $$WEIGHT^XLFMSMT: Valid units**

**Output**

returns: Returns the weight measurement.

**Example 1**

Converting U.S. weight to Metric weight:

```
>S X=$$WEIGHT^XLFMSMT(12,"LB","G")

>W X
5448 G
```

**Example 2**

Converting Metric weight to U.S. weight:

```
>S X=$$WEIGHT^XLFMSMT(5448,"g","lb")

>W X
12.011 LB
```

# String Functions—XLFSTR

These functions are provided to help process strings.

## $$CJ^XLFSTR(): Center Justify String

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | String Functions | |
| **IA #** | 10104 | |
| **Description** | This extrinsic function returns a center justified character string. | |
| **Format** | `$$CJ^XLFSTR(s,i[,p])` | |
| **Input Parameters** | s: | (required) Character string. |
| | i: | (required) Field size. |
| | p: | (optional) Pad character. |
| **Output** | returns: | Returns the Center justified string. |

**Example 1**

```
>W "[",$$CJ^XLFSTR("SUE",10),"]"
[   SUE    ]
```

**Example 2**

```
>W "[",$$CJ^XLFSTR("SUE",10,"-"),"]" >Enter>
[---SUE----]
```

## $$INVERT^XLFSTR(): Invert String

**Reference Type** Supported

**Category** String Functions

**IA #** 10104

**Description** This extrinsic function returns an inverted string. It inverts the order of the characters in a string.

**Format** `$$INVERT^XLFSTR(x)`

**Input Parameters**

| | | |
|---|---|---|
| | x: | (required) Character string. |

**Output**

| | | |
|---|---|---|
| | returns: | Returns the inverted string. |

**Example**

```
>S X=$$INVERT^XLFSTR("ABC")

>W X
CBA
```

## $$LJ^XLFSTR(): Left Justify String

**Reference Type** Supported

**Category** String Functions

**IA #** 10104

**Description** This extrinsic function returns a left justified character string.

**Format** `$$LJ^XLFSTR(s,i[,p])`

**Input Parameters**

| | | |
|---|---|---|
| | s: | (required) Character string. |
| | i: | (required) Field size. |
| | p: | (optional) Pad character. |

**Output**

| | | |
|---|---|---|
| | returns: | Returns the left justified string. |

**Example 1**

```
>W "[",$$LJ^XLFSTR("TOM",10),"]"
[TOM       ]
```

**Example 2**

```
>W "[",$$LJ^XLFSTR("TOM",10,"-"),"]"
[TOM-------]
```

## $$LOW^XLFSTR(): Convert String to Lowercase

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | String Functions | |
| **IA #** | 10104 | |
| **Description** | This extrinsic function returns an input string converted to all Lowercase. | |
| **Format** | `$$LOW^XLFSTR(x)` | |
| **Input Parameters** | x: | (required) Character string. |
| **Output** | returns: | Returns the input stringconverted to all lowercase. |

**Example**

```
>S X=$$LOW^XLFSTR("JUSTICE")

>W X
justice
```

# $$REPEAT^XLFSTR(): Repeat String

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | String Functions | |
| **IA #** | 10104 | |
| **Description** | This extrinsic function returns a string that repeats the value of **x** for **y** number of times. | |
| **Format** | `$$REPEAT^XLFSTR(x[,y])` | |
| **Input Parameters** | x: | (required) Character string to be repeated. |
| | y: | (optional) Number of times to repeat the string in **x**. |
| **Output** | returns: | Returns the repeated string. |

### Example 1

```
>S X=$$REPEAT^XLFSTR("-",10)

>W X
----------
```

### Example 2

```
>S X=$$REPEAT^XLFSTR("blue water ",5)

>W X
blue water blue water blue water blue water blue water
```

# $$REPLACE^XLFSTR(): Replace Strings

**Reference Type** Supported

**Category** String Functions

**IA #** 10104

**Description** This extrinsic function uses a multi-character $Translate to return a string with the specified string replaced.

**Format** `$$REPLACE^XLFSTR(in,.spec)`

**Input Parameters**

| | |
|---|---|
| in: | (required) Input string. |
| .spec: | (required) An array passed by reference. |

**Output**

| | |
|---|---|
| returns: | Returns the replaced string. |

### Example 1

```
>SET spec("aa")="a",spec("pqr")="alabama"
>S X=$$REPLACE^XLFSTR("aaaaaaapqraaaaaaa",.spec)

>W X
aaaaalabamaaaaa
```

### Example 2

```
>SET spec("F")="VA File",spec("M")="Man"
>S X=$$REPLACE^XLFSTR("FM",.spec)

>W X
VA FileMan
```

# $$RJ^XLFSTR(): Right Justify String

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | String Functions | |
| **IA #** | 10104 | |
| **Description** | This extrinsic function returns a right justified character string. | |
| **Format** | `$$RJ^XLFSTR(s,i[,p])` | |
| **Input Parameters** | s: | (required) Character string. |
| | i: | (required) Field size. |
| | p: | (optional) Pad character. |
| **Output** | returns: | Returns the right justified string. |

**Example 1**

```
>W "[",$$RJ^XLFSTR("TOM",10),"]"
[       TOM]
```

**Example 2**

```
>W "[",$$RJ^XLFSTR("TOM",10,"-"),"]"
[-------TOM]
```

## $$SENTENCE^XLFSTR(): Convert String to Sentence Case

**Reference Type** Supported

**Category** String Functions

**IA #** 10104

**Description** Released with Kernel Patch XU*8.0*400, this extrinsic function returns an input string converted to Sentence case. The initial character of each sentence in the input string will be capitalized and the remaining characters in that sentence are returned as all lowercase. The first character of the string begins a sentence. Subsequent sentences are identified as beginning after a a period (**.**), exclamation point (**!**), or question mark (**?**).

**Format** `$$SENTENCE^XLFSTR(x)`

**Input Parameters** x: (required) Character string.

**Output** returns: Returns the string converted to Sentence case format.

**Example**

```
>S X=$$SENTENCE^XLFSTR("HELLO WORLD!!! THIS IS A CAPITALIZED SENTENCE. this isn't.")

>W X
Hello world!!! This is a capitalized sentence. This isn't.
```

## $$STRIP^XLFSTR(): Strip a String

**Reference Type** Supported

**Category** String Functions

**IA #** 10104

**Description** This extrinsic function returns a string stripped of all instances of a specified character.

**Format** `$$STRIP^XLFSTR(x,y)`

**Input Parameters** x: (required) Character string.

y: (required) The character to strip out of the string.

**Output** returns: Returns the string stripped of specified character.

**Example 1**

```
>S X=$$STRIP^XLFSTR("hello","e")

>W X
hllo
```

**Example 2**

```
>S X=$$STRIP^XLFSTR("Mississippi","i")

>W X
Mssssspp
```

## $$TITLE^XLFSTR(): Convert String to Title Case

**Reference Type** Supported

**Category** String Functions

**IA #** 10104

**Description** Released with Kernel Patch XU*8.0*400, this extrinsic function returns an input string converted to Title case. The initial letter of the first block of characters (i.e., word) in the input string is capitalized and the remaining characters of that first word are returned as all lowercase. Also, the initial letter of any subsequent word in the input string is capitalized and the remaining characters in that word are returned as all lowercase. A word is identified when it is preceded by at least one space, except for the first word in the string.

**Format** `$$TITLE^XLFSTR(x)`

**Input Parameters** x: (required) Character string.

**Output** returns: Returns the string converted to Title case format.

**Example**

```
>S X=$$TITLE^XLFSTR("HELLO WORLD!!! THIS IS A title-form SENTENCE. so is this.")

>W X
Hello World!!! This Is A Title-form Sentence. So Is This.
```

# $$TRIM^XLFSTR(): Trim String

**Reference Type** Supported

**Category** String Functions

**IA #** 10104

**Description** This extrinsic function trims spaces or other specified characters from the left, right, or both ends of an input string.

**Format** `$$TRIM^XLFSTR(s[,f][,c])`

**Input Parameters**

s: (required) Character string.

f: (optional) This flag can have the following value:

"LR" (default)—Trim characters from both ends of the string.

"L"—Trim characters from the left/beginning of the string.

"R"—Trim characters from the right/end of the string.

c: (optional) Set this parameter to the character to trim from the input string. This parameter defaults to a space.

**Output** returns: Returns the trimmed string.

### Example 1

In this example, we are trimming the spaces from both the left and right end of the string (the brackets are added to more clearly display the trimmed string):

```
>S X="["_$$TRIM^XLFSTR("  A B C  ")_"]"

>W X
[A B C]
```

The second input parameter defaults to "LR" and the third input parameter defaults to spaces.

### Example 2

In this example, we are trimming the slashes from both the left and right end of the string (the brackets are added to more clearly display the trimmed string):

```
>S X="["_$$TRIM^XLFSTR("//A B C//",,"/")_"]"

>W X
[A B C]
```

The second input parameter defaults to "LR."

**Example 3**

In this example, we are trimming the slashes from the left end of the string (the brackets are added to more clearly display the trimmed string):

```
>S X="["_$$TRIM^XLFSTR("//A B C//","L","/")_"]"

>W X
[A B C//]
```

**Example 4**

In this example, we are trimming the slashes from the right end of the string (the brackets are added to more clearly display the trimmed string):

```
>S X="["_$$TRIM^XLFSTR("//A B C//","r","/")_"]"

>W X
[//A B C]
```

## $$UP^XLFSTR(): Convert String to Uppercase

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | String Functions | |
| **IA #** | 10104 | |
| **Description** | This extrinsic function returns an input string converted to all Uppercase. | |
| **Format** | `$$UP^XLFSTR(x)` | |
| **Input Parameters** | x: | (required) Character string. |
| **Output** | returns: | Returns the string converted to all uppercase. |

**Example**

```
>S X=$$UP^XLFSTR("freedom")

>W X
FREEDOM
```

# Utility Functions—XLFUTL

These functions are provided to help with a variety of tasks.

## $$BASE^XLFUTL(): Convert Between Two Bases

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Utility Functions | |
| **IA #** | 2622 | |
| **Description** | This extrinsic function converts a number from one base to another. The base must be between 2 and 16, both from and to. | |
| **Format** | `$$BASE^XLFUTL(n,from,to)` | |
| **Input Parameters** | n: | (required) Number to convert. |
| | from: | (required) Base of number being converted. |
| | to: | (required) Base to which the number is to be converted. |
| **Output** | returns: | Returns the converted number from one base to another. |

### Example 1

```
>S X=$$BASE^XLFUTL(1111,2,16)

>W X
F
```

### Example 2

```
>S X=$$BASE^XLFUTL(15,10,16)

>W X
F
```

### Example 3

```
>S X=$$BASE^XLFUTL("FF",16,10)

>W X
255
```

# $$CCD^XLFUTL(): Append Check Digit

**Reference Type** Supported

**Category** Utility Functions

**IA #** 2622

**Description** This extrinsic function returns a number appended with a computed check digit. To check if the original number corresponds with the appended check digit, use the $$VCD^XLFUTL(): Verify Integrity API.

**Format** `$$CCD^XLFUTL(x)`

**Input Parameters** x: (required) Integer for which the check digit is computed.

**REF:** Please refer to "The Taylor Report" in Computerworld magazine, 1975, for the algorithm.

**Output** returns: Returns the number with appended check digit.

### Example 1

```
>S X=$$CCD^XLFUTL(99889)

>W X
998898
```

### Example 2

```
>S X=$$CCD^XLFUTL(7654321)

>W X
76543214
```

# $$CNV^XLFUTL(): Convert Base 10 to Another Base

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | Utility Functions | |
| **IA #** | 2622 | |
| **Description** | This extrinsic function converts a number from Base 10 to another base, which must be between 2 and 16. | |
| **Format** | `$$CNV^XLFUTL(n,base)` | |
| **Input Parameters** | n: | (required) Base 10 number to convert. |
| | base: | (required) The base to which the number is to be converted. |
| **Output** | returns: | Returns the converted number to specified base. |

### Example 1

```
>S X=$$CNV^XLFUTL(15,2)

>W X
1111
```

### Example 2

```
>S X=$$CNV^XLFUTL(255,2)

>W X
11111111
```

### Example 3

```
>S X=$$CNV^XLFUTL(255,8)

>W X
377
```

## $$DEC^XLFUTL(): Convert Another Base to Base 10

**Reference Type** Supported

**Category** Utility Functions

**IA #** 2622

**Description** This extrinsic function converts a number from a specified base, which must be between 2 and 16, to Base 10.

**Format** `$$DEC^XLFUTL(n,base)`

**Input Parameters**

| | |
|---|---|
| n: | (required) Number to convert. |
| base: | (required) Base of number being converted. |

**Output**

| | |
|---|---|
| returns: | Returns the converted number in Base 10. |

**Example**

```
>S X=$$DEC^XLFUTL("FF",16)

>W X
255
```

## $$VCD^XLFUTL(): Verify Integrity

**Reference Type** Supported

**Category** Utility Functions

**IA #** 2622

**Description** This extrinsic function verifies the integrity of a number with an appended check digit. The check digit must be appended by the $$CCD^XLFUTL(): Append Check Digit API.

**Format** `$$VCD^XLFUTL(number)`

**Input Parameters**

| | |
|---|---|
| number: | (required) Number to verify, including appended check digit. |

**Output**

| | |
|---|---|
| returns: | Returns:<br>1—Number corresponds to check digit.<br>0—Number does *not* correspond to check digit. |

### Example 1

```
>S X=$$VCD^XLFUTL(76543214)

>W X
1
```

### Example 2

Transposing "32" to "23":

```
>S X=$$VCD^XLFUTL(76542314)

>W X
0
```

# 27. XML: Programmer Tools

## Application Program Interface (API)

Several APIs are available for programmers to work with the EXtensible Markup Language (XML). These APIs are described below.

### $$ATTRIB^MXMLDOM(): XML—Get Attribute Name

**Reference Type** Supported

**Category** XML

**IA #** 3561

**Description** This extrinsic function retrieves the first or next attribute associated with the specified node.

**Format** `$$ATTRIB^MXMLDOM(handle,node[,attrib])`

**Input Parameters**

| | |
|---|---|
| handle: | (required) The value (integer) returned by the $$EN^MXMLDOM(): XML—Initial Processing, Build In-memory Image API, which created the in-memory document image. |
| node: | (required) The node (integer) whose attribute name is being retrieved. |
| attrib: | (optional) The name (string) of the last attribute retrieved by this call. If null or missing, the first attribute associated with the specified node is returned. Otherwise, the next attribute in the list is returned. |

**Output**

| | |
|---|---|
| returns: | Returns the name (string) of the first or next attribute associated with the specified node, or null if there are none remaining. |

**Example**

# $$CHILD^MXMLDOM(): XML—Get Child Node

**Reference Type** Supported

**Category** XML

**IA #** 3561

**Description** This extrinsic function retrieves the node of the first or next child of a given parent node, or zero (0) if there are none remaining.

**Format** `$$CHILD^MXMLDOM(handle,parent[,child])`

**Input Parameters**

| | |
|---|---|
| handle: | (required) The value (integer) returned by the $$EN^MXMLDOM(): XML—Initial Processing, Build In-memory Image API, which created the in-memory document image. |
| parent: | (required) The node (integer) whose children are being retrieved. |
| child: | (optional) If specified, this is the last child node (integer) retrieved. The function will return the next child in the list. If the parameter is zero or missing, the first child is returned. |

**Output**

returns: Returns:

Child Node—The next child node (integer).

Zero (0)—If there are none remaining.

**Example**

## $$CMNT^MXMLDOM(): XML—Extract Comment Text (True/False)

**Reference Type** Supported

**Category** XML

**IA #** 3561

**Description** This extrinsic function extracts comment text associated with the specified node.

**Format** `$$CMNT^MXMLDOM(handle,node,text)`

**Input Parameters**

handle: (required) The value (integer) returned by the $$EN^MXMLDOM(): XML—Initial Processing, Build In-memory Image API, which created the in-memory document image.

node: (required) The node (integer) in the document tree that is being referenced by this API.

text: (required) This input parameter (string) must contain a closed local or global array reference that is to receive the text. The specified array is deleted before being populated.

**Output**

returns: Returns a Boolean value:

True (non-zero)—Text was retrieved.

False (zero)—Text was *not* retrieved.

**Example**

## CMNT^MXMLDOM(): XML—Extract Comment Text (True/False)

**Reference Type** Supported

**Category** XML

**IA #** 3561

**Description** This API extracts comment text associated with the specified node.

**Format** `$$CMNT^MXMLDOM(handle,node,text)`

**Input Parameters**

handle: (required) The value (integer) returned by the $$EN^MXMLDOM(): XML—Initial Processing, Build In-memory Image API, which created the in-memory document image.

node: (required) The node (integer) in the document tree that is being referenced by this API.

text: (required) This input parameter (string) must contain a closed local or global array reference that is to receive the text. The specified array is deleted before being populated.

**Output**

returns: Returns a Boolean value:

True (non-zero)—Text was retrieved.

False (zero)—Text was *not* retrieved.

**Example**

## DELETE^MXMLDOM(): XML—Delete Document Instance

**Reference Type** Supported

**Category** XML

**IA #** 3561

**Description** This API deletes the specified document instance. A client application should always call this API when finished with a document instance.

**Format** `DELETE^MXMLDOM(handle)`

**Input Parameters**

| | |
|---|---|
| handle | (required) The value (integer) returned by the $$EN^MXMLDOM(): XML—Initial Processing, Build In-memory Image API, which created the in-memory document image. |

**Output** none

**Example**

## $$EN^MXMLDOM(): XML—Initial Processing, Build In-memory Image

**Reference Type** Supported

**Category** XML

**IA #** 3561

**Description** This extrinsic function performs initial processing of the XML document. The client application must first call this entry point to build the in-memory image of the document before the remaining methods can be applied. The return value is a handle to the document instance that was created and is used by the remaining API calls to identify a specific document instance. The parameters for this entry point are listed by type, requirement (yes or no), and description.

**Format** `$$EN^MXMLDOM(doc[,opt])`

**Input Parameters**

doc: (required) This string is either a closed reference to a global root containing the document or a filename and path reference identifying the document on the host system. If a global root is passed, the document either must be stored in standard VA FileMan word-processing format or may occur in sequentially numbered nodes below the root node. Thus, if the global reference is "^XYZ", the global must be of one of the following formats:

- ^XYZ(1,0) = "LINE 1"

  ^XYZ(2,0) = "LINE 2" ...

Or

- ^XYZ(1) = "LINE 1"

  ^XYZ(2) = "LINE 2" ...

opt: (optional) This string is a list of option flags that control parser behavior. Recognized option flags are:

W—Do not report warnings to the client.

V—Do not validate the document. If specified, the parser only checks for conformance.

1—Terminate parsing on encountering a validation error. (By default, the parser terminates only when a conformance error is encountered.)

0—Terminate parsing on encountering a warning.

**Output** returns: Returns:

Non-zero Handle of Document Instance—Parsing completed successfully.

ZeroHandle of Document Instance.

This handle is passed to all other API methods to indicate which document instance is being referenced. This allows for multiple document instances to be processed concurrently.

**Example**

## $$NAME^MXMLDOM(): XML—Get Element Name

**Reference Type** Supported

**Category** XML

**IA #** 3561

**Description** This extrinsic function retrieves the name of the element at the specified node within the document parse tree.

**Format** `$$NAME^MXMLDOM(handle,node)`

**Input Parameters** handle: (required) The value (integer) returned by the $$EN^MXMLDOM(): XML—Initial Processing, Build In-memory Image API, which created the in-memory document image.

node: (required) The node (integer) for which the associated element name is being retrieved.

**Output** returns: Returns the name (string) of the element associated with the specified node.

**Example**

# $$PARENT^MXMLDOM(): XML—Get Parent Node

**Reference Type** Supported

**Category** XML

**IA #** 3561

**Description** This extrinsic function retrieves the parent node of the specified node, or zero (0) if there is none.

**Format**

```
$$PARENT^MXMLDOM(handle,node)
```

**Input Parameters**

handle: (required) The value (integer) returned by the $$EN^MXMLDOM(): XML—Initial Processing, Build In-memory Image API, which created the in-memory document image.

node: (required) The node (integer) in the document tree whose parent is being retrieved.

**Output**

returns: Returns:

Parent Node—The parent node (string) of the specified node.

Zero (0)—If there is no parent.

**Example**

# $$SIBLING^MXMLDOM(): XML—Get Sibling Node

**Reference Type** Supported

**Category** XML

**IA #** 3561

**Description** This extrinsic function retrieves the node of the specified node's immediate sibling, or zero (0) if there is none.

**Format** `$$SIBLING^MXMLDOM(handle,node)`

**Input Parameters**

handle: (required) The value (integer) returned by the $$EN^MXMLDOM(): XML—Initial Processing, Build In-memory Image API, which created the in-memory document image.

node: (required) The node (integer) in the document tree whose sibling is being retrieved.

**Output**

returns: Returns:

Node—The node (integer) corresponding to the immediate sibling of the specified node.

Zero (0)—If there is no node (integer) corresponding to the immediate sibling of the specified node.

**Example**

## $$TEXT^MXMLDOM(): XML—Get Text (True/False)

**Reference Type** Supported

**Category** XML

**IA #** 3561

**Description** This extrinsic function extracts non-markup text associated with the specified node.

**Format** `$$TEXT^MXMLDOM(handle,node,text)`

**Input Parameters**

| | |
|---|---|
| handle: | (required) The value (integer) returned by the $$EN^MXMLDOM(): XML—Initial Processing, Build In-memory Image API, which created the in-memory document image. |
| node: | (required) The node (integer) in the document tree that is being referenced by this API. |
| text: | (required) This input parameter (string) must contain a closed local or global array reference that is to receive the text. The specified array is deleted before being populated. |

**Output**

| | |
|---|---|
| returns: | Returns a Boolean value:<br>True (non-zero)—Text was retrieved.<br>False (zero)—Text was *not* retrieved. |

**Example**

# TEXT^MXMLDOM(): XML—Get Text (True/False)

**Reference Type** Supported

**Category** XML

**IA #** 3561

**Description** This API extracts non-markup text associated with the specified node.

**Format** `TEXT^MXMLDOM(handle,node,text)`

**Input Parameters**

| | |
|---|---|
| handle: | (required) The value (integer) returned by the $$EN^MXMLDOM(): XML—Initial Processing, Build In-memory Image API, which created the in-memory document image. |
| node: | (required) The node (integer) in the document tree that is being referenced by this API. |
| text: | (required) This input parameter (string) must contain a closed local or global array reference that is to receive the text. The specified array is deleted before being populated. |

**Output**

returns: Returns a Boolean value:

True (non-zero)—Text was retrieved.

False (zero)—Text was *not* retrieved.

**Example**

# $$VALUE^MXMLDOM(): XML—Get Attribute Value

**Reference Type** Supported

**Category** XML

**IA #** 3561

**Description** This extrinsic function retrieves the value associated with the named attribute.

**Format** `$$VALUE^MXMLDOM(handle,node[,attrib])`

**Input Parameters**

| | |
|---|---|
| handle: | (required) The value (integer) returned by the $$EN^MXMLDOM(): XML—Initial Processing, Build In-memory Image API, which created the in-memory document image. |
| node: | (required) The node (integer) whose attribute value is being retrieved. |
| attrib: | (optional) The name of the attribute (string) whose value is being retrieved by this API. |

**Output** returns: Returns the value associated with the specified attribute.

**Example**

## EN^MXMLPRSE(): XML—Event Driven API

**Reference Type** Supported

**Category** XML

**IA #** 4149

**Description** This API is based on the well-established Simple API for XML (SAX) interface employed by many XML parsers. This API has a single method.

**Format** `EN^MXMLPRSE(doc[,cbk][,opt])`

**Input Parameters**

doc: (required) This string is either a closed reference to a global root containing the document or a filename and path reference identifying the document on the host system. If a global root is passed, the document either must be stored in standard VA FileMan word-processing format or may occur in sequentially numbered nodes below the root node. Thus, if the global reference is "^XYZ", the global must be of one of the following formats:

- ^XYZ(1,0) = "LINE 1"

  ^XYZ(2,0) = "LINE 2"...

Or

- ^XYZ(1) = "LINE 1"

  ^XYZ(2) = "LINE 2"...

cbk: (optional) This is a local array, passed by reference that contains a list of parse events and the entry points for the handlers of those events. The format for each entry is:

CBK(<*event type*>) = <*entry point*>

The entry point must reference a valid entry point in an existing M routine and should be of the format *tag^routine*. The entry should not contain any formal parameter references. The application developer is responsible for ensuring that the actual entry point contains the appropriate number of formal parameters for the event type. For example, client application might register its STARTELEMENT event handler as follows:

CBK("STARTELEMENT") = "STELE^CLNT"

The actual entry point in the CLNT routine must include two formal parameters as in the following example:

STELE(ELE,ATR) <handler code>

For the types of supported events and their required parameters, see the "Details" topic that follows.

opt: (optional) This is a list of option flags (string) that control parser behavior. Recognized option flags are:

W—Do not report warnings to the client.

V—Validate the document. If not specified, the parser only checks for conformance.

1—Terminate parsing on encountering a validation error. (By default, the parser terminates only when a conformance error is encountered.)

0—Terminate parsing on encountering a warning.

**Output** returns: Returns the XML parsed string.

**Details**

The VistA XML Parser recognizes the following event types:

| Event Type | Parameter(s) | Description |
|---|---|---|
| **STARTDOCUMENT** | None | Notifies the client that document parsing has commenced. |
| **ENDDOCUMENT** | None | Notifies the client that document parsing has completed. |
| **DOCTYPE** | ROOT<br>PUBID<br>SYSID | Notifies the client that a DOCTYPE declaration has been encountered. The name of the document root is given by ROOT. The public and system identifiers of the external document type definition are given by PUBID and SYSID, respectively. |
| **STARTELEMENT** | NAME<br>ATTRLIST | An element (tag) has been encountered. The name of the element is given in NAME. The list of attributes and their values is provided in the local array ATTRLST in the format:<br>ATTRLST(<name>) = <value> |
| **ENDELEMENT** | NAME | A closing element (tag) has been encountered. The name of the element is given in *NAME*. |
| **CHARACTERS** | TEXT | Non-markup content has been encountered. TEXT contains the text. Line breaks within the original document are represented as carriage return/line feed character sequences. The parser does not necessarily pass an entire line of the original document to the client with each event of this type. |
| **PI** | TARGET<br>TEXT | The parser has encountered a processing instruction. TARGET is the target application for the processing instruction. TEXT is a local array containing the parameters for the instruction. |
| **EXTERNAL** | SYSID<br>PUBID<br>GLOBAL | The parser has encountered an external entity reference whose system and public identifiers are given by SYSID and PUBID, |

| Event Type | Parameter(s) | Description |
|---|---|---|
| | | respectively. If the event handler elects to retrieve the entity rather than allowing the parser to do so, it should pass the global root of the retrieved entity in the GLOBAL parameter. If the event handler wishes to suppress retrieval of the entity altogether, it should set both SYSID and PUBID to null. |
| **NOTATION** | NAME<br>SYSID<br>PUBIC | The parser has encountered a notation declaration. The notation name is given by NAME. The system and public identifiers associated with the notation are given by SYSID and PUBIC, respectively. |
| **COMMENT** | TEXT | The parser has encountered a comment. TEXT is the text of the comment. |
| **ERROR** | ERR | The parser has encountered an error during the processing of a document. ERR is a local array containing information about the error. The format is:<br>ERR("SEV") = Severity of the error where zero (0) is a warning, 1 is a validation error, and 2 is a conformance error.<br>• ERR("MSG")—Brief text description of the error.<br>• ERR("ARG")—The token value the triggered the error (optional).<br>• ERR("LIN")—The number of the line being processed when the error occurred.<br>• ERR("POS")—The character position within the line where the error occurred.<br>• ERR("XML")—The original document text of the line where the error occurred. |

**Example**

A sample client of the event-driven API is provided in the routine MXMLTEST. This routine has an entry point EN(DOC,OPT), where DOC and OPT are the same parameters as described above in for the parser entry point. This sample application simply prints a summary of the parsing events as they occur.

## $$SYMENC^MXMLUTL(): XML—Encoded Strings in Messages

| | | |
|---|---|---|
| **Reference Type** | Supported | |
| **Category** | XML | |
| **IA #** | 4153 | |
| **Description** | This extrinsic function replaces reserved XML symbols in a string with their XML encoding for strings used in an extensible markup language (XML) message. | |
| **Format** | `$$SYMENC^MXMLUTL(str)` | |
| **Input Parameters** | str: | (required) String to be encoded in an XML message. |
| **Output** | returns: | Returns the input string with XML encoding replacing reserved XML symbols. |

**Example**

```
>S X=$$SYMENC^MXMLUTL("This line isn't &""<XML>"" safe as is.")

>W X
This line isn&apos;t &amp;&quot;&lt;XML&gt;&quot; safe as is.
```

# $$XMLHDR^MXMLUTL: XML—Message Headers

**Reference Type** Supported

**Category** XML

**IA #** 4153

**Description** This extrinsic function returns a standard extensible markup language (XML) header for encoding XML messages.

**Format** `$$XMLHDR^MXMLUTL`

**Input Parameters** none

**Output** returns: Returns a standard XML header.

## Example

```
>S X=$$XMLHDR^MXMLUTL

>W X
<?xml version="1.0" encoding="utf-8" ?>
```

# Glossary

| | |
|---|---|
| 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. |
| CAPACITY MANAGEMENT | The process of assessing a system's capacity and evaluating its efficiency relative to workload in an attempt to optimize system performance. 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.<br><br>(See also: ENCRYPTION.) |
| 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. |
| 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). |

| | |
|---|---|
| DIFROM | VA FileMan utility that gathers all software components and changes them into routines (namespaceI* routines) so that they can be exported and installed in another VA FileMan environment. |
| 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 DR variable 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(0) | A local variable that holds the FILE MANAGER ACCESS CODE of the signed-on user. |
| 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 and Verify codes). |
| 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 (i.e., FILE MANAGER ACCESS CODE field). |
| 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. |
| GO-HOME JUMP | A menu jump that returns the user to the primary menu presented at signon. It is specified by entering two carets ("^^") at the menu's select prompt. It resembles the Rubber-band Jump but without an option specification after the carets. |
| HELP PROCESSOR | A Kernel module that provides a system for creating and displaying online 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. |
| 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 (#3.5) 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 can stand in as a substitute. |

| | |
|---|---|
| INDEX (%INDEX) | A Kernel utility used to verify routines and other M code associated with a software application. Checking is done according to current ANSI MUMPS standards and VistA programming standards. This tool can be invoked through an option or from direct mode (>D ^%INDEX). |
| INIT | Initialization of an software application. INIT* routines are built by VA FileMan's DIFROM and, when run, recreate a set of files and other software components. |
| JUMP | In VistA applications, the Jump command allows you to go from a particular field within an option to another field within that same option. You can also Jump from one menu option to another menu option without having to respond to all the prompts in between. To jump, type a caret ("^", uppercase-6 key on most keyboards) and then type the name of the field or option you wish to jump to.<br><br>(See also GO-HOME JUMP, PHANTOM JUMP, RUBBER-BAND JUMP, 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 Kernel and can be set up as an alternate editor. |
| MANAGER ACCOUNT | A UCI that can be referenced by non-manager accounts (e.g., production accounts). Like a library, the MGR UCI holds percent routines and globals (e.g., ^%ZOSF) for shared use by other UCIs. |
| 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 (e.g., 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 (e.g., 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 can define user-specific MENU templates via the corresponding Common option. |

| | |
|---|---|
| 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. |
| PAC | **P**rogrammer **A**ccess **C**ode. An optional user attribute that can function as a second level password into programmer mode. |
| PART 3 OF THE KERNEL INIT | See FILE ACCESS SECURITY SYSTEM. |
| 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. |
| PHANTOM JUMP | Menu jumping in the background. Used by the menu system to check menu pathway restrictions. |
| PRIMARY MENUS | The list of options presented at signon. Each user must have a PRIMARY MENU OPTION 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. |
| 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)=@. |
| PROTOCOL | An entry in the PROTOCOL file (#101). Used by the Order Entry/Results Reporting (OE/RR) software to support the ordering of medical tests and other activities. Kernel includes several protocol-type options for enhanced menu displays within the OE/RR software. |
| QUEUING | Requesting that a job be processed in the background rather than in the foreground within the current session. Kernel's TaskMan module 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 can be invoked and the job prepared for processing, but the output can only be generated during the specified time periods. |
| 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 (#3.54). |

| | |
|---|---|
| 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 carets ("^^", uppercase-6 on most keyboards) followed by an option's menu text or synonym (e.g., ^^Print Option File). If the two carets are not followed by an option specification, the user is returned to the primary menu.<br><br>(See also: GO-HOME JUMP.) |
| SCHEDULING OPTIONS | A way of ordering TaskMan to run an option at a designated time with a specified rescheduling frequency (e.g., once per week). |
| 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 MENU OPTIONS | 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 (e.g., application coordinators) who have been so authorized. SMD is a part of Menu Manager. |
| SERVER OPTION | In VistA, an entry in the OPTION file (#19). An automated mail protocol that is activated by sending a message to the server with the "S.server" syntax. A server option's activity is specified in the OPTION file (#19) and can be the running of a routine or the placement of data into a file. |
| SIGNON/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 signons is maintained. |
| SPECIAL QUEUEING | An option attribute indicating that TaskMan should automatically run the option whenever the system reboots. |
| SPOOLER | An entry in the DEVICE file (#3.5). It uses the associated operating system's spool facility, whether it's a global, device, or host file. 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. Kernel will subsequently transfer the text to a global for subsequent despooling (printing). |
| SYNONYM | In VistA, a field in the OPTION file (#19). Options can be selected by their menu text or synonym.<br><br>(See also: MENU TEXT.) |

| | |
|---|---|
| TASKMAN | The Kernel module that schedules and processes background tasks (also called Task Manager). |
| TIMED READ | The amount of time Kernel will wait for a user response to an interactive READ command before starting to halt the process. |
| UP-ARROW JUMP | In the menu system, entering a carets ("^", uppercase-6 on most keyboards) followed by an option name accomplishes a jump to the target option without needing to take the usual steps through the menu pathway. |
| 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 Kernel to provide an interface between VistA software 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 can be referenced by application programmers so that separate versions of the software need not be written for each operating system. |

**REF:** For a comprehensive list of commonly used infrastructure- and security-related terms and definitions, please visit the ISS Glossary Web page at the following Web address:

http://vaww.vista.med.va.gov/iss/glossary.asp

For a comprehensive list of acronyms, please visit the ISS Acronyms Web site at the following Web address:

http://vaww.vista.med.va.gov/iss/acronyms/index.asp

# 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 software application 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 Callback (Tag^Routine Reference) | Parameter Used by Checkpoint | Checkpoint Name |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# KIDS Build Checklist (Page 6 of 6)

**Post-Installation Worksheet:**

| Task | Checkpoint Callback (Tag^Routine Reference) | Parameter Used by Checkpoint | Checkpoint Name |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Index

## $


$$%H^XLFDT, 26-4
$$ABS^XLFMTH, 26-37
$$ACCESS^XQCHK, 14-10
$$ACOS^XLFMTH, 26-38
$$ACOSDEG^XLFMTH, 26-38
$$ACOSH^XLFHYPER, 26-28
$$ACOT^XLFMTH, 26-39
$$ACOTDEG^XLFMTH, 26-40
$$ACOTH^XLFHYPER, 26-29
$$ACSC^XLFMTH, 26-40
$$ACSCDEG^XLFMTH, 26-41
$$ACSCH^XLFHYPER, 26-29
$$ACTIVE^XUAF4, 12-1
$$ACTIVE^XUSER, 24-6
$$ACTJ^%ZOSV, 17-4
$$ADD^XPDMENU, 14-5
$$ADD^XUSERNEW, 20-15
$$ADDRESS^XLFNSLK, 5-1
$$ASEC^XLFMTH, 26-42
$$ASECDEG^XLFMTH, 26-42
$$ASECH^XLFHYPER, 26-30
$$ASIN^XLFMTH, 26-43
$$ASINDEG^XLFMTH, 26-44
$$ASINH^XLFHYPER, 26-31
$$ASKSTOP^%ZTLOAD, 22-34
$$ATAN^XLFMTH, 26-44
$$ATANDEG^XLFMTH, 26-45
$$ATANH^XLFHYPER, 26-31
$$ATTRIB^MXMLDOM, 27-1
$$AVJ^%ZOSV, 17-4
$$BASE^XLFUTL, 26-82
$$BLDNAME^XLFNAME, 16-1
$$BSA^XLFMSMT, 26-65
$$CCD^XLFUTL, 26-83
$$CHECKAV^XUSRB, 20-17
$$CHECKAV^XUVERIFY, 20-25
$$CHILD^MXMLDOM, 27-2
$$CHKSUM^XUSESIG1, 6-2
$$CIRN^XUAF4, 12-4
$$CJ^XLFSTR, 26-72
$$CLEANC^XLFNAME, 16-4
$$CMNT^MXMLDOM, 27-3
$$CMP^XUSESIG1, 6-3
$$CNV^XLFUTL, 26-84
$$CODE2TXT^XUA4A72, 24-1
$$COMCP^XPDUTL, 13-55
$$COS^XLFMTH, 26-46
$$COSDEG^XLFMTH, 26-46
$$COSH^XLFHYPER, 26-32
$$COT^XLFMTH, 26-47
$$COTDEG^XLFMTH, 26-48
$$COTH^XLFHYPER, 26-33
$$CREATE^XUSAP, 20-12
$$CSC^XLFMTH, 26-48
$$CSCDEG^XLFMTH, 26-49
$$CSCH^XLFHYPER, 26-33
$$CURCP^XPDUTL, 13-56
$$CURRSURO^XQALSURO, 3-33
$$DE^XUSESIG1, 6-4
$$DEA^XUSER, 24-8
$$DEC^XLFUTL, 26-85
$$DECDMS^XLFMTH, 26-50
$$DECRYP^XUSRB1, 20-21
$$DEFDIR^%ZISH, 11-3
$$DEL^%ZISH, 11-4
$$DEV^XUTMDEVQ, 22-11
$$DMSDEC^XLFMTH, 26-51
$$DOW^XLFDT, 26-5
$$DT^XLFDT, 26-6
$$DTIME^XUP, 24-4
$$DTR^XLFMTH, 26-51
$$E^XLFMTH, 26-52
$$EC^%ZOSV, 7-1
$$EN^MXMLDOM, 27-6
$$EN^XUA4A71, 15-2
$$EN^XUSESIG1, 6-4
$$ENCRYP^XUSRB1, 20-22
$$ESBLOCK^XUSESIG1, 6-5
$$EXP^XLFMTH, 26-53
$$FIPS^XIPUTIL, 2-2
$$FIPSCHK^XIPUTIL, 2-3
$$FMADD^XLFDT, 26-6
$$FMDIFF^XLFDT, 26-7
$$FMNAME^XLFNAME, 16-6
$$FMTE^XLFDT, 26-9
$$FMTH^XLFDT, 26-13
$$FMTHL7^XLFDT, 26-14
$$FTG^%ZISH, 11-5
$$GATF^%ZISH, 11-7
$$GET^XUA4A72, 24-2
$$GET^XUPARAM, 20-6

$$GET1^DID, 13-33
$$GETSURO^XQALSURO, 3-34
$$GTF^%ZISH, 11-8
$$HADD^XLFDT, 26-15
$$HANDLE^XUSRB4, 20-23
$$HDIFF^XLFDT, 26-16
$$HL7TFM^XLFDT, 26-17
$$HLNAME^XLFNAME, 16-9
$$HTE^XLFDT, 26-19
$$HTFM^XLFDT, 26-21
$$ID^XUAF4, 12-6
$$IDX^XUAF4, 12-7
$$IEN^XUAF4, 12-8
$$IEN^XUMF, 12-24
$$IEN2CODE^XUA4A72, 24-3
$$INHIBIT^XUSRB, 20-18
$$INVERT^XLFSTR, 26-73
$$JOB^%ZTLOAD, 22-38
$$KCHK^XUSRB, 24-16
$$KSP^XUPARAM, 20-7
$$LAST^XPDUTL, 13-57
$$LEGACY^XUAF4, 12-9
$$LENGTH^XLFMSMT, 26-66
$$LGR^%ZOSV, 17-6
$$LIST^%ZISH, 11-9
$$LJ^XLFSTR, 26-73
$$LKUP^XPDKEY, 18-3
$$LKUP^XUAF4, 12-10
$$LKUP^XUPARAM, 20-8
$$LN^XLFMTH, 26-53
$$LOG^XLFMTH, 26-54
$$LOOKUP^XUSER, 24-12
$$LOW^XLFSTR, 26-74
$$MADD^XUAF4, 12-12
$$MAX^XLFMTH, 26-55
$$MIN^XLFMTH, 26-56
$$MV^%ZISH, 11-10
$$NAM^XUSER, 24-14
$$NAME^MXMLDOM, 27-7
$$NAME^XUAF4, 12-13
$$NAMEFMT^XLFNAME, 16-13
$$NEWCP^XPDUTL, 13-59
$$NEWERR^%ZTER, 7-4
$$NNT^XUAF4, 12-14
$$NODEV^XUTMDEVQ, 22-15
$$NOW^XLFDT, 26-22
$$NS^XUAF4, 12-14
$$O99^XUAF4, 12-15
$$OPTDE^XPDUTL, 13-60
$$OS^%ZOSV, 17-8
$$PADD^XUAF4, 12-16
$$PARCP^XPDUTL, 13-61
$$PARENT^MXMLDOM, 27-8
$$PATCH^XPDUTL, 13-62
$$PENDING^XQALBUTL, 3-9
$$PI^XLFMTH, 26-56
$$PKG^XPDUTL, 13-62
$$PKGPAT^XPDIP, 13-53
$$PKGPEND^XQALBUTL, 3-10
$$PRNT^XUAF4, 12-18
$$PROD^XUPROD, 20-9
$$PRODE^XPDUTL, 13-63
$$PROVIDER^XUSER, 24-15
$$PSET^%ZTLOAD, 22-41
$$PWD^%ZISH, 11-12
$$PWR^XLFMTH, 26-57
$$QQ^XUTMDEVQ, 22-17
$$RENAME^XPDKEY, 18-4
$$REPEAT^XLFSTR, 26-75
$$REPLACE^XLFSTR, 26-76
$$REQQ^XUTMDEVQ, 22-20
$$RES^XUDHSET, 4-5
$$REWIND^%ZIS, 4-21
$$RF^XUAF4, 12-19
$$RJ^XLFSTR, 26-77
$$RT^XUAF4, 12-20
$$RTD^XLFMTH, 26-58
$$RTNUP^XPDUTL, 13-64
$$S^%ZTLOAD, 22-48
$$SCH^XLFDT, 26-23
$$SD^XLFMTH, 26-59
$$SEC^XLFDT, 26-25
$$SEC^XLFMTH, 26-60
$$SECDEG^XLFMTH, 26-60
$$SECH^XLFHYPER, 26-34
$$SENTENCE^XLFSTR, 26-78
$$SETUP1^XQALERT, 3-25
$$SIBLING^MXMLDOM, 27-9
$$SIN^XLFMTH, 26-61
$$SINDEG^XLFMTH, 26-62
$$SINH^XLFHYPER, 26-35
$$SQRT^XLFMTH, 26-62
$$STA^XUAF4, 12-22
$$STATUS^%ZISH, 11-13
$$STRIP^XLFSTR, 26-78
$$SYMENC^MXMLUTL, 27-17
$$TAN^XLFMTH, 26-63
$$TANDEG^XLFMTH, 26-64
$$TANH^XLFHYPER, 26-36
$$TEMP^XLFMSMT, 26-67
$$TEXT^MXMLDOM, 27-10
$$TF^XUAF4, 12-22

$$TITLE^XLFSTR, 26-79
$$TM^%ZTLOAD, 22-51
$$TRIM^XLFSTR, 26-80
$$TZ^XLFDT, 26-26
$$UP^XLFSTR, 26-81
$$UPCP^XPDUTL, 13-65
$$VALUE^MXMLDOM, 27-12
$$VCD^XLFUTL, 26-85
$$VER^XPDUTL, 13-66
$$VERCP^XPDUTL, 13-67
$$VERSION^%ZOSV, 17-13
$$VERSION^XPDUTL, 13-68
$$VOLUME^XLFMSMT, 26-68
$$WEIGHT^XLFMSMT, 26-70
$$WHAT^XUAF4, 12-23
$$WITHIN^XLFDT, 26-27
$$XMLHDR^MXMLUTL, 27-18

## ^

^%ZIS, 4-6
^%ZISC, 4-22
^%ZTER, 7-2
^%ZTLOAD, 22-5
^XGF Direct Mode Utilities, 25-1
^XGFDEMO Direct Mode Utility, 25-1
^XQDATE, 15-1
^XUP Direct Mode Utility, 20-1
^XUP Routine, 14-4
^XUS Direct Mode Utility, 20-1
^XUSCLEAN Direct Mode Utility, 20-2
^XUSEC Global, 18-1
^XUSESIG, 6-1
^XUVERIFY, 20-24
^XUWORKDY, 15-3

## A

Aborting an Installation During the Pre-Install Routine (KIDS), 13-33
Aborting Installations During the Environment Check (KIDS), 13-28
Accessing Questions and Answers (KIDS), 13-37
Acronyms (ISS)
  Home Page Web Address, Glossary, 6
ACTION^XQALERT, 3-15
ACTION^XQH4, 10-4
Adding New Users
  $$ADD^XUSERNEW, 20-15
Address Hygiene
  $$FIPS^XIPUTIL, 2-2
  $$FIPSCHK^XIPUTIL, 2-3
  APIs, 2-1
  CCODE^XIPUTIL, 2-1
  POSTAL^XIPUTIL, 2-4
  POSTALB^XIPUTIL, 2-6
  Programmer Tools, 2-1
Advanced Build Techniques (KIDS), 13-26
AHISTORY^XQALBUTL, 3-4
AK.Keyname Cross-reference, 18-1
ALERT File (#8992), 3-1, 3-3, 3-15, 3-32
Alert Identifier, 3-2
ALERT TRACKING File (#8992.1), 3-2, 3-4, 3-5, 3-6, 3-13, 3-14, 3-18, 3-25
ALERTDAT^XQALBUTL, 3-6
Alerts
  $$CURRSURO^XQALSURO, 3-33
  $$GETSURO^XQALSURO, 3-34
  $$PENDING^XQALBUTL, 3-9
  $$PKGPEND^XQALBUTL, 3-10
  $$SETUP1^XQALERT, 3-25
  ACTION^XQALERT, 3-15
  AHISTORY^XQALBUTL, 3-4
  Alert Identifier, 3-2
  ALERTDAT^XQALBUTL, 3-6
  APIs, 3-4
  DELETE^XQALERT, 3-16, 3-17
  DELSTAT^XQALBUTL, 3-7
  FORWARD^XQALFWD, 3-31
  GETACT^XQALERT, 3-18
  Glossary, 3-3
  NOTIPURG^XQALBUTL, 3-8
  Package Identifier, 3-2
  PATIENT^XQALERT, 3-19
  Programmer Tools, 3-1
  PTPURG^XQALBUTL, 3-11
  RECIPURG^XQALBUTL, 3-12
  REMVSURO^XQALSURO, 3-35
  SETSURO1^XQALSURO, 3-36
  SETUP^XQALERT, 3-21
  USER^XQALERT, 3-30
  USERDATA^XQALBUTL, 3-13
  USERLIST^XQALBUTL, 3-14
Alpha/Beta Tracking (KIDS), 13-45
APIs
  XRT0 Output Parameter, Start Time, 17-11
  XRTN Input Parameter, Routine Name, 17-12
APP PROXY ALLOWED Field (#.11), 20-12
Appending Text to a Server Bulletin or Mailman Reply, 19-1
Appendix A—KIDS Build Checklists, A, 1

Application Program Interface (API)
Address Hygiene, 2-1
Alerts, 3-4
Device Handler, 4-2
DNS, 5-1
Electronic Signatures, 6-1
Error Processing, 7-1
Field Monitoring, 8-1
Help Processor, 10-2
Host Files, 11-1
Institution File, 12-1
Menu Manager, 14-5
Miscellaneous, 15-1
Name Standardization, 16-1
Operating System, 17-1
Security Keys, 18-2
Signon/Security, 20-6
Spooling, 21-2
TaskMan, 22-11
Unwinder, 23-1
User, 24-1
XGF Function Library, 25-2
XLF Function Library, 26-1
XML, 27-1
Application Programming Interface (API)
KIDS, 13-49
Application Proxy User, 20-12, 20-13
Ask if Production Account Option, 20-9
Ask Installation Questions, How to (KIDS), 13-35
Assumptions About the Reader, xxx
AUTO MENU, 14-2
AVHLPTXT^XUS2, 20-11

## B

BMES^XPDUTL, 13-54
Build Checklists (KIDS), 13-5
Build Entries (KIDS), 13-2
BUILD File (#9.6), 13-2, 13-7, 13-27, 13-31, 13-33, 13-42, 13-62, 13-66
Build Name (KIDS), 13-7

## C

CALL^%ZISTCP, 4-36
Calling
^%ZTLOAD to Create Tasks (TaskMan), 22-1
^%ZTLOAD Within a Task (TaskMan), 22-7
Device Handler (^%ZIS) Within a Task (TaskMan), 22-8
EN^XUTMDEVQ to Create Tasks (TaskMan), 22-2
Callout Boxes, xxviii
Capacity Management
Response Time Measures
APIs
XRT0 Output Parameter, Start Time, 17-11
XRTN Input Parameter, Routine Name, 17-12
Capacity Planning
National Database, 17-7
CCODE^XIPUTIL, 2-1
CDSYS^XUAF4, 12-2
Checking
For Background Execution
ZTQUEUED (TaskMan), 22-6
For Stop Requests (TaskMan), 22-5
Checklists
KIDS, A, 1
Checkpoint Parameter Node, 13-39
Checkpoints *With* Callbacks, 13-39
Checkpoints *Without* Callbacks (Data Storage), 13-41
CHGA^XGF, 25-2
CHILDREN^XUAF4, 12-2
Choosing What Data to Send with a File (KIDS), 13-12
CLEAN^XGF, 25-4
CLEAR^XGF, 25-5
CLOSE^%ZISH, 11-2
CLOSE^%ZISTCP, 4-37
CLOSE^%ZISUTL, 4-38
CLOSEST PRINTER Field, 4-14
CMNT^MXMLDOM, 27-4
Collecting Error Reports (Developer's Domain), 13-47
Contents, vii
Controlling
The Disable Options/Protocols Prompt (KIDS), 13-29
The Move Routines to Other CPUs Prompt (KIDS), 13-30
The Queueing of the Install Prompt (KIDS), 13-29
Convert
$H to External Format, 26-19
$H to VA FileMan Date Format, 26-21
$H/VA FileMan date to Seconds, 26-25

Another Base to Base 10, 26-85
Base 10 to Another Base, 26-84
Between Two Bases, 26-82
Decimals to Degrees:Minutes:Seconds, 26-50
Degrees to Radians, 26-51
Degrees:Minutes:Seconds to Decimal, 26-51
Domain Name to IP Addresses, 5-1
HL7 Date to VA FileMan Date, 26-17
HL7 Formatted Name to Name, 16-6
Length Measurement, 26-66
Name to HL7 Formatted Name, 16-9
Radians to Degrees, 26-58
Seconds to $H, 26-4
String to Lowercase, 26-74
String to Soundex, 15-2
String to Uppercase, 26-81
Temperature Measurement, 26-67
VA FileMan Date to $H, 26-13
VA FileMan Date to External Format, 26-9
VA FileMan Date to HL7 Date, 26-14
Volume Measurement, 26-68
Weight Measurement, 26-70
Copy Build to Build (KIDS), 13-4
COUNTY CODE File (#5.13), 2-4, 2-6
Create a Build Using Namespace (KIDS), 13-3
Creating
Tasks Using Scheduled Options (TaskMan), 22-2
Creating a Package-specific User Termination Action, 20-4
Creating Builds (KIDS), 13-1
Creating Options, 14-1
Creating Transport Globals that Install Efficiently (KIDS), 13-25
Customizing a Server Bulletin, 19-2
CVC^XUSRB, 20-18

## D

Data Dictionary
Data Dictionary Utilities Menu, xxx
Listings, xxx
Data Dictionary Cleanup (KIDS), 13-15
Data Dictionary Update (KIDS), 13-8
Databases
Capacity Planning National Database, 17-7
Date Functions
$$$H^XLFDT, 26-4
$$DOW^XLFDT, 26-5
$$DT^XLFDT, 26-6
$$FMADD^XLFDT, 26-6
$$FMDIFF^XLFDT, 26-7
$$FMTE^XLFDT, 26-9
$$FMTH^XLFDT, 26-13
$$FMTHL7^XLFDT, 26-14
$$HADD^XLFDT, 26-15
$$HDIFF^XLFDT, 26-16
$$HL7TFM^XLFDT, 26-17
$$HTE^XLFDT, 26-19
$$HTFM^XLFDT, 26-21
$$NOW^XLFDT, 26-22
$$SCH^XLFDT, 26-23
$$SEC^XLFDT, 26-25
$$TZ^XLFDT, 26-26
$$WITHIN^XLFDT, 26-27
Date Functions (XLF), 26-4
DE^XUSHSHP, 6-6
DEA# Field (#53.2), 24-8
DEFAULT TIMED READ (SECONDS) Field (#210), 24-4
DEL^XPDKEY, 18-2
DELCOMP^XLFNAME2, 16-23
Delete a Routine or Skip Installing (KIDS), 13-28
Delete Old (>14d) Alerts Option, 3-21, 3-23, 3-27
DELETE^MXMLDOM, 27-5
DELETE^XPDMENU, 14-6
DELETE^XQALERT, 3-16
DELETEA^XQALERT, 3-17
DELSTAT^XQALBUTL, 3-7
DESC^%ZTLOAD, 22-35
Determining How Data is Installed at the Receiving Site (KIDS), 13-13
DEVICE File (#3.5), 3-32, 4-6, 4-8, 4-9, 4-12, 4-13, 4-14, 4-15, 4-20, 11-1, 24-4
Device Handler
$$RES^XUDHSET, 4-5
$$REWIND^%ZIS, 4-21
$I, 4-10
^%ZIS, 4-6
^%ZISC, 4-22
APIs, 4-2
CALL^%ZISTCP, 4-36
CLOSE^%ZISTCP, 4-37
CLOSE^%ZISUTL, 4-38
Device Type, 4-13
DEVICE^XUDHGUI, 4-2
ENDR^%ZISS, 4-25
ENS^%ZISS, 4-26
GKILL^%ZISS, 4-33
GSET^%ZISS, 4-34

Help Frames, 4-18
HLP1^%ZIS, 4-18
HLP2^%ZIS, 4-18
HOME^%ZIS, 4-19
KILL^%ZISS, 4-35
Multiple Devices and ^%ZIS, 4-17
OPEN^%ZISUTL, 4-39
PKILL^%ZISP, 4-23
Programmer Tools, 4-1
RMDEV^%ZISUTL, 4-41
SAVDEV^%ZISUTL, 4-42
Subtype, 4-12
USE^%ZISUTL, 4-43
DEVICE^XUDHGUI, 4-2
Devices
Rewinding, 4-21
Dialog Entries (KIDS), 13-19
DIALOG File (#.84), 13-19
DIFROM, 13-1, 13-2, 13-10, 13-26, 13-32, 13-33
DIFROM Variable, 13-27, 13-33
Direct Mode Utilities
^XGF, 25-1
^XGFDEMO, 25-1
^XUP, 20-1
^XUS, 20-1
^XUSCLEAN, 20-2
^ZU, 20-2
H^XUS, 20-2
Menu Manager, 14-4
Signon/Security, 20-1
DISABLE, 13-29
Discontinuation
USER TERMINATE ROUTINE, 20-4
DISP^XQORM1, 23-5
DISP^XUTMOPT, 22-22
DIV4^XUSER, 24-11
DIVGET^XUSRB2, 24-17
DIVSET^XUSRB2, 24-18
DLAYGO
^DIC Calls, 9-2
^DIE Calls, 9-3
When Navigating to Files, 9-2
DNS
APIs, 5-1
DNS IP Field (#8989.3,51), 5-1
Documentation
History, iii
Symbols, xxvii
DOLRO^%ZOSV, 17-5
Domain, 20-7
DOMAIN File (#4.2), 20-7
Domain Name Service (DNS)
$$ADDRESS^XLFNSLK, 5-1
MAIL^XLFNSLK, 5-2
Programmer Tools, 5-1
DQ^%ZTLOAD, 22-35
DSD^ZISPL, 21-2
DSDOC^ZISPL, 21-2
DUZ("AG"),, 20-1
DUZ(0), 9-1
DUZ(2), 20-1

## E

Edit a Build
Components
Dialog entries, 13-19
Forms, 13-20
Options, 13-17
Protocols, 13-17
Routines, 13-18
Templates, 13-20
Components (KIDS), 13-16
File List
Data Dictionary Update (KIDS), 13-8
DD (Full or Partial) (KIDS), 13-10
Sending Security Codes (KIDS), 13-8
Files (KIDS), 13-8
Name & Version, Build Information (KIDS), 13-7
Edit a Build (KIDS), 13-4
Edit a Build—Screen 4 (KIDS), 13-34
Edit Options, 14-1
EDIT^XUTMOPT, 22-22
Electronic Signatures
$$CHKSUM^XUSESIG1, 6-2
$$CMP^XUSESIG1, 6-3
$$DE^XUSESIG1, 6-4
$$EN^XUSESIG1, 6-4
$$ESBLOCK^XUSESIG1, 6-5
^XUSESIG, 6-1
APIs, 6-1
DE^XUSHSHP, 6-6
EN^XUSHSHP, 6-7
HASH^XUSHSHP, 6-8
Programmer Tools, 6-1
SIG^XUSEIG, 6-2
EN^MXMLPRSE, 27-13
EN^XPDIJ, 13-52
EN^XQH, 10-2
EN^XQOR, 23-1

EN^XQORM, 23-4
EN^XUSHSHP, 6-7
EN^XUTMDEVQ, 22-13
EN^XUTMTP, 22-25
EN1^XQH, 10-3
EN1^XQOR, 23-2
ENDR^%ZISS, 4-25
ENS^%ZISS, 4-26
Entry Action Options, 14-2
Entry and Exit Execute Statements, 10-1
Environment Check is Run Twice (KIDS), 13-27
Environment Check Routine (KIDS), 13-26
ERROR LOG File (#3.075), 7-2
ERROR MESSAGES File (#3.076), 7-2
Error Processing
  $$NEWERR^%ZTER, 7-4
  ^%ZTER, 7-2
  APIs, 7-1
  Programmer Tools, 7-1
  UNWIND^%ZTER, 7-5
Errors Logged in Alpha/Beta Test (QUEUED) option, 13-46
EVE Menu, 13-1
EVS Anonymous Directories, xxxi
Exit Action Options, 14-2
EXIT^XPDID, 13-49
Exporting Globals (KIDS), 13-24

## F

F4^XUAF4, 12-4
Field Level Protection, 9-1
Field Monitoring
  APIs, 8-1
  OPKG^XUHUI, 8-1
  Programmer Tools, 8-1
Fields
  APP PROXY ALLOWED (#.11), 20-12
  CLOSEST PRINTER, 4-14
  DEA# (#53.2), 24-8
  DEFAULT TIMED READ (SECONDS) (#210), 24-4
  DNS IP (#8989.3,51), 5-1
  OPEN PARAMETERS, 4-8, 4-12
  PACKAGE FILE LINK, 13-43, 13-45
  PRE-TRANSPORTATION ROUTINE f(#900), 13-31
  Protection, 9-1
  TIME ZONE (#1), 26-26
  TIMED READ (# OF SECONDS) (#200.1), 24-4
  TIMED READ (# OF SECONDS) (#51.1), 24-4
  USE PARAMETERS, 4-12
  USER CLASS (#9.5), 20-12
  USER TERMINATE ROUTINE, 20-4
  USER TERMINATE TAG, 20-4
Figures and Tables, xxiii
File Access Security
  DLAYGO
    ^DIC Calls, 9-2
    ^DIE Calls, 9-3
    When Navigating to Files, 9-2
  Field Level Protection, 9-1
  File Navigation, 9-1
  Programmer Tools, 9-1
File Navigation, 9-1
Files
  ALERT (#8992), 3-1, 3-3, 3-15, 3-32
  ALERT TRACKING (#8992.1), 3-2, 3-4, 3-5, 3-6, 3-13, 3-14, 3-18, 3-25
  BUILD (#9.6), 13-2, 13-7, 13-27, 13-31, 13-33, 13-42, 13-62, 13-66
  COUNTY CODE (#5.13), 2-4, 2-6
  DEVICE (#3.5), 3-32, 4-6, 4-8, 4-9, 4-12, 4-13, 4-14, 4-15, 4-20, 11-1, 24-4
  DIALOG (#.84), 13-19
  DOMAIN (#4.2), 20-7
  ERROR LOG (#3.075), 7-2
  ERROR MESSAGES (#3.076), 7-2
  HELP FRAME (#9.2), 10-1, 10-2, 10-3, 10-4
  HL7 MESSAGE TEXT (#772), 12-26
  HOLIDAY (#40.5), 15-3
  INDEX (#.11), 16-24
  INSTALL (#9.7), 13-49, 13-52, 13-54, 13-58, 13-59, 13-61, 13-65
  INSTITUTION (#4), 12-1, 12-4, 12-6, 12-7, 12-8, 12-9, 12-10, 12-11, 12-12, 12-13, 12-14, 12-15, 12-16, 12-18, 12-19, 12-20, 12-22, 12-23, 12-25, 20-7, 24-8, 24-11
  INSTITUTION ASSOCIATION TYPES (#4.05), 12-17, 12-21
  KERNEL PARAMETERS (#8989.2), 20-6, 20-8
  KERNEL SYSTEM PARAMETERS (#8989.3), 5-1, 11-3, 13-48, 20-7, 20-15, 24-4
  MAILMAN SITE PARAMETERS (#4.3), 26-26

MAILMAN TIME ZONE (#4.4), 26-17, 26-26
NAME COMPONENTS (#20), 16-3, 16-9, 16-11, 16-13, 16-17, 16-23, 16-24, 16-25
NAME COMPONENTS File (#20), 16-1, 16-23
NEW PERSON (#200), 3-9, 3-10, 3-12, 3-13, 3-32, 3-33, 3-34, 3-35, 3-36, 6-2, 6-5, 6-7, 6-8, 8-2, 8-3, 16-3, 16-11, 16-17, 16-24, 18-1, 18-2, 20-3, 20-4, 20-12, 20-13, 20-15, 24-2, 24-4, 24-6, 24-7, 24-8, 24-11, 24-12, 24-14, 24-15, 24-16
OBJECT (#2005), 10-1
OPTION (#19), 8-1, 13-17, 14-2, 14-3, 14-9, 20-14, 22-3, 22-22, 22-23, 22-24, 23-2
OPTION SCHEDULING (#19.2), 13-17, 22-1, 22-3, 22-22
PACKAGE (#9.4), 13-2, 13-34, 13-42, 13-43, 13-44, 13-45, 13-53, 13-68, 20-4
PATIENT (#2), 3-2, 3-11, 3-19, 3-20
PERSON CLASS (#8932.1), 24-1, 24-3
PROTOCOL (#101), 8-1, 23-1, 23-2
REMOTE PROCEDURE (#8994), 20-12
ROUTINE (#9.8), 13-18
SECURITY KEY (#19.1), 18-1, 18-2
SIGN-ON LOG (#3.081), 20-2
SPOOL DATA (#3.519), 21-2
SPOOL DOCUMENT (#3.51), 21-2
STATE (#5), 2-1, 2-4, 2-6
TASK SYNC FLAG (#14.8), 22-9, 22-10
TASKS (#14.4), 22-4, 22-5, 22-6, 22-35, 22-36
TERMINAL TYPE (#3.2), 4-8, 4-12, 4-13, 4-24, 4-25, 4-27
USER CLASS (#201), 20-12
VOLUME SET (#14.5), 22-37
XQAB ERRORS LOGGED (#8991.5), 13-47
Forced Queuing, 4-16
Form Feeds, 4-12, 4-22
Forms (KIDS), 13-20
FORWARD^XQALFWD, 3-31
FRAME^XGF, 25-6
Full DD (All Fields) (KIDS), 13-10

## G

GETACT^XQALERT, 3-18
GETENV^%ZOSV, 17-6
GETPEER^%ZOSV, 20-26
GKILL^%ZISS, 4-33
Globals
^%ZOSF, 17-1
^%ZRTL, 17-12
^XUSEC, 18-1, 18-2, 18-4
XTMP, 13-21, 13-28, 13-29, 17-5, 17-7, 20-23, 22-8
XUTL, 23-5
Glossary, 1
Alerts, 3-3
Glossary (ISS)
Home Page Web Address, Glossary, 6
GSET^%ZISS, 4-34

## H

H^XUS, 20-2, 20-10
HASH^XUSHSHP, 6-8
Header Options, 14-2
Help
At Prompts, xxix
Online, xxix
Question Marks, xxix
HELP FRAME File (#9.2), 10-1, 10-2, 10-3, 10-4
Help processor
ACTION^XQH4, 10-4
EN^XQH, 10-2
EN1^XQH, 10-3
Help Processor
APIs, 10-2
Entry and Exit Execute Statements, 10-1
Link to the OBJECT File, 10-1
Programmer Tools, 10-1
History, Revisions to Documentation and Patches, iii
HL7 MESSAGE TEXT File (#772), 12-26
HLP1^%ZIS, 4-18
HLP2^%ZIS, 4-18
HOLIDAY File (#40.5), 15-3
HOLIDAY global, 15-3
Home Pages
Adobe Acrobat Quick Guide Web Address, xxxi
Adobe Web Address, xxxi
Health Systems Design and Development Web Address, xxx
ISS Acronyms Home Page Web Address, Glossary, 6
ISS Glossary Home Page Web Address, Glossary, 6
Kernel Home Page Web Address, xxx

VistA Documentation Library (VDL) Home Page Web Address, xxxi
HOME^%ZIS, 4-19
Host Files
$$DEFDIR^%ZISH, 11-3
$$DEL^%ZISH, 11-4
$$FTG^%ZISH, 11-5
$$GATF^%ZISH, 11-7
$$GTF^%ZISH, 11-8
$$LIST^%ZISH, 11-9
$$MV^%ZISH, 11-10
$$PWD^%ZISH, 11-12
$$STATUS^%ZISH, 11-13
APIs, 11-1
CLOSE^%ZISH, 11-2
OPEN^%ZISH, 11-11
Programmer Tools, 11-1
How KIDS Matches Incoming Entries with Existing Entries, 13-14
How to
Ask Installation Questions (KIDS), 13-35
Obtain Technical Information Online, xxix
Use this Manual, xxvii
Write Code to Queue Tasks, 22-1
Hunt Groups, 4-7, 4-12
Hyperbolic Trigonometric Functions
$$ACOSH^XLFHYPER, 26-28
$$ACOTH^XLFHYPER, 26-29
$$ACSCH^XLFHYPER, 26-29
$$ASECH^XLFHYPER, 26-30
$$ASINH^XLFHYPER, 26-31
$$ATANH^XLFHYPER, 26-31
$$COSH^XLFHYPER, 26-32
$$COTH^XLFHYPER, 26-33
$$CSCH^XLFHYPER, 26-33
$$SECH^XLFHYPER, 26-34
$$SINH^XLFHYPER, 26-35
$$TANH^XLFHYPER, 26-36
Hyperbolic Trigonometric Functions (XLF), 26-28

## I

INDEX File (#.11), 16-24
INIT^XPDID, 13-50
INITKB^XGF, 25-7
INSTALL File (#9.7), 13-49, 13-52, 13-54, 13-58, 13-59, 13-61, 13-65
Institution, 20-7
INSTITUTION ASSOCIATION TYPES File (#4.05), 12-17, 12-21
Institution File
$$ACTIVE^XUAF4, 12-1
$$CIRN^XUAF4, 12-4
$$ID^XUAF4, 12-6
$$IDX^XUAF4, 12-7
$$IEN^XUAF4, 12-8
$$IEN^XUMF, 12-24
$$LEGACY^XUAF4, 12-9
$$LKUP^XUAF4, 12-10
$$MADD^XUAF4, 12-12
$$NAME^XUAF4, 12-13
$$NNT^XUAF4, 12-14
$$NS^XUAF4, 12-14
$$O99^XUAF4, 12-15
$$PADD^XUAF4, 12-16
$$PRNT^XUAF4, 12-18
$$RF^XUAF4, 12-19
$$RT^XUAF4, 12-20
$$STA^XUAF4, 12-22
$$TF^XUAF4, 12-22
$$WHAT^XUAF4, 12-23
APIs, 12-1
CDSYS^XUAF4, 12-2
CHILDREN^XUAF4, 12-2
F4^XUAF4, 12-4
LOOKUP^XUAF4, 12-11
MAIN^XUMFI, 12-25
MAIN^XUMFP, 12-27
PARENT^XUAF4, 12-17
Programmer Tools, 12-1
SIBLING^XUAF4, 12-21
INSTITUTION File (#4), 12-1, 12-4, 12-6, 12-7, 12-8, 12-9, 12-10, 12-11, 12-12, 12-13, 12-14, 12-15, 12-16, 12-18, 12-19, 12-20, 12-22, 12-23, 12-25, 20-7, 24-8, 24-11
INTRO^XUSRB, 20-19
Introduction, 1-1
IOXY^XGF, 25-8
ISQED^%ZTLOAD, 22-36
ISS Acronyms
Home Page Web Address, Glossary, 6
ISS Glossary
Home Page Web Address, Glossary, 6

## K

Kernel
Home Page Web Address, xxx
Kernel Installation & Distribution System Menu, 13-1
Kernel Management Menu, 20-9

KERNEL PARAMETERS File (#8989.2), 20-6, 20-8
KERNEL SYSTEM PARAMETERS File (#8989.3), 5-1, 11-3, 13-48, 20-7, 20-15, 24-4
Key Lookup, 18-1
Key Variables
  KIDS, 13-27, 13-33
  Servers, 19-1
  Tasks, 22-3
Keywords, xxix
KIDS
  $$PKG^XPDUTL, 13-62
  $$PKGPAT^XPDIP, 13-53
  $$VER^XPDUTL, 13-66
  $$VERSION^XPDUTL, 13-68
  Alpha/Beta Tracking, 13-45
  APIs, 13-49
  Build Checklists, 13-5
  Build Entries, 13-2
  Build Name, 13-7
  Checkpoint Parameter Node, 13-39
  Checkpoints *With* Callbacks, 13-39
  Checkpoints *Without* Callbacks (Data Storage), 13-41
  Choosing What Data to Send with a File, 13-12
  Collecting Error Reports (Developer's Domain), 13-47
  Copy Build to Build, 13-4
  Create a Build Using Namespace, 13-3
  Creating Builds, 13-1
  Data Dictionary Cleanup, 13-15
  Data Dictionary Update, 13-8
  Determining How Data is Installed at the Receiving Site, 13-13
  Edit a Build, 13-4
    Components, 13-16
    Dialog Entries, 13-19
    File List
      DD (Full or Partial), 13-10
    Files, 13-8
    Forms, 13-20
    Name & Version, Build Information, 13-7
    Options and Protocols, 13-17
    Routines, 13-18
    Templates, 13-20
  Edit a Build—Screen 4, 13-34
  EN^XPDIJ, 13-52
  Environment Check, 13-26
    $$PATCH^XPDUTL, 13-62
    $$RTNUP^XPDUTL, 13-64
    Aborting Installations, 13-28
    DIFROM Variable, 13-27
    DISABLE Scheduled Options, Options, and Protocols Prompt, 13-29
    Key Variables, 13-27
    Move routines to other CPUs Prompt, 13-30
    Queueing the Install Prompt, 13-29
    Routine Install Options, 13-28
    Run Twice, 13-27
    Sample Routine, 13-31
    Self-Contained Routine, 13-26
    Verifying Patch Installation, 13-28
    Version Numbers, 13-27
    XPDENV Variable, 13-27
    XPDNM Variable, 13-27
  EXIT^XPDID, 13-49
  Exporting Globals, 13-24
  Full DD (All Fields), 13-10
  How KIDS Matches Incoming Entries with Existing Entries, 13-14
  How to Ask Installation Questions, 13-35
  INIT^XPDID, 13-50
  Installation Questions
    M Code, 13-36
    Questions and answers, 13-37
    Skipping, 13-36
    Subscripts, 13-36
    Where Asked, 13-38
  Limited Resolution of Pointers, 13-15
  M Code in Questions, 13-36
  Multi-Package Builds, 13-24
  NEW the DIFROM Variable When Calling MailMan, 13-33
  Options, 13-1
  Package File Link, 13-43
  Partial DD (Some Fields), 13-10
  Pre- and Post-Install
    Aborting installations, 13-33
  Pre- and Post-Install Routines
    $$COMCP^XPDUTL, 13-55
    $$CURCP^XPDUTL, 13-56
    $$LAST^XPDUTL, 13-57
    $$NEWCP^XPDUTL, 13-59
    $$OPTDE^XPDUTL, 13-60
    $$PARCP^XPDUTL, 13-61
    $$PRODE^XPDUTL, 13-63
    $$UPCP^XPDUTL, 13-65
    $$VERCP^XPDUTL, 13-67
    BMES^XPDUTL, 13-54
    Checkpoint Parameter Node, 13-39

Checkpoints *Without* Callbacks, 13-41
DIFROM Variable, 13-33
Key Variables, 13-33
MES^XPDUTL, 13-58
Sample Routine, 13-40
XPDNM Variable, 13-33
ZTQUEUED Variable, 13-33
Pre- and Post-Install Routines:Special Features, 13-32
PRE-TRANSPORTATION ROUTINE Field (#900), 13-31
Programmer Tools, 13-1
Advanced Build Techniques, 13-26
Question Subscripts, 13-36
Re-Indexing Files, 13-15
Required Build, 13-42
Send Alpha/Beta Usage to Developers Option, 13-47
Sending Security Codes, 13-8
Setting a File's Package Revision Data Node (Post-Install), 13-33
Setting up Alpha/Beta Tracking in the Build Entry, 13-46
Site Option to Report Errors, 13-46
Skipping Installation Questions, 13-36
Splitting a Distribution Across Diskettes, 13-22
TITLE^XPDID, 13-50
Track Package Nationally, 13-45
Transporting a distribution
Efficient builds, 13-25
Transporting a Distribution, 13-20
Turning Off Alpha/Beta Tracking, 13-47
Update the Status Bar During Pre- and Post-Install Routines, 13-34
UPDATE^XPDID, 13-51
Using Checkpoints (Pre- and Post-Install Routines), 13-38
When to Transport More than One Transport Global in a Distribution, 13-23
Where Questions Are Asked During Installations, 13-38
KIDS Build Checklists, A, 1
KILL^%ZISS, 4-35
KILL^%ZTLOAD, 22-4, 22-6, 22-39
KILL^XUSCLEAN, 20-14

## L

Limited Resolution of Pointers (KIDS), 13-15
Link
Package File Link, 13-43
Link to the OBJECT File, 10-1
List File Attributes Option, xxx
LKOPT^XPDMENU, 14-7
LOGOUT^XUSRB, 20-19
LOGRSRC^%ZOSV, 17-7
LOOKUP^XUAF4, 12-11
Lowercase
$$LOW^XLFSTR, 26-74

## M

M Code in Questions (KIDS), 13-36
MAIL^XLFNSLK, 5-2
MAILMAN SITE PARAMETERS File (#4.3), 26-26
MAILMAN TIME ZONE File (#4.4), 26-17, 26-26
MAIN^XUMFI, 12-25
MAIN^XUMFP, 12-27
Math Functions
$$ABS^XLFMTH, 26-37
$$ACOS^XLFMTH, 26-38
$$ACOSDEG^XLFMTH, 26-38
$$ACOT^XLFMTH, 26-39
$$ACOTDEG^XLFMTH, 26-40
$$ACSC^XLFMTH, 26-40
$$ACSCDEG^XLFMTH, 26-41
$$ASEC^XLFMTH, 26-42
$$ASECDEG^XLFMTH, 26-42
$$ASIN^XLFMTH, 26-43
$$ASINDEG^XLFMTH, 26-44
$$ATAN^XLFMTH, 26-44
$$ATANDEG^XLFMTH, 26-45
$$COS^XLFMTH, 26-46
$$COSDEG^XLFMTH, 26-46
$$COT^XLFMTH, 26-47
$$COTDEG^XLFMTH, 26-48
$$CSC^XLFMTH, 26-48
$$CSCDEG^XLFMTH, 26-49
$$DECDMS^XLFMTH, 26-50
$$DMSDEC^XLFMTH, 26-51
$$DTR^XLFMTH, 26-51
$$E^XLFMTH, 26-52
$$EXP^XLFMTH, 26-53
$$LN^XLFMTH, 26-53
$$LOG^XLFMTH, 26-54
$$MAX^XLFMTH, 26-55
$$MIN^XLFMTH, 26-56
$$PI^XLFMTH, 26-56
$$PWR^XLFMTH, 26-57

$$RTD^XLFMTH, 26-58
$$SD^XLFMTH, 26-59
$$SEC^XLFMTH, 26-60
$$SECDEG^XLFMTH, 26-60
$$SIN^XLFMTH, 26-61
$$SINDEG^XLFMTH, 26-62
$$SQRT^XLFMTH, 26-62
$$TAN^XLFMTH, 26-63
$$TANDEG^XLFMTH, 26-64
Math Functions (XLF), 26-37
Measurement Functions
$$BSA^XLFMSMT, 26-65
$$LENGTH^XLFMSMT, 26-66
$$TEMP^XLFMSMT, 26-67
$$VOLUME^XLFMSMT, 26-68
$$WEIGHT^XLFMSMT, 26-70
Measurement Functions (XLF), 26-65
Menu Manager
$$ACCESS^XQCHK, 14-10
APIs, 14-5
Creating Options, 14-1
DELETE^XPDMENU, 14-6
Direct Mode Utilities, 14-4
LKOPT^XPDMENU, 14-7
NEXT^XQ92, 14-9
OP^XQCHK, 14-12
Option Types, 14-1
OUT^XPDMENU, 14-8
Programmer Tools, 14-1
RENAME^XPDMENU, 14-9
SSADD^XPDMENU, 14-5
Variables for Programmer Use, 14-2
XQ1, 14-4
XQMM("A") Variable, 14-2
XQMM("B") Variable, 14-3
XQMM("J") Variable, 14-3
XQMM("N") Variable, 14-3
XQUIT Variable, 14-2
Menus
Data Dictionary Utilities, xxx
EVE, 13-1
Kernel Installation & Distribution System, 13-1
Kernel Management Menu, 20-9
Programmer Options, 13-1
Systems Manager Menu, 13-1
XPD MAIN, 13-1
XUKERNEL, 20-9
XUPROG, 13-1
MES^XPDUTL, 13-58
Miscellaneous
$$EN^XUA4A71, 15-2
^XQDATE, 15-1
^XUWORKDY, 15-3
APIs, 15-1
Programmer Tools, 15-1
Move routines to other CPUs Prompt (KIDS), 13-30
MSG^XQOR, 23-3
Multi-Package Builds (KIDS), 13-24
MUMPS OPERATING SYSTEM file (#.7), 13-14
MXMLDOM
$$ATTRIB^MXMLDOM, 27-1
$$CHILD^MXMLDOM, 27-2
$$CMNT^MXMLDOM, 27-3
$$EN^MXMLDOM, 27-6
$$NAME^MXMLDOM, 27-7
$$PARENT^MXMLDOM, 27-8
$$SIBLING^MXMLDOM, 27-9
$$TEXT^MXMLDOM, 27-10
$$VALUE^MXMLDOM, 27-12
CMNT^MXMLDOM, 27-4
DELETE^MXMLDOM, 27-5
EN^MXMLPRSE, 27-13
TEXT^MXMLDOM, 27-11
MXMLUTL
$$SYMENC^MXMLUTL, 27-17
$$XMLHDR^MXMLUTL, 27-18

## N

NAME COMPONENTS File (#20), 16-1, 16-3, 16-9, 16-11, 16-13, 16-17, 16-23, 16-24, 16-25
Name Standardization
$$BLDNAME^XLFNAME, 16-1
$$CLEANC^XLFNAME, 16-4
$$FMNAME^XLFNAME, 16-6
$$HLNAME^XLFNAME, 16-9
$$NAMEFMT^XLFNAME, 16-13
APIs, 16-1
DELCOMP^XLFNAME2, 16-23
NAMECOMP^XLFNAME, 16-12
Programmer Tools, 16-1
STDNAME^XLFNAME, 16-18
UPDCOMP^XLFNAME2, 16-24
NAMECOMP^XLFNAME, 16-12
National Database
Capacity Planning, 17-7
Navigation
DLAYGO, 9-2

Files, 9-1
NEW PERSON File (#200), 3-9, 3-10, 3-12, 3-13, 3-32, 3-33, 3-34, 3-35, 3-36, 6-2, 6-5, 6-7, 6-8, 8-2, 8-3, 16-3, 16-11, 16-17, 16-24, 18-1, 18-2, 20-3, 20-4, 20-12, 20-13, 20-15, 24-2, 24-4, 24-6, 24-7, 24-8, 24-11, 24-12, 24-14, 24-15, 24-16
NEW the DIFROM Variable When Calling MailMan (KIDS), 13-33
NEXT^XQ92, 14-9
Nodes
ACTJ, 17-1
AVJ, 17-1
BRK, 17-1
DEL, 17-1
EOFF, 17-1
EOT, 17-2
ERRTN, 17-2
ETRP, 17-2
JOBPARAM, 17-2
LABOFF, 17-2
LOAD, 17-2
LPC, 17-2
MAGTAPE, 17-2
MAXSIZ, 17-2
MGR, 17-2
MTBOT, 17-2
MTERR, 17-2
MTONLINE, 17-2
MTWPROT, 17-2
NBRK, 17-2
NO-PASSALL, 17-2
NO-TYPE-AHEAD, 17-2
OS, 17-3
PASSALL, 17-3
PRIINQ, 17-3
PRIORITY, 17-3
PROD, 17-3
PROGMODE, 17-3
RD, 17-3
RESJOB, 17-3
RM, 17-3
RSEL, 17-3
RSUM, 17-3
SAVE, 17-3
SIZE, 17-3
SS, 17-3
TEST, 17-3
TMK, 17-3
TRAP, 17-3
TRMOFF, 17-3
TRMON, 17-3
TRMRD, 17-3
UCI, 17-3
UCICHECK, 17-3
UPPERCASE, 17-3
VOL, 17-3
XY, 17-3
ZD, 17-3
NOTIPURG^XQALBUTL, 3-8

## O

OBJECT File (#2005), 10-1
Obsolete
$$NEWERR^%ZTER, 7-4
^XQDATE, 15-1
^XUWORKDY, 15-3
D H^XUS, 20-2
T0^%ZOSV, 17-11
T1^%ZOSV, 17-12
USER TERMINATE ROUTINE Option, 20-4
Online
Documentation, xxix
Technical Information, How to Obtain, xxix
Online Help Frames, xxix
OP^XQCHK, 14-12
OPEN PARAMETERS Field, 4-8, 4-12
OPEN^%ZISH, 11-11
OPEN^%ZISUTL, 4-39
Operating System
APIs, 17-1
Operating System Interface
$$ACTJ^%ZOSV, 17-4
$$AVJ^%ZOSV, 17-4
$$EC^%ZOSV, 7-1
$$LGR^%ZOSV, 17-6
$$OS^%ZOSV, 17-8
$$VERSION^%ZOSV, 17-13
DOLRO^%ZOSV, 17-5
GETENV^%ZOSV, 17-6
LOGRSRC^%ZOSV, 17-7
Programmer Tools, 17-1
SETENV^%ZOSV, 17-9
SETNM^%ZOSV, 17-10
T0^%ZOSV, 17-11
T1^%ZOSV, 17-12
OPKG^XUHUI, 8-1
OPTION File (#19), 8-1, 13-17, 14-2, 14-3, 14-9, 20-14, 22-3, 22-22, 22-23, 22-24, 23-2
Entry Action, 14-2
Exit Action, 14-2

Header, 14-2
OPTION SCHEDULING File (#19.2), 13-17, 22-1, 22-3, 22-22
OPTION^%ZTLOAD, 22-40
Options
Ask if Production Account Option, 20-9
Creating, 14-1
Data Dictionary Utilities, xxx
Delete Old (>14d) Alerts, 3-21, 3-23, 3-27
EVE, 13-1
Kernel Installation & Distribution System, 13-1
Kernel Management Menu, 20-9
KIDS, 13-1, 13-17
List File Attributes, xxx
Programmer Options, 13-1
Regularly Scheduled, 14-2
Startup PROD check, 20-9
Systems Manager Menu, 13-1
Types, 14-1
USER TERMINATE ROUTINE (Obsolete), 20-4
XPD BUILD NAMESPACE, 13-3
XPD COPY BUILD, 13-4
XPD MAIN, 13-1
XQAB ERROR LOG SERVER, 13-47
XQAB ERROR LOG XMIT, 13-47
XQUIT (Menu Manager), 14-2
XU SID ASK, 20-9
XU SID STARTUP, 20-9
XU USER SIGN-ON, 20-3
XU USER TERMINATE, 20-4
XUEDITOPT, 14-1
XUKERNEL, 20-9
XUPROG, 13-1
OPTSTAT^XUTMOPT, 22-23
Orientation, Kernel Programmer Manual, xxvii
OUT^XPDMENU, 14-8
OWNSKEY^XUSRB, 18-5

## P

PACKAGE File (#9.4), 13-2, 13-34, 13-42, 13-43, 13-44, 13-45, 13-53, 13-68, 20-4
Package File Link (KIDS), 13-43
PACKAGE FILE LINK Field, 13-43, 13-45
Package Identifier
Alert Identifier, 3-2
Conventions, 3-2
Package Revision Data Node, 13-33
Page Length, 4-12
Parameters, Site, 20-7
PARCP^XPDUTL, 13-39
PARENT^XUAF4, 12-17
Part 3 of Kernel Install, 9-1
Partial DD (Some Fields) (KIDS), 13-10
Patches
History, vi
PATIENT File (#2), 3-2, 3-11, 3-19, 3-20
PATIENT^XQALERT, 3-19
PCLEAR^%ZTLOAD, 22-40
PERSON CLASS File (#8932.1), 24-1, 24-3
Phantom Jump, 14-3
PKI
$$DEA^XUSER, 24-8
PKILL^%ZISP, 4-23
POSTAL^XIPUTIL, 2-4
POSTALB^XIPUTIL, 2-6
Post-Execution Commands
ZTREQ (TaskMan), 22-6
Post-execution commands - ZTREQ, 22-6
PRD^DILFD, 13-33
Pre- and Post-Install Routines
Special Features (KIDS), 13-32
PREP^XGF, 25-9
PRE-TRANSPORTATION ROUTINE Field (#900), 13-31
Programmer Options Menu, 13-1
Programmer Tools
Address Hygiene, 2-1
Alerts, 3-1
Device Handler, 4-1
Domain Name Service (DNS), 5-1
Electronic Signatures, 6-1
Error Processing, 7-1
Field Monitoring, 8-1
File Access Security, 9-1
Help Processor, 10-1
Host Files, 11-1
Institution File, 12-1
KIDS, 13-1
Menu Manager, 14-1
Miscellaneous, 15-1
Name Standardization, 16-1
Operating System Interface, 17-1
Public Key Infrastructure (PKI), 24-8
Security Keys, 18-1
Servers, 19-1
Signon/Security, 20-1
Spooling, 21-1
TaskMan, 22-1
Unwinder, 23-1

User, 24-1
XGF Function Library, 25-1
XLF Function Library, 26-1
XML, 27-1
PROTOCOL File (#101), 8-1, 23-1, 23-2
Protocols
KIDS, 13-17
Proxy
Application Proxy User, 20-12, 20-13
PSET^%ZISP, 4-24
PTPURG^XQALBUTL, 3-11
Public Key Infrastructure (PKI)
Programmer Tools, 24-8
Purging the Task Record (TaskMan), 22-5

## Q

Question Mark Help, xxix
Question Subscripts (KIDS), 13-36
Queueing the Install Prompt (KIDS), 13-29
Queuers
Non-interactive, 22-31
Queuers (TaskMan), 22-1
^%ZTLOAD, 22-1
EN^XUTMDEVQ, 22-2
Scheduled Options, 22-2
Queuing, 4-7, 4-9, 4-11
Spooler), 21-1

## R

READ^XGF, 25-10
Reader, Assumptions About the, xxx
RECIPURG^XQALBUTL, 3-12
Reference Materials, xxx
Reference Type
Controlled Subscription
$$CHECKAV^XUSRB, 20-17
$$CREATE^XUSAP, 20-12
$$KCHK^XUSRB, 24-16
^XUSESIG, 6-1
AVHLPTXT^XUS2, 20-11
CVC^XUSRB, 20-18
DELCOMP^XLFNAME2, 16-23
DIV4^XUSER, 24-11
DIVGET^XUSRB2, 24-17
DIVSET^XUSRB2, 24-18
DOLRO^%ZOSV, 17-5
DSD^ZISPL, 21-2
DSDOC^ZISPL, 21-2
EN^XPDIJ, 13-52
EN^XUTMTP, 22-25
GETPEER^%ZOSV, 20-26
INTRO^XUSRB, 20-19
LOGOUT^XUSRB, 20-19
MAIN^XUMFI, 12-25
MAIN^XUMFP, 12-27
SETUP^XUSRB, 20-20
UPDCOMP^XLFNAME2, 16-24
USERINFO^XUSRB2, 24-19
VALIDAV^XUSRB, 20-21
WITNESS^XUVERIFY, 20-26
Supported
$$%H^XLFDT, 26-4
$$ABS^XLFMTH, 26-37
$$ACCESS^XQCHK, 14-10
$$ACOS^XLFMTH, 26-38
$$ACOSDEG^XLFMTH, 26-38
$$ACOSH^XLFHYPER, 26-28
$$ACOT^XLFMTH, 26-39
$$ACOTDEG^XLFMTH, 26-40
$$ACOTH^XLFHYPER, 26-29
$$ACSC^XLFMTH, 26-40
$$ACSCDEG^XLFMTH, 26-41
$$ACSCH^XLFHYPER, 26-29
$$ACTIVE^XUAF4, 12-1
$$ACTIVE^XUSER, 24-6
$$ACTJ^%ZOSV, 17-4
$$ADD^XUSERNEW, 20-15
$$ADDRESS^XLFNSLK, 5-1
$$ASEC^XLFMTH, 26-42
$$ASECDEG^XLFMTH, 26-42
$$ASECH^XLFHYPER, 26-30
$$ASIN^XLFMTH, 26-43
$$ASINDEG^XLFMTH, 26-44
$$ASINH^XLFHYPER, 26-31
$$ASKSTOP^%ZTLOAD, 22-34
$$ATAN^XLFMTH, 26-44
$$ATANDEG^XLFMTH, 26-45
$$ATANH^XLFHYPER, 26-31
$$ATTRIB^MXMLDOM, 27-1
$$AVJ^%ZOSV, 17-4
$$BASE^XLFUTL, 26-82
$$BLDNAME^XLFNAME, 16-1
$$BSA^XLFMSMT, 26-65
$$CCD^XLFUTL, 26-83
$$CHECKAV^XUVERIFY, 20-25
$$CHILD^MXMLDOM, 27-2
$$CHKSUM^XUSESIG1, 6-2
$$CIRN^XUAF4, 12-4
$$CJ^XLFSTR, 26-72
$$CLEANC^XLFNAME, 16-4

$$CMNT^MXMLDOM, 27-3
$$CMP^XUSESIG1, 6-3
$$CNV^XLFUTL, 26-84
$$CODE2TXT^XUA4A72, 24-1
$$COMCP^XPDUTL, 13-55
$$COS^XLFMTH, 26-46
$$COSDEG^XLFMTH, 26-46
$$COSH^XLFHYPER, 26-32
$$COT^XLFMTH, 26-47
$$COTDEG^XLFMTH, 26-48
$$COTH^XLFHYPER, 26-33
$$CSC^XLFMTH, 26-48
$$CSCDEG^XLFMTH, 26-49
$$CSCH^XLFHYPER, 26-33
$$CURCP^XPDUTL, 13-56
$$CURRSURO^XQALSURO, 3-33
$$DE^XUSESIG1, 6-4
$$DEA^XUSER, 24-8
$$DEC^XLFUTL, 26-85
$$DECDMS^XLFMTH, 26-50
$$DECRYP^XUSRB1, 20-21
$$DEFDIR^%ZISH, 11-3
$$DEL^%ZISH, 11-4
$$DEV^XUTMDEVQ, 22-11
$$DMSDEC^XLFMTH, 26-51
$$DOW^XLFDT, 26-5
$$DT^XLFDT, 26-6
$$DTIME^XUP, 24-4
$$DTR^XLFMTH, 26-51
$$E^XLFMTH, 26-52
$$EC^%ZOSV, 7-1
$$EN^MXMLDOM, 27-6
$$EN^XUSESIG1, 6-4
$$ENCRYP^XUSRB1, 20-22
$$ESBLOCK^XUSESIG1, 6-5
$$EXP^XLFMTH, 26-53
$$FIPS^XIPUTIL, 2-2
$$FIPSCHK^XIPUTIL, 2-3
$$FMADD^XLFDT, 26-6
$$FMDIFF^XLFDT, 26-7
$$FMNAME^XLFNAME, 16-6
$$FMTE^XLFDT, 26-9
$$FMTH^XLFDT, 26-13
$$FMTHL7^XLFDT, 26-14
$$FTG^%ZISH, 11-5
$$GATF^%ZISH, 11-7
$$GET^XUA4A72, 24-2
$$GET^XUPARAM, 20-6
$$GETSURO^XQALSURO, 3-34
$$GTF^%ZISH, 11-8
$$HADD^XLFDT, 26-15
$$HANDLE^XUSRB4, 20-23
$$HDIFF^XLFDT, 26-16
$$HL7TFM^XLFDT, 26-17
$$HLNAME^XLFNAME, 16-9
$$HTE^XLFDT, 26-19
$$HTFM^XLFDT, 26-21
$$ID^XUAF4, 12-6
$$IDX^XUAF4, 12-7
$$IEN^XUAF4, 12-8
$$IEN^XUMF, 12-24
$$IEN2CODE^XUA4A72, 24-3
$$INHIBIT^XUSRB, 20-18
$$INVERT^XLFSTR, 26-73
$$JOB^%ZTLOAD, 22-38
$$KSP^XUPARAM, 20-7
$$LAST^XPDUTL, 13-57
$$LEGACY^XUAF4, 12-9
$$LENGTH^XLFMSMT, 26-66
$$LGR^%ZOSV, 17-6
$$LIST^%ZISH, 11-9
$$LJ^XLFSTR, 26-73
$$LKUP^XPDKEY, 18-3
$$LKUP^XUAF4, 12-10
$$LKUP^XUPARAM, 20-8
$$LN^XLFMTH, 26-53
$$LOG^XLFMTH, 26-54
$$LOOKUP^XUSER, 24-12
$$LOW^XLFSTR, 26-74
$$MADD^XUAF4, 12-12
$$MAX^XLFMTH, 26-55
$$MIN^XLFMTH, 26-56
$$MV^%ZISH, 11-10
$$NAM^XUSER, 24-14
$$NAME^MXMLDOM, 27-7
$$NAME^XUAF4, 12-13
$$NAMEFMT^XLFNAME, 16-13
$$NEWCP^XPDUTL, 13-59
$$NEWERR^%ZTER, 7-4
$$NNT^XUAF4, 12-14
$$NODEV^XUTMDEVQ, 22-15
$$NOW^XLFDT, 26-22
$$NS^XUAF4, 12-14
$$O99^XUAF4, 12-15
$$OPTDE^XPDUTL, 13-60
$$OS^%ZOSV, 17-8
$$PADD^XUAF4, 12-16
$$PARCP^XPDUTL, 13-61
$$PARENT^MXMLDOM, 27-8
$$PATCH^XPDUTL, 13-62
$$PENDING^XQALBUTL, 3-9
$$PI^XLFMTH, 26-56

$$PKG^XPDUTL, 13-62
$$PKGPAT^XPDIP, 13-53
$$PKGPEND^XQALBUTL, 3-10
$$PRNT^XUAF4, 12-18
$$PROD^XUPROD, 20-9
$$PRODE^XPDUTL, 13-63
$$PROVIDER^XUSER, 24-15
$$PSET^%ZTLOAD, 22-41
$$PWD^%ZISH, 11-12
$$PWR^XLFMTH, 26-57
$$QQ^XUTMDEVQ, 22-17
$$RENAME^XPDKEY, 18-4
$$REPEAT^XLFSTR, 26-75
$$REPLACE^XLFSTR, 26-76
$$REQQ^XUTMDEVQ, 22-20
$$RES^XUDHSET, 4-5
$$REWIND^%ZIS, 4-21
$$RF^XUAF4, 12-19
$$RJ^XLFSTR, 26-77
$$RT^XUAF4, 12-20
$$RTD^XLFMTH, 26-58
$$RTNUP^XPDUTL, 13-64
$$S^%ZTLOAD, 22-48
$$SCH^XLFDT, 26-23
$$SD^XLFMTH, 26-59
$$SEC^XLFDT, 26-25
$$SEC^XLFMTH, 26-60
$$SECDEG^XLFMTH, 26-60
$$SECH^XLFHYPER, 26-34
$$SENTENCE^XLFSTR, 26-78
$$SETUP1^XQALERT, 3-25
$$SIBLING^MXMLDOM, 27-9
$$SIN^XLFMTH, 26-61
$$SINDEG^XLFMTH, 26-62
$$SINH^XLFHYPER, 26-35
$$SQRT^XLFMTH, 26-62
$$STA^XUAF4, 12-22
$$STATUS^%ZISH, 11-13
$$STRIP^XLFSTR, 26-78
$$SYMENC^MXMLUTL, 27-17
$$TAN^XLFMTH, 26-63
$$TANDEG^XLFMTH, 26-64
$$TANH^XLFHYPER, 26-36
$$TEMP^XLFMSMT, 26-67
$$TEXT^MXMLDOM, 27-10
$$TF^XUAF4, 12-22
$$TITLE^XLFSTR, 26-79
$$TM^%ZTLOAD, 22-51
$$TRIM^XLFSTR, 26-80
$$TZ^XLFDT, 26-26
$$UP^XLFSTR, 26-81
$$UPCP^XPDUTL, 13-65
$$VALUE^MXMLDOM, 27-12
$$VCD^XLFUTL, 26-85
$$VER^XPDUTL, 13-66
$$VERCP^XPDUTL, 13-67
$$VERSION^%ZOSV, 17-13
$$VERSION^XPDUTL, 13-68
$$VOLUME^XLFMSMT, 26-68
$$WEIGHT^XLFMSMT, 26-70
$$WHAT^XUAF4, 12-23
$$WITHIN^XLFDT, 26-27
$$XMLHDR^MXMLUTL, 27-18
^%ZIS, 4-6
^%ZISC, 4-22
^%ZTER, 7-2
^%ZTLOAD, 22-26
^XQDATE, 15-1
^XUP, 20-1
^XUS, 20-1
^XUSCLEAN, 20-2
^XUVERIFY, 20-24
^ZU, 20-2
ACTION^XQALERT, 3-15
ACTION^XQH4, 10-4
AHISTORY^XQALBUTL, 3-4
ALERTDAT^XQALBUTL, 3-6
BMES^XPDUTL, 13-54
CALL^%ZISTCP, 4-36
CCODE^XIPUTIL, 2-1
CDSYS^XUAF4, 12-2
CHGA^XGF, 25-2
CHILDREN^XUAF4, 12-2
CLEAN^XGF, 25-4
CLEAR^XGF, 25-5
CLOSE^%ZISH, 11-2
CLOSE^%ZISTCP, 4-37
CLOSE^%ZISUTL, 4-38
CMNT^MXMLDOM, 27-4
DE^XUSHSHP, 6-6
DEL^XPDKEY, 18-2
DELETE^MXMLDOM, 27-5
DELETE^XPDMENU, 14-6
DELETE^XQALERT, 3-16
DELETEA^XQALERT, 3-17
DELSTAT^XQALBUTL, 3-7
DESC^%ZTLOAD, 22-35
DEVICE^XUDHGUI, 4-2
DISP^XQORM1, 23-5
DISP^XUTMOPT, 22-22
DQ^%ZTLOAD, 22-35
EDIT^XUTMOPT, 22-22

EN^MXMLPRSE, 27-13
EN^XQH, 10-2
EN^XQOR, 23-1
EN^XQORM, 23-4
EN^XUA4A71, 15-2, 15-3
EN^XUSHSHP, 6-7
EN^XUTMDEVQ, 22-13
EN1^XQH, 10-3
EN1^XQOR, 23-2
ENDR^%ZISS, 4-25
ENS^%ZISS, 4-26
EXIT^XPDID, 13-49
F4^XUAF4, 12-4
FORWARD^XQALFWD, 3-31
FRAME^XGF, 25-6
GETACT^XQALERT, 3-18
GETENV^%ZOSV, 17-6
GKILL^%ZISS, 4-33
GSET^%ZISS, 4-34
H^XUS, 20-2, 20-10
HASH^XUSHSHP, 6-8
HLP1^%ZIS, 4-18
HLP2^%ZIS, 4-18
HOME^%ZIS, 4-19
INIT^XPDID, 13-50
INITKB^XGF, 25-7
IOXY^XGF, 25-8
ISQED^%ZTLOAD, 22-36
KILL^%ZISS, 4-35
KILL^%ZTLOAD, 22-39
KILL^XUSCLEAN, 20-14
LKOPT^XPDMENU, 14-7
LOGRSRC^%ZOSV, 17-7
LOOKUP^XUAF4, 12-11
MAIL^XLFNSLK, 5-2
MES^XPDUTL, 13-58
MSG^XQOR, 23-3
NAMECOMP^XLFNAME, 16-12
NOTIPURG^XQALBUTL, 3-8
OP^XQ92, 14-9
OP^XQCHK, 14-12
OPEN^%ZISH, 11-11
OPEN^%ZISUTL, 4-39
OPKG^XUHUI, 8-1
OPTION^%ZTLOAD, 22-40
OPTSTAT^XUTMOPT, 22-23
OUT^XPDMENU, 14-8
OWNSKEY^XUSRB, 18-5
PARENT^XUAF4, 12-17
PATIENT^XQALERT, 3-19
PCLEAR^%ZTLOAD, 22-40
PKILL^%ZISP, 4-23
POSTAL^XIPUTIL, 2-4
POSTALB^XIPUTIL, 2-6
PREP^XGF, 25-9
PSET^%ZISP, 4-24
PTPURG^XQALBUTL, 3-11
READ^XGF, 25-10
RECIPURG^XQALBUTL, 3-12
REMVSURO^XQALSURO, 3-35
RENAME^XPDMENU, 14-9
REQ^%ZTLOAD, 22-42
RESCH^XUTMOPT, 22-24
RESETKB^XGF, 25-12
RESTORE^XGF, 25-13
RMDEV^%ZISUTL, 4-41
RTN^%ZTLOAD, 22-47
SAVDEV^%ZISUTL, 4-42
SAVE^XGF, 25-14
SAY^XGF, 25-15
SAYU, 25-17
SET^XUPARAM, 20-8
SET^XUS1A, 20-10
SETA^XGF, 25-19
SETENV^%ZOSV, 17-9
SETNM^%ZOSV, 17-10
SETSURO1^XQALSURO, 3-36
SETUP^XQALERT, 3-21
SIBLING^XUAF4, 12-21
SIG^XUSESIG, 6-2
SSADD^XPDMENU, 14-5
STAT^%ZTLOAD, 22-49
STDNAME^XLFNAME, 16-18
T0^%ZOSV, 17-11
T1^%ZOSV, 17-12
TEXT^MXMLDOM, 27-11
TITLE^XPDID, 13-50
UNWIND^%ZTER, 7-5
UPDATE^XPDID, 13-51
USE^%ZISUTL, 4-43
USER^XQALERT, 3-30
USERDATA^XQALBUTL, 3-13
USERLIST^XQALBUTL, 3-14
WIN^XGF, 25-21
XREF^XQORM, 23-5
ZTSAVE^%ZTLOAD, 22-51
Regularly Scheduled Options, 14-2
Re-Indexing Files (KIDS), 13-15
REMOTE PROCEDURE File (#8994), 20-12
REMVSURO^XQALSURO, 3-35
RENAME^XPDMENU, 14-9
REQ^%ZTLOAD, 22-42

Required Builds (KIDS), 13-42
RESCH^XUTMOPT, 22-24
RESETKB^XGF, 25-12
Resource Devices
  SYNC FLAGs, 4-47
RESTORE^XGF, 25-13
Revision History, iii
  Documentation, iii
  Patches, vi
Rewinding Devices, 4-21
Right Margin, 4-12, 4-15
RMDEV^%ZISUTL, 4-41
ROUTINE File (#9.8), 13-18
Routine Install Options (KIDS), 13-28
Routines
  ^XUP, 14-4
  KIDS, 13-18
  XQ1, 14-4
RT logging, 17-11
RTN^%ZTLOAD, 22-47
RUM, 17-7

## S

S^%ZTLOAD, 22-4
*SAC*, xxxi
SAVDEV^%ZISUTL, 4-42
SAVE^XGF, 25-14
SAY^XGF, 25-15
SAYU^XGF, 25-17
SECURITY KEY File (#19.1), 18-1, 18-2
Security Keys
  $$KCHK^XUSRB, 24-16
  $$LKUP^XPDKEY, 18-3
  $$RENAME^XPDKEY, 18-4
  APIs, 18-2
  DEL^XPDKEY, 18-2
  Key Lookup, 18-1
  OWNSKEY^XUSRB, 18-5
  Person Lookup, 18-1
  Programmer Tools, 18-1
  XUPROG, 13-1
Selecting Templates (KIDS), 13-20
Self-Contained Routine (KIDS), 13-26
Send Alpha/Beta Usage to Developers Option, 13-47
Sending Security Codes (KIDS), 13-8
Servers
  Appending Text to a Server Bulletin or Mailman Reply, 19-1
  Customizing a Server Bulletin, 19-2
  Key Variables, 19-1
  Programmer Tools, 19-1
  Tools for Processing Server Requests, 19-1
SET^XUPARAM, 20-8
SET^XUS1A, 20-10
SETA^XGF, 25-19
SETENV^%ZOSV, 17-9
SETNM^%ZOSV, 17-10
SETSURO1^XQALSURO, 3-36
Setting a File's Package Revision Data Node (Post-Install) (KIDS), 13-33
Setting up Alpha/Beta Tracking in the Build Entry, 13-46
SETUP^XQALERT, 3-21
SETUP^XUSRB, 20-20
SIBLING^XUAF4, 12-21
SIG^XUSESIG, 6-2
SIGN-ON LOG File (#3.081), 20-2
Signon/Security
    $$PROD^XUPROD, 20-9
  $$ADD^XUSERNEW, 20-15
  $$CHECKAV^XUSRB, 20-17
  $$CHECKAV^XUVERIFY, 20-25
  $$CREATE^XUSAP, 20-12
  $$DECRYP^XUSRB1, 20-21
  $$ENCRYP^XUSRB1, 20-22
  $$GET^XUPARAM, 20-6
  $$HANDLE^XUSRB4, 20-23
  $$INHIBIT^XUSRB, 20-18
  $$KSP^XUPARAM, 20-7
  $$LKUP^XUPARAM, 20-8
  ^XUVERIFY, 20-24
  APIs, 20-6
  AVHLPTXT^XUS2, 20-11
  Creating a Package-specific User Termination Action, 20-4
  CVC^XUSRB, 20-18
  Direct Mode Utilities, 20-1
  GETPEER^%ZOSV, 20-26
  H^XUS, 20-10
  INTRO^XUSRB, 20-19
  KILL^XUSCLEAN, 20-14
  LOGOUT^XUSRB, 20-19
  Programmer Tools, 20-1
  SET^XUPARAM, 20-8
  SET^XUS1A, 20-10
  SETUP^XUSRB, 20-20
  VALIDAV^XUSRB, 20-21
  WITNESS^XUVERIFY, 20-26
  XU USER SIGN-ON Option, 20-3
  XU USER TERMINATE Option, 20-4

Signon/security Functions
SIG^XUSESIG, 6-2
Signon/Security Functions
^XUP, 20-1
^XUS, 20-1
^XUSCLEAN, 20-2
^ZU, 20-2
H^XUS, 20-2
Site Option to Report Errors (KIDS), 13-46
Site Parameters, 20-7
Skip Installing or Delete a Routine (KIDS), 13-28
Skipping Installation Questions (KIDS), 13-36
Slave Printers, 4-11
Software-wide Variables, Protecting, 20-14
Soundex
$$EN^XUA4A71, 15-2
Splitting a Distribution Across Diskettes (KIDS), 13-22
SPOOL DATA File (#3.519), 21-2
SPOOL DOCUMENT File (#3.51), 21-2
Spooling
APIs, 21-2
DSD^ZISPL, 21-2
DSDOC^ZISPL, 21-2
Programmer Tools, 21-1
Site Parameters, 20-7
Spool Device, 4-11
Startup PROD check Option, 20-9
STAT^%ZTLOAD, 22-49
STATE File (#5), 2-1, 2-4, 2-6
STDNAME^XLFNAME, 16-18
Stop Requests, Checking for (TaskMan), 22-5
Stopping tasks, 22-48
String Functions
$$CJ^XLFSTR, 26-72
$$INVERT^XLFSTR, 26-73
$$LJ^XLFSTR, 26-73
$$LOW^XLFSTR, 26-74
$$REPEAT^XLFSTR, 26-75
$$REPLACE^XLFSTR, 26-76
$$RJ^XLFSTR, 26-77
$$SENTENCE^XLFSTR, 26-78
$$STRIP^XLFSTR, 26-78
$$TITLE^XLFSTR, 26-79
$$TRIM^XLFSTR, 26-80
$$UP^XLFSTR, 26-81
String Functions (XLF), 26-72
Subtype, 4-12
Symbols
Found in the Documentation, xxvii
SYNC FLAG, 22-9
SYNC FLAGs, 4-47
SYNC FLAGs to Control Sequences of Tasks, 22-8
Systems Manager Menu, 13-1

## T

T0^%ZOSV, 17-11
T1^%ZOSV, 17-12
TASK SYNC FLAG File (#14.8), 22-9, 22-10
TaskMan
$$ASKSTOP^%ZTLOAD, 22-34
$$DEV^XUTMDEVQ, 22-11
$$JOB^%ZTLOAD, 22-38
$$NODEV^XUTMDEVQ, 22-15
$$PSET^%ZTLOAD, 22-41
$$QQ^XUTMDEVQ, 22-17
$$REQQ^XUTMDEVQ, 22-20
$$S^%ZTLOAD, 22-48
$$TM^%ZTLOAD, 22-51
^%ZTLOAD, 4-47, 22-26
APIs, 22-11
DESC^%ZTLOAD, 22-35
DISP^XUTMOPT, 22-22
DQ^%ZTLOAD, 22-35
EDIT^XUTMOPT, 22-22
EN^XUTMDEVQ, 22-13
EN^XUTMTP, 22-25
How to Write Code to Queue Tasks, 22-1
ISQED^%ZTLOAD, 22-36
KILL^%ZTLOAD, 22-39
OPTION^%ZTLOAD, 22-40
OPTSTAT^XUTMOPT, 22-23
PCLEAR^%ZTLOAD, 22-40
Programmer Tools, 22-1
Queuers, 22-1
REQ^%ZTLOAD, 22-42
RESCH^XUTMOPT, 22-24
RTN^%ZTLOAD, 22-47
SYNC FLAGs, 4-47
Task Status, 22-49
Tasks, 22-3
ZTSAVE^%ZTLOAD, 22-51
TaskMan (DCL context), 22-28
Tasks
^%ZIS Call Within a Task, 22-8
^%ZTLOAD call within a task, 22-7
Destination, 22-4
Device, 22-4
DT Variable, 22-3

DUZ Array, 22-3
Error Trap, 22-4
IO* Array, 22-3
Post-execution commands, 22-6
Priority, 22-4
Purging the Task Record, 22-5
Queuing with no I/O device, 22-31
S^%ZTLOAD, 22-5
Saved Variables, 22-4
Stop Requests, 22-5
SYNC FLAGs, 22-8
TaskMan, 22-3
Tools, 22-4
Two-step tasks, 22-8
ZTDESC Variable, 22-4
ZTDTH Variable, 22-4
ZTIO Variable, 22-4
ZTQUEUED variable, 22-6
ZTQUEUED Variable, 22-4
ZTREQ, 22-6
ZTREQ Variable, 22-6
ZTRTN Variable, 22-4
ZTSK Variable, 22-4
ZTSTAT Variable, 22-9
ZTSTOP Variable, 22-5
TASKS File (#14.4), 22-4, 22-5, 22-6, 22-35, 22-36
Templates (KIDS), 13-20
Terminal Server, 4-7, 4-11
TERMINAL TYPE File (#3.2), 4-8, 4-12, 4-13, 4-24, 4-25, 4-27
Termination Action, Creating, 20-4
TEXT^MXMLDOM, 27-11
TIME ZONE field (#1), 26-26
TIMED READ (# OF SECONDS) Field (#200.1), 24-4
TIMED READ (# OF SECONDS) Field (#51.1), 24-4
TITLE^XPDID, 13-50
Toolkit Queuable Options menu
Errors Logged in Alpha/Beta Test (QUEUED) option, 13-46
Tools for Processing Server Requests, 19-1
Track Package Nationally (KIDS), 13-45
Transporting a Distribution (KIDS), 13-20
Turning Off Alpha/Beta Tracking, 13-47
Types
Options, 14-1

## U

UNWIND^%ZTER, 7-5
Unwinder
APIs, 23-1
DISP^XQORM1, 23-5
EN^XQOR, 23-1
EN^XQORM, 23-4
EN1^XQOR, 23-2
MSG^XQOR, 23-3
Programmer Tools, 23-1
XREF^XQORM, 23-5
UPCP^XPDUTL, 13-39
Update the Status Bar During Pre- and Post-Install Routines (KIDS), 13-34
UPDATE^XPDID, 13-51
UPDCOMP^XLFNAME2, 16-24
URLs
Adobe Acrobat Quick Guide Web Address, xxxi
Adobe Home Page Web Address, xxxi
Health Systems Design and Development Home Page Web Address, xxx
Kernel Home Page Web Address, xxx
Use of
DIDEL in ^DIE Calls, 9-3
DLAYGO in ^DIC Calls, 9-2
DLAYGO When Navigating to Files, 9-2
USE PARAMETERS, 4-8
USE PARAMETERS Field, 4-12
Use this Manual, How to, xxvii
USE^%ZISUTL, 4-43
User
$$ACTIVE^XUSER, 24-6
$$CODE2TXT^XUA4A72, 24-1
$$DTIME^XUP, 24-4
$$GET^XUA4A72, 24-2
$$IEN2CODE^XUA4A72, 24-3
$$LOOKUP^XUSER, 24-12
$$NAM^XUSER, 24-14
$$PROVIDER^XUSER, 24-15
APIs, 24-1
DIV4^XUSER, 24-11
DIVGET^XUSRB2, 24-17
DIVSET^XUSRB2, 24-18
Programmer Tools, 24-1
USERINFO^XUSRB2, 24-19
USER CLASS Field (#9.5), 20-12
USER CLASS File (#201), 20-12
USER TERMINATE ROUTINE Field, 20-4

USER TERMINATE ROUTINE Option (Obsolete), 20-4
USER TERMINATE TAG Field, 20-4
User Termination Action, Creating, 20-4
USER^XQALERT, 3-30
USERDATA^XQALBUTL, 3-13
USERINFO^XUSRB2, 24-19
USERLIST^XQALBUTL, 3-14
Using Checkpoints (Pre- and Post-Install Routines), 13-38
Using SYNC FLAGs to Control Sequences of Tasks (TaskMan), 22-8
Utility Functions
  $$BASE^XLFUTL, 26-82
  $$CCD^XLFUTL, 26-83
  $$CNV^XLFUTL, 26-84
  $$DEC^XLFUTL, 26-85
  $$VCD^XLFUTL, 26-85
Utility Functions (XLF), 26-82

## V

VALIDAV^XUSRB, 20-21
Variables
  DIFROM, 13-27
  DIFROM (KIDS), 13-33
  KIDS, 13-27, 13-33
  Programmer Use in Menu Manager, 14-2
  Servers, 19-1
  Tasks, 22-3
  XPDENV, 13-27
  XPDNM, 13-27
  XPDNM (KIDS), 13-33
  XQMM("A") (Menu Manager), 14-2
  XQMM("B") (Menu Manager), 14-3
  XQMM("J") (Menu Manager), 14-3
  XQMM("N") (Menu Manager), 14-3
  XQUIT (Menu Manager), 14-2
  ZTQUEUED (KIDS), 13-33
Verifying Patch Installation (KIDS), 13-28
Version Numbers (KIDS), 13-27
VistA Documentation Library (VDL)
  Home Page Web Address, xxxi
VOLUME SET File (#14.5), 22-37

## W

Web Pages
  Adobe Acrobat Quick Guide Web Address, xxxi
  Adobe Home Page Web Address, xxxi
  Health Systems Design and Development Home Page Web Address, xxx
  ISS Acronyms Home Page Web Address, Glossary, 6
  ISS Glossary Home Page Web Address, Glossary, 6
  Kernel Home Page Web Address, xxx
  VistA Documentation Library (VDL) Home Page Web Address, xxxi
When to Transport More than One Transport Global in a Distribution (KIDS), 13-23
Where Questions Are Asked During Installations (KIDS), 13-38
WIN^XGF, 25-21
WITNESS^XUVERIFY, 20-26
Workday Calculation, 15-3
Writing Two-step Tasks (TaskMan), 22-8

## X

XGF Direct Mode Utilities, 25-1
XGF Function Library
  $$READ^XGF, 25-10
  ^XGFDEMO, 25-1
  APIs, 25-2
  CHGA^XGF, 25-2
  CLEAN^XGF, 25-4
  CLEAR^XGF, 25-5
  Demo Program, 25-1
  FRAME^XGF, 25-6
  INITKB^XGF, 25-7
  IOXY^XGF, 25-8
  PREP^XGF, 25-9
  Programmer Tools, 25-1
  RESETKB^XGF, 25-12
  RESTORE^XGF, 25-13
  SAVE^XGF, 25-14
  SAY^XGF, 25-15
  SAYU^XGF, 25-17
  SETA^XGF, 25-19
  WIN^XGF, 25-21
XGFDEMO
  ^XGFDEMO, 25-1
XIPUTIL
  $$FIPS^XIPUTIL, 2-2
  $$FIPSCHK^XIPUTIL, 2-3
  CCODE^XIPUTIL, 2-1
  POSTAL^XIPUTIL, 2-4
  POSTALB^XIPUTIL, 2-6
XLF Function Library
  $$%H^XLFDT, 26-4

$$ABS^XLFMTH, 26-37
$$ACOS^XLFMTH, 26-38
$$ACOSDEG^XLFMTH, 26-38
$$ACOSH^XLFHYPER, 26-28
$$ACOT^XLFMTH, 26-39
$$ACOTDEG^XLFMTH, 26-40
$$ACOTH^XLFHYPER, 26-29
$$ACSC^XLFMTH, 26-40
$$ACSCDEG^XLFMTH, 26-41
$$ACSCH^XLFHYPER, 26-29
$$ASEC^XLFMTH, 26-42
$$ASECDEG^XLFMTH, 26-42
$$ASECH^XLFHYPER, 26-30
$$ASIN^XLFMTH, 26-43
$$ASINDEG^XLFMTH, 26-44
$$ASINH^XLFHYPER, 26-31
$$ATAN^XLFMTH, 26-44
$$ATANDEG^XLFMTH, 26-45
$$ATANH^XLFHYPER, 26-31
$$BASE^XLFUTL, 26-82
$$BSA^XLFMSMT, 26-65
$$CCD^XLFUTL, 26-83
$$CJ^XLFSTR, 26-72
$$CNV^XLFUTL, 26-84
$$COS^XLFMTH, 26-46
$$COSDEG^XLFMTH, 26-46
$$COSH^XLFHYPER, 26-32
$$COT^XLFMTH, 26-47
$$COTDEG^XLFMTH, 26-48
$$COTH^XLFHYPER, 26-33
$$CSC^XLFMTH, 26-48
$$CSCDEG^XLFMTH, 26-49
$$CSCH^XLFHYPER, 26-33
$$DEC^XLFUTL, 26-85
$$DECDMS^XLFMTH, 26-50
$$DMSDEC^XLFMTH, 26-51
$$DOW^XLFDT, 26-5
$$DT^XLFDT, 26-6
$$DTR^XLFMTH, 26-51
$$E^XLFMTH, 26-52
$$EXP^XLFMTH, 26-53
$$FMADD^XLFDT, 26-6
$$FMDIFF^XLFDT, 26-7
$$FMTE^XLFDT, 26-9
$$FMTH^XLFDT, 26-13
$$FMTHL7^XLFDT, 26-14
$$HADD^XLFDT, 26-15
$$HDIFF^XLFDT, 26-16
$$HL7TFM^XLFDT, 26-17
$$HTE^XLFDT, 26-19
$$HTFM^XLFDT, 26-21
$$INVERT^XLFSTR, 26-73
$$LENGTH^XLFMSMT, 26-66
$$LJ^XLFSTR, 26-73
$$LN^XLFMTH, 26-53
$$LOG^XLFMTH, 26-54
$$LOW^XLFSTR, 26-74
$$MAX^XLFMTH, 26-55
$$MIN^XLFMTH, 26-56
$$NOW^XLFDT, 26-22
$$PI^XLFMTH, 26-56
$$PWR^XLFMTH, 26-57
$$REPEAT^XLFSTR, 26-75
$$REPLACE^XLFSTR, 26-76
$$RJ^XLFSTR, 26-77
$$RTD^XLFMTH, 26-58
$$SCH^XLFDT, 26-23
$$SEC^XLFDT, 26-25
$$SEC^XLFMTH, 26-60
$$SECDEG^XLFMTH, 26-60
$$SECH^XLFHYPER, 26-34
$$SENTENCE^XLFSTR, 26-78
$$SIN^XLFMTH, 26-61
$$SINDEG^XLFMTH, 26-62
$$SINH^XLFHYPER, 26-35
$$SQRT^XLFMTH, 26-62
$$STRIP^XLFSTR, 26-78
$$TAN^XLFMTH, 26-63
$$TANDEG^XLFMTH, 26-64
$$TANH^XLFHYPER, 26-36
$$TEMP^XLFMSMT, 26-67
$$TITLE^XLFSTR, 26-79
$$TRIM^XLFSTR, 26-80
$$TZ^XLFDT, 26-26
$$UP^XLFSTR, 26-81
$$VCD^XLFUTL, 26-85
$$VOLUME^XLFMSMT, 26-68
$$WEIGHT^XLFMSMT, 26-70
$$WITHIN^XLFDT, 26-27
APIs, 26-1
Date Functions, 26-4
Hyperbolic Trigonometric Functions, 26-28
Math Functions, 26-37
Measurement Functions, 26-65
Programmer Tools, 26-1
String Functions, 26-72
Utility Functions, 26-82
XLFDT
$$%H^XLFDT, 26-4
$$DOW^XLFDT, 26-5
$$DT^XLFDT, 26-6
$$FMADD^XLFDT, 26-6

$$FMDIFF^XLFDT, 26-7
$$FMTE^XLFDT, 26-9
$$FMTH^XLFDT, 26-13
$$FMTHL7^XLFDT, 26-14
$$HADD^XLFDT, 26-15
$$HDIFF^XLFDT, 26-16
$$HL7TFM^XLFDT, 26-17
$$HTE^XLFDT, 26-19
$$HTFM^XLFDT, 26-21
$$NOW^XLFDT, 26-22
$$SCH^XLFDT, 26-23
$$SEC^XLFDT, 26-25
$$TZ^XLFDT, 26-26
$$WITHIN^XLFDT, 26-27
Date Functions), 26-4
XLFHYPER
$$ACOSH^XLFHYPER, 26-28
$$ACOTH^XLFHYPER, 26-29
$$ACSCH^XLFHYPER, 26-29
$$ASECH^XLFHYPER, 26-30
$$ASINH^XLFHYPER, 26-31
$$ATANH^XLFHYPER, 26-31
$$COSH^XLFHYPER, 26-32
$$COTH^XLFHYPER, 26-33
$$CSCH^XLFHYPER, 26-33
$$SECH^XLFHYPER, 26-34
$$SINH^XLFHYPER, 26-35
$$TANH^XLFHYPER, 26-36
Hyperbolic Trigonometric Functions), 26-28
XLFMSMT
$$BSA^XLFMSMT, 26-65
$$LENGTH^XLFMSMT, 26-66
$$TEMP^XLFMSMT, 26-67
$$VOLUME^XLFMSMT, 26-68
$$WEIGHT^XLFMSMT, 26-70
Measurement Functions), 26-65
XLFMTH
$$ABS^XLFMTH, 26-37
$$ACOS^XLFMTH, 26-38
$$ACOSDEG^XLFMTH, 26-38
$$ACOT^XLFMTH, 26-39
$$ACOTDEG^XLFMTH, 26-40
$$ACSC^XLFMTH, 26-40
$$ACSCDEG^XLFMTH, 26-41
$$ASEC^XLFMTH, 26-42
$$ASECDEG^XLFMTH, 26-42
$$ASIN^XLFMTH, 26-43
$$ASINDEG^XLFMTH, 26-44
$$ATAN^XLFMTH, 26-44
$$ATANDEG^XLFMTH, 26-45
$$COS^XLFMTH, 26-46
$$COSDEG^XLFMTH, 26-46
$$COT^XLFMTH, 26-47
$$COTDEG^XLFMTH, 26-48
$$CSC^XLFMTH, 26-48
$$CSCDEG^XLFMTH, 26-49
$$DECDMS^XLFMTH, 26-50
$$DMSDEC^XLFMTH, 26-51
$$DTR^XLFMTH, 26-51
$$E^XLFMTH, 26-52
$$EXP^XLFMTH, 26-53
$$LN^XLFMTH, 26-53
$$LOG^XLFMTH, 26-54
$$MAX^XLFMTH, 26-55
$$MIN^XLFMTH, 26-56
$$PI^XLFMTH, 26-56
$$PWR^XLFMTH, 26-57
$$RTD^XLFMTH, 26-58
$$SD^XLFMTH, 26-59
$$SEC^XLFMTH, 26-60
$$SECDEG^XLFMTH, 26-60
$$SIN^XLFMTH, 26-61
$$SINDEG^XLFMTH, 26-62
$$SQRT^XLFMTH, 26-62
$$TAN^XLFMTH, 26-63
$$TANDEG^XLFMTH, 26-64
Math Functions), 26-37
XLFNAME
$$BLDNAME^XLFNAME, 16-1
$$CLEANC^XLFNAME, 16-4
$$FMNAME^XLFNAME, 16-6
$$HLNAME^XLFNAME, 16-9
$$NAMEFMT^XLFNAME, 16-13
NAMECOMP^XLFNAME, 16-12
STDNAME^XLFNAME, 16-18
XLFNAME2
DELCOMP^XLFNAME2, 16-23
UPDCOMP^XLFNAME2, 16-24
XLFNSLK
$$ADDRESS^XLFNSLK, 5-1
MAIL^XLFNSLK, 5-2
XLFSTR
$$CJ^XLFSTR, 26-72
$$INVERT^XLFSTR, 26-73
$$LJ^XLFSTR, 26-73
$$LOW^XLFSTR, 26-74
$$REPEAT^XLFSTR, 26-75
$$REPLACE^XLFSTR, 26-76
$$RJ^XLFSTR, 26-77
$$SENTENCE^XLFSTR, 26-78
$$STRIP^XLFSTR, 26-78
$$TITLE^XLFSTR, 26-79

$$TRIM^XLFSTR, 26-80
$$UP^XLFSTR, 26-81
String Functions), 26-72
XLFUTL
$$BASE^XLFUTL, 26-82
$$CCD^XLFUTL, 26-83
$$CNV^XLFUTL, 26-84
$$DEC^XLFUTL, 26-85
$$VCD^XLFUTL, 26-85
Utility Functions), 26-82
XML
$$ATTRIB^MXMLDOM, 27-1
$$CHILD^MXMLDOM, 27-2
$$CMNT^MXMLDOM, 27-3
$$EN^MXMLDOM, 27-6
$$NAME^MXMLDOM, 27-7
$$PARENT^MXMLDOM, 27-8
$$SIBLING^MXMLDOM, 27-9
$$SYMENC^MXMLUTL, 27-17
$$TEXT^MXMLDOM, 27-10
$$VALUE^MXMLDOM, 27-12
$$XMLHDR^MXMLUTL, 27-18
APIs, 27-1
CMNT^MXMLDOM, 27-4
DELETE^MXMLDOM, 27-5
EN^MXMLPRSE, 27-13
Programmer Tools, 27-1
TEXT^MXMLDOM, 27-11
XPD BUILD NAMESPACE Option, 13-3
XPD COPY BUILD Option, 13-4
XPD MAIN Menu, 13-1
XPDENV Variable, 13-27
XPDID
EXIT^XPDID, 13-49
INIT^XPDID, 13-50
TITLE^XPDID, 13-50
UPDATE^XPDID, 13-51
XPDIJ
EN^XPDIJ, 13-52
XPDIP
$$PKGPAT^XPDIP, 13-53
XPDKEY
$$LKUP^XPDKEY, 18-3
$$RENAME^XPDKEY, 18-4
DEL^XPDKEY, 18-2
XPDMENU
$$ADD^XPDMENU, 14-5
DELETE^XPDMENU, 14-6
LKOPT^XPDMENU, 14-7
OUT^XPDMENU, 14-8
RENAME^XPDMENU, 14-9
XPDNM Variable, 13-27, 13-33
XPDUTL
$$COMCP^XPDUTL, 13-55
$$CURCP^XPDUTL, 13-56
$$LAST^XPDUTL, 13-57
$$NEWCP^XPDUTL, 13-59
$$OPTDE^XPDUTL, 13-60
$$PARCP^XPDUTL, 13-61
$$PATCH^XPDUTL, 13-62
$$PKG^XPDUTL, 13-62
$$PRODE^XPDUTL, 13-63
$$RTNUP^XPDUTL, 13-64
$$UPCP^XPDUTL, 13-65
$$VER^XPDUTL, 13-66
$$VERCP^XPDUTL, 13-67
$$VERSION^XPDUTL, 13-68
BMES^XPDUTL, 13-54
MES^XPDUTL, 13-58
XQ1 Routine, 14-4
XQ92
NEXT^XQ92, 14-9
XQAB ERROR LOG SERVER Option, 13-47
XQAB ERROR LOG XMIT Option, 13-47
XQAB ERRORS LOGGED File (#8991.5), 13-47
XQALBUTL
$$PENDING^XQALBUTL, 3-9
$$PKGPEND^XQALBUTL, 3-10
AHISTORY^XQALBUTL, 3-4
ALERTDAT^XQALBUTL, 3-6
DELSTAT^XQALBUTL, 3-7
NOTIPURG^XQALBUTL, 3-8
PTPURG^XQALBUTL, 3-11
RECIPURG^XQALBUTL, 3-12
USERDATA^XQALBUTL, 3-13
USERLIST^XQALBUTL, 3-14
XQALERT
$$SETUP1^XQALERT, 3-25
ACTION^XQALERT, 3-15
DELETE^XQALERT, 3-16
DELETEA^XQALERT, 3-17
GETACT^XQALERT, 3-18
PATIENT^XQALERT, 3-19
SETUP^XQALERT, 3-21
USER^XQALERT, 3-30
XQALFWD
FORWARD^XQALFWD, 3-31
XQALSURO
$$CURRSURO^XQALSURO, 3-33
$$GETSURO^XQALSURO, 3-34
REMVSURO^XQALSURO, 3-35

SETSURO1^XQALSURO, 3-36
XQCHK
$$ACCESS^XQCHK, 14-10
OP^XQCHK, 14-12
XQDATE
^XQDATE, 15-1
XQH
EN^XQH, 10-2
EN1^XQH, 10-3
XQH4
ACTION^XQH4, 10-4
XQOR
EN^XQOR, 23-1
EN1^XQOR, 23-2
MSG^XQOR, 23-3
XQORM
EN^XQORM, 23-4
XREF^XQORM, 23-5
XQORM1
DISP^XQORM1, 23-5
XQUIT Variable, 14-2
XREF^XQORM, 23-5
XTMP Global, 13-21, 13-28, 13-29, 17-5, 17-7, 20-23, 22-8
XU SID ASK Option, 20-9
XU SID STARTUP Option, 20-9
XU USER SIGN-ON Extended Action, 20-10
XU USER SIGN-ON Option, 20-3
Package-specific Signon Actions, 20-3
XU USER TERMINATE Option, 20-4
XUA4A71
$$EN^XUA4A71, 15-2
XUA4A72
$$CODE2TXT^XUA4A72, 24-1
$$GET^XUA4A72, 24-2
$$IEN2CODE^XUA4A72, 24-3
XUAF4
$$ACTIVE^XUAF4, 12-1
$$CIRN^XUAF4, 12-4
$$ID^XUAF4, 12-6
$$IDX^XUAF4, 12-7
$$IEN^XUAF4, 12-8
$$LEGACY^XUAF4, 12-9
$$LKUP^XUAF4, 12-10
$$MADD^XUAF4, 12-12
$$NAME^XUAF4, 12-13
$$NNT^XUAF4, 12-14
$$NS^XUAF4, 12-14
$$O99^XUAF4, 12-15
$$PADD^XUAF4, 12-16
$$PRNT^XUAF4, 12-18
$$RF^XUAF4, 12-19
$$RT^XUAF4, 12-20
$$STA^XUAF4, 12-22
$$TF^XUAF4, 12-22
$$WHAT^XUAF4, 12-23
CDSYS^XUAF4, 12-2
CHILDREN^XUAF4, 12-2
F4^XUAF4, 12-4
LOOKUP^XUAF4, 12-11
PARENT^XUAF4, 12-17
SIBLING^XUAF4, 12-21
XUDHGUI
DEVICE^XUDHGUI, 4-2
XUDHSET
$$RES^XUDHSET, 4-5
XUEDITOPT Option, 14-1
XUHUI
OPKG^XUHUI, 8-1
XUKERNEL Menu, 20-9
XUMF
$$IEN^XUMF, 12-24
XUMFI
MAIN^XUMFI, 12-25
XUMFP
MAIN^XUMFP, 12-27
XUP
$$DTIME^XUP, 24-4
^XUP Direct Mode Utility, 20-1
XUP Routine, 14-4
XUPARAM
$$GET^XUPARAM, 20-6
$$KSP^XUPARAM, 20-7
$$LKUP^XUPARAM, 20-8
SET^XUPARAM, 20-8
XUPROD
$$PROD^XUPROD, 20-9
XUPROG Menu, 13-1
XUPROG Security Key, 13-1
XUS
^XUS Direct Mode Utility, 20-1
H^XUS, 20-10
H^XUS Direct Mode Utility, 20-2
XUS1A
SET^XUS1A, 20-10
XUS2
AVHLPTXT^XUS2, 20-11
XUSCLEAN
$$CREATE^XUSAP, 20-12
^XUSCLEAN Direct Mode Utility, 20-2
KILL^XUSCLEAN, 20-14
XUSEC Global, 18-2, 18-4

XUSER
$$ACTIVE^XUSER, 24-6
$$DEA^XUSER, 24-8
$$LOOKUP^XUSER, 24-12
$$NAM^XUSER, 24-14
$$PROVIDER^XUSER, 24-15
DIV4^XUSER, 24-11
XUSERNEW
$$ADD^XUSERNEW, 20-15
XUSESIG
^XUSESIG, 6-1
SIG^XUSESIG, 6-2
XUSESIG1
$$CHKSUM^XUSESIG1, 6-2
$$CMP^XUSESIG1, 6-3
$$DE^XUSESIG1, 6-4
$$EN^XUSESIG1, 6-4
$$ESBLOCK^XUSESIG1, 6-5
XUSHSHP
DE^XUSHSHP, 6-6
EN^XUSHSHP, 6-7
HASH^XUSHSHP, 6-8
XUSPF200 Key, 20-15
XUSRB
$$CHECKAV^XUSRB, 20-17
$$INHIBIT^XUSRB, 20-18
$$KCHK^XUSRB, 24-16
CVC^XUSRB, 20-18
INTRO^XUSRB, 20-19
LOGOUT^XUSRB, 20-19
OWNSKEY^XUSRB, 18-5
SETUP^XUSRB, 20-20
VALIDAV^XUSRB, 20-21
XUSRB1
$$DECRYP^XUSRB1, 20-21
$$ENCRYP^XUSRB1, 20-22
XUSRB2
DIVGET^XUSRB2, 24-17
DIVSET^XUSRB2, 24-18
USERINFO^XUSRB2, 24-19
XUSRB4
$$HANDLE^XUSRB4, 20-23
XUTL Global, 23-5
XUTMDEVQ
$$DEV^XUTMDEVQ, 22-11
$$NODEV^XUTMDEVQ, 22-15
$$QQ^XUTMDEVQ, 22-17
$$REQQ^XUTMDEVQ, 22-20
EN^XUTMDEVQ, 22-13
XUTMOPT
DISP^XUTMOPT, 22-22
EDIT^XUTMOPT, 22-22
OPTSTAT^XUTMOPT, 22-23
RESCH^XUTMOPT, 22-24
XUTMTP
EN^XUTMTP, 22-25
XUVERIFY
$$CHECKAV^XUVERIFY, 20-25
^XUVERIFY, 20-24
WITNESS^XUVERIFY, 20-26
XUWORKDY
^XUWORKDY, 15-3

## Z

ZIS
$$REWIND^%ZIS, 4-21
^%ZIS, 4-6
HLP1^%ZIS, 4-18
HLP2^%ZIS, 4-18
HOME^%ZIS, 4-19
ZISC
^%ZISC, 4-22
ZISH
$$DEFDIR^%ZISH, 11-3
$$DEL^%ZISH, 11-4
$$FTG^%ZISH, 11-5
$$GATF^%ZISH, 11-7
$$GTF^%ZISH, 11-8
$$LIST^%ZISH, 11-9
$$MV^%ZISH, 11-10
$$PWD^%ZISH, 11-12
$$STATUS^%ZISH, 11-13
CLOSE^%ZISH, 11-2
OPEN^%ZISH, 11-11
ZISP
PKILL^%ZISP, 4-23
PSET^%ZISP, 4-24
ZISPL
DSD^ZISPL, 21-2
DSDOC^ZISPL, 21-2
ZISS
ENDR^%ZISS, 4-25
ENS^%ZISS, 4-26
GKILL^%ZISS, 4-33
GSET^%ZISS, 4-34
KILL^%ZISS, 4-35
ZISTCP
CALL^%ZISTCP, 4-36
CLOSE^%ZISTCP, 4-37
ZISUTL
CLOSE^%ZISUTL, 4-38

OPEN^%ZISUTL, 4-39
RMDEV^%ZISUTL, 4-41
SAVDEV^%ZISUTL, 4-42
USE^%ZISUTL, 4-43
ZOSF Global, 17-1
ZOSF Nodes
ACTJ, 17-1
AVJ, 17-1
BRK, 17-1
DEL, 17-1
EOFF, 17-1
EON, 17-1
EOT, 17-2
ERRTN, 17-2
ETRP, 17-2
GD, 17-2
JOBPARAM, 17-2
LABOFF, 17-2
LOAD, 17-2
LPC, 17-2
MAGTAPE, 17-2
MAXSIZ, 17-2
MGR, 17-2
MTBOT, 17-2
MTERR, 17-2
MTONLINE, 17-2
MTWPROT, 17-2
NBRK, 17-2
NO-PASSALL, 17-2
NO-TYPE-AHEAD, 17-2
OS, 17-3
PASSALL, 17-3
PRIINQ, 17-3
PRIORITY, 17-3
PROD, 17-3
PROGMODE, 17-3
RD, 17-3
RESJOB, 17-3
RM, 17-3
RSEL, 17-3
RSUM, 17-3
SAVE, 17-3
SIZE, 17-3
SS, 17-3
TEST, 17-3
TMK, 17-3
TRAP, 17-3
TRMOFF, 17-3
TRMON, 17-3
TRMRD, 17-3
TYPE-AHEAD, 17-3
UCI, 17-3
UCICHECK, 17-3
UPPERCASE, 17-3
VOL, 17-3
XY, 17-3
ZD, 17-3
ZOSV
$$ACTJ^%ZOSV, 17-4
$$AVJ^%ZOSV, 17-4
$$EC^%ZOSV, 7-1
$$LGR^%ZOSV, 17-6
$$OS^%ZOSV, 17-8
$$VERSION^%ZOSV, 17-13
DOLRO^%ZOSV, 17-5
GETENV^%ZOSV, 17-6
GETPEER^%ZOSV, 20-26
LOGRSRC^%ZOSV, 17-7
SETENV^%ZOSV, 17-9
SETNM^%ZOSV, 17-10
T0^%ZOSV, 17-11
T1^%ZOSV, 17-12
ZRTL Global, 17-12
ZSTOP, 22-5
ZTER
$$NEWERR^%ZTER, 7-4
^%ZTER, 7-2
UNWIND^%ZTER, 7-5
ZTLOAD, 4-47
$$ASKSTOP^%ZTLOAD, 22-34
$$JOB^%ZTLOAD, 22-38
$$S^%ZTLOAD, 22-48
$$TM^%ZTLOAD, 22-51
^%ZTLOAD, 22-26
DESC^%ZTLOAD, 22-35
DQ^%ZTLOAD, 22-35
ISQED^%ZTLOAD, 22-36
KILL^%ZTLOAD, 22-39
OPTION^%ZTLOAD, 22-40
PCLEAR^%ZTLOAD, 22-40
REQ^%ZTLOAD, 22-42
RTN^%ZTLOAD, 22-47
STAT^%ZTLOAD, 22-49
ZTSAVE^%ZTLOAD, 22-51
ZTQUEUED variable, 22-6
ZTQUEUED Variable, 13-33, 22-4
ZTREQ, 22-6
ZTREQ variable, 22-4
ZTREQ Variable, 22-6
ZTSAVE^%ZTLOAD, 22-51
ZTSTAT Variable, 22-9
ZTSTOP Variable, 22-4, 22-5

ZU ^ZU Direct Mode Utility, 20-2