Kernel 8.0 & Kernel Toolkit 7.3 Developer's Guide

May 2021

Department of Veterans Affairs (VA)

Office of Information and Technology (OIT)

Enterprise Program Management Office (EPMO)

Revision History

Date	Revision	Description	Author
05/18/2021	11.17	Updates: • Updated the DEL^XPDKEY(): Delete Security Key API. • Section 15.3.1.1: Changed to say "use ^DIR API for **READ**s." • Section 15.3.1.3; Table 15: Added the **XPDGREF** variable.	VistA Kernel/VistA Infrastructure (VI) Development Team
01/28/2021	11.16	Updated the following APIs for Kernel Patch XU*8.0*739; Added the optional **date** input parameter to each: • $$DEA^XUSER()—Get User's DEA Number. Also, added an Example 5 (missing) and 6 (new). • $$SDEA^XUSER()—Check for Prescribing Privileges. • $$DETOX^XUSER()—Get Detox/Maintenance ID Number. • Updated the $$PROVIDER^XUSER(): Providers in New Person File: Added the missing **xuf** input parameter **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	VistA Infrastructure (VI) Development Team
04/21/2020	11.15	Updates: • Patch XU*8.0*723 functionality was backed-out via Kernel Patch XU*8.0*729, because the functionality was *not* needed. • Deleted the $$CERNER^XUAF4(): Check if Site has been Converted to Cerner. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	VistA Infrastructure (VI) Development Team
03/12/2020	11.14	Added new APIs: • Patch XU*8.0*723: $$CERNER^XUAF4(): Check if	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		Site has been Converted to Cerner. • DUZ^XUP(): Set the DUZ Variable. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	
11/04/2019	11.13	Tech Edits for Kernel Patches XU*8.0*607 and 608: Kernel Lock Manager Utility. Added the following: • Section 17, "Lock Manager: Developer Tools." • APIs: ○ CLEANUP^XULMU(): Execute the Housecleaning Stack ○ SETCLEAN^XULMU(): Register a Cleanup Routine ○ UNCLEAN^XULMU(): Remove Entries from the Housecleaning Stack ○ ADDPAT^XULMU(): Add Patient Identifiers for a Computable File Reference ○ PAT^XULMU(): Get a Standard Set of Patient Identifiers **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	VistA Infrastructure (VI) Development Team
10/09/2018	11.12	Tech Edits for Kernel Patch XU*8.0*672. Added the following APIs: • LOCK^XPDMENU(): Set LOCK Field in OPTION File • RLOCK^XPDMENU(): Set REVERSE/NEGATIVE Field in OPTION File **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	VistA Infrastructure (VI) Development Team
04/09/2018	11.11	Tech Edits:	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		• Updated the $$CHKDGT^XUSNPI(): Validate NPI Format API. • Updated the $$NPI^XUSNPI(): Get NPI from Files #200, #4, or #355.93 API. • Updated the $$QI^XUSNPI(): Get Provider Entities API. • Added the $$NPIUSED^XUSNPI1(): Returns an Error or Warning if an NPI is in Use API. • Updated ^XUWORKDY: Workday Calculation (Obsolete) API. • Updated $$EN^XUWORKDY: Number of Workdays Calculation API. • Updated $$WORKDAY^XUWORKDY: Workday Validation API. • Updated $$WORKPLUS^XUWORKDY: Workday Offset Calculation API. • Added the $$PKGVER^XPDIP(): Update API based on addition to existing Integration Control Registration #2067. • Updated the $$PKGPAT^XPDIP(): Update Patch History API. • Updated the $$CCD^XLFUTL(): Append Check Digit API to include note regarding algorithms. • Made format and content updates throughout this document related to synchronizing the online HTML and Word document APIs. • Updated Section 27.8.3 to add text regarding use of APIs to programmatically update parameter file entries. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	

Date	Revision	Description	Author
12/08/2017	11.10	Tech Edits: • Changed the following sections based on updates with Kernel Patch XU*8.0*657: ○ Updated Section 5.1, "Overview." ○ Added Section 5.2.1, "$$FILE^XLFSHAN(): Returns SHA Hash for Specified FileMan File or Subfile Entry." ○ Added Section 5.2.2, "$$GLOBAL^XLFSHAN(): Returns SHA Hash for a Global." ○ Added Section 5.2.3, "$$HOSTFILE^XLFSHAN(): Returns SHA Hash for Specified Host File." ○ Added Section 5.2.4, "$$LSHAN^XLFSHAN(): Returns SHA Hash for a Long Message." ○ Added Section 5.2.5, "$$ROUTINE^XLFSHAN(): Returns SHA Hash for a VistA Routine." ○ Added Section 5.2.6, "$$SHAN^XLFSHAN(): Returns SHA Hash for a Message." ○ Added Section 21.3.1, "$$CPUTIME^XLFSHAN: Return System and User CPU Time." ○ Added Section 21.3.2, "$$ETIMEMS^XLFSHAN(): Return Elapsed Time in Milliseconds." ○ Updated Section 31.1, "Overview." ○ Added Section 31.3, "Bitwise Logic Functions—XLFSHAN." ○ Added Section 31.3.1, "$$AND^XLFSHAN(): Bitwise Logical AND."	VistA Infrastructure (VI) Development Team VA FileMan 23 Development Team

Date	Revision	Description	Author
		○ Added Section 31.3.2, "$$OR^XLFSHAN(): Bitwise Logical OR." ○ Added Section 31.3.3, "$$XOR^XLFSHAN(): Bitwise Logical XOR." • Changed reference in Section 13.1.2, "$$DEFDIR^%ZISH(): Get Default Host File Directory API," from "NT" to "Windows" in the **df** input parameter. • Updated Section 27.12.3.1.7. • Updated Section 27.12.3.1.9. • Updated Section 27.12.3.1.11. • Added new Section 31.12, "JSON Conversion Functions—XLFJSON," based on updates received for Kernel Patch XU*8.0*680. Kernel Patch XU*8.0*680 was created as part of the VA FileMan 23 Project. • Corrected the format for the $$%H^XLFDT(): Convert Seconds to $H API. • Changed all references from "OI&T" back to "OIT" throughout (again). • Made style and format updates throughout. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	
11/30/2016	11.9	Tech Edits: • Updated the "Orientation" section. • Updated the format in Section 32.2.4; removed the "$$." **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	VistA Infrastructure (VI) Development Team
08/10/2016	11.8	Tech Edits: • Updated VA Directive reference in the "Software Disclaimer" section.	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		• Added VPID caution note to Sections 4.1.1 and 4.1.2. • Updated "Security ID (SECID)" in Section 4.1.3. • Added "IPv6-ready" note to Sections 6.2.16, 6.2.17, 7.1.1, and 7.1.2. • Updated the **IPADDRESS** variable description in Section 6.2.16. • Added an IPv6 example to Section 7.1.2. • Added reference to Kernel Toolkit patch XT*7.3*138 in Sections 27.4.1, 27.4.3, 27.4.5, and 27.4.6. • Added the .xt8meth input parameter and reference links to Section 27.4.3. • Added reference to Kernel Toolkit patch XT*7.3*138 in Section 27.7.1. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	
07/19/2016	11.7	Tech Edits: • Updated the "XML Parser (VistA): Developer Tools" section. Added overview content from the standalone *VistA XML Parser Technical and User Documentation* (Patches XT*7.3*58 & 67). • Updated the $$PATCH^XPDUTL(): Verify Patch Installation API to correct the example based on NSD Incident I6524269FY16. • Replaced "Integration Agreement (IA)" with "Integration Control Registration (ICR)" throughout the document. • Updated the $$CREATE^XUSAP: Create Application Proxy User API based on feedback from developer.	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		• Added the "Developing a File Merge Capability" section (content taken from *Kernel Toolkit 7.3 User Manual*). • Added Caution note regarding modification of Kernel routines in the "Software Disclaimer" section. • Removed all API tables used to format API data for Section 508 conformance. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	
10/20/2015	11.6	Tech Edits: • Updated the following APIs: ○ $$GETURL^XTHC10: Return URL Data Using HTTP API. ○ $$MAKEURL^XTHCURL: Creates a URL from Components API. ○ $$ENCODE^XTHCURL: Encodes a Query String API. ○ $$PARSEURL^XTHCURL: Parses a URL API. ○ $$DECODE^XTHCUTL: Decodes a String API. • Corrected Example 2 in the SAY^XGF(): Screen String API. • Added the "^XTMP Global: Developer Tools" section. • Updated Table 14 in Section 15.2.4.5. • Added "Data Security: Developer Tools" section and APIs based on Kernel Patch XU*8.0*655. The following APIS were added: ○ $$AESDECR^XUSHSH(): Returns Plaintext String Value for AES Encrypted Ciphertext Entry API. ○ $$AESENCR^XUSHSH(): Returns AES Encrypted Ciphertext for String Entry API.	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		○ $$B64DECD ^XUSHSH(): Returns Decoded Value for a Base64 String Entry API. ○ $$B64ENCD^XUSHSH(): Returns Base64 Encoded Value for a String Entry API. ○ $$RSADECR^XUSHSH(): Returns Plaintext String Value for RSA Encrypted Ciphertext Entry API. ○ $$RSAENCR^XUSHSH(): Returns RSA Encrypted Ciphertext for String Entry API. ○ $$SHAHASH^XUSHSH(): Returns SHA Hash for a String Entry API. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	
06/11/2015	11.5	Updated the following: • Merged (and then deleted) the "Toolkit—VistA XML Parser APIs" section into the "XML Parser (VistA): Developer Tools" section, since they had duplicate API content. • Updated document for Kernel Toolkit patch XT*7.3*81. Added the "Toolkit—M Unit" section. • Updated document for Kernel Patches XU*8.0*605 and 638. Added the following APIs to the "Application Programming Interface (API)" section in the "XLF Function Library: Developer Tools" section: ○ Added the "$$CONVERT^XLFIPV():" API. ○ Added the "$$FORCEIP4^XLFIPV(): Convert any IP Address to IPv4" API. ○ Added the "$$FORCEIP6^XLFIPV():	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		Convert any IP Address to IPv6" API. ○ Added the "$$VALIDATE^XLFIPV(): Validate IP Address Format" API. ○ Added the "$$VERSION^XLFIPV: Show System Settings for IPv6" API. • Updated the $$SCH^XLFDT(): Next Scheduled Runtime API: Added Example 4. • Updated the "$$ADDRESS^XLFNSLK(): Convert Domain Name to IP Addresses" API for changes to IPv4 and IPv6 in Kernel Patch XU*8.0*638. • Merged the DELSTAT^XQALBUTL API content with the DELSTAT^XQALBUTL(): API. • Added the following APIs in this manual to the online HTML APIs: ○ Toolkit—Duplicate Record Merge ○ Toolkit—KERMIT APIs ○ Toolkit—Multi-Term Look-Up (MTLU) APIs ○ Toolkit—M Unit Utility ○ Toolkit—Parameter Tools • Reformatted document to follow latest documentation standards and formatting rules. Also, formatted document for online presentation vs. print presentation (i.e., for double-sided printing). These changes include: ○ Revised section page setup. ○ Removed section headers. ○ Revised document footers. ○ Removed blank pages between sections. ○ Revised all heading style formatting.	

Date	Revision	Description	Author
		Software Versions: **Kernel 8.0** **Toolkit 7.3**	
09/24/2014	11.4	Updated the following: • $$LOOKUP^XUSER(): New Person File Lookup API: minor corrections and used example in this guide to match and scrub examples in online API. • $$NAME^XUSER(): Get Name of User API: fixed index entries. • "$$DEA^XUSER()—Get User's DEA Number" API: Added ien input parameter and Example 4. • Added statement to Section 15.2.4.4 as per Remedy Ticket #63050. • Changed all references from "OIT" to "OI&T" throughout. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	VistA Infrastructure (VI) Development Team
04/07/2014	11.3	Added a patch reference note and made minor edits/updates to the following APIs: • ^%ZIS: Standard Device Call API. • REQ^%ZTLOAD: Requeue a Task API. • Updated the ^%ZOSF(): Operating System-dependent Logic Global API. Changed reference in ("LOAD") from "DIE" to "DIF". • Added patch release reference note to $$GET^XUA4A72(): Get Specialty and Subspecialty for a User and $$IEN2CODE^XUA4A72(): Get VA Code APIs. • Redacted document for the following information: ○ Names (replaced with role and initials).	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		◦ Production IP addresses and ports. ◦ VA Intranet websites. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	
05/31/2013	11.2	Updates: • Updated document for Word accessibility issues for Section 508 conformance. • Made general style and format updates as needed. • Updated/Corrected all URLs (active and inactive) • Updated document for Section 508 conformance; ◦ Added bookmarks (identifiers) to all tables. ◦ Changed all floating callout boxes to in-line boxes. ◦ Added screen tips to all active URLs. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	VistA Infrastructure (VI) Development Team
04/30/2013	11.1	Updates: • Updated document for Kernel Patch XU*8.0*580. Added the following APIs to the "Application Programming Interface (API)" section in the "User: Developer Tools" section: ◦ Updated the "$$DEA^XUSER()—Get User's DEA Number" API. ◦ Added the "$$DETOX^XUSER()—Get Detox/Maintenance ID Number" API. ◦ Added the "$$SDEA^XUSER()—Check for Prescribing Privileges" API.	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		○ Added the "$$VDEA^XUSER()—Check if User Can Sign Controlled Substance Orders" API. • Reformatted document to follow current style guides and standards. • Replaced references from "*VA FileMan Getting Started Manual*" to "*VA FileMan User Manual*," since the next VA FileMan 22.*n* software version will create a new "*VA FileMan Getting Started Manual*." • Updated the **ZTCPU** input variable description in the ^%ZTLOAD: Queue a Task API, as per email feedback on 10/04/12 from developer. • HD0000000748766: Updated the following APIs; ○ $$ID^XUAF4(): Institution Identifier ○ $$IDX^XUAF4(): Institution IEN (Using Coding System & ID) ○ $$IEN^XUMF(): Institution IEN (Using IFN, Coding System, & ID) • HD0000000598920: Added documentation for the XPD NO_EPP_DELETE parameter to the new "Key Parameters during Pre- and Post-Install Routines" section, as requested by developer. • HD0000000389572: Removed the obsolete Section 11.2, "Link to the OBJECT File", as per email discussion on 03/23/2010; see Remedy Ticket #HD0000000389572. • Patch XU*8.0*546: Removed Support for Device Hunt Groups. This includes removal of the *HUNT GROUP (#29) and HUNT	

Date	Revision	Description	Author
		GROUP DEVICE (#30) fields in the DEVICE (#3.5) file. Sites had to remove any HUNT GROUP devices before installing this patch using VA FileMan to find any existing Hunt Groups. Removed any references to "Hunt Groups" from this document. • Added the following XPDPROT APIs released with Kernel Patch XU*8.0*547: ○ $$ADD^XPDPROT(): Add Child Protocol to Parent Protocol. ○ $$DELETE^XPDPROT(): Delete Child Protocol from Parent Protocol. ○ FIND^XPDPROT(): Find All Parents for a Protocol. ○ $$LKPROT^XPDPROT(): Look Up Protocol IEN. ○ OUT^XPDPROT(): Edit Protocol's Out of Order Message. ○ RENAME^XPDPROT(): Rename Protocol. ○ $$TYPE^XPDPROT(): Get Protocol Type. • Added blue font highlighting and underline to signify internal links to figures, tables, or sections for ease of use, similar to what one sees to hyperlinks on a Web page. • Updated document for Section 508 conformance using word's built-in Accessibility check: ○ Added table bookmarks. ○ Added screen tips for all URL links. ○ Changed all floating callout boxes to in-line, causing reformatting of numerous dialogue screen captures. **Software Versions:** **Kernel 8.0**	

Date	Revision	Description	Author
		Toolkit 7.3	
07/26/2012	11.0	Updates: • $$SETUP1^XQALERT: Send Alerts. Corrected the descriptions for the **XQAARCH** and **XQASUPV** variables based on feedback from developer. • Updated the "OPEN^%ZISUTL(): Open Device with Handle" API. Corrected reference to the CLOSE^%ZISUTL(): Close Device with Handle API, based on feedback from developer. • Added the "XU USER START-UP Option" section. The XU USER START-UP option was added with Kernel Patch XU*8.0*593. • Reordered sections in Section 26, "Toolkit: Developer Tools," to discuss all APIs before general Toolkit developer tools/options. • Added/Promoted the "XINDEX" section based on the following: ○ Open Source Electronic Health Record Agent (OSEHRA) software quality certification dashboard review of VistA Freedom of Information Act (FOIA) code using the XINDEX tool. ○ Code review and updates by developer related to Kernel Toolkit patch XT*7.3*132. ○ Created a new VA Intranet Kernel Toolkit XINDEX website. • Updated the "%Index of Routines Option—XINDEX" based on addition of new XINDEX section and feedback from developer related to Kernel Toolkit patch XT*7.3*132. • Added the TOUCH^XUSCLEAN: Notify Kernel of Tasks that Run 7 Days or Longer API to this document after already being	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		added to VA Intranet online Kernel APIs; based on email dated 02/08/11. • Revised all version numbers in the "Revision History" section. • Updated the "Orientation" section. • Updated the overall document for current national documentation standards and style guides. For example: ○ Changed all Heading *n* styles to use Arial font. ○ Changed all Heading *n* styles to be left justified. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	
10/18/2011	10.1	Updates: • Updated the "STDNAME^XLFNAME(): Name Standardization Routine" API for Kernel Patch XU*8.0*535. • Updated formatting and internal styles. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	VistA Infrastructure (VI) Development Team
09/15/2011	10.0	Updates: • Made opt parameter optional in the$$EN^MXMLDOM(): XML—Initial Processing, Build In-memory Image API. • Added Cautionary Note to the $$CREATE^XUSAP: Create Application Proxy User API. • Updated the $$SCH^XLFDT(): Next Scheduled Runtime API examples, as per suggestion by developer via email. • Updated the $$SCREEN^XTID(): Get Screening Condition (Term/Concept) API based on Remedy #HD0000000391324.	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		• Made other minor format, style, grammar, and punctuation updates. • Updated ^%ZTER: Kernel Standard Error Recording Routine API to remove statement about NEWing all variables. This does *not* apply for this API. • Changed all reference to NEWing variables from "NEW all variables." to "NEW all *non*-namespaced variables" and removed follow-up explanation throughout the document. • Updated $$DELETE^XPDMENU(): Delete Menu Item API. Corrected documentation to show this as an extrinsic function. • Updated $$LKOPT^XPDMENU(): Look Up Option IEN API. Corrected documentation to show this as an extrinsic function. • Added the new $$TYPE^XPDMENU(): Get Option Type API. • Added Section 26.5, "Toolkit—HTTP Client APIs." and the following APIs: ○ $$GETURL^XTHC10: Return URL Data Using HTTP. ○ $$ENCODE^XTHCURL: Encodes a Query String. ○ $$MAKEURL^XTHCURL: Creates a URL from Components. ○ $$PARSEURL^XTHCURL: Parses a URL. ○ $$DECODE^XTHCUTL: Decodes a String. • Updates Section 14.2.4.3.2, "Sending Security Codes" to include reference to VA FileMan FILESEC^DDMOD to set security access.	

Date	Revision	Description	Author
		• Updated/Clarified Section 14.2.4.3.5, "Partial DD (Some Fields)," and added Figure 54. KIDS—Partial DD: Choosing DD levels (top level and Multiple) to send. • Added NOTE regarding Class 3 and FORCED queuing related to Kernel Patches XU*8.0*546/556 to the top of Section 5, "Device Handler: Developer Tools." • Updated the "$$LAST^XPDUTL(): Last Software Patch" API based on Kernel Patch XU*8.0*559. • Added the **XPDNM("TST")** and **XPDNM("SEQ")** variables to Table 9. KIDS—Key variables during the environment check and Table 14. KIDS—Key variables during the pre- and post-install routines, as per Kernel Patch XU*8.0*559. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	
03/18/2010	9.0	Added the text "Any routine that is specified is automatically sent by KIDS. You do *not* have to list the routine in the Build Components section." to the following sections: • 14.3.1, "Environment Check Routine." • 14.3.3, "Pre- and Post-Install Routines: Special Features." **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	VistA Infrastructure (VI) Development Team
11/16/2009	8.0	Updates: • Added the SUROFOR^XQALSURO(): Return a Surrogate's List of Users API. • Deleted SUROLIST^XQALSUR1 API and added the	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		SUROLIST^XQALSURO(): List Surrogates for a User API. • Updated APIs to change input parameter to Input Variable for EN^XQH: Display Help Frames and EN1^XQH: Display Help Frames APIs. • Updated input variable for ^%ZTER: Kernel Standard Error Recording Routine API. • Updated WITNESS^XUVERIFY(): Return IEN of Users with A/V Codes & Security Keys API. • Updated Section 17, "Miscellaneous: Developer Tools." Added the following sections from the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide* to the *Kernel 8.0 & Kernel Toolkit 7.3 Developer's Guide*, because the functions documented are more developer-related than system management-related: o Programmer Options Menu o ^%Z Editor • Updated Section 26, "Toolkit: Developer Tools." Added the following sections from the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide* to the *Kernel 8.0 & Kernel Toolkit 7.3 Developer's Guide*, because the functions documented are more developer-related than system management-related: o Toolkit—Routine Tools o Toolkit—Verification Tools • Updated the introductory content in Section 29, "XGF Function Library: Developer Tools." Moved the XGF Function Library content from the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide* to the *Kernel 8.0 & Kernel Toolkit 7.3 Developer's Guide*, because the functions	

Date	Revision	Description	Author
		documented are more developer-related than system management-related. • Reviewed and updated all sections for minor format changes (e.g., bulleted lists and tables), style updates, spelling, and grammar fixes. • Added **GSEL** node to ^%ZOSF(): Operating System-dependent Logic Global API. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	
07/09/2009	7.4	Updates: • After developer re-review, corrected reference type from “Controlled Subscription” back to “Supported” for the $$OS^%ZOSV: Get Operating System Information API and updated the ICR # to 10097. Updated the FORUM ICR. • Added ICR # 10097 to the $$VERSION^%ZOSV(): Get OS Version Number or Name API. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	VistA Infrastructure (VI) Development Team
07/02/2009	7.3	Updates: • Corrected reference type from “Supported” to Controlled Subscription” for the $$OS^%ZOSV: Get Operating System Information API. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	VistA Infrastructure (VI) Development Team
06/23/2009	7.2	Updates: • Added new section, “Long Running Tasks—Using ^%ZIS" to Section 25.	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		• Renamed "Writing Two-step Tasks" section to "Long Running Tasks—Writing Two-step Tasks" in Section 25. • Reformatted document to add outline numbering. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	
05/04/2009	7.1	Updates: • Patch XT*7.3*111, released FEB 13, 2009. Included new section titled "Toolkit—Data Standardization APIs" in the Toolkit: Developer Tools section. • Background: Toolkit—Developed Data Standardization APIs to support Data Standardization's effort to allow the mapping of one term to another term. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	VistA Infrastructure (VI) Development Team
04/27/2009	7.0	Updates: • Updated $$SCREEN^XTID(): Get Screening Condition (Term/Concept) API (ICR # 4631) for Kernel Toolkit patch XT*7.3*108. • Updated ^XUWORKDY: Workday Calculation (Obsolete) API. • Added $$EN^XUWORKDY: Number of Workdays Calculation API. • Added $$WORKDAY^XUWORKDY: Workday Validation API. • Added $$WORKPLUS^XUWORKDY: Workday Offset Calculation API. • Updated $$PATCH^XPDUTL(): Verify Patch Installation. • Updated the "Orientation" section.	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		• Updated organizational references. • Minor format updates (e.g., reordered the document Revision History table to display latest to earliest). • Other minor format updates to correspond with the latest standards and style guides. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	
10/28/2008	6.3	Updates: • Table 26: Added "DEV" entity and corrected the OE/RR LIST file number from "101.21" to the correct "100.21" file number. • Updated references to the CHCKSUM^XTSUMBLD direct mode utility and added references to CHECK^XTSUMBLD and CHECK1^XTSUMBLD routines in Table 28 in Section 26, "Toolkit: Developer Tools." • Minor format updates. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	VistA Infrastructure (VI) Development Team
10/01/2008	6.2	Updates: • Minor format updates (e.g., reordered document Revision History table to display latest to earliest). • DE^XUSHSHP: Decrypt Data String API. • EN^XUSHSHP: Encrypt Data String API. • HASH^XUSHSHP: Hash Electronic Signature Code. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
08/07/2008	6.1	Updates: • Made general formatting and organizational reference changes where appropriate. • Changed references from "%INDEX" to "XINDEX" where appropriate. • Updated Table 8, last two entries. • Updated "PRE-TRANSPORTATION ROUTINE (#900) Field" section to show use of the **XPDGREF** variable in Pre-install, Environment Check, and/or Post-install routines. • Removed Appendix A—KIDS Build Checklists (Obsolete). • API Updates: ○ $$MV^%ZISH(): Rename Host File. ○ $$NODEV^XUTMDEVQ(): Force Queuing—No Device Selection—Updated input parameters. ○ $$INSTALDT^XPDUTL(): Return All Install Dates/Times. ○ UPDATE^XPDID(): Update Install Progress Bar. ○ Moved INIT^XPDID: Progress Bar Emulator: Initialize Device and Draw Box Borders API to "Miscellaneous: Developer Tools" section. ○ Moved TITLE^XPDID(): Progress Bar Emulator: Display Title Text API to "Miscellaneous: Developer Tools" section. ○ Moved EXIT^XPDID(): Progress Bar Emulator: Restore Screen, Clean Up Variables, and Display Text API to "Miscellaneous: Developer Tools" section. ○ OP^XQCHK(): Current Option Check.	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		○ ENDR^%ZISS: Set Up Specific Screen Handling Variables. ○ $$ASKSTOP^%ZTLOAD: Stop TaskMan Task. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	
01/07/2008	6.0	Updates: • $$CJ^XLFSTR(): Center Justify String. • $$LJ^XLFSTR(): Left Justify String. • $$RJ^XLFSTR(): Right Justify String. • DELETE^XQALERT: Clear Obsolete Alerts. • DELETEA^XQALERT: Clear Obsolete Alerts. • SETUP^XQALERT: Send Alerts. • $$SETUP1^XQALERT: Send Alerts. • FORWARD^XQALFWD(): Forward Alerts. • REMVSURO^XQALSURO(): Remove Surrogates for Alerts. • SUROLIST^XQALSURO(): List Surrogates for a User. • SETSURO1^XQALSURO(): Establish a Surrogate for Alerts. • GETIREF^XTID(): Get IREF (Term/Concept). • $$GETMASTR^XTID(): Get Master VUID Flag (Term/Concept). • $$GETSTAT^XTID(): Get Status Information (Term/Concept). • $$GETVUID^XTID(): Get VUID (Term/Concept). • $$SCREEN^XTID(): Get Screening Condition (Term/Concept) API (ICR # 4631).	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		• $$SETMASTR^XTID(): Set Master VUID Flag (Term/Concept). • $$SETSTAT^XTID(): Set Status Information (Term/Concept). • $$SETVUID^XTID(): Set VUID (Term/Concept). • $$IEN^XUPS(): Get IEN Using VPID in File #200—Changed references to IENS to IEN. • $$NNT^XUAF4(): Institution Station Name, Number, and Type—Output order was previously incorrect, should be Name, Number, and type *not* Number, Name, and Type. • $$NODEV^XUTMDEVQ(): Force Queuing—No Device Selection—Updated input parameters. • $$OPTDE^XPDUTL(): Disable/Enable an Option. • ^%ZIS: Standard Device Call—Added output parameters. • ^%ZOSF(): Operating System-dependent Logic Global. General Updates: • Updated the "Re-Indexing Files" section based on Remedy Ticket #63087. • Updated references to the VDL. • Updated the "Alpha/Beta Tracking" section in Section 14. Merged information from the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide* into the *Kernel 8.0 & Kernel Toolkit 7.3 Developer's Guide* (this manual) in order to avoid duplication and confusion with instructions/procedures. • Removed all but one reference to HSD&D; kept as a placeholder for now. • Removed obsolete references to MSM, PDP, 486, VAX Alpha, etc. and changed/updated references	

Date	Revision	Description	Author
		to DSM for OpenVMS to Caché where appropriate. **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	
02/08/2007	5.0	Updates: • Merging the Kernel Toolkit documentation set with the Kernel documentation set. Moving all Kernel Toolkit content to the appropriate Kernel manual and section. In the *Kernel 8.0 & Kernel Toolkit 7.3 Developer's Guide*, the following Kernel Toolkit APIs and Direct Mode Utilities have been added to the new “Toolkit” Section: ○ Toolkit—Alerts APIs ○ Toolkit—Duplicate Record Merge APIs ○ Toolkit—KERMIT APIs ○ Toolkit—Multi-Term Look-Up (MTLU) APIs ○ Toolkit—Parameter Tools APIs ○ Toolkit—VistA XML Parser APIs ○ Toolkit—VHA Unique ID (VUID) APIs **NOTE:** Adding Kernel Toolkit APIs to the Kernel APIs VA Intranet Website in the near future. • Added new National Provider Identifier (NPI)-related APIs section. APIs released with Kernel Patch XU*8.0*410: ○ $$CHKDGT^XUSNPI (ICR # 4532) ○ $$NPI^XUSNPI (ICR # 4532) ○ $$QI^XUSNPI (ICR # 4532)	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		◦ $$TAXIND^XUSTAX (ICR # 4911) ◦ $$TAXORG^XUSTAX (ICR # 4911 • Added new Common Services-related APIs section. APIs released with Kernel Patches XU*8.0*309 and 325: ◦ $$VPID^XUPS (ICR # 4574) ◦ $$IEN^XUPS (ICR # 4574) ◦ EN1^XUPSQRY (ICR # 4575) • Changed Kernel document title references to: ◦ *Kernel 8.0 & Kernel Toolkit 7.3 Developer's Guide* (previously known as the *Kernel Programmer Manual*). ◦ *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide* (previously known as the *Kernel Systems Manual*). **Software Versions:** **Kernel 8.0** **Toolkit 7.3**	
06/20/2006	4.1	Updates: • Corrected output array subscript in the F4^XUAF4 API from "STATION NUMER" to "STATION NUMBER (Remedy #HD0000000147298). • Updated document format to follow latest Guidelines and SOP. **Kernel 8.0**	VistA Infrastructure (VI) Development Team
01/23/2006	4.0	Updates: • $$QQ^XUTMDEVQ, updated description (XU*8.0*389). • Changed REQQ^XUTMDEVQ to $$REQQ^XUTMDEVQ; updated description (XU*8.0*389). • Updated REQ^%ZTLOAD and ^%ZTLOAD APIs.	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		• Changed $$SENTCASE^XLFSTR to $$SENTENCE^XLFSTR (XU*8.0*400). **Kernel 8.0**	
12/15/2005	3.8	Added the following APIs (via patches currently *not* yet released): • $$CREATE^XUSAP (XU*8.0*361) • $$SENTCASE^XLFSTR (XU*8.0*400) • $$TITLE^XLFSTR (XU*8.0*400) • Changed Job^%ZTLOAD to $$JOB^%ZTLOAD **Kernel 8.0**	VistA Infrastructure (VI) Development Team
10/19/2005	3.7	Updated the SETUP^XQALERT API based on feedback from the user community and developers. **Kernel 8.0**	VistA Infrastructure (VI) Development Team
09/28/2005	3.6	Added the $$HANDLE^XUSRB4 and REQQ^XUTMDEVQ APIs. **Kernel 8.0**	VistA Infrastructure (VI) Development Team
09/22/2005	3.5	Updated APIs: • SETUP^XQALERT • SETUP^XUSRB • OWNSKEY^XUSRB • DQ^%ZTLOAD • ISQED^%ZTLOAD • KILL^%ZTLOAD • PCLEAR^%ZTLOAD • STAT^%ZTLOAD Added APIs: • ASKSTOP^%ZTLOAD • DESC^%ZTLOAD • JOB^%ZTLOAD • OPTION^%ZTLOAD • $$PSET^%ZTLOAD • RTN^%ZTLOAD • $$S^%ZTLOAD • ZTSAVE^%ZTLOAD **Kernel 8.0**	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
04/14/2005	3.4	Categorized CRC XLF functions into a new category (i.e., "CRC" vs. "Other"). **Kernel 8.0**	VistA Infrastructure (VI) Development Team
03/02/2005	3.3	Corrected various APIs. Reordered all APIs under each category: 1) by routine name and 2) by tag name. **Kernel 8.0**	VistA Infrastructure (VI) Development Team
02/10/2005	3.2	Updates: • ^%ZTLOAD: Queue a Task • REQ^%ZTLOAD: Requeue a Task • Added three new XUTMDEVQ APIs (Kernel Patch XU*8.0*275). **Kernel 8.0**	VistA Infrastructure (VI) Development Team
12/20/2004	3.1	Reviewed document and edited for the "Data Scrubbing" and the "PDF 508 Compliance" projects. **Data Scrubbing—**Changed all patient/user TEST data to conform to OIT standards and conventions as indicated below: The first three digits (prefix) of any Social Security Numbers (SSN) start with "**000**" or "**666**." Format patient or user names as follows: XUPATIENT,[N] or XUUSER,[N] respectively, where the N is a number written out and incremented with each new entry (e.g., XUPATIENT, ONE, XUPATIENT, TWO, etc.). Changed other personal demographic-related data (e.g., addresses, phones, IP addresses, etc.) to be generic. **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). **Kernel 8.0**	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
12/09/2004	3.0	Updated various APIs based on developer feedback. Also, making minor edits as we begin populating the HTML versions of the APIs. **Kernel 8.0**	VistA Infrastructure (VI) Development Team
12/24/2003	2.0	Kernel 8.0 documentation reformatting/revision. This is the initial *Kernel 8.0 & Kernel Toolkit 7.3 Developer's Guide*. Created this manual by extracting all developer-specific content from the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide* (original release date of July 1995). The *Kernel 8.0 & Kernel Toolkit 7.3 Developer's Guide* includes added/updated Direct Mode Utilities and Application Programming Interface (API) information (e.g., Reference Type, Category, Integration Control Registration number. etc.). It also includes APIs for previous Kernel APIs never before documented (i.e., includes APIs that were previously only documented in patch descriptions, Integration Control Registrations, or separate supplemental documentation). **NOTE:** This manual also includes the Kernel Toolkit APIs. Due to time constraints, *not* all released Kernel patches with developer-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 the *Kernel 8.0 & Kernel Toolkit 7.3 Developer's Guide*. As time allows, we will be updating this manual with all released patch information that affects its content.	VistA Infrastructure (VI) Development Team

Date	Revision	Description	Author
		Kernel 8.0	
07/1995	1.0	Initial Kernel 8.0 software and documentation release. **Kernel 8.0**	VistA Infrastructure (VI) Development Team

Patch Revisions

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

Table of Contents

List of Figures

List of Tables

Orientation

How to Use this Manual

This manual provides advice and instruction about Kernel 8.0 & Kernel Toolkit 7.3 Application Programming Interfaces (APIs), Direct Mode Utilities, and other information for Veterans Health Information Systems and Technology Architecture (VistA) application developers.

Intended Audience

The intended audience of this manual is the following stakeholders:

- Enterprise Program Management Office (EPMO)—VistA legacy development teams.
- System Administrators—System administrators at Department of Veterans Affairs (VA) regional and local sites who are responsible for computer management and system security on the VistA M Servers.
- Information Security Officers (ISOs)—Personnel at VA sites responsible for system security.
- Product Support (PS).

Disclaimers

Software Disclaimer

This software was developed at the Department of Veterans Affairs (VA) by employees of the Federal Government in the course of their official duties. Pursuant to title 17 Section 105 of the United States Code this software is *not* subject to copyright protection and is in the public domain. VA assumes no responsibility whatsoever for its use by other parties, and makes no guarantees, expressed or implied, about its quality, reliability, or any other characteristic. We would appreciate acknowledgement if the software is used. This software can be redistributed freely provided that any derivative works bear some notice that they are derived from it.

CAUTION: Kernel routines should *never* be modified at the site. If there is an immediate national requirement, the changes should be made by emergency Kernel patch. Kernel software is subject to FDA regulations requiring Blood Bank Review, among other limitations. Line 3 of all Kernel routines states:

Per VA Directive 6402 (pending signature), this routine should not be modified.

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).

Documentation Disclaimer

This manual provides an overall explanation of using kernel; 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 Internet and Intranet SharePoint sites and websites for a general orientation to VistA. For example, visit the Office of Information and Technology (OIT) Enterprise Program Management Office (EPMO) Intranet Website.

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

Documentation Conventions

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. Table 1 gives a description of each of these symbols:

Table 1: Documentation Symbol Descriptions

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

- 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) begin with either "**000**" or "**666**".
  - Patient and user names are formatted as follows:
    - *<Application Name/Abbreviation/Namespace>*PATIENT,*<N>*
    - *<Application Name/Abbreviation/Namespace>*USER,*<N>*

Where:

– *<Application Name/Abbreviation/Namespace>* is defined in the Approved Application Abbreviations document.

– *<N>* represents the first name as a number spelled out and incremented with each new entry.

For example, in Kernel (XU or KRN) test patient and user names would be documented as follows:

KRNPATIENT,ONE; KRNPATIENT,TWO; KRNPATIENT,THREE; … KRNPATIENT,14; etc.

KRNUSER,ONE; KRNUSER,TWO; KRNUSER,THREE; … KRNUSER,14; etc.

- "Snapshots" of computer online displays (i.e., screen captures/dialogues) and computer source code is shown in a *non*-proportional font and may be enclosed within a box.
  - User's responses to online prompts are **boldface** and (optionally) highlighted in yellow (e.g., **<Enter>**).
  - Emphasis within a dialogue box is **boldface** and (optionally) highlighted in blue (e.g., STANDARD LISTENER: RUNNING).
  - Some software code reserved/key words are **boldface** with alternate color font.
  - References to "**<Enter>**" within these snapshots indicate that the user should press the **Enter** key on the keyboard. Other special keys are represented within **< >** angle brackets. For example, pressing the **PF1** key can be represented as pressing **<PF1>**.
  - Author's comments 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 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 is 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 are used with regards to APIs:
  - The following API types are documented:
    - **Supported:**

      This applies where any VistA application may use the attributes/functions defined by the Integration Control Registration (ICR); these are also called “Public”. An example is an ICR that describes a standard API. The package that creates/maintains the Supported Reference *must* ensure it is recorded as a Supported Reference in the ICR database. There is no need for other VistA packages to request an ICR to use these references; they are open to all by default.
    - **Controlled Subscription:**

      Describes attributes/functions that *must* be controlled in their use. The decision to restrict the Integration Control Registration (ICR) is based on the maturity of the custodian package. Typically, these ICRs are created by the requesting package based on their independent examination of the custodian package's features. For the ICR to be approved the custodian grants permission to other VistA packages to use the attributes/functions of the ICR; permission is granted on a one-by-one basis where each is based on a solicitation by the requesting package.

    Private APIs are *not* documented.
  - Headings for developer 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 is 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 shown in lowercase and variables are shown in uppercase. This is to convey that the 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. In addition, 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 security 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 (e.g., CamelCase).

Documentation Navigation

This document uses Microsoft® Word's built-in navigation for internal hyperlinks. To add **Back** and **Forward** navigation buttons to the toolbar, do the following:

1. Right-click anywhere on the customizable Toolbar in Word (*not* the Ribbon section).
2. Select **Customize Quick Access Toolbar** from the secondary menu.
3. Select the drop-down arrow in the "Choose commands from:" box.
4. Select **All Commands** from the displayed list.
5. Scroll through the command list in the left column until you see the **Back** command (circle with arrow pointing left).
6. Select/Highlight the **Back** command and select **Add** to add it to your customized toolbar.
7. Scroll through the command list in the left column until you see the **Forward** command (circle with arrow pointing right).
8. Select/Highlight the **Forward** command and select **Add** to add it to the customized toolbar.
9. Select **OK**.

You can now use these **Back** and **Forward** command buttons in the Toolbar to navigate back and forth in the Word document when selecting hyperlinks within the document.

NOTE: This is a one-time setup and is automatically available in any other Word document once you install it on the Toolbar.

How to Obtain Technical Information Online

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

NOTE: Methods of obtaining specific technical information online is indicated where applicable under the appropriate section.

REF: For further information, see the *Kernel 8.0 & Kernel Toolkit 7.3 Technical Manual*.

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 VistA M Server-based 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** [DILIST] option on the **Data Dictionary Utilities** [DI DDU] menu in VA FileMan to print formatted data dictionaries.

REF: For details about obtaining data dictionaries and about the formats available, see the “List File Attributes” section in the “File Management” section in the *VA FileMan Advanced User Manual*.

Assumptions

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

Reference Materials

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

- *Kernel Release Notes*
- *Kernel Installation Guide*
- *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide*
- *Kernel 8.0 & Kernel Toolkit 7.3 Developer's Guide* (this manual)
- *Kernel 8.0 & Kernel Toolkit 7.3 Technical Manual*
- *Kernel Security Tools Manual*
- Kernel VA Intranet Website.

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

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, which is freely distributed by Adobe® Systems Incorporated at: http://www.adobe.com/

VistA documentation can be downloaded from the VA Software Document Library (VDL): http://www.va.gov/vdl/

REF: Kernel manuals are located on the VDL at: http://www.va.gov/vdl/application.asp?appid=10

VistA documentation and software can also be downloaded from the Product Support (PS) Anonymous Directories.

1 Introduction

1.1 Overview

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

You can find developer information in the sections and sub-sections of this manual that contain "Developer Tools" in their titles. You might want to concentrate on those sections 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: Developer Tools" section.

Kernel provides developers with a number of tools. These tools include Application Programming 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 8.0 & Kernel Toolkit 7.3 Developer's Guide* is divided into sections, based on the following functional API/Direct Mode Utility categories within Kernel (listed alphabetically):

- Address Hygiene: Developer Tools
- Alerts: Developer Tools
- Common Services: Developer Tools
- Data Security: Developer Tools
- Device Handler: Developer Tools
- Domain Name Service (DNS): Developer Tools
- Electronic Signatures: Developer Tools
- Error Processing: Developer Tools
- Field Monitoring: Developer Tools
- File Access Security: Developer Tools
- Help Processor: Developer Tools
- Host Files: Developer Tools
- Institution File: Developer Tools
- Kernel Installation and Distribution System (KIDS): Developer Tools

- Menu Manager: Developer Tools
- Miscellaneous: Developer Tools
- Name Standardization: Developer Tools
- National Provider Identifier (NPI): Developer Tools
- Operating System (OS) Interface: Developer Tools
- Security Keys: Developer Tools
- Server Options: Developer Tools
- Signon/Security: Developer Tools
- Spooling: Developer Tools
- TaskMan: Developer Tools
- Toolkit: Developer Tools
    - Toolkit—Data Standardization
    - Toolkit—Duplicate Record Merge
    - Developing a File Merge Capability
    - Toolkit—HTTP Client
    - Toolkit—KERMIT APIs
    - Toolkit—Multi-Term Look-Up (MTLU) APIs
    - Toolkit—M Unit
    - Toolkit—Parameter Tools
    - Toolkit—VHA Unique ID (VUID) APIs
    - Toolkit—Routine Tools
    - Toolkit—Verification Tools
    - XINDEX

- Unwinder: Developer Tools
- User: Developer Tools
- XGF Function Library: Developer Tools
- XLF Function Library: Developer Tools
    - Bitwise Logic Functions—XLFSHAN
    - CRC Functions—XLFCRC
    - Date Functions—XLFDT
    - Hyperbolic Trigonometric Functions—XLFHYPER

    - Mathematical Functions—XLFMTH
    - Measurement Functions—XLFMSMT
    - String Functions—XLFSTR
    - Utility Functions—XLFUTL
    - IP Address Functions—XLFIPV
    - JSON Conversion Functions—XLFJSON

- XML Parser (VistA): Developer Tools

REF: For general user information and system manager information, see the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide*.

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 8.0 & Kernel Toolkit 7.3 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*.

1.2 API Information

Each API displays the following information in the order listed:

1. **API Name** (required):

   This is the name of the API and is followed by a colon and a brief descriptive phrase of its use. It is written in one of the following formats:

   - **^ROUTINE or TAG^ROUTINE**—This format is used when the API is an entry point that does *not* take any input parameters in a parameter list (i.e., no parenthesis following the routine name).
   - **TAG^ROUTINE()**—This format is used when the API is a **procedure**. Parentheses following the routine name indicate that the API may take input parameters.

- **$$TAG^ROUTINE()**—This format is used when the API is an extrinsic **function**. Parentheses following the routine name indicate that the API may take input parameters.

For example:

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

In this case "**MAIL**" is the tag name, "**XLFNSLK**" is the routine name, and the parenthesis indicate that this API may take input parameters. The lack of "**$$**" preceding the tag name indicates that this API is a **procedure**. The brief text that follows the colon gives you a general idea of what this API does.

Another example:

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

In this case "**ADDRESS**" is the tag name, "**XLFNSLK**" is the routine name, and the parenthesis indicate that this API may take input parameters. The "**$$**" preceding the tag name indicates that this API is an extrinsic **function**. The brief text that follows the colon gives you a general idea of what this API does.

2. **Reference Type** (required):

   The Reference Type indicates the Integration Control Registration (ICR) for the API:

   - **Supported Reference**—An API of this type is open for use by any VistA application. It has been recorded as a Supported Reference in the IA database on FORUM. VistA software applications do *not* need to request an IA to use it.
   - **Controlled Subscription Reference**—An API of this type is controlled in its use. Permission to use the API is granted by the custodial package (software application, such as Kernel) on a case-by-case basis.

   NOTE: Private APIs are *not* documented in this manual.

3. **Category** (required):

   The Category indicates the general category to which the API belongs.

4. **Integration Control Registration** (required):

   The Integration Control Registration indicates the Supported or Controlled Subscription Reference Integration Control Registration (ICR) number for the API.

5. **Description** (required):

   This section provides an overall description of the API. Please include the patch reference ID (e.g., XU*8.0*999) if this API is being released via a patch.

6. **Format** (required):

   This section displays the format (usage) of the API. Optional parameters appear inside rectangular brackets **[ ]**. For example, tag^routine(x[,y]), the **x** input parameter is required and the **y** input parameter is optional. Rectangular brackets around a leading period **[.]** in front of a parameter indicate that you can optionally pass that parameter by reference.

7. **Input Parameters / Input Variables** (optional):

   This section lists all input parameters/variables for the API:

   - **Input Parameters**—Input passed in a parameter list to procedure and function APIs. For documentation purposes only, parameters are shown in lowercase.
   - **Input Variables**—Input variables passed through the symbol table to APIs without a parameter list. For documentation purposes only, variables are shown in uppercase.

   All input parameters *must* indicate whether they are "required" or "optional."

8. Output / Output Parameters / Output Variables (optional):

   This section lists all output or output variables returned by the API:

   - **Output**—Output returned through a "pass by reference" variable from a procedure or the return value of an extrinsic function API.
   - **Output Parameters**—Output parameters returned by the API.
   - **Output Variables**—Output variables returned through the symbol table from an API.

9. **Details** (optional):

   This section provides any additional information regarding the use of the API. This should include anything *not* already included in the API "Description" section.

10. **Examples** (required):

    This section provides one or more examples demonstrating the use/functionality of the API (*not* all APIs have examples).

2 Address Hygiene: Developer Tools

2.1 Application Programming Interface (API)

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

2.1.1 CCODE^XIPUTIL(): FIPS Code Data

Reference Type:	Supported	
Category:	Address Hygiene	
ICR #:	3618	
Description:	The CCODE^XIPUTIL API returns all the data associated for a Federal Information Processing Standards (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 (#5) file
- **XIPC("ERROR")**—Errors encountered during lookup

2.1.1.1 Example

Figure 1: CCODE^XIPUTIL API—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
```

2.1.2 $$FIPS^XIPUTIL(): FIPS Code for ZIP Code

Reference Type:	Supported	
Category:	Address Hygiene	
ICR #:	3618	
Description:	The $$FIPS^XIPUTIL extrinsic function returns the Federal Information Processing Standard (FIPS) Code associated with the Postal Code.	
Format:	`$$FIPS^XIPUTIL(pcode)`	
Input Parameters:	**pcode**:	(required) Postal Code for which the FIPS Code is returned.
Output:	returns:	Returns the FIPS Code.

2.1.2.1 Example

Figure 2: $$FIPS^XIPUTIL API—Example

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

>W X
54041
```

2.1.3 $$FIPSCHK^XIPUTIL(): Check for FIPS Code

Reference Type:	Supported	
Category:	Address Hygiene	
ICR #:	3618	
Description:	The $$FIPSCHK^XIPUTIL extrinsic function answers the question as to whether or not a Federal Information Processing Standard (FIPS) code exists. It returns the following: • **IEN**—Internal Entry Number, if the FIPS code exists. • **Zero (0)**—FIPS Code does *not* exist.	
Format:	`$$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.

2.1.3.1 Examples

2.1.3.1.1 Example 1

Figure 3: $$FIPSCHK^XIPUTIL API—Example 1

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

>W X
335
```

2.1.3.1.2 Example 2

Figure 4: $$FIPSCHK^XIPUTIL API—Example 2

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

>W X
0
```

2.1.4 POSTAL^XIPUTIL(): ZIP Code Information

Reference Type:	Supported	
Category:	Address Hygiene	
ICR #:	3618	
Description:	The POSTAL^XIPUTIL API returns United States Postal Service (USPS)-related data/information in an output array (see "Output Parameters") 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 United States Postal Service (USPS) assigned to this PCODE.
- **XIP("CITY ABBREVIATION")**—USPS assigned abbreviation.
- **XIP("CITY KEY")**—USPS assigned city key.
- **XIP("COUNTY")**—County associated with this PCODE.
- **XIP("COUNTY POINTER")**—Pointer to the county in the COUNTY CODE (#5.13) file.
- **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 preferred (DEFAULT) city key.
- **XIP("STATE")**—State associated with this PCODE.
- **XIP("STATE POINTER")**—Pointer to the state in the STATE (#5) file.

- **XIP("UNIQUE KEY")**—Unique lookup value.
- **XIP("ERROR")**—Errors encountered during lookup.

2.1.4.1 Examples

2.1.4.1.1 Example 1

Figure 5: POSTAL^XIPUTIL API—Example 1

```
>S ZCODE=99991

>S ZTMP=""

>D POSTAL^XIPUTIL(ZCODE,.ZTMP)

>ZW ZTMP,ZCODE
ZCODE=99991
ZTMP=
ZTMP("CITY")=ANYCITY1
ZTMP("CITY ABBREVIATION")=
ZTMP("CITY KEY")=Z22802
ZTMP("COUNTY")=ANYCOUNTY1
ZTMP("COUNTY POINTER")=2910
ZTMP("FIPS CODE")=06075
ZTMP("INACTIVE DATE")=
ZTMP("LATITUDE")=39:00N
ZTMP("LONGITUDE")=80:28W
ZTMP("POSTAL CODE")=99991
ZTMP("PREFERRED CITY KEY")=Z22802
ZTMP("STATE")=ANYSTATE1
ZTMP("STATE POINTER")=6
ZTMP("UNIQUE KEY")=999919Z22802
```

2.1.4.1.2 Example 2

Figure 6: POSTAL^XIPUTIL API—Example 2

```
>S ZCODE=99992

>S ZTMP=""

>D POSTAL^XIPUTIL(ZCODE,.ZTMP)

>ZW ZTMP,ZCODE
ZCODE=99992
ZTMP=
ZTMP("CITY")=ANYCITY2
ZTMP("CITY ABBREVIATION")=
ZTMP("CITY KEY")=Z22296
ZTMP("COUNTY")=ANYCOUNTY2
ZTMP("COUNTY POINTER")=2912
ZTMP("FIPS CODE")=06001
ZTMP("INACTIVE DATE")=
ZTMP("POSTAL CODE")=99992
ZTMP("PREFERRED CITY KEY")=Z22296
ZTMP("STATE")=ANYSTATE2
ZTMP("STATE POINTER")=6
ZTMP("UNIQUE KEY")=99992Z22296
```

2.1.5 POSTALB^XIPUTIL(): Active ZIP Codes

Reference Type: Supported

Category: Address Hygiene

ICR #: 3618

Description: The POSTALB^XIPUTIL 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 United States Postal Service (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 the COUNTY CODE (#5.13) file.
- **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 PREFERRED (DEFAULT) city key.
- **XIP(*n*,"STATE")**—State associated with this **pcode**.
- **XIP(*n*,"STATE POINTER")**—Pointer to the state in the STATE (#5) file.
- **XIP(*n*,"UNIQUE KEY")**—Unique lookup value.
- **XIP("ERROR")**—Errors encountered during lookup.

2.1.5.1 Example

Figure 7: POSTALB^XIPUTIL API—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 CHEL
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: Developer Tools

3.1 Overview

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 APIs. The SETUP^XQALERT API creates 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 system administrators 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 have sent an alert, one way you can confirm that the alert was sent is to use the VA FileMan **Inquire to File Entries** [DIINQUIRE] option, and examine the entry in the ALERT (#8992) file for the users to whom you sent the alert.

Figure 8: Alerts—Creating an Alert for a User (e.g., #14)

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

Figure 9: Alerts—Checking that the Alert was Sent

```
>D Q^DI

Select OPTION: INQ <Enter> UIRE TO FILE ENTRIES

OUTPUT FROM WHAT FILE: ALERT
Select ALERT RECIPIENT: `14 <Enter> XUUSER,14
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: XUUSER,15
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
```

3.2 Package Identifier vs. Alert Identifier

3.2.1 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.

3.2.2 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.
- **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 [#8992.1] file).

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).

3.3 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 (#2) file.
- **notificationcode** 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.

3.4 Glossary of Terms for Alerts

Table 2: Alerts—Related Terms and Definitions

Term	Definition
ALERTS	An alert notifies one or more users of a matter requiring immediate attention. Alerts function as brief notices that are distinct from mail messages or triggered bulletins. Alerts are designed to provide interactive notification of pending computing activities (e.g., the need to reorder supplies or review a patient's clinical test results). Along with the alert message is an indication that the **View Alerts** [XQALERT] common option should be chosen to take further action. An alert includes any specifications made by the developer 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 (#8992) file.
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 developer 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

Term	Definition
	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 developer 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.

3.5 Application Programming Interface (API)

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

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

Reference Type: Supported

Category: Alerts

ICR #: 2788

Description: The AHISTORY^XQALBUTLAPI returns information from the ALERT TRACKING (#8992.1) file for alerts with the **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 of 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 (#8992.1) file is returned descendent from the specified **root**.

NOTE: A more user (developer) friendly call would be the ALERTDAT^XQALBUTL(): Get Alert Tracking File Information 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 (#8992.1) file.

3.5.1.1 Example

Figure 10 illustrates the use of the AHISTORY^XQALBUTL API and the format of the data returned.

Figure 10: AHISTORY^XQALBUTL API—Example: Sample Use and Format of 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)=20^2990212.112954^2990212.145609^2990212.145621^2990212.14
5621
XXXROOT(20,"B",20,1)=
```

Figure 11 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 (#8992.1) file:

Figure 11: AHISTORY^XQALBUTL API—Example: Basic Structure of Nodes Taken from the Global for this Entry as seen via a Global Map View of the ALERT TRACKING (#8992.1) File

```
^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(): Get Alert Tracking File Information 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.

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

Reference Type:	Supported	
Category:	Alerts	
ICR #:	2788	
Description:	The ALERTDAT^XQALBUTL API returns information from the ALERT TRACKING (#8992.1) file for alerts with the **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 of 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 (#8992.1) file descendent from the specified **root**. • **NULL**—If the specified alert is *not* present, the array root is returned with a **NULL** value.

3.5.2.1 Example

Figure 12: ALERTDAT^XQALBUTL API—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 provides the field names if they are desired.

3.5.3 DELSTAT^XQALBUTL(): Get Recipient Information and Alert Status

Reference Type:	Supported
Category:	Alerts
ICR #:	3197
Description:	The DELSTAT^XQALBUTL 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 **xqaid** value for generating an alert by a software application. This value identifies the most recent alert generated with this **xqaid** value and that alert generates 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 **KILL**ed prior to building the list. For example: • **DUZ^1**—If alert deleted. • **DUZ^0**—If alert *not* deleted.

3.5.3.1 Example

Figure 13: DELSTAT^XQALBUTL API—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:

Figure 14: DELSTAT^XQALBUTL API—Example: Sample 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
```

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

Reference Type: Supported

Category: Alerts

ICR #: 3010

Description: The NOTIPURG^XQALBUTL 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.

3.5.5 $$PENDING^XQALBUTL(): Pending Alerts for a User

Reference Type: Supported

Category: Alerts

ICR #: 2788

Description: The $$PENDING^XQALBUT 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 (#200) file for the desired user.

xqaid: (required) This is the value of 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: • **1—YES**, alert is pending. • **0—NO**, alert is *not* pending.

3.5.5.1 Examples

3.5.5.1.1 Example 1

Figure 15 is an example of an alert *not* pending:

Figure 15: $$PENDING^XQALBUTL API—Example 1

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

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

3.5.5.1.2 Example 2

Figure 16 is an example of an alert pending:

Figure 16: $$PENDING^XQALBUTL API—Example 2

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

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

3.5.6 $$PKGPEND^XQALBUTL(): Pending Alerts for a User in Specified Software

Reference Type: Supported

Category: Alerts

ICR #: 2788

Description: The $$PKGPEND^XQALBUTL 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 (#200) file 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 the user specified by the **xqauser** input parameter 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.

3.5.6.1 Examples

3.5.6.1.1 Example 1

Figure 17 is an example of an alert *not* pending:

Figure 17: $$PKGPEND^XQALBUTL API—Example 1

```
>S XQKG="XU"

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

3.5.6.1.2 Example 2

Figure 18 is an example of an alert pending (one or more):

Figure 18: $$PKGPEND^XQALBUTL API—Example 2

```
>S XQKG="XU"

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

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

Reference Type: Supported

Category: Alerts

ICR #: 3010

Description: The PTPURG^XQALBUTL 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 [#2] file) for which alerts are deleted.

Output: none.

3.5.8 RECIPURG^XQALBUTL(): Purge User Alerts

Reference Type: Supported

Category: Alerts

ICR #: 3010

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

Format: `RECIPURG^XQALBUTL(duz)`

Input Parameters: **duz**: (required) Internal Entry Number (IEN in the NEW PERSON [#200] file) of the user who received alerts is deleted.

Output: none.

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

Reference Type:	Supported	
Category:	Alerts	
ICR #:	2788	
Description:	The USERDATA^XQALBUTL API returns recipients of the alert with the **xqaid** input parameter as its alert ID from the ALERT TRACKING (#8992.1) file 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 of 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 (#200) file 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 (#8992.1) file descendent from the specified **root**. • **NULL**—If the specified alert is *not* present, the array root is returned with a **NULL** value.

3.5.9.1 Example

Figure 19: USERDATA^XQALBUTL API—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")=
```

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

Reference Type: Supported

Category: Alerts

ICR #: 2788

Description: The USERLIST^XQALBUTL API returns recipients of the alert with the **xqaid** input parameter as its alert ID from the ALERT TRACKING (#8992.1) file 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 of 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 (#8992.1) file descendent from the specified **root**. • **NULL**—If the specified alert is *not* present, the array **root** is returned with a **NULL** value.

3.5.10.1 Example

Figure 20: USERLIST^XQALBUTL API—Example

```
>D USERLIST^XQALBUTL(XQAID)

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

3.5.11 ACTION^XQALERT(): Process an Alert

Reference Type:	Supported	
Category:	Alerts	
ICR #:	10081	
Description:	The ACTION^XQALERT 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 [#8992] file). This contains three semicolon-delimited pieces: 1. Original software application identifier. 2. **DUZ** of the alert creator. 3. VA FileMan date and time the alert was created.
Output:	none.	

3.5.12 DELETE^XQALERT: Clear Obsolete Alerts

Reference Type:	Supported
Category:	Alerts
ICR #:	10081

Description: The DELETE^XQALERT 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, see the "Package Identifier vs. Alert Identifier" section.

Format:
```
DELETE^XQALERT
```

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables: **XQAID:** (required) Alert Identifier of the alert to delete. It *must* be a complete Alert Identifier, containing all three semicolon pieces:

1. The first semicolon piece (Package Identifier) *must* be in the same form as the alert creator defined it.
2. The second piece being the **DUZ** of the user who created the alert.
3. The third piece being the time the alert was created.

NOTE: The second and third pieces are defined by the Alert Handler.

XQAKILL: (optional) **XQAKILL** determines how the alert is deleted.

- 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** (i.e., the alert is deleted only from one user), regardless of how it is actually set.

Output: none**.**

3.5.13 DELETEA^XQALERT: Clear Obsolete Alerts

Reference Type: Supported

Category: Alerts

ICR #: 10081

Description: The DELETEA^XQALERT 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 variable (multiple alerts can potentially share the same software application identifier).

REF: For more information on software application identifiers, see the "Package Identifier vs. Alert Identifier" section in this section.

Format: `DELETEA^XQALERT`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables: **XQAID:** (required) All alerts whose software application identifier matches the value of this input parameter is deleted, for the alert recipients designated by the **XQAKILL** input variable.

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 does *not* delete matching alerts.

XQAKILL: (optional) **XQAKILL** determines how the alert is deleted. If **XQAKILL** is:

- **Undefined or Zero (0)**—The Alert Handler deletes matching alerts for all recipients.
- **Set to 1**—Alert Handler deletes matching alerts for the current user, as identified by **DUZ**.

  NOTE: Using a value of **1** only makes sense if the current user is also a recipient of the alert, however.

Output: none.

3.5.14 GETACT^XQALERT(): Return Alert Variables

Reference Type:	Supported
Category:	Alerts
ICR #:	10081
Description:	The GETACT^XQALERT API returns to the calling routine the required variables to act on a specific alert.
Format:	`GETACT^XQALERT(alertid)`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

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

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

Reference Type:	Supported
Category:	Alerts
ICR #:	10081

Description: The PATIENT^XQALERT API allows you to return an array of all alerts for a particular patient that are either:

- Open.
- 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 (#2) file.

REF: For information on CPRS conventions for the format of the Package Identifier, see the "Package Identifier vs. Alert Identifier" section.

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 [#2] file) 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.

In addition, 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.

3.5.16 SETUP^XQALERT: Send Alerts

Reference Type: Supported

Category: Alerts

ICR #: 10081

Description: The SETUP^XQALERT API sends alerts to users; however, the *preferred* API to use 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 this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

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 (#8992.1) file. 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 proximately **220** years).

XQACNDEL: (optional) Setting a value in the **XQACNDEL** variable prior to calling this API causes the CAN DELETE WITHOUT PROCESSING (#.1) field in the ALERT (#8992) file to be set. A value in this field indicates that the alert can be deleted by the user without having processed it.

XQADATA: (optional) Use this to store a software application-specific data string, in any format. It is 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 opens the correct application to process the alert within the GUI environment.

NOTE: 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 do *not* set **XQAID**, you are *not* 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, see the "Package Identifier vs. Alert Identifier" section.

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 navigates 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** [XQALERT DELETE OLD] option forwards alert to recipient's supervisor based on 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** [XQALERT DELETE OLD] 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 is displayed in a ScreenMan form within the roll and 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 and scroll environment.

Output: none.

3.5.16.1 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 any option specified in the **XQAOPT** or **XQAROU** input variables.
3. 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.
4. 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, 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. 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.

5. Finally, the Alert Handler cleans up by **KILL**ing **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.

3.5.16.2 Example

Figure 21: SETUP^XQALERT API—Example: Call to Send an Alert Sample

```
;send an alert
;assume DFN is for patient XUPATIENT,ONE
N
XQA,XQAARCH,XQADATA,XQAFLG,XQAGUID,XQAID,XQAMSG,XQAOPT,XQAROU,XQASUPV,XQAS
URO,
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 22: SETUP^XQALERT API—Example: Resulting Alert, from View Alerts Option

```
Select Systems Manager Menu Option: "VA

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

3.5.17 $$SETUP1^XQALERT: Send Alerts

Reference Type:	Supported
Category:	Alerts
ICR #:	10081
Description:	The $$SETUP1^XQALERT extrinsic function sends alerts to users. This is the *preferred* API rather than SETUP^XQALERT: Send Alerts API.

- 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 this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

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 (#8992.1) file. Default time period is **30** days. Users can specify a different number of days using this input variable. **NOTE:** Critical patient data, as part of medical records, should be retained for at least **65** years, which is **23,725** days. To retain information forever, a value of **100000** is *recommended* (good for about **273+** years). Sites may *not* have sufficient disk storage space to accommodate this need, however.
	XQACNDEL:	(optional) Setting a value in the **XQACNDEL** variable prior to calling this API causes the CAN DELETE WITHOUT PROCESSING (#.1) field in the ALERT (#8992) file to be set. A value in this field indicates that the alert can be deleted by the user *without* having processed it.
	XQADATA:	(optional) Use this variable to store a software application-specific data string, in any format. It is 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 opens the correct application to process the alert within the GUI environment.

NOTE: Currently, this functionality has *not* 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 do *not* set **XQAID**, you are *not* 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, see the "Package Identifier vs. Alert Identifier" section.

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 navigates 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.
XQAREVUE:	(optional) This variable sets the DAYS FOR BACKUP REVIEWER (#.15) field in the ALERTS (#8992) file. It *must* be an integer from **1** to **15**.
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) Supervisor forwarding. Number of days to wait before **Delete Old (>14d) Alerts** [XQALERT DELETE OLD] option forwards alert to recipient's supervisor, if unprocessed by recipient. Can be a number from **1** to **30**. Supervisor is determined from the recipient's NEW PERSON (#200) file entry pointer to the SERVICE/SECTION (#49) file, and then the entry (if any) in the pointed-to Service/Section's CHIEF field.
XQASURO:	(optional) Number of days to wait before **Delete Old (>14d) Alerts** [XQALERT DELETE OLD] 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 are displayed in a ScreenMan form within the roll-and-scroll environment. **NOTE:** It was also intended to be displayed within a text display box within the GUI

		environment. Currently, CPRS has *not* implemented this functionality, so it can only be viewed in the roll-and-scroll environment.
Output:	returns:	Returns: • **1**—The alert was sent successfully. • **0**—The alert was *not* sent successfully, in which case the **XQALERR** variable contains a text string indicating the reason that the alert was *not* sent.
Output Variables:	**XQALERR:**	Returns: • **NULL**—It the alert was sent successfully, this variable is **NULL**. • **Text String**—If the alert was *not* sent successfully, this variable contains a text string that indicates the reason that the alert was *not* sent.

3.5.17.1 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 any option specified 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, the alert is deleted for the current user only.
   - To prevent the alert from being deleted, **KILL XQAKILL** during Step 2. 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. 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 **KILL**ing **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.

3.5.17.2 Example

Figure 23: $$SETUP1^XQALERT API—Example: Call to Send an Alert Sample

```
;send an alert
;assume DFN is for patient XUPATIENT,ONE
N
XQA,XQAARCH,XQADATA,XQAFLG,XQAGUID,XQAID,XQAMSG,XQAOPT,XQAROU,XQASUPV,XQAS
URO,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."
S VAR=$$SETUP1^XQALERT I 'XQALERR W !,"ERROR IN ALERT: ",XQALERR
Q
```

Figure 24: $$SETUP1^XQALERT API—Example: Resulting Alert, from View Alerts Option

```
Select Systems Manager Menu Option: "VA

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

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

Reference Type: Supported

Category: Alerts

ICR #: 10081

Description: The USER^XQALERT 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 do *not* pass a number, it uses the current user's **DUZ**.

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

In addition, 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.

3.5.18.1 Example

Figure 25: USER^XQALERT API—Example

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

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

3.5.19 FORWARD^XQALFWD(): Forward Alerts

Reference Type:	Supported
Category:	Alerts
ICR #:	3009
Description:	The FORWARD^XQALFWD 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 [#8992.01,.02] field in the ALERT DATE/TIME [#8992.01,.01] Multiple field of the current user's entry in the ALERT [#8992] file). 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 [#200] file). You do *not* need to precede the user's IEN with a grave accent (`). 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 (#3.5) file. You can specify the device either by name or by Internal Entry Number (IEN). If specifying by IEN, precede the IEN with an accent grave (e.g., `202).

type: (required) Indicates the method of forwarding desired. The options are the single characters:

- **A**—Forward as an Alert.
- **M**—Forward as a Mail Message.
- **P**—Print a copy of the alert.

If the value passed is *not* **A**, **M**, or **P**, then no action is taken.

comment: (optional) A character string to use as a comment to accompany the alert when it is forwarded.

Output: none.

3.5.19.1 Example

Figure 26: FORWARD^XQALFWD API—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(A6AALRT(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
```

3.5.20 $$CURRSURO^XQALSURO(): Get Current Surrogate for Alerts

Reference Type:	Supported	
Category:	Alerts	
ICR #:	2790	
Description:	The $$CURRSURO^XQALSURO extrinsic function obtains the current surrogate for alerts (if any) for the user with **DUZ** specified by the **xqauser** input parameter.	
Format:	`$$CURRSURO^XQALSURO(xqauser)`	
Input Parameters:	**xqauser**:	(required) This is the Internal Entry Number (IEN, **DUZ** value) in the NEW PERSON (#200) file 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.

3.5.21 $$GETSURO^XQALSURO(): Get Current Surrogate Information

Reference Type:	Supported	
Category:	Alerts	
ICR #:	3213	

Description: The $$GETSURO^XQALSURO 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 (#200) file:

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

If there is no surrogate, the result is:

```
^^^
```

If either of the start or end dates and times is *not* specified, a **NULL** value is returned for that piece of the return string.

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

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

Input Parameters: **xqauser**: (required) This is the Internal Entry Number (IEN) in the NEW PERSON (#200) file 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 (#200) file.
- **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.

3.5.21.1 Example

Figure 27: $$GETSURO^XQALSURO API—Example

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

>W X

2327^XUUSER,FOUR^3000929.1630^3001006.0800
```

This indicates that user **#2327** (Four Xuuser) becomes active as surrogate at **4:30 PM 9/29/00** and remains surrogate until **8:00 am** on **10/06/00**.

3.5.22 REMVSURO^XQALSURO(): Remove Surrogates for Alerts

Reference Type:	Supported
Category:	Alerts
ICR #:	2790
Description:	The REMVSURO^XQALSURO API removes any surrogates for alerts for the specified user.
Format:	`REMVSURO^XQALSURO(xqauser[,.xqalsuro][,.xqalstrt])`

Input Parameters:	**xqauser**:	(required) This is the Internal Entry Number (IEN, **DUZ** value) in the NEW PERSON (#200) file for the specified user.
	xqalsuro:	(optional) IEN of user in the NEW PERSON (#200) file. If passed, only the user who is passed is removed from the list of surrogates. If *not* passed, only the current surrogate is removed (if any).
	xqalstrt:	(optional) If passed, the surrogate is removed only from the start date indicated. If *not* passed, the surrogate is removed starting from the date of the current surrogate (if any). If there is no current surrogate, no entries are removed.
Output:	none.	

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

Reference Type:	Supported
Category:	Alerts
ICR #:	3213

Description: The SETSURO1^XQALSURO API establishes a surrogate for alerts. It should be used instead of the (obsolete) 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:

- **IEN (value > 0; True)**—Surrogate was created successfully.
- **Text String (False)**—Text explaining why the surrogate was *not* created.

Previously, the (obsolete) 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 (obsoelte) SETSURO^XQALSURO API (i.e., ICR #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 [#200] file) for which the surrogate should act in receiving alerts.
xqalsuro:	(required) Surrogate's **DUZ** number (i.e., Internal Entry Number in the NEW PERSON [#200] file) for the user who receives and processes 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: • **IEN (value > 0; True)**—Surrogate was created successfully. • **Text String (False)**—Text explaining why the surrogate was *not* created.

3.5.23.1 Example

Figure 28: SETSURO1^XQALSURO—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
```

3.5.24 SUROFOR^XQALSURO(): Return a Surrogate's List of Users

Reference Type:	Supported	
Category:	Alerts	
ICR #:	3213	
Description:	The SUROFOR^XQALSURO API returns a list of users for which the user, as defined by the **xqauser** input parameter, is acting as a surrogate.	
Format:	`SUROFOR^XQALSURO(xqauser,.xqalist)`	
Input Parameters:	**xqauser**:	(required) This is the Internal Entry Number (IEN, **DUZ** value) in the NEW PERSON (#200) file for the specified user.
	xqalist:	(required) Passed by reference; it contains the name of the output array.

Output:	**xqalist**:	The output contains the list of users for whom the specified user is currently acting as a surrogate. The data in the list includes the: • User's internal entry number (**DUZ**). • User's name. • Start and end dates for the surrogate period. Set to a number equal to the count of the total number of surrogates returned in the list: **XQALIST(*n*)** Where ***n*** is a sequential integer starting with **1**. Each entry in the array contains: **IEN^Name^Start Date/Time^End Date/Time**

3.5.24.1 Example

Figure 29: SUROFOR^XQALSURO API—Example

```
>S XQAUSER=DUZ
>D SUROFOR^XQALSURO(XQAUSER,.USERLIST)
```

Returns:

Figure 30: SUROFOR^XQALSURO API—Example: Returns

```
USERLIST=count
USERLIST(1)=IEN2^NEWPERSON,USER2^STARTDATETIME^ENDDATETIME
USERLIST(2)=3^NAME,USER3^3050407.1227^3050406

>ZW USERLIST
OUTPUT=2
OUTPUT(1)="5206652^PERSON,FIRST^3071113.141547^3071113.142"
OUTPUT(2)="5206656^PERSON,SECOND^3071114^3071114.08"
```

3.5.25 SUROLIST^XQALSURO(): List Surrogates for a User

Reference Type:	Supported
Category:	Alerts
ICR #:	3213

Description: The SUROLIST^XQALSURO API returns a list of current or future surrogates for the user that is defined by the **xqauser** input parameter. It also sets the following surrogate fields in the ALERT (#8992) file if there is a current surrogate for this user:

- SURROGATE FOR ALERTS (#.02)
- SURROGATE START DATE/TIME (#.03)
- SURROGATE END DATE/TIME (#.04)

Format: `SUROLIST^XQALSURO(xqauser,.xqalist)`

Input Parameters:	**xqauser**:	(required) This is the Internal Entry Number (IEN, **DUZ** value) in the NEW PERSON (#200) file for the specified user.
	xqalist:	(required) Passed by reference; it contains the name of the output array.
Output:	**xqalist**:	The output contains the list of current and future surrogates for the specified user. The data in the list includes the following:

- User's internal entry number (**DUZ**).
- User's name.
- Start and end dates for the surrogate period.

Set to a number equal to the count of the total number of surrogates returned in the list:

XQALIST(*n*)

Where ***n*** is a sequential integer starting with **1**. Each entry in the array contains:

IEN^Name^Start Date/Time^End Date/Time

3.5.25.1 Example

Figure 31: SUROLIST^XQALSURO API—Example

```
>D SUROLIST^XQALSURO(duz,.output)

>ZW OUTPUT
OUTPUT=2
OUTPUT(1)="5206652^PERSON,FIRST^3071113.141547^3071113.142"
OUTPUT(2)="5206656^PERSON,SECOND^3071114^3071114.08"
```

4 Common Services: Developer Tools

4.1 Application Programming Interface (API)

The following are Common Services APIs available for developers. These APIs are described below.

4.1.1 $$IEN^XUPS(): Get IEN Using VPID in File #200

Reference Type: Supported

Category: Common Services

ICR #: 4574

Description: The $$IEN^XUPS extrinsic function accepts the VA Person ID (VPID) of an entry in the NEW PERSON (#200) file and returns the Internal Entry Number (IEN)/**DUZ**.

CAUTION: VPID has *not* been fully implemented in the VA. VPID was the user identifier within the *canceled* Enterprise Single Sign-On (ESSO) project. The current Identity and Access Management (IAM) 2-Factor Authentication (2FA) project uses Security ID (SecID) as the unique identifier. VPID APIs and fields will be deprecated in a *future* Kernel patch. Developers are encouraged to remove all references to these APIs in their code.

NOTE: This API was released with Kernel Patch XU*8.0*309.

Format: `$$IEN^XUPS(vpid)`

Input Parameters:	**vpid**:	(required) The VA Person ID (VPID).
Output:	returns:	Returns the Internal Entry Number (IEN)/**DUZ** of the NEW PERSON (#200) file.

4.1.2 $$VPID^XUPS(): Get VPID Using IEN in File #200

Reference Type: Supported

Category: Common Services

ICR #: 4574

Description: The $$VPID^XUPS extrinsic function accepts the internal entry number (IEN)/**DUZ** of an entry in the NEW PERSON (#200) file and returns the VA Person ID (VPID) for the selected user.

NOTE: This API was released with Kernel Patch XU*8.0*309.

CAUTION: VPID has *not* been fully implemented in the VA. VPID was the user identifier within the *canceled* Enterprise Single Sign-On (ESSO) project. The current Identity and Access Management (IAM) 2-Factor Authentication (2FA) project uses Security ID (SecID) as the unique identifier. VPID APIs and fields will be deprecated in a *future* Kernel patch. Developers are encouraged to remove all references to these APIs in their code.

Format: `$$VPID^XUPS(duz)`

Input Parameters:	**duz**:	(required) The Internal Entry Number (IEN) in the NEW PERSON (#200) file.
Output:	returns:	Returns the VA Person ID (VPID) for the entry found in the NEW PERSON (#200) file.

4.1.3 EN1^XUPSQRY(): Query New Person File

Reference Type: Controlled Subscription

Category: Common Services

ICR #: 4575

Description: The **XUPS PERSONQUERY** RPC uses the EN1^XUPSQRY API. This API provides the functionality to query the NEW PERSON (#200) file. The calling application can query the NEW PERSON (#200) file by using either the Security ID (**SECID**) of the requested entry or part/all of a last name. Other optional parameters can be passed to the call as additional filters.

NOTE: This API was released with Kernel Patch XU*8.0*325.

Format:

```
EN1^XUPSQRY(result,xupsecid,xupslnam[,xupsfnam][,xupsssn][,
xupsprov][,xupsstn][,xupsmnm][,xupsdate])
```

Input Parameters:	**result**:	(required) Name of the subscripted return array. In every API that is used as an RPC, the first parameter is the return array.
	xupsecid:	(required) This parameter contains the **SECID** for the requested user. Either the **SECID** or last name is required.
	xupslnam:	(required) This parameter contains all or part of a last name. A last name or **SECID** are required input variables.
	xupsfnam:	(optional) This parameter is set to **NULL** or the full or partial first name.
	xupsssn:	(optional) This parameter is set to **NULL** or contains the **9** digits of the Social Security Number (SSN).
	xupsprov:	(optional) This parameter is set to **NULL** or **P**. If set to **P**, it screens for providers (person with active user class).
	xupsstn:	(optional) This parameter is set to **NULL** or the Station Number.
	xupsmnm:	(optional) This parameter is set to the maximum number of entries (**1-50**) to be returned. Defaults to **50**.
	xupsdate:	(optional) This parameter contains the date used to determine if person class is active. Defaults to current date.
Output Parameters:	**result()**:	Returns a subscripted output array of the input value/subscripted array (i.e., list) with the following possible values shown: • **^TMP($J,“XUPSQRY”,1)—1** if found, **0** if *not* found • **^TMP($J,“XUPSQRY”,n,0)—** VPID^IEN^LastName~First Name~Middle Name^SSN^DOB^SEX^ • **^TMP($J,“XUPSQRY”,n,1)—**Provider Type^ • **^TMP($J,“XUPSQRY”,n,2)—**Provider Classification^ • **^TMP($J,“XUPSQRY”,n,3)—**Provider Area of Specialization^

- **^TMP($J,“XUPSQRY”,n,4)**—VA CODE^X12 CODE^Specialty Code^end-of-record character “|”|

5 Data Security: Developer Tools

5.1 Overview

Developers can use data security tools to protect information from unauthorized viewing.

Federal Information Processing Standards Publication 180-4 (FIPS PUB 180-4) specifies secure hash algorithms for computing a condensed representation of electronic data (message). The hash algorithms specified in this Standard are called secure because, for a given algorithm, it is computationally infeasible to find either of the following:

1. A message that corresponds to a given message digest.
2. Two different messages that produce the same message digest.

Any change to a message, with a very high probability, results in a different message digest.

Released with Kernel Patch XU*8.0*655, the Secure Hash Algorithm (SHA) is a family of one-way cryptographic hash functions. The input data is often called the message, and the hash value is often called the message digest. Cryptographic hash functions are used in the following:

- Digital signatures.
- Message authentication codes.
- Other forms of authentication.

They can also be used to:

- Detect duplicate data.
- Uniquely identify files.
- Detect accidental data corruption as checksums.

In information security contexts, cryptographic hash values are sometimes called digital fingerprints.

Additional SHA utilities were released with Kernel Patch XU*8.0*657. These utilities include hashes for the following:

- Specified VA FileMan file or subfile.
- Specified host file.
- Specified routine.
- Message that is too long to be passes as a single string.
- Message that can be passed in a single string.

Encryption is the process of using a mathematical algorithm to transform information so that it becomes unreadable. The information is then available only to those who possess the key that can be used for decryption. Patch XU*8.0*655 distributed several encryption utilities, including:

- **AES (Advanced Encryption Standard)** encryption.
- **RSA (Rivest–Shamir–Adleman)** encryption.

Binary-to-text encoding schemes are used to represent binary data in an ASCII string format. They are commonly used when there is a need to store or transfer data over media that is designed to deal with textual data to ensure that the data remains intact *without* modification during transport. Patch XU*8.0*655 also included Base 64 encoding and decoding utilities.

5.2 Application Programming Interface (API)

Several APIs for hashing, encoding/decoding, or encryption/decryption of input of various formats are available for developers to work with data security. These APIs are supported under Integration Control Registration (ICR) #6189 and are described below.

5.2.1 $$FILE^XLFSHAN(): Returns SHA Hash for Specified FileMan File or Subfile Entry

Reference Type: Supported

Category: Data Security

ICR #: 6157

Description: The $$FILE^XLFSHAN extrinsic function returns the **SHA** hash for a specified file entry. It uses the VA FileMan GETS^DIQ API to extract the data from the file. The input parameters match the input parameters for GETS^DIQ.

NOTE: The *VA FileMan Developer's Guide* is located on the VDL at: https://www.va.gov/vdl/application.asp?appid=5

NOTE: This API was released with Kernel Patch XU*8.0*657.

Format: `$$FILE^XLFSHAN(hashlen,filenum,iens[,field][,flags])`

Input Parameters: **hashlen**: (required) The hash length in bits:

- 160 (SHA-1)
- 224 (SHA-224)
- 256 (SHA-256)
- 384 (SHA-384)
- 512 (SHA-512)

filenum:	(required) VA FileMan file or subfile number.
iens:	(required) Standard VA FileMan IENS indicating internal entry numbers, as documented in the *VA FileMan Developer's Guide*.
field:	(optional) Can be one of the following: • A single field number. • A list of field numbers, separated by semicolons. • A range of field numbers, in the form ***M***:***N;*** where ***M*** and ***N*** are the end points of the inclusive range. All field numbers within this range are retrieved. • Asterisk (*****) for all fields at the top-level (no sub-Multiple record). • Double asterisk (******) for all fields including all fields and data in sub-Multiple fields. • Field number of a multiple followed by an ***** to indicate all fields and records in the sub-Multiple for that field. If this parameter is *not* passed, it defaults to ******, which extracts all fields.
flags:	(optional) Flags to control processing. The possible values are: • **E**—Returns **E**xternal values in nodes ending with **E**. • **I**—Returns **I**nternal values in nodes ending with **I**; otherwise, external is returned. • **N**—Does *not* return **NULL** values. • **R**—**R**esolves field numbers to field names in target array subscripts. • **Z**—WORD-PROCESSING fields include **Z**ero nodes. • **A#**—**A**udit Trail is used to retrieve the value of "**FIELD**" at a particular point in time. # is a date/time in VA FileMan internal format

(e.g., 3021015.08). The values retrieved are the (audited) values of the fields as of that date/time.

Output: returns: Returns:

- **SHA hash**—If successful.
- **Zero (0)**—If the file could *not* be opened or found.
- **-1**—If an error occurs.

5.2.1.1 Example

Figure 32: $$FILE^XLFSHAN API—Example

```
>W $$FILE^XLFSHAN(512,200,"10000000407,")
8FE96A435D69989EFC30FC852260990BECB030247657B9CA1CDB9D103097B51792648254 77
0D88E292592CC06C36D22C3E502F790050B8ADBB035C89F59FB8A7
```

5.2.2 $$GLOBAL^XLFSHAN(): Returns SHA Hash for a Global

Reference Type: Supported

Category: Data Security

ICR #: 6157

Description: The $$GLOBAL^XLFSHAN extrinsic function returns the **SHA** hash of a specified global, in contrast with the $$FILE^XLFSHAN API, which returns the hash for a particular entry in a global.

NOTE: This API was released with Kernel Patch XU*8.0*657.

Format: `$$GLOBAL^XLFSHAN(hashlen,filenum,dataonly)`

Input Parameters: **hashlen**: (required) The hash length in bits:

- 160 (SHA-1)
- 224 (SHA-224)
- 256 (SHA-256)
- 384 (SHA-384)
- 512 (SHA-512)

	filenum:	(required) VA FileMan file number.
	dataonly:	(required) Scope of the hash: • **0**—Global location of the data is to be included in the hash computation. • **1**—Hash is computed only for the data.
Output:	returns:	Returns: • **SHA hash**—If successful. • **Zero (0)**—If there is an error.

5.2.2.1 Example

Figure 33: $$GLOBAL^XLFSHAN API—Example

```
>W $$GLOBAL^XLFSHAN(256,200,0)
714CE00DE20E30700229F95F69DBAE34262CF30576EA03852CFBE0D0DC2BE611
```

5.2.3 $$HOSTFILE^XLFSHAN(): Returns SHA Hash for Specified Host File

Reference Type: Supported

Category: Data Security

ICR #: 6157

Description: The $$HOSTFILE^XLFSHAN extrinsic function returns the **SHA** hash for a specified host file. It uses the $$FTG^%ZISH API to load the host file for processing.

NOTE: This API was released with Kernel Patch XU*8.0*657.

Format: `$$HOSTFILE^XLFSHAN(hashlen,path,filename)`

Input Parameters: **hashlen**: (required) The hash length in bits:

- 160 (SHA-1)
- 224 (SHA-224)
- 256 (SHA-256)
- 384 (SHA-384)
- 512 (SHA-512)

	path:	(required) Full path, up to but *not* including the filename.
	filename:	(required) Name of the file.
Output:	returns:	Returns: • **SHA hash**—If successful. • **Zero (0)**—If the host file could *not* be opened/found.

5.2.3.1 Example

Figure 34: $$HOSTFILE^XLFSHAN API—Example

```
>W $$HOSTFILE^XLFSHAN(160,"Z:\Cache2014\","cache.cpf")
F11F3595604296A1F8BCF13AA7F2744FB9EB1675
```

5.2.4 $$LSHAN^XLFSHAN(): Returns SHA Hash for a Long Message

Reference Type: Supported

Category: Data Security

ICR #: 6157

Description: The $$LSHAN^XLFSHAN extrinsic function returns the **SHA** hash of a message that is too long to be passed as a single string. The message is passed in **^TMP($J,MSUB)**. The message should be broken into blocks that are exactly **64** bytes/characters long except for the last one. **^TMP($J,MSG,N)** is the *N*th block of the message; where *N* runs from **1** to **NBLOCKS**.

NOTE: This API was released with Kernel Patch XU*8.0*657.

Format: `$$LSHAN^XLFSHAN(hashlen,msub,nblocks)`

Input Parameters:	**hashlen**:	(required) The hash length in bits: • 160 (SHA-1) • 224 (SHA-224) • 256 (SHA-256) • 384 (SHA-384) • 512 (SHA-512)
	msub:	(required) The **^TMP($J,msub)** subscript in which the message is passed.
	nblocks:	(required) The number of blocks in the message.
Output:	returns:	Returns: • **SHA hash**—If successful. • **Zero (0)**—If there is an error.

5.2.4.1 Example

Figure 35: $$LSHAN^XLFSHAN API—Example

```
>S ^TMP($J,"MSG",1)= "test line one"
>S ^TMP($J,"MSG",2)= "test line two"
>W $$LSHAN^XLFSHAN(224,"MSG",2)
42E2C4B559757087BFA5834F43C2C50740984766910C1B4EEC79A350
```

5.2.5 $$ROUTINE^XLFSHAN(): Returns SHA Hash for a VistA Routine

Reference Type: Supported

Category: Data Security

ICR #: 6157

Description: The $$ROUTINE^XLFSHAN extrinsic function returns the **SHA** hash for a specified VistA routine.

NOTE: This API was released with Kernel Patch XU*8.0*657.

Format: `$$ROUTINE^XLFSHAN(hashlen,routine)`

Input Parameters:	**hashlen**:	(required) The hash length in bits: • 160 (SHA-1) • 224 (SHA-224) • 256 (SHA-256) • 384 (SHA-384) • 512 (SHA-512)
	routine:	(required) The name of the routine.
Output:	returns:	Returns: • **SHA hash**—If successful. • **Zero (0)**—If the routine could *not* be opened/found.

5.2.5.1 Example

Figure 36: $$ROUTINE^XLFSHAN API—Example

```
>W $$ROUTINE^XLFSHAN(384,"XUCERT")
54BA28936CE7CEC515305AE4BBD07FC4FD7620ACF0EAD0AF6A9E5BFBEEF24794DA414C0C33A6
C0C3B90005D70A2BFE4D
```

5.2.6 $$SHAN^XLFSHAN(): Returns SHA Hash for a Message

Reference Type: Supported

Category: Data Security

ICR #: 6157

Description: The $$SHAN^XLFSHAN extrinsic function returns the **SHA** hash of a message.

NOTE: This API was released with Kernel Patch XU*8.0*657.

Format: `$$SHAN^XLFSHAN(hashlen,message)`

Input Parameters:	**hashlen**:	(required) The hash length in bits: • 160 (SHA-1) • 224 (SHA-224) • 256 (SHA-256) • 384 (SHA-384) • 512 (SHA-512)
	message:	(required) The message string.
Output:	returns:	Returns: • **SHA hash**—If successful. • **Zero (0)**—If there is an error.

5.2.6.1 Example

Figure 37: $$SHAN^XLFSHAN API—Example

```
>W $$SHAN^XLFSHAN(256,"this is a test")
2E99758548972A8E8822AD47FA1017FF72F06F3FF6A016851F45C398732BC50C
```

5.2.7 $$AESDECR^XUSHSH(): Returns Plaintext String Value for AES Encrypted Ciphertext Entry

Reference Type: Supported

Category: Data Security

ICR #: 6189

Description: The $$AESDECR^XUSHSH extrinsic function returns the string value of an Advanced Encryption Standard (AES) encrypted ciphertext entry. AES is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001.

NOTE: This API was released with Kernel Patch XU*8.0*655.

Format: `$$AESDECR^XUSHSH(text,key[,iv])`

Input Parameters:	**text**:	(required) The ciphertext string to be decrypted.
	key:	(required) The input key material **16**, **24**, or **32** characters long.
	iv:	(optional) The initialization vector. If this argument is present, it *must* be **16** characters long.
Output:	returns:	Returns the plaintext value of the AES encrypted ciphertext entry in the **text** input parameter.

5.2.7.1 Example

Figure 38: $$AESDECR^XUSHSH API—Example

```
>W
$$AESDECR^XUSHSH($$B64DECD^XUSHSH("STbvalBtOxy754eRo15Bkg=="),"Encr4pt10nK
3y")
This is a test
```

5.2.8 $$AESENCR^XUSHSH(): Returns AES Encrypted Ciphertext for String Entry

Reference Type: Supported

Category: Data Security

ICR #: 6189

Description: The $$AESENCR^XUSHSH extrinsic function returns the Advanced Encryption Standard (AES) encrypted ciphertext for a string entry. AES is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001.

NOTE: This API was released with Kernel Patch XU*8.0*655.

Format: `$$AESENCR^XUSHSH(text,key[,iv])`

Input Parameters:	**text**:	(required) The plaintext string to be encrypted.
	key:	(required) The input key material **16**, **24**, or **32** characters long.
	iv:	(optional) The initialization vector. If this argument is present, it *must* be **16** characters long.
Output:	returns:	Returns the AES encrypted ciphertext for the string entry in the **text** input parameter.

5.2.8.1 Example

NOTE: The AES encryption API returns Unicode ciphertext, which does *not* properly display on an ASCII roll-and-scroll terminal; so the example demonstrated output is **Base64** encoded before display.

Figure 39: $$AESENCR^XUSHSH API—Example

```
>W $$B64ENCD^XUSHSH($$AESENCR^XUSHSH("This is a test","Encr4pt10nK3y"))
STbvalBtOxy754eRo15Bkg==
```

5.2.9 $$B64DECD ^XUSHSH(): Returns Decoded Value for a Base64 String Entry

Reference Type:	Supported	
Category:	Data Security	
ICR #:	6189	
Description:	The $$B64DECD ^XUSHSH extrinsic function returns the decoded value for a **Base64** string entry. **Base64** is a binary-to-text encoding scheme that represents binary data in an ASCII string format by translating it into a **radix-64** representation. **Base64** encoding is commonly used when there is a need to encode binary data that needs to be stored and transferred over media that is designed to deal with textual data. **NOTE:** This API was released with Kernel Patch XU*8.0*655.	
Format:	`$$B64DECD^XUSHSH(x)`	
Input Parameters:	**x**:	(required) The string to be decoded.
Output:	returns:	Returns the decoded value for the **Base64** input parameter.

5.2.9.1 Example

Figure 40: $$B64DECD ^XUSHSH API—Example

```
>W $$B64DECD^XUSHSH("VGhpcyBpcyBhIHRlc3Q=")
This is a test
```

5.2.10 $$B64ENCD^XUSHSH(): Returns Base64 Encoded Value for a String Entry

Reference Type:	Supported
Category:	Data Security
ICR #:	6189

Description: The $$B64ENCD^XUSHSH extrinsic function returns the **Base64** encoded value for a string entry. **Base64** is a binary-to-text encoding scheme that represents binary data in an ASCII string format by translating it into a **radix-64** representation. **Base64** encoding is commonly used when there is a need to encode binary data that needs to be stored and transferred over media that is designed to deal with textual data.

NOTE: This API was released with Kernel Patch XU*8.0*655.

Format:	`$$B64ENCD^XUSHSH(x)`	
Input Parameters:	**x**:	(required) The string to be encoded.
Output:	returns:	Returns the **Base64** encoded value of the input parameter.

5.2.10.1 Example

Figure 41: $$B64ENCD^XUSHSH API—Example

```
>W $$B64ENCD^XUSHSH("This is a test")
VGhpcyBpcyBhIHRlc3Q=
```

5.2.11 $$RSADECR^XUSHSH(): Returns Plaintext String Value for RSA Encrypted Ciphertext Entry

Reference Type:	Supported	
Category:	Data Security	
ICR #:	6189	
Description:	The $$RSADECR^XUSHSH extrinsic function returns the plaintext string value for an **RSA** encrypted ciphertext entry. **RSA** is a public-key encryption system that is widely used for secure data transmission. The encryption key is public and differs from the decryption key, which is kept secret. **NOTE:** This API was released with Kernel Patch XU*8.0*655.	
Format:	`$$RSADECR^XUSHSH(text,key[,pwd][,enc])`	
Input Parameters:	**text**:	(required) The **RSA** encrypted ciphertext string to be decrypted.
	key:	(required) The **RSA** private key corresponding to the **RSA** public key that was used for encryption, Privacy Enhanced Mail (**PEM**) encoded.
	pwd:	(optional) The private key password.
	enc:	(optional) Encoding - Public-Key Cryptography Standards (PKCS) #1 v2.1 encoding method: • **1**—Optimal Asymmetric Encryption Padding (OAEP; default). • **2**—PKCS 1-v1_5.
Output:	returns:	Returns the plaintext string value for the **RSA** encrypted ciphertext **text** input parameter.

5.2.11.1 Example

NOTE: "**hgwds**" is the alias of a certificate installed in Caché through the management portal for demonstration purposes. The private key used to decrypt the ciphertext was *not* available, so that function is *not* demonstrated here.

5.2.12 $$RSAENCR^XUSHSH(): Returns RSA Encrypted Ciphertext for String Entry

Reference Type:	Supported
Category:	Data Security
ICR #:	6189
Description:	The $$RSAENCR^XUSHSH extrinsic function returns the **RSA** encrypted ciphertext for a string entry. **RSA** is a public-key encryption system that is widely used for secure data transmission. The encryption key is public and differs from the decryption key, which is kept secret.

NOTE: This API was released with Kernel Patch XU*8.0*655.

Format: `$$RSAENCR^XUSHSH(text,cert[,cafile][,crlfile][,enc])`

Input Parameters:	**text**:	(required) The plaintext string to be encrypted.
	cert:	(required) An **X.509** certificate containing the **RSA** public key to be used for encryption, in **PEM** encoded or binary Distinguished Encoding Rules (**DER**) format. The length of the plaintext *cannot* be greater than the length of the modulus of the **RSA** public key contained in the certificate minus **42** bytes.
	cafile:	(optional) The name of a file containing the trusted Certificate Authority **X.509** Certificates in **PEM**-encoded format, one of which was used to sign the certificate.
	crlfile:	(optional) The name of a file containing **X.509** Certificate Revocation Lists in **PEM**-encoded format that should be checked to verify the status of the certificate.
	enc:	(optional) Encoding - PKCS #1 v2.1 encoding method: • **1**—Optimal Asymmetric Encryption Padding (OAEP; default). • **2**—PKCS 1-v1_5.
Output:	returns:	Returns the **RSA** encrypted ciphertext value of the text input parameter.

5.2.12.1 Example

NOTE: The **RSA** encryption API returns Unicode ciphertext, which does *not* properly display on an ASCII roll-and-scroll terminal; so the example demonstrated output is **Base64** encoded before display.

Figure 42: $$RSAENCR^XUSHSH API—Example

```
>S TEXT="This is a test"

>S CREDSET=##class(%SYS.X509Credentials).GetByAlias("hgwds")

>S CERT=CREDSET.Certificate

>W $$B64ENCD^XUSHSH($$RSAENCR^XUSHSH(TEXT,CERT,,,1))
PbFxIUBA+Mu5F4rtFHVJOusYfqFOm99eyhp3jYTBBIteSMYE1J+dHFqSePGtGXInBIy2f6gVxTvf
WQyy8Le92tbqADftPsGKlBISaA1O3v2r0oxYQkwR6FPub3y/r92b6l/StwAzImMF9EP6vqLt/IOK
1eu4UD+sT5qesGB9zgAmEfQgitT3qhXZJZUAbIi//NZbLiWVtGF+99GSa77VyMXkWqKiSVZZHCLG
yUGgPn8SwFXEsZNs+STuFaQn6jialrn04NOuaqXEDSZu1qGpn5WE3fNcWeLZE5sXJX8rG0uW5R/O
lx/Xlk3L2GhqELELsgzJY0RG5fp8wT58cJKqwQ==
```

5.2.13 $$SHAHASH^XUSHSH(): Returns SHA Hash for a String Entry

Reference Type: Supported

Category: Data Security

ICR #: 6189

Description: The $$SHAHASH^XUSHSH extrinsic function returns the Secure Hash Algorithm (**SHA**) hash for a string entry. It uses an input variable to specify the length in bits of the desired hash.

NOTE: This API was released with Kernel Patch XU*8.0*655.

Format: `$$SHAHASH^XUSHSH(n,x[,flag])`

Input Parameters: **n:** (required) Length in bits of the desired hash:

- 160 (SHA-1)
- 224 (SHA-224)
- 256 (SHA-256)
- 384 (SHA-384)
- 512 (SHA-512)

	x:	(required) String to be hashed.
	flag:	(optional) Flag to control format of hash: • **H**—Hexadecimal (default) • **B—Base64** Encoded
Output:	returns:	Returns **SHA** hash for a string entry.

5.2.13.1 Examples

5.2.13.1.1 Example 1

Figure 43: $$SHAHASH^XUSHSH API—Example 1

```
>W $$SHAHASH^XUSHSH(256,"This is a test")
C7BE1ED902FB8DD4D48997C6452F5D7E509FBCDBE2808B16BCF4EDCE4C07D14E
>
```

5.2.13.1.2 Example 2

Figure 44: $$SHAHASH^XUSHSH API—Example 1

```
>W $$SHAHASH^XUSHSH(256,"This is a test","B")
x74e2QL7jdTUiZfGRS9dflCfvNvigIsWvPTtzkwH0U4=
>
```

6 Device Handler: Developer Tools

6.1 Overview

The Device Handler provides a common user interface and developer API for using output devices. This section describes the Device Handler's developer 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.

NOTE: As of Kernel Patch XU*8.0*546 (and Informational Patch XU*8.0*556), Class 3 routines that are *not* written to permit queuing no longer output to devices where the QUEUING (#5.5) field in the DEVICE (#3.5) file is set to FORCED. Sites that have completed the Linux upgrade checklist, should have already addressed this issue.

REF: For more specific details, see Kernel Patches XU*8.0*546 and 556.

6.2 Application Programming Interface (API)

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

6.2.1 DEVICE^XUDHGUI(): GUI Device Lookup

Reference Type:	Supported
Category:	Device Handler
ICR #:	3771
Description:	The DEVICE^XUDHGUI 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. **NOTE:** This API was released 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**—Returns devices whose name starts with **P**. • **P***—Returns 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**—Up.
- **-1**—Down.

right_margin_range:(optional) This parameter specifies 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
```

6.2.1.1 Examples

6.2.1.1.1 Example 1

This example stores/displays a list of all devices that begin with **P** in an array (e.g., DEVICES), without passing a **direction** or **right_margin_range** parameter:

Figure 45: DEVICE^XUDHGUI API—Example 1: Store Devices

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

The DEVICES array displays the following results:

Figure 46: DEVICE^XUDHGUI API—Example 1: Display Sample Results

```
>ZW DEVICES
DEVICES(1)=358^P-MESSAGE-HFS^P-MESSAGE-HFS^HFS FILE=>MESSAGE^255^256
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^MWI WINDOW DOCUMENT BOX^80^256
```

6.2.1.1.2 Example 2

This example stores/displays 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**:

Figure 47: DEVICE^XUDHGUI API—Example 2: Store Devices

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

The DEVICES array displays the following results:

Figure 48: DEVICE^XUDHGUI API—Example 2: Display Sample Results

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

6.2.1.1.3 Example 3

This example stores/displays 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**:

Figure 49: DEVICE^XUDHGUI API—Example 3: Store Devices

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

The DEVICES array displays the following results:

Figure 50: DEVICE^XUDHGUI API—Example 3: Display Sample Results

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

6.2.1.1.4 Example 4

This example stores/displays 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:

Figure 51: DEVICE^XUDHGUI API—Example 4: Store Devices

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

The DEVICES array displays the following results:

Figure 52: DEVICE^XUDHGUI API—Example 4: Display Sample Results

```
>ZW DEVICES
DEVICES(1)=358^P-MESSAGE-HFS^P-MESSAGE-HFS^HFS FILE=>MESSAGE^255^256
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^MWI WINDOW DOCUMENT BOX^80^256
DEVICES(6)=202^C6_SDD_MX3 ROUTINE^ROUTINE <C6_SDD_MX3 ROUTINE>^Next to
USER2's Office^80^59
DEVICES(7)=428^SDD DUPLEX P10^SDD DUPLEX P10^SSD DUPLEX PRINTER NEXT TO
USER1^80^60
DEVICES(8)=429^SDD P10^SDD P10^Printer next to USER1.^80^60
DEVICES(9)=329^C6_SDD_MX3 P10^SS10 <C6_SDD_MX3 P10>^Near USER2's
Office^80^59
DEVICES(10)=330^C6_SDD_MX3 P12^SS12 <C6_SDD_MX3 P12>^Near USER2's
Office^96^57
DEVICES(11)=331^C6_SDD_MX3 P16^SS16 <C6_SDD_MX3 P16>^Near USER2's
Office^255^58
DEVICES(12)=349^C6_SDD_MX3 P16P8L^SS16P8L <C6_SDD_MX3 P16P8L>^Near USER2's
Office^117^79
DEVICES(13)=202^C6_SDD_MX3 ROUTINE^SSR <C6_SDD_MX3 ROUTINE>^Next to
USER2's Office^80^59
DEVICES(14)=427^SUP$PRT TEST^SUP$PRT TEST^DISK FILE^132^58
DEVICES(15)=283^SYS$INPUT^SYS$INPUT^SYS$INPUT;^132^64
DEVICES(16)=198^VMS FILE^VMS FILE^DISK^80^64
DEVICES(17)=349^C6_SDD_MX3 P16P8L^VPM <C6_SDD_MX3 P16P8L>^Near USER2's
Office^117^79
DEVICES(18)=291^VTB255^VTB255^RMS FILE^255^99999
DEVICES(19)=288^ZBROWSE^ZBROWSE^RMS FILE^255^99999
```

6.2.2 $$RES^XUDHSET(): Set Up Resource Device

Reference Type: Supported

Category: Device Handler

ICR #: 2232

Description: The $$RES^XUDHSET 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

6.2.3 ^%ZIS: Standard Device Call

Reference Type: Supported

Category: Device Handler

ICR #: 10086

Description: The ^%ZIS API allows you to select a device.

All input variables are optional. *Non*-namespaced variables that are defined and later **KILL**ed 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, it 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, see the "Form Feeds" section in the "Special Device Issues" section.

Format: `^%ZIS`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variable:	**%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 is prompted with the right margin query.
- **N—NO OPENING:** The Device Handler returns 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 (#3.5) file, is presented at the default response to the device prompt.
- **Q—QUEUING ALLOWED:** The job can be queued to run later. There is no automatic link between the Device Handler and 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 defines the **IO("Q")** 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, see the "TaskMan: Developer Tools" section.

- **0—DON'T USE IO(0)**: The Device Handler does *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, it returns a positive number in **POP**.
- **L—RESET IO("ZIO"):** If **%ZIS** contains **L**, the **IO("ZIO")** output variable is 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**. The possible values are:

- **RW** (which may *not* work in all environments)—**READ/WRITE** access.
- **R—READ** Only access.
- **W—WRITE** access.
- **A**—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 (#3.5) file. 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 (#3.5) file, TERMINAL TYPE (#3.2) file, 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 (#3.5) file is 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** are *not* 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**.).\

NOTE: If **IOP** is set to **NULL**, the device handler defaults to the **HOME** device.

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 (#3.5) file, by setting **IOP="P"** or **IOP="p"**. If there is *not* a closest printer associated with the home device at the time of the call, device selection fails and **POP** is returned with a positive value.

You can 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 (#3.5) file rather than as FREE TEXT.

Output Variables:	**IO:**	If a device is successfully opened, **IO** is returned with the device **$I** value of the selected device. If an abnormal exit occurs, **POP** is returned with a positive numeric value and **IO** is returned as **NULL**. **CAUTION: Because the returned value of IO can be changed, since December 1990, developers have been advised to check for a positive value in POP rather for IO equal to NULL when determining if an abnormal exit occurred.**
	IO(0):	**HOME DEVICE**—Contains the **$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, **KILL**s, and checks the existence of **IO(1,IO)**. **NOTE:** This array should *not* be altered by applications outside of Kernel.
	IO("CLNM"):	This variable holds the name of the remote system. It is defined via the RPC Broker.
	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 **KILL**ed when a call is made to ^%ZIS or HOME^%ZIS: Reset Home Device IO Variables APIs.

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 saves **IO("HFSIO")** and **IOPAR** so that the scheduled task opens the appropriate Host file. **NOTE:** This variable is **KILL**ed when a call is made to ^%ZIS or HOME^%ZIS: Reset Home Device IO Variables APIs.
IO("IP"):	This variable holds the Internet Protocol (IP) of the remote system.
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 is undefined. **NOTE:** This variable is **KILL**ed when a call is made to ^%ZIS or HOME^%ZIS: Reset Home Device IO Variables APIs.
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 exists temporarily, during spooling, and is **KILL**ed upon normal exit. **NOTE:** This variable is **KILL**ed when a call is made to ^%ZIS or HOME^%ZIS: Reset Home Device IO Variables APIs.

IO("T"):	TaskMan call.
IO("ZIO"):	**TERMINAL SERVER PORT**—If **%ZIS["L"**, both physical port and server names are returned in **IO("ZIO")** under Caché. 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 **WRITE**s 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 (#3.5) file. TaskMan uses the **IOCPU** input variable as an indicator of where the job should ultimately be run.
IOF:	**FORM FEED**—This output variable issues a form feed when writing its value with indirection; that is, **W @IOF**.
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 (#3.5) file.
IOPAR:	**OPEN PARAMETERS**—This variable returns any OPEN PARAMETERS that may have been defined for the selected device, for example, a magnetic tape drive. If the **OPEN PARAMETERS** input variable has *not* been defined, **IOPAR** is returned as **NULL**. **NOTE:** When a device is closed, this variable gets set to **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**. **NOTE:** When a device is closed, this variable gets set to **NULL**.

IOS:	**DEVICE NUMBER**—The DEVICE (#3.5) file 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 (#3.2) file.
IOST(0):	**SUBTYPE NUMBER**—This variable returns the Internal Entry Number (IEN) of the selected device's subtype as recorded in the TERMINAL TYPE (#3.2) file.
IOT:	**TYPE OF DEVICE**—The DEVICE (#3.5) file 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 is 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 has the same result.

The second condition, unavailability, is met if the Device Handler *cannot* open the selected device. 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 (#3.5) file. A device name may have been used that is *not* found as a **.01** field entry. If the device is selected with **P** for the closest printer, the CLOSEST PRINTER field in the DEVICE (#3.5) file may be **NULL**.

If the exit is abnormal, returning **POP** with a positive value, the following output variables are 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.

6.2.3.1 Examples

6.2.3.1.1 Example 1

Figure 53 is a simplified example; the process of issuing form feeds is *not* shown.

Figure 53: ^%ZIS API—Example

```
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
```

6.2.3.1.2 Example 2

The **IOP** variable can be defined to pass a string to the Device Handler so that no user interaction is 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]]
```

6.2.3.1.3 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]
```

6.2.3.1.4 Example 4

The following shows how a device named "RXPRINTER" in the DEVICE (#3.5) file can be opened *without user interaction*:

```
>S IOP="RXPRINTER" D ^%ZIS Q:POP
```

6.2.3.1.5 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.

6.2.3.1.6 Example 6

The **IOP** variable can be set to FORCED 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
```

6.2.3.1.7 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, see the "Spooling" section in the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide*.

In this example:

- **DEVNAM** contains the name of the device to be opened.
- **IO("DOC")** contains the spool document name.
- **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.
- **"MYDOC"** is the name of the spool document.

6.2.3.1.8 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 a grave accent character (`) followed by the IEN value of 202:

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

6.2.3.2 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 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 section.

6.2.3.3 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, see the "Host Files" section.

6.2.4 HLP1^%ZIS: Display Brief Device Help

Reference Type:	Supported
Category:	Device Handler
ICR #:	10086
Description:	The HLP1^%ZIS 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.

6.2.5 HLP2^%ZIS: Display Device Help Frames

Reference Type: Supported

Category: Device Handler

ICR #: 10086

Description: The HLP2^%ZIS 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.

6.2.6 HOME^%ZIS: Reset Home Device IO Variables

Reference Type: Supported

Category: Device Handler

ICR #: 10086

Description: The HOME^%ZIS 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.

NOTE: Developers have been advised that Kernel 8.0 is the last version of Kernel to support the ***obsolete*** 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 this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

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 (#3.5) file.
IOS:	Internal Entry Number (IEN) of last selected input/output device from the DEVICE (#3.5) file.
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.

6.2.7 $$REWIND^%ZIS(): Rewind Devices

Reference Type: Supported

Category: Device Handler

ICR #: 10086

Description: The $$REWIND^%ZIS 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 the **IO** variable, which is returned by ^%ZIS.
	iot:	(required) The "Type" of device to be rewound, in the same format as the **IOT** variable, which is returned by ^%ZIS.
	iopar:	(required) The "Open Parameters" for the selected device, in the same format as the **IOPAR** variable, which is returned by ^%ZIS.
Output:	returns:	Returns: • **1**—Device was rewound successfully. • **0**—Device was *not* rewound successfully.

6.2.7.1 Example

Figure 54: $$REWIND^%ZIS API—Example

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

6.2.8 ^%ZISC: Close Device

Reference Type: Supported

Category: Device Handler

ICR #: 10089

Description: The $$XMLHDR^MXMLUTL 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")** is no longer 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. Kernel versions beyond Kernel 8.0 will *not* export **^%ZIS("C")**.

Format: `^%ZISC`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables:	See ^%ZIS:	For a list of input variables, see the normal device output variables from the ^%ZIS: Standard Device Call API.
Output Variables:	See ^%ZIS:	For a list of output variables, see the normal device output variables from the ^%ZIS: Standard Device Call API.

6.2.8.1 Example

Figure 55: ^%ZISC API—Example

```
>D ^%ZISC
```

6.2.9 PKILL^%ZISP: Kill Special Printer Variables

Reference Type:	Supported
Category:	Device Handler
ICR #:	3172
Description:	The PKILL^%ZISP API **KILL**s printer-specific Device Handler variables. All output parameters defined by the PSET^%ZISP: Set Up Special Printer Variables API are **KILL**ed.
Format:	PKILL^%ZISP
Input Parameters:	none.
Output:	none.

6.2.10 PSET^%ZISP: Set Up Special Printer Variables

Reference Type:	Supported
Category:	Device Handler
ICR #:	3172
Description:	The PSET^%ZISP API defines a set of variables that toggle special printer modes. The corresponding fields in the TERMINAL TYPE (#3.2) file 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 this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

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.

6.2.10.1 Example

To toggle a printer mode with one of PSET^%ZISP's output variables, **WRITE** the variable to the printer using indirection, as follows:

Figure 56: PSET^%ZISP API—Example

```
>D PSET^%ZISP
>W @IOBARON
```

6.2.11 ENDR^%ZISS: Set Up Specific Screen Handling Variables

Reference Type:	Supported
Category:	Device Handler
ICR #:	10088
Description:	The ENDR^%ZISS 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 this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables:

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

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 are created.

Output Variables:	See ENS^%ZISS:	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.

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

Reference Type: Supported

Category: Device Handler

ICR #: 10088

Description: The ENS^%ZISS API is used for screen management. It sets up screen handling variables and other terminal type attributes.

Format: `ENS^%ZISS`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables:	**IOST(0):**	(required) Internal entry number of the selected device's subtype as recorded in the TERMINAL TYPE (#3.2) file.
	%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, see the "Output Variables" section.

NOTE: Not all characteristics are possible on all terminal types for all output variables. The **IOEFLD** and **IOSTBM** variables are used with indirection. Also, **IOSTBM** requires the setting of **IOTM** and **IOBM** as input variables for the top and bottom margins.

Output Variables:	**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:	Doublewide 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:	Singlewide 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** • **DO** • **ENTER** • **FIND** • **HELP** • **INSERT** • **IOCUB** • **IOCUD** • **IOCUF** • **IOCUU** • **KP0** • **KP1** • **KP2** • **KP3** • **KP4** • **KP5** • **KP6** • **KP7** • **KP8** • **KP9** • **MINUS** • **NEXTSCRN** • **PERIOD** • **PF1** • **PF2** • **PF3**

- **PF4**
- **PREVSCRN**
- **REMOVE**
- **SELECT**

6.2.13 GKILL^%ZISS: KILL Graphic Variables

Reference Type:	Supported
Category:	Device Handler
ICR #:	10088
Description:	The GKILL^%ZISS API is used for screen management. It **KILL**s graphic variables used in screen handling. All output parameters set up by the GSET^%ZISS: Set Up Graphic Variables API are **KILL**ed.
Format:	`GKILL^%ZISS`
Input Parameters:	none.
Output:	none.

6.2.14 GSET^%ZISS: Set Up Graphic Variables

Reference Type:	Supported
Category:	Device Handler
ICR #:	10088
Description:	The GSET^%ZISS 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, since they already have the high range of ASCII codes.
Format:	`GSET^%ZISS`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

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.

6.2.14.1 Example

Figure 57: GSET^%ZISS API—Example

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

6.2.15 KILL^%ZISS: KILL Screen Handling Variables

Reference Type: Supported

Category: Device Handler

ICR #: 10088

Description: The KILL^%ZISS API is used for screen management. It **KILL**s 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 **KILL**ed by this call.

Format: `KILL^%ZISS`

Input Parameters: none.

Output: none.

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

Reference Type:	Supported
Category:	Device Handler
ICR #:	2118
Description:	The CALL^%ZISTCP API makes a TCP/IP connection to a remote system.

NOTE: This API is **IPv6** compliant.

Format: `CALL^%ZISTCP`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables:

IPADDRESS: (required) This is the Internet Protocol (IP) address of the Host system to which it connects. It *must* be in either of the following formats:

- **IPv4**—Format of four numbers separated by dots (e.g., 99.99.9.999)
- **IPv6**—Format of six numbers separated by colons (e.g., fe80::206a:b21b:fbd5:c93).

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 the **IO** variable holds the implementation value that references 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 the ^%ZIS: Standard Device Call API.

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

Reference Type:	Supported
Category:	Device Handler
ICR #:	2118
Description:	The CLOSE^%ZISTCP API closes 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.

NOTE: This API is **IPv6** compliant.

Format: `CLOSE^%ZISTCP`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables:	See CALL^%ZISTCP:	For a list of input variables, see CALL^%ZISTCP: Make TCP/IP Connection (Remote System) API.
Output Variables:	See CALL^%ZISTCP:	For a list of output variables, see CALL^%ZISTCP: Make TCP/IP Connection (Remote System) API.

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

Reference Type: Supported

Category: Device Handler

ICR #: 2119

Description: The CLOSE^%ZISUTL 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.

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

Reference Type: Supported

Category: Device Handler

ICR #: 2119

Description: Use the OPEN^%ZISUTL API 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^%ZISUTL(): Close Device with Handle, 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 RMDEV^%ZISUTL(): Delete Data Given a Handle and CLOSE^%ZISUTL(): Close Device with 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. **REF:** For more information, see the ^%ZIS API 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). **REF:** For more information on these variables, see 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

6.2.19.1 Example

Figure 58: OPEN^%ZISUTL API—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
```

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

Reference Type:	Supported
Category:	Device Handler
ICR #:	2119
Description:	The RMDEV^%ZISUTL 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.	

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

Reference Type: Supported

Category: Device Handler

ICR #: 2119

Description: The SAVDEV^%ZISUTL 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.	

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

Reference Type: Supported

Category: Device Handler

ICR #: 2119

Description: The USE^%ZISUTL 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.

6.3 Special Device Issues

This section discusses the following special devices and device issues:

- Form Feeds
- Resources

6.3.1 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 **WRITE**s a form feed (**W @IOF**) to reset the value of **$Y** to **zero**. Therefore, applications 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** variable when closing the device. Thus, application developers can use the **IONOFF** variable to suppress form feeds by setting it just before calling ^%ZISC: Close Device API to close the device.

6.3.1.1 How to Check if Current Device is a CRT

You should use the following code to test if the current device is a CRT:

```
>I $E(IOST,1,2)'="C-"
```

If it returns:

- **False**—Current device is a CRT.
- **True**—Assume that the current device is a printer.

6.3.1.2 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:

6.3.1.2.1 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**.

6.3.1.2.2 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 Figure 59 and Figure 60 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.

Figure 59: Device Handler—Issuing Form Feeds following Current Guidelines

```
ROU      ;SAMPLE ROUTINE
         S IOP="DEVNAM" D ^%ZIS G EXIT:POP
         I $D(IO("Q")) S ZTRTN="DQ^ROU",ZTDESC="SAMPLE REPORT" D
^%ZTLOAD,HOME^%ZIS Q
.
.
.
DQ       ;SAMPLE REPORT
         S (END,PAGE)=0
         U IO D @("HDR"_(2-($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 60: Device Handler—Alternate Approach following Current Guidelines

```
ROU      ;SAMPLE ROUTINE
         S IOP="DEVNAM" D ^%ZIS G EXIT:POP
         I $D(IO("Q")) S ZTRTN="DQ^ROU",ZTDESC="SAMPLE REPORT" D
^%ZTLOAD,HOME^%ZIS Q
.
.
.
DQ       ;SAMPLE REPORT
         S (END,PAGE)=0
         U IO F  Q:END  D
         .D HDR:$Y+5>IOSL Q
         .W !,....
         .W !,...
               .
               .
               .
         D ^%ZISC Q
HDR      ;SAMPLE HEADER
         I PAGE,$E(IOST,1,2)="C-" W !,"Press RETURN to continue or '^' to
exit: " R X:DTIME S END='$T!(X="^") Q:END
HDR1     W:'($E(IOST,1,2)'="C-"&'PAGE) @IOF
HDR2     S PAGE=PAGE+1 W ?20,"SAMPLE HEADING",?(IOM-10),"PAGE:
",$J(PAGE,3)
```

6.3.2 Resources

6.3.2.1 Queuing to a Resource

You can only use resources through calls to ^%ZTLOAD. They *cannot* be directly manipulated (except by TaskMan). To use a resource, you need to set the **ZTIO** input variable to the name of the resource. For example:

```
>S ZTIO="ZZRES",ZTRTN="tag^routine",ZTDTH=$H
>S ZTDESC="First task in a series"
>D ^%ZTLOAD
```

Since the name of the resource is part of the call, application developers *must* include installation procedures so that system administrators are 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 FLAG**s, see the "TaskMan: Developer Tools" section.

7 Domain Name Service (DNS): Developer Tools

7.1 Application Programming Interface (API)

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

7.1.1 $$ADDRESS^XLFNSLK(): Convert Domain Name to IP Addresses

Reference Type:	Supported	
Category:	Domain Name Service (DNS)	
ICR #:	3056	
Description:	The $$ADDRESS^XLFNSLK extrinsic function calls the Domain Name Service (DNS) to convert a domain name into its IP addresses. The IP addresses of the DNS being called are in the DNS IP (#8989.3,51) field in the KERNEL SYSTEM PARAMETERS (#8989.3) file. **NOTE:** This API is **IPv6** compliant.	
Format:	$$ADDRESS^XLFNSLK(domain_name[,type])	
Input Parameters:	**domain_name**:	(required) This is the fully qualified domain name (e.g., REDACTED.VA.GOV).
	type:	(optional) This input parameter is from the set A: **IPv4** address (the default), AAAA: **IPv6** address, CNAME: alias.
Output:	returns:	Returns a comma-separated list of IP addresses that are associated with the input domain.

7.1.1.1 Examples

7.1.1.1.1 Example 1

Figure 61: $$ADDRESS^XLFNSLK API—Example 1

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

>W X
99.9.99.999
```

7.1.1.1.2 Example 2

Figure 62: $$ADDRESS^XLFNSLK API—Example 2

```
>S X=$$ADDRESS^XLFNSLK("www.google.com","AAAA" )

>W X
2607:F8B0:400E:0C02:0000:0000:0000:0067
```

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

Reference Type: Supported

Category: Domain Name Service (DNS)

ICR #: 3056

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

NOTE: This API is **IPv6** compliant.

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., REDACTED.VA.GOV).

Output Parameters:

.return: Returns data in the array passed in by reference. The data is subscripted by priority. The **domain_name** parameter is a fully qualified domain name (e.g., REDACTED.VA.GOV).

7.1.2.1 Examples

7.1.2.1.1 IPv4 Example

Figure 63: MAIL^XLFNSLK API Example: IPv4

```
>K ZX D MAIL^XLFNSLK(.ZX,"REDACTED.VA.GOV") ZW ZX
ZX=2
ZX(5)=a2.REDACTED.VA.GOV.^REDACTED
ZX(10)=a1.REDACTED.VA.GOV.^REDACTED
```

7.1.2.1.2 IPv6 Example

Figure 64: MAIL^XLFNSLK API Example: IPv6

```
>K ZX D MAIL^XLFNSLK(.ZX,"GMAIL.COM") ZW ZX
ZX=5
ZX(5)="gmail-smtp-in.l.google.COM.^2607:F8B0:4001:0C0E:0000:0000:0000:001A"
ZX(10)="alt1.gmail-smtp-
in.l.google.COM.^2607:F8B0:400D:0C0C:0000:0000:0000:001B"
ZX(20)="alt2.gmail-smtp-
in.l.google.COM.^2607:F8B0:400C:0C0A:0000:0000:0000:001B"
ZX(30)="alt3.gmail-smtp-
in.l.google.COM.^2A00:1450:400C:0C08:0000:0000:0000:001B"
ZX(40)="alt4.gmail-smtp-
in.l.google.COM.^2A00:1450:400B:0C03:0000:0000:0000:001B"
```

8 Electronic Signatures: Developer Tools

8.1 Application Programming Interface (API)

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

8.1.1 ^XUSESIG: Set Up Electronic Signature Code

Reference Type:	Controlled Subscription
Category:	Electronic Signatures
ICR #:	936
Description:	The ^XUSESIG 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.

8.1.2 SIG^XUSESIG(): Verify Electronic Signature Code

Reference Type:	Supported	
Category:	Electronic Signatures	
ICR #:	10050	
Description:	The SIG^XUSESIG 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 (#200) file is returned in **x1**. Otherwise, **x1** is returned as **NULL**.

8.1.3 $$CHKSUM^XUSESIG1(): Build Checksum for Global Root

Reference Type:	Supported	
Category:	Electronic Signatures	
ICR #:	1557	
Description:	The $$CHKSUM^XUSESIG1 extrinsic function takes a global root (**$name_value**) and builds a checksum for all data in the root. **NOTE:** The **flag** input parameter is no longer used. Previously, It was used when there was more than one checksum algorithm.	
Format:	`$$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.

8.1.4 $$CMP^XUSESIG1(): Compare Checksum to $Name_Value

Reference Type:	Supported	
Category:	Electronic Signatures	
ICR #:	1557	
Description:	The $$CMP^XUSESIG1 extrinsic function compares the checksum passed in to the calculated value from the **$name_value** input parameter. It Returns the following: • **1**—Match. • **0**—No match.	
Format:	`$$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: • **1**—Match. • **0**—No match.

8.1.5 $$DE^XUSESIG1(): Decode String

Reference Type:	Supported	
Category:	Electronic Signatures	
ICR #:	1557	
Description:	The $$DE^XUSESIG1 extrinsic function decodes the input string using the checksum as the key.	
Format:	`$$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.

8.1.6 $$EN^XUSESIG1(): Encode ESBLOCK

Reference Type:	Supported	
Category:	Electronic Signatures	
ICR #:	1557	
Description:	The $$EN^XUSESIG1 extrinsic function encodes the **ESBLOCK** using the checksum as the key.	
Format:	`$$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**.

8.1.7 $$ESBLOCK^XUSESIG1(): E-Sig Fields Required for Hash

Reference Type:	Supported	
Category:	Electronic Signatures	
ICR #:	1557	
Description:	The $$ESBLOCK^XUSESIG1 extrinsic function returns the set of fields from the NEW PERSON (#200) file that are needed as part of the hash for an acceptable electronic signature (E-Sig). These fields include the following: • E-Sig Block • E-Sig Title • Degree • Current Date/Time If the Internal Entry Number (IEN) is *not* passed in, then it uses the **DUZ**.	
Format:	`$$ESBLOCK^XUSESIG1([ien])`	
Input Parameters:	**ien**:	(optional) This is the Internal Entry Number (IEN) of the NEW PERSON (#200) file entry for which data is requested. The default is to use the **DUZ** of the current user.
Output:	returns:	Returns the following fields: • E-Sig Block • E-Sig Title • Degree • Current Date/Time

8.1.8 DE^XUSHSHP: Decrypt Data String

Reference Type:	Supported
Category:	Electronic Signatures
ICR #:	10045
Description:	The DE^XUSHSHP 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`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

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

8.1.9 EN^XUSHSHP: Encrypt Data String

Reference Type: Supported

Category: Electronic Signatures

ICR #: 10045

Description: The EN^XUSHSHP 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 variables. Typically, this API would be used to encrypt strings within a document.

Format: EN^XUSHSHP

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables:	**X:**	(required) The string to be encrypted (e.g., the contents of the SIGNATURE BLOCK PRINTED NAME field in the NEW PERSON [#200] file).
	X1:	(required) An identification number (e.g., **DUZ**).
	X2:	(required) A document number (or the number one).
Output Variables:	**X:**	Encrypted string.

8.1.10 HASH^XUSHSHP: Hash Electronic Signature Code

Reference Type:	Supported
Category:	Electronic Signatures
ICR #:	10045
Description:	The HASH^XUSHSHP 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 [#200] file).
Format:	`HASH^XUSHSHP`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables:	**X:**	(required) Electronic Signature code as entered by the user.
Output Variables:	**X:**	Hashed form of the electronic signature code submitted as input to function.

9 Error Processing: Developer Tools

9.1 Direct Mode Utilities

These direct mode utilities can be run from Programmer mode. They are *not*, however, APIs; instead, they are provided for convenience.

9.1.1 >D ^XTER

You can call the **^XTER** direct mode utility from Programmer mode. It is the same as using the **Error Trap Display** [XUERTRAP] option.

9.1.2 >D ^XTERPUR

You can call the **^XTERPUR** direct mode utility from Programmer mode. It is the same as using the **Clean Error Trap** [XUERTRP CLEAN] option.

9.2 Application Programming Interface (API)

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

9.2.1 $$EC^%ZOSV: Get Error Code

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

9.2.1.1 Example

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

9.2.2 ^%ZTER: Kernel Standard Error Recording Routine

Reference Type:	Supported
Category:	Error Processing
ICR #:	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 (#3.075) file.

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 (#3.075) file if it decides that it needs to. ^%ZTER gathers all available information such as local symbols and last global reference and stores that information in an entry in the ERROR LOG (#3.075) file.

The simple example below shows an application that replaces the standard Kernel Error Trap with its own Error Trap. When an error occurs, and the application's Error Trap routine is called, it calls $$EC^%ZOSV to see what type of error occurred. If an end-of-file (EOF) error occurs, it lets the application continue. Otherwise, it calls ^%ZTER to record the error, and then quits to terminate the application.

NOTE: The recording mechanism of ^%ZTER also functions in the absence of an error. In a debug mode, this would enable a developer to record local symbols and global structures at predetermined places within code execution for later checking.

NOTE: As of Kernel Patch XU*8.0*431, the ^%ZTER error trap routine checks a count (limit) in the ERROR TRAP SUMMARY (#3.077) file and stops recording errors once this limit has been reached. This limit is initialized to **10** but can be changed by the sites. To change the value, use VA FileMan to edit the ERROR LIMIT (#520.1) field in the KERNEL SYSTEM PARAMETERS (#8989.3) file.

Format: `^%ZTER`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables:	**%ZT:**	(optional) The **%ZT** array can be used to identify a global node whose descendants should be recorded in the error log. When called within the standard Kernel Error Trap, **%ZT** is set to record the user's location in the menu system:

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

Output Variables:	**%ZTERROR:**	Calls to the error recorder always return this variable. It has the error name and error type as its first and second caret-delimited (^) 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 (#3.076) file.

9.2.2.1 Examples

9.2.2.1.1 Example 1

Figure 65 is an example of the Error Trap:

Figure 65: Error Trap—Example

```
ZXGP ; 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^ZXGP"
         ;
         ; 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
         ;
```

9.2.2.1.2 Example 2

To test the error limit set in the ERROR LIMIT (#520.1) field in the KERNEL SYSTEM PARAMETERS (#8989.3) file, run the following:

```
>F I=1:1:20 D APPERROR^%ZTER("My Application Error")
```

Check the error trap and see how many errors with "My Application Error" get recorded in the Kernel error log (i.e., ERROR LOG [#3.075] file). If the value in the ERROR LIMIT (#520.1) field is set to **10**, there should just be **10** occurrences of the “My Error” error in the Kernel error log.

NOTE: For more information, see the "APPERROR^%ZTER" API.

9.2.3 APPERR^%ZTER: Set Application Error Name in Kernel Error Trap Log

Reference Type:	Supported
Category:	Error Processing
ICR #:	1621

Description: The APPERR^%ZTER API sets the "application error" text passed in as the error name in the Kernel error trap log (i.e., ERROR LOG [#3.075] file).

NOTE: The APPERR^%ZTER API replaces the need to set **$ZE** before calling the ^%ZTER: Kernel Standard Error Recording Routine API:

Before:

```
>S $ZE="application error" D ^%ZTER
```

After:

```
>D APPERROR^%ZTER("application error")
```

NOTE: This API was released with Kernel Patch XU*8.0*431.

Format: `APPERROR^%ZTER("application error")`

Input Parameters: **"application error"**: This input parameter is the "application error" name that gets displayed in the Kernel error trap log (i.e., ERROR LOG [#3.075] file).

Output: returns: Displays the "application error" text passed in as the error name in the Kernel error trap log (i.e., ERROR LOG [#3.075] file).

9.2.3.1 Example

```
>DO APPERROR^%ZTER("My Application Error")
```

Check the Kernel error trap and see if there is an error called "My Application Error".

9.2.4 $$NEWERR^%ZTER: Verify Support of Standard Error Trapping (Obsolete)

NOTE: The $$NEWERR^%ZTER API is obsolete, because all VA systems support the standard error trapping.

Reference Type:	Supported
Category:	Error Processing
ICR #:	1621
Description:	The $$NEWERR^%ZTER extrinsic function reports if the current platform supports the standard error trapping. It returns: • **1**—If the standard error trapping is supported. • **0**—For all other cases.
Format:	`$$NEWERR^%ZTER`
Input Parameters:	none.
Output:	returns: Returns: • **1**—If the standard error trapping is supported. • **0**—For all other cases.

9.2.5 UNWIND^%ZTER: Quit Back to Calling Routine

Reference Type:	Supported
Category:	Error Processing
ICR #:	1621
Description:	Use the UNWIND^%ZTER API 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.

9.2.5.1 Example

Main:

Figure 66: UNWIND^%ZTER API—Main Code Example

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

Usage:

Figure 67: UNWIND^%ZTER API—Usage

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

10 Field Monitoring: Developer Tools

10.1 Application Programming Interface (API)

The OPKG^XUHUI API is available for developers to work with field monitoring, which is described below.

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

Reference Type:	Supported	
Category:	Field Monitoring	
ICR #:	3589	
Description:	The OPKG^XUHUI 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 to use the PROTOCOL (#101) file or the OPTION (#19) file. If this parameter is **NULL**, the default value is used (i.e., **101**): • **101 (default)**—PROTOCOL (#101) file is used. • **19**—The OPTION (#19) file is 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 is used (i.e., **S**): • **S (default)**—The data being passed is from the **SET**ting of the cross-reference. • **K**—The data being passed is from the **KILL**ing 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.

10.1.1.1 Example

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

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

10.1.1.1.1 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 in Figure 68:

Figure 68: OPKG^XUHUI API—Example of Creating New Style Cross-references

```
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 (#200) file 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" (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)
```

10.1.1.1.2 Sample Scenario

Change a monitored (cross-referenced) field value in the NEW PERSON (#200) file, as shown in Figure 69:

Figure 69: OPKG^XUHUI API—Sample Scenario

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

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

In this example, the ONE XUUSER's Date of Birth (DOB) was changed from **07/04/50** to **12/24/49** and also changed the 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 section), you should see this data passed to the "XUHUI FIELD CHANGE EVENT" protocol (see the "Internal Results for Developers" section).

10.1.1.1.3 Internal Results for Developers

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 New Style Cross-References” section).

Figure 70: OPKG^XUHUI API—Example of Internal Results

```
------------------------------------------------------------------------
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=XUUSER,ONE
X(1)=XUUSER,ONE
X(2)=
X(3)=2491224
X(4)=000558888
------------------------------------------------------------------------
```

Old values are in this array.

```
X1=XUUSER,ONE
X1(1)=XUUSER,ONE
X1(2)=
X1(3)=2500704
X1(4)=000220000
------------------------------------------------------------------------
```

New values are in this array.

```
X2=XUUSER,ONE
X2(1)=XUUSER,ONE
X2(2)=
X2(3)=2491224
X2(4)=000558888
------------------------------------------------------------------------
```

“S” = Set Logic is being executed, “K” = Kill logic being executed.

```
XUHUIA=S
XUHUIDA=70
XUHUIFIL=200
XUHUIFLD=
```

“DA” array, File number, and Field numbers if available.

```
XUHUINM=XUHUI FIELD CHANGE EVENT
```

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

```
XUHUIOP=101
```

File number of where to find the Extended Action.

The "X" array.

```
XUHUIX=XUUSER,ONE
XUHUIX(1)=XUUSER,ONE
XUHUIX(2)=
XUHUIX(3)=2491224
XUHUIX(4)=000558888
```

The "X1" array.

```
XUHUIX1=XUUSER,ONE
XUHUIX1(1)=XUUSER,ONE
XUHUIX1(2)=
XUHUIX1(3)=2500704
XUHUIX1(4)=000220000
```

The "X2" array.

```
XUHUIX2=XUUSER,ONE
XUHUIX2(1)=XUUSER,ONE
XUHUIX2(2)=
XUHUIX2(3)=2491224
XUHUIX2(4)=000558888
XUHUIXR=AXUHUI
```

Name of cross-reference being executed by DIK.

11 File Access Security: Developer Tools

11.1 Overview

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, see the "File Access Security" section in the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide*.

11.2 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 Programmers Manual*.

11.3 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, requires that the user have:

- **WRITE** access to change data in the pointed-to file.
- **DELETE** access to delete an entry.
- (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 Learn As You Go (LAYGO), as specified in the data dictionary, and the pointed-to file also allows LAYGO, the user does *not* need explicit file access to add entries. If the pointed-to file is protected, however, the user needs 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.

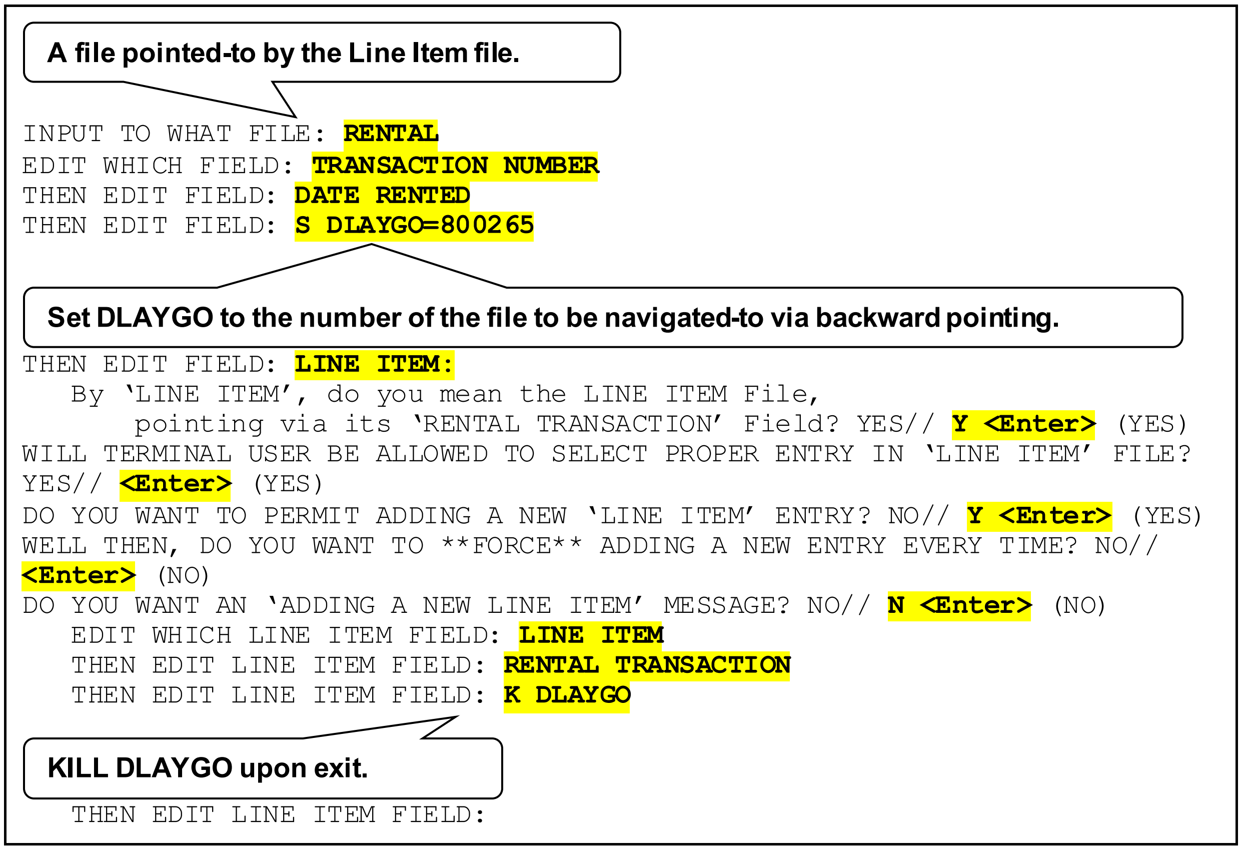
11.4 Use of DLAYGO When Navigating to Files

Use of input templates or VA FileMan ^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** access 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 system administrators 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 in Figure 71:

Figure 71: File Access Security—Setting DLAYGO in a Template


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.

```
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:
```

11.5 Use of DLAYGO in ^DIC Calls

When a user attempts to add an entry at the top level of a file in a VA FileMan ^DIC call, their file access security is checked for **LAYGO** access to the file. Developers 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, see the *VA FileMan Developer's Guide*.

11.6 Use of DIDEL in ^DIE Calls

When a user attempts to delete an entry at the top level of a file in a VA FileMan ^DIE call, their file access security is checked for **DELETE** access to the file. Developers 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, see the *VA FileMan Developer's Guide*.

12 Help Processor: Developer Tools

12.1 Entry and Exit Execute Statements

The HELP FRAME (#9.2) file contains two fields for the entry of M code:

- **Entry Execute Statement**—Code in the Entry Execute Statement is executed just before the help frame is displayed.
- **Exit Execute Statement**—Code in the Exit Execute Statement is executed afterwards.

12.2 Application Programming Interface (API)

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

12.2.1 EN^XQH: Display Help Frames

Reference Type:	Supported
Category:	Help Processor
ICR #:	10074
Description:	The EN^XQH 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`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables:	**XQH:**	(required) Help Frame name (the **.01** value from the HELP FRAME [#9.2] file).
Output:	none.	

12.2.2 EN1^XQH: Display Help Frames

Reference Type:	Supported
Category:	Help Processor
ICR #:	10074
Description:	The EN1^XQH API displays a help frame as ACTION^XQH4(): Print Help Frame Tree does, except that it does *not* 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`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variable:	**XQH:**	(required) Help Frame name (the **.01** value from the HELP FRAME [#9.2] file).
Output:	none.	

12.2.3 ACTION^XQH4(): Print Help Frame Tree

Reference Type:	Supported
Category:	Help Processor
ICR #:	10080
Description:	The ACTION^XQH4 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 (#9.2) file. Should be set to the **NAME** of the top-level help frame for which a listing is desired.
	xqfmt:	(optional) Specifies the output format. Value of **xqfmt** can be: • **T—**Text of help frames only (default). • **R—**Text of help frames, plus a table of related frames and keywords (if any) for each help frame. • **C—**Complete listing (text of help frames, table of related frames for each help frame, and internal help frame names).
Output:	none.	

13 Host Files: Developer Tools

13.1 Application Programming Interface (API)

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

The traditional method of working with Host File System (HFS) files prior to Kernel 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 [#3.5] file). For example:

Figure 72: Host Files—Opening a Host File Using the ^%ZIS API

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

Kernel 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.
$$GATF^%ZISH	pend records from a global to a Host file.
$$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.

Table 3 lists definitions that apply for the Host file APIs:

Table 3: Host File APIs—Definitions

Term	Definition
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 **READ**s 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 does *not* exist, it is created. • **A—PEND**; use the file for writing but start writing at the end of the current file. If the file does *not* exist, it is created. • **B—BINARY** file.

13.1.1 CLOSE^%ZISH(): Close Host File

Reference Type: Supported

Category: Host Files

ICR #: 2320

Description: The CLOSE^%ZISH 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.

13.1.1.1 Example

Figure 73: CLOSE^%ZISH API—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")
```

13.1.2 $$DEFDIR^%ZISH(): Get Default Host File Directory

Reference Type:	Supported
Category:	Host Files
ICR #:	2320

Description: The $$DEFDIR^%ZISH 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 (#8989.3) file.
- **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 Windows 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.

13.1.3 $$DEL^%ZISH(): Delete Host File

Reference Type:	Supported
Category:	Host Files
ICR #:	2320

Description: The $$DEL^%ZISH 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")=""`
		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.

13.1.3.1 Example

Figure 74: $$DEL^%ZISH API—Example

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

13.1.4 $$FTG^%ZISH(): Load Host File into Global

Reference Type: Supported

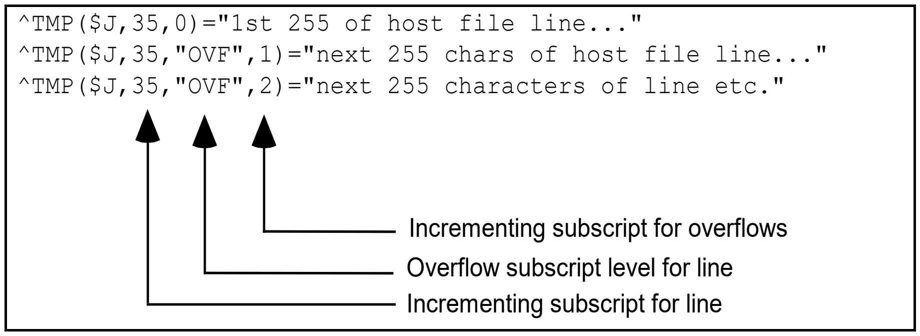
Category: Host Files

ICR #: 2320

Description: The $$FTG^%ZISH extrinsic function loads a Host file into a global. Each line of the Host file becomes the value of one node in the global. You do *not* need to open the Host file before making this call; it is opened and closed by $$FTG^%ZISH.

If a line from a Host file exceeds **255** characters in length, the overflows are stored in overflow nodes for that line, as follows:

Figure 75: Host Files—Overflow Lines in a Host File Sample



```
^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."
```

Incrementing subscript for overflows
Overflow subscript level for line
Incrementing subscript for line

Format:	`$$FTG^%ZISH(path,filename,global_ref,inc_subscr[,ovfsub])`	
Input Parameters:	**path**:	(required) Full path, up to but *not* including the filename.
	filename:	(required) Name of the file to open.
	global_ref:	(required) Global reference to **WRITE** Host file to, in fully resolved (closed root) format. This function does *not* **KILL** the global before writing to it. At least one subscript *must* be numeric. This is the incrementing subscript (i.e., the subscript that $$FTG^%ZISH increments 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 increments the third subscript [e.g., **^TMP(115,1,x)**], but **WRITE**s nodes at the full global reference [e.g., **^TMP(115,1,x,0)**].
	ovfsub:	(optional) Name of subscript level at which overflow nodes for lines (if any) should be stored. Overflows occur if a line is greater than **255** characters. Further overflows occur for every additional **255** characters. The default subscript name at which overflows are stored for a line is "**OVF**".
Output:	returns:	Returns: • **1**—Success. • **0**—Failure.

13.1.4.1 Example

Figure 76: $$FTG^%ZISH API—Example

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

13.1.5 $$GATF^%ZISH(): Copy Global to Host File

Reference Type:	Supported	
Category:	Host Files	
ICR #:	2320	
Description:	The $$GATF^%ZISH extrinsic function is used in the same way as the $$GTF^%ZISH(): Copy Global to Host File API. The one difference is that if the file already exists, $$GATF^%ZISH appends global nodes to the existing file rather than truncating the existing file first. **REF:** For more information, see the $$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^%ZISH increments the third subscript [e.g., **^TMP(115,1,x)**], but **READ**s 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.

13.1.6 $$GTF^%ZISH(): Copy Global to Host File

Reference Type:	Supported	
Category:	Host Files	
ICR #:	2320	
Description:	The $$GTF^%ZISH extrinsic function **WRITE**s the values of nodes in a global (at the subscript level you specify) to a Host file. If the Host file already exists, it is truncated to length **zero** (**0**) before the copy. You do *not* need to open the Host file before making this call. The Host file is opened (in **WRITE** mode) and closed by $$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^%ZISH increments the third subscript [e.g., **^TMP(115,1,x)**], but **READ**s 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.

13.1.6.1 Example

Figure 77: $$GTF^%ZISH API—Example

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

13.1.7 $$LIST^%ZISH(): List Directory

Reference Type:	Supported	
Category:	Host Files	
ICR #:	2320	
Description:	The $$LIST^%ZISH extrinsic function returns a list of file names in the current directory. The list is returned in an array in the variable named by the third parameter.	
Format:	`$$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*")=""`
		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.

13.1.7.1 Example

Figure 78: $$LIST^%ZISH API—Example

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

13.1.8 $$MV^%ZISH(): Rename Host File

Reference Type:	Supported	
Category:	Host Files	
ICR #:	2320	
Description:	The $$MV^%ZISH extrinsic function renames a Host file. The function performs the renaming, regardless of the underlying operating system, by first copying the file to the new name/location and then deleting the original file at the old name/location.	
Format:	`$$MV^%ZISH([path1,]filename1[,path2],filename2)`	
Input Parameters:	**path1**:	(optional) Full path of the original file, up to but *not* including the filename. If **NULL**, it defaults to $$DEFDIR^%ZISH.
	filename1:	(required) Name of the original file.
	path2:	(optional) Full path of renamed file, up to but *not* including the filename. If **NULL**, it defaults to $$DEFDIR^%ZISH.
	filename2:	(required) Name of the renamed file.
Output:	returns:	Returns: • **1**—Success. • **0**—Failure.

13.1.8.1 Example

Figure 79: $$MV^%ZISH API—Example

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

13.1.9 OPEN^%ZISH(): Open Host File

Reference Type:	Supported	
Category:	Host Files	
ICR #:	2320	
Description:	The OPEN^%ZISH 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 is used.
	filename:	(required) Name of the file to open.
	mode:	(required) Mode to open file: • **W—WRITE**. • **R—READ**. • **A—PEND**. • **B—BLOCK** (fixed record size).
	max:	(optional) Maximum record size for a new file.
	subtype:	(optional) File subtype.
Output Variables:	**POP:**	Returns the following values: • **Zero (0)**—File was opened successfully. • **Positive Value**—File was *not* opened.
	IO:	Name of the opened file in the format to use for M **USE** and **CLOSE** commands.

13.1.9.1 Example

Figure 80: OPEN^%ZISH API—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")
```

13.1.10 $$PWD^%ZISH: Get Current Directory

Reference Type: Supported

Category: Host Files

ICR #: 2320

Description: The $$PWD^%ZISH 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.

13.1.10.1 Example

Figure 81: $$PWD^%ZISH API—Example

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

13.1.11 $$STATUS^%ZISH: Return End-of-File Status

Reference Type: Supported

Category: Host Files

ICR #: 2320

Description: The $$STATUS^%ZISH extrinsic function returns the current end-of-file status. If end-of-file has been reached, $$STATUS^%ZISH returns:

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

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.

13.1.11.1 Example

Figure 82: $$STATUS^%ZISH API—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")
```

14 Institution File: Developer Tools

14.1 Application Programming Interface (API)

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

14.1.1 $$ACTIVE^XUAF4(): Institution Active Facility (True/False)

Reference Type:	Supported	
Category:	Institution File	
ICR #:	2171	
Description:	The $$ACTIVE^XUAF4 extrinsic function, given the Internal Entry Number (IEN) in the INSTITUTION (#4) file, returns the Boolean value for the question—is this an active facility? It checks to see if the INACTIVE FACILITY FLAG (#101) field is *not* set.	
Format:	`$$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 (#101) field has a value indicating it is inactive.

14.1.2 CDSYS^XUAF4(): Coding System Name

Reference Type:	Supported	
Category:	Institution File	
ICR #:	2171	
Description:	The CDSYS^XUAF4 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
```

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

Reference Type: Supported

Category: Institution File

ICR #: 2171

Description: The CHILDREN^XUAF4 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); has the grave accent (`) 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
```

14.1.4 $$CIRN^XUAF4(): Institution CIRN-enabled Field Value

Reference Type: Supported

Category: Institution File

ICR #: 2171

Description: The $$CIRN^XUAF4 extrinsic function returns the value of the CIRN-enabled field from the INSTITUTION (#4) file.

Format: `$$CIRN^XUAF4(inst[,value])`

Input Parameters:	**inst**:	(required) Institution lookup value, any of the following: • Internal Entry Number (IEN); has the grave accent (`) 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.

14.1.5 F4^XUAF4(): Institution Data for a Station Number

Reference Type: Supported

Category: Institution File

ICR #: 2171

Description: The F4^XUAF4 API returns the Internal Entry Number (IEN) and other institution data, including historical information, for a given STATION NUMBER (#99) in the INSTITUTION (#4) file.

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: • **A**—Active entries only. • **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).

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.

14.1.5.1 Example

Figure 83: F4^XUAF4 API—Example

```
>D F4^XUAF4("528A8",.ARRAY)

>ZW ARRAY
ARRAY=7020
ARRAY("INACTIVE")=0
ARRAY("NAME")=REDACTED
ARRAY("REALIGNED FROM")=500^500^3000701
ARRAY("STATION NUMBER")=528A8
ARRAY("TYPE")=VAMC
ARRAY("VA NAME")=REDACTED - REDACTED DIVISION
```

14.1.6 $$ID^XUAF4(): Institution Identifier

Reference Type: Supported

Category: Institution File

ICR #: 2171

Description: The $$ID^XUAF4 extrinsic function returns the Identifier (ID) of an INSTITUTION (#4) file 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 of the INSTITUTION (#4) file. To see the existing coding system in the file:

```
>D CDSYS^XUAF4(.Y)
```

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

Output: returns: Returns the INSTITUTION (#4) file Identifier (ID) associated with the given Coding System and IEN.

14.1.7 $$IDX^XUAF4(): Institution IEN (Using Coding System & ID)

Reference Type: Supported

Category: Institution File

ICR #: 2171

Description: The $$IDX^XUAF4 extrinsic function returns the Internal Entry Number (IEN) of an INSTITUTION (#4) file 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 of the INSTITUTION (#4) file. To see the existing coding system in the file:

```
>D CDSYS^XUAF4(.Y)
```

id: (required) **ID** is the identifier associated with the coding system. The station number, for example, is the identifier for the **VASTANUM** coding system and **NPI** number is the **ID** for the **NPI** coding system.

Output:

returns: Returns the INSTITUTION (#4) file Internal Entry Number (IEN) associated with the given Coding System and Identifier (**ID**).

14.1.8 $$IEN^XUAF4(): IEN for Station Number

Reference Type: Supported

Category: Institution File

ICR #: 2171

Description: The $$IEN^XUAF4 extrinsic function returns the Internal Entry Number (IEN) of the entry for a given STATION NUMBER (#99) field in the INSTITUTION (#4) file.

Format: `$$IEN^XUAF4(sta)`

Input Parameters:

sta: (required) Station Number.

Output:

returns: Returns:

- **IEN**—Internal Entry Number.
- **NULL**—Error.

14.1.8.1 Example

Figure 84: $$IEN^XUAF4 API—Example

```
>S X=$$IEN^XUAF4("528A5")

>W X
532
```

14.1.9 $$LEGACY^XUAF4(): Institution Realigned/Legacy (True/False)

Reference Type: Supported

Category: Institution File

ICR #: 2171

Description: The $$LEGACY^XUAF4 extrinsic function, given the STATION NUMBER (#99) field in the INSTITUTION (#4) file, 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 (#99) field value in the INSTITUTION (#4) file 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.

14.1.10 $$LKUP^XUAF4(): Institution Lookup

Reference Type: Supported

Category: Institution File

ICR #: 2171

Description: The $$LKUP^XUAF4 extrinsic function returns the IEN or **zero (0)** when doing a lookup on the INSTITUTION (#4) file.

Format: `$$LKUP^XUAF4(inst)`

Input Parameters:	**inst**:	(required) Institution lookup value, any of the following: • Internal Entry Number (IEN); has the grave accent (`) in front of it. • Station Number. • Station Name.
Output:	returns:	Returns: • **IEN**—Internal Entry Number. • **Zero (0)**.

14.1.11 LOOKUP^XUAF4(): Look Up Institution Identifier

Reference Type:	Supported	
Category:	Institution File	
ICR #:	2171	
Description:	The LOOKUP^XUAF4 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 (#.02) field of the IDENTIFIER (#9999) Multiple field in the INSTITUTION (#4) file.	
Format:	`LOOKUP^XUAF4()`	
Input Parameters:	See IX^DIC	For input information, see the IX^DIC documentation in the *VA FileMan Developer's Guide*.
Output:	See IX^DIC	For output information, see the IX^DIC documentation in the *VA FileMan Developer's Guide.*

14.1.11.1 Example

Figure 85: LOOKUP^XUAF4 API—Example

```
Select INSTITUTION CODING SYSTEM: DMIS
                            ID: 0037
DMIS   0037   WALTER REED        DC   USAH       688CN
```

14.1.12 $$MADD^XUAF4(): Institution Mailing Address

Reference Type:	Supported	
Category:	Institution File	
ICR #:	2171	
Description:	The $$MADD^XUAF4 extrinsic function returns the mailing address information for an institution in a caret-delimited string (i.e., streetaddr^city^state^zip) for a given Internal Entry Number (IEN) in the INSTITUTION (#4) file.	
Format:	`$$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`

14.1.13 $$NAME^XUAF4(): Institution Official Name

Reference Type:	Supported	
Category:	Institution File	
ICR #:	2171	
Description:	The $$NAME^XUAF4 extrinsic function returns the OFFICIAL NAME (#100) field value in the INSTITUTION (#4) file for an institution given its Internal Entry Number (IEN). However, if Field #100 is **NULL**, the NAME (#.01) field in the INSTITUTION (#4) file is returned.	
Format:	`$$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 (#100) field value in the INSTITUTION (#4) file**—If Field #100 is *not* **NULL**.
- **NAME (#.01) field value in the INSTITUTION (#4) file**—If Field #100 is **NULL**.

14.1.14 $$NNT^XUAF4(): Institution Station Name, Number, and Type

Reference Type:	Supported	
Category:	Institution File	
ICR #:	2171	
Description:	The $$NNT^XUAF4 extrinsic function returns the station information for an institution in a caret-delimited string (i.e., station_name^station_number^station_type) for a given Internal Entry Number (IEN) in the INSTITUTION (#4) file.	
Format:	`$$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_name^station_number^station_type`

14.1.15 $$NS^XUAF4(): Institution Name and Station Number

Reference Type:	Supported	
Category:	Institution File	
ICR #:	2171	
Description:	The $$NS^XUAF4 extrinsic function returns the institution information in a caret-delimited string (i.e., institution_name^station_number) for a given Internal Entry Number (IEN) in the INSTITUTION (#4) file.	
Format:	`$$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`

14.1.16 $$O99^XUAF4(): IEN of Merged Station Number

Reference Type:	Supported	
Category:	Institution File	
ICR #:	2171	
Description:	The $$O99^XUAF4 extrinsic function returns the Internal Entry Number (IEN) of the valid STATION NUMBER field (#99) in the INSTITUTION (#4) file, if this entry was merged during the INSTITUTION (#4) file cleanup process (e.g., due to a duplicate STATION NUMBER [#99] field). This function may be used by application developers to re-point their INSTITUTION (#4) file references to a valid entry complete with Station Number.	
Format:	`$$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 (#4) file entry with a valid STATION NUMBER field (#99)—the Station Number deleted from the input IEN during the cleanup process (i.e., Kernel Patch XU*8.0*206).

14.1.16.1 Example

Figure 86: $$O99^XUAF4 API—Example

```
>S NEWIEN=$$O99^XUAF4(6538)

>W NEWIEN
6164
>W ^DIC(4,6164,99)
519HB^^^
```

14.1.17 $$PADD^ XUAF4(): Institution Physical Address

Reference Type:	Supported	
Category:	Institution File	
ICR #:	2171	
Description:	The $$PADD^ XUAF4 extrinsic function returns the physical address information for an institution in a caret-delimited string (streetaddr^city^state^zip) for a given Internal Entry Number (IEN) in the INSTITUTION (#4) file.	
Format:	`$$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`

14.1.18 PARENT^XUAF4(): Parent Institution Lookup

Reference Type: Supported

Category: Institution File

ICR #: 2171

Description: The PARENT^XUAF4 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); has the grave accent (`) in front of it.
- Station Number.
- Station Name.

type: (optional) Type of institution from the INSTITUTION ASSOCIATION TYPES (#4.05 file, 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 **type** input parameter, you get an array that only has one **PIEN** in it. If the **type** parameter is left blank, it finds *all* parents for the institution and lists them in the array.

14.1.19 $$PRNT^XUAF4(): Institution Parent Facility

Reference Type:	Supported	
Category:	Institution File	
ICR #:	2171	
Description:	The $$PRNT^XUAF4 extrinsic function returns the parent facility institution information in a caret-delimited string (ien^station_number^name) for a given child facility STATION NUMBER (#99) field in the INSTITUTION (#4) file.	
Format:	`$$PRNT^XUAF4(sta)`	
Input Parameters:	**sta**:	(required) The STATION NUMBER (#99) field value in the INSTITUTION (#4) file 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`

14.1.20 $$RF^XUAF4(): Realigned From Institution Information

Reference Type:	Supported	
Category:	Institution File	
ICR #:		
Description:	The $$RF^XUAF4 extrinsic function returns the information that is pointed to in the REALIGNED FROM (#.06) field in the HISTORY (#999) Multiple field in a caret-delimited string (ien^station_number^effective_date) for a given Internal Entry Number (IEN) in the INSTITUTION (#4) file.	
Format:	`$$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`

14.1.20.1 Example

Figure 87: $$RF^XUAF4 API—Example

```
>S IEN=$$RF^XUAF4(7020)

>W IEN
500^500^3000701
```

14.1.21 $$RT^XUAF4(): Realigned To Institution Information

Reference Type:	Supported	
Category:	Institution File	
ICR #:		
Description:	The $$RT^XUAF4 extrinsic function returns the information that is pointed to in the REALIGNED TO (#.05) field in the HISTORY (#999) Multiple field in a caret-delimited string (ien^station_number^effective_date) for a given Internal Entry Number (IEN) in the INSTITUTION (#4) file.	
Format:	`$$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`

14.1.21.1 Example

Figure 88: $$RT^XUAF4 API—Example

```
>S IEN=$$RT^XUAF4(500)

>W IEN
7020^528A8^3000701
```

14.1.22 SIBLING^XUAF4(): Sibling Institution Lookup

Reference Type: Supported

Category: Institution File

ICR #: 2171

Description: The SIBLING^XUAF4 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); has the grave accent (`) in front of it.
- Station Number.
- Station Name.

type: (optional) Type of institution from the INSTITUTION ASSOCIATION TYPES (#4.05) file (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 **type** input parameter, you get an array that only has one **PIEN** in it. If the **type** parameter is left blank, it finds *all* parents for the institution and lists them in the array. Also, the input site (i.e., **child** input parameter) is included in the list.

14.1.23 $$STA^XUAF4(): Station Number for IEN

Reference Type:	Supported	
Category:	Institution File	
ICR #:	2171	
Description:	The $$STA^XUAF4 extrinsic function returns the STATION NUMBER (#99) field for the entry of a given Internal Entry Number (IEN) in the INSTITUTION (#4) file.	
Format:	`$$STA^XUAF4(ien)`	
Input Parameters:	**ien**:	(required) Internal Entry Number (IEN) of the institution in question.
Output:	returns:	Returns the Station Number.

14.1.23.1 Example

Figure 89: $$STA^XUAF4API—Example

```
>S STA=$$STA^XUAF4(7020)

>W STA
528A8
```

14.1.24 $$TF^XUAF4(): Treating Facility (True/False)

Reference Type:	Supported	
Category:	Institution File	
ICR #:	2171	
Description:	The $$TF^XUAF4 extrinsic function, given the Internal Entry Number (IEN) in the INSTITUTION (#4) file, returns the Boolean value for the question—is this a medical treating facility?	
Format:	`$$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.

14.1.24.1 Example

Figure 90: $$TF^XUAF4 API—Example

```
>S TF=$$TF^XUAF4(7020)

>W TF
1
```

14.1.25 $$WHAT^XUAF4(): Institution Single Field Information

Reference Type:	Supported	
Category:	Institution File	
ICR #:	2171	
Description:	The $$WHAT^XUAF4 extrinsic function returns the data from a single field given the Internal Entry Number (IEN) and the specific field requested in the INSTITUTION (#4) file.	
Format:	`$$WHAT^XUAF4(ien,field)`	
Input Parameters:	**ien**:	(required) Internal Entry Number (IEN) of the institution in question (pointer value to the INSTITUTION (#4) file.
	field:	(required) field number of the field in question.
Output:	returns:	Returns the value in the specified field.

14.1.26 $$IEN^XUMF(): Institution IEN (Using IFN, Coding System, & ID)

Reference Type:	Supported	
Category:	Institution File	
ICR #:	3795	
Description:	The $$IEN^XUMF extrinsic function returns the Internal Entry Number (IEN) for a given Internal File Number (IFN), Coding System, and Identifier (**ID**).	
Format:	`$$IEN^XUMF(ifn,cdsys,id)`	
Input Parameters:	**ifn**:	(required) Internal File Number (IFN).
	cdsys:	(required) **CDSYS** is an existing coding system of the INSTITUTION (#4) file. To see the existing coding system in the file: `>D CDSYS^XUAF4(.Y)`
	id:	(required) **ID** is the identifier associated with the coding system. The station number, for example, is the identifier for the **VASTANUM** coding system and **NPI** number is the **ID** for the **NPI** coding system.
Output:	returns:	Returns the Internal Entry Number (IEN) of the institution requested.

14.1.27 MAIN^XUMFI(): HL7 Master File Message Builder

Reference Type: Controlled Subscription

Category: Institution File

ICR #: 2171

Description: The MAIN^XUMFI 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 the following segments:

- Master File Notification (MFN)
- Master File Query (MFQ)
- Master File Response (MFR)

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:	See MAIN^XUMFP	For a description of the Input parameters for this API, see the "MAIN^XUMFP(): Master File Parameters" API.
Output Parameters & Output:	See MAIN^XUMFP	For a description of the Output Parameters and Output for this API, see the "MAIN^XUMFP(): Master File Parameters" API.

14.1.27.1 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) implemented several Application Programming Interfaces (APIs). After the IFR patch was installed and the cleanup performed, the STATION NUMBER (#99) field was a unique key to the INSTITUTION (#4) file.

14.1.27.2 Example

Figure 91: MAIN^XUMFI API—Example

```
>D MAIN^XUMFI(4,18723,1,.PARAM,.ERROR)
```

From the HL7 MESSAGE TEXT (#772) file, you would see the following:

Figure 92: MAIN^XUMFI API—Sample Output

```
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
```

14.1.28 MAIN^XUMFP(): Master File Parameters

Reference Type:	Controlled Subscription	
Category:	Institution File	
ICR #:	3354	
Description:	The MAIN^XUMFP 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 developer can 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"** NEW entry (pass by value). Example: **IEN="NEW"**

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.

Input / Output Parameters:

param: Parameter array:

- PARAM("PROTOCOL"): IEN PROTOCOL (#101) file.
- PARAM("BROADCAST"): Broadcast message to all VistA sites.
- PARAM("LLNK"): Logical link in HLL("LINKS",*n*) format.

For more **param** array options, see the "Details" section.

Output: error: Returns:

```
1^Error message text
```

14.1.28.1 Details

Table 4: MAIN^XUMFP(): Master File Parameters API—QRD: Query Definition

Parameter	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

Table 5: MAIN^XUMFP(): Master File Parameters API—XCN Data Type of QRD WHO Parameter

Component	Value	Description
1ST component		One of the following:
NAME		Value of NAME (#.01) field 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.

Table 6: MAIN^XUMFP(): Master File Parameters API—CE Data Type of QRD WHAT Parameter

Component	Value	Description
1ST component	4	Identifier
2nd component	IFN	Text
3rd component	VA FM	Name of Coding System

Table 7: MAIN^XUMFP(): Master File Parameters API—MFI: Master File Identification

Parameter	Description
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

Table 8: MAIN^XUMFP(): Master File Parameters API—MFE: Master File Entry

Parameter	Description
PARAM("RLEC")	Record-Level Event Code
PARAM("MFNCID")	**MFN** Control ID
PARAM("MFEEDT")	Effective Date/Time
PARAM("PKV")	Primary Key Value

Table 9: MAIN^XUMFP(): Master File Parameters API—[Z...] Segments Parameters

Parameter	Description
PARAM("SEG",SEG)=""	HL7 segment name
PARAM("SEG",SEG,"SEQ",SEQ,FLD#)	segment sequence # and field

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*)**

Where:

- ***n*** is the component sequence number.
- ***X*** is the external value from VA FileMan.

P(segment_sequence,HLCS,n)=FM_external_value

Table 10: MAIN^XUMFP(): Master File Parameters API—Files Involving Sub-Records and Extended Reference

Parameter	Description
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)

For Example, **MFE PKV** node is:

^TMP("XUMF MFS",$J,"PARAM",IEN,"PKV")

14.1.28.2 Example

Figure 93 is an example of a query (**MFQ**) for a group records array:

Figure 93: MAIN^XUMFP API—Example

```
>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:

Figure 94: MAIN^XUMFP API—Displaying ^TMP Global for 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
```

15 Kernel Installation and Distribution System (KIDS): Developer Tools

15.1 KIDS Build-related Options

To get to the **KIDS: Kernel Installation & Distribution System** [XPD MAIN] menu (locked with the XUPROG security key) choose the **Programmer Options** [XUPROG] menu option on the **Kernel Systems Manager Menu** [EVE], as shown in Figure 95:

Figure 95: KIDS—Edits and Distribution Menu Options

```
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
          M 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:
```

15.2 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, see the "KIDS: System Management—Installations" and "KIDS: System Management—Utilities" sections in the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide*.

Table 11 provides a functional listing of the KIDS options supporting software application (package) export:

Table 11: KIDS—Options Supporting Software Application Builds and Exports

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

This section covers each of these tasks, describing how to accomplish the tasks using KIDS options.

15.2.1 Build Entries

KIDS stores the definition of a software application in the BUILD (#9.6) file. Individual entries in the BUILD (#9.6) file 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 (#9.6) file.

Unlike DIFROM, where you re-used the same PACKAGE (#9.4) file entry each time you exported a new version of a software application, with KIDS you create a new BUILD (#9.6) file 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**

Figure 96: KIDS—Choosing a Build Type Sample

```
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
```

The following KIDS options, described below, support creating and maintaining build entries:

- Create a Build Using Namespace
- Copy Build to Build
- Edit a Build

15.2.2 Create a Build Using Namespace

You can quickly create a build entry and populate its components by namespace. The **Create a Build Using Namespace** [XPD BUILD 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 namespaces and populates the build entry with all matches it finds on the system. You can then use the **Edit a Build** [XPD EDIT BUILD] option 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.

The following are Kernel 8.0 component types (listed alphabetically):

- Bulletin
- Dialog
- Form
- Function
- Help Frame
- HL7 Application Parameter
- HL Logical Link
- HL Lower Level Protocol
- Input Template
- List Template
- Mail Group
- Option
- Print Template
- Protocol
- Remote Procedure
- Routine
- Security Key
- Sort Template

Figure 97: KIDS—Populating a Build Entry by Namespace

```
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.
```

15.2.3 Copy Build to Build

You can create a new build entry based on a previous entry using the **Copy Build to Build** [XPD COPY 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.

Figure 98: KIDS—Copying a Build Entry

```
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.
```

15.2.4 Edit a Build

Using the Edit a Build option [XPD EDIT BUILD], 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 option [XPD EDIT BUILD]. The following sections describe in detail each part of a build entry and how you can edit each part.

15.2.4.1 KIDS Build Screens

KIDS Build Screens are designed in conjunction with the **Edit a Build** option to help you plan your build entries.

Table 12: KIDS—Functional Layout, Edit a Build

Screen	Build Section	Build Sub-Section
Screen 1	Build Name	
	Date Distributed	
	Description	
	Environment Check Routine	
	Pre-Install Routine	
	Post-Install Routine	
	Pre-Transportation Routine	
Screen 2	Files and Data	Partial DD Definition
		Send Data Definition
Screen 3	Build Components	Print Template
		Sort Template
		Input Template
		Form
		Function
		Dialog
		Bulletin
		Mail Group
		Help Frame
		Routine
		Option
		Security Key
		Protocol
		List Template
		HL7 Application Parameter
		HL Lower Level Protocol
		HL Logical Link
		Remote Procedure

Screen	Build Section	Build Sub-Section
Screen 4	Install Questions	
	Required Builds	
	Package File Link	
	Package Tracking	

15.2.4.2 Edit a Build: Name & Version, Build Information

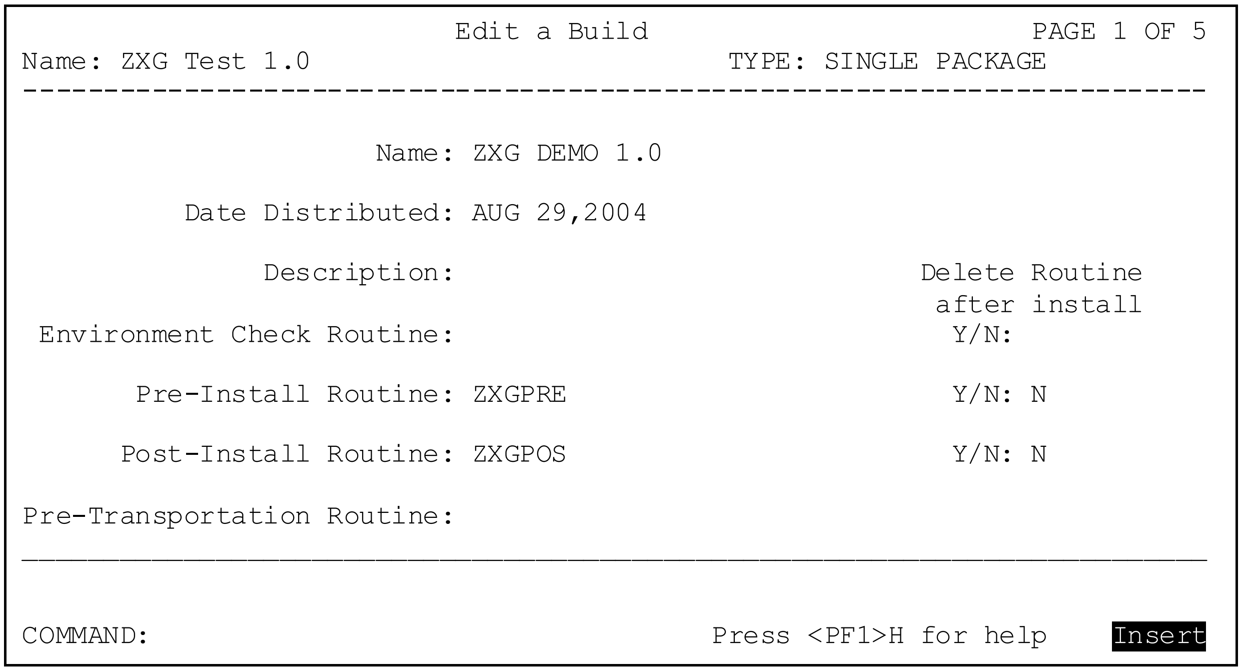
When you invoke the **Edit a Build** [XPD EDIT BUILD] option, KIDS loads a four-page ScreenMan form. The first screen of the form lets you edit the following software application settings:

- Name
- Date Distributed
- Description
- Environment Check Routine
- Pre-Install Routine
- Post-Install Routine
- Pre-Transportation Routine

15.2.4.2.1 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 (#9.6) file. 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.

Figure 99: KIDS—Screen 1 of Edit a Build Sample



```
                           Edit a Build                         PAGE 1 OF 5
Name: ZXG Test 1.0                           TYPE: SINGLE PACKAGE
---------------------------------------------------------------------------

                      Name: ZXG DEMO 1.0

          Date Distributed: AUG 29,2004

               Description:                            Delete Routine
                                                        after install
 Environment Check Routine:                               Y/N:

       Pre-Install Routine: ZXGPRE                        Y/N: N

      Post-Install Routine: ZXGPOS                        Y/N: N

Pre-Transportation Routine:
___________________________________________________________________________


COMMAND:                                    Press <PF1>H for help    Insert
```

15.2.4.3 Edit a Build: Files

The second screen of the **Edit a Build** option 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.

15.2.4.3.1 Data Dictionary Update

The installing site is *not* asked whether they want to override data dictionary updates; data dictionary updates are determined entirely by how the developer exports 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:

- **YES**—If you answer **YES** to Update the Data Dictionary, the data dictionary is updated at the installing site.
- **NO**—If you answer **NO** to Update the Data Dictionary, 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 does *not* 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).

15.2.4.3.2 Sending Security Codes

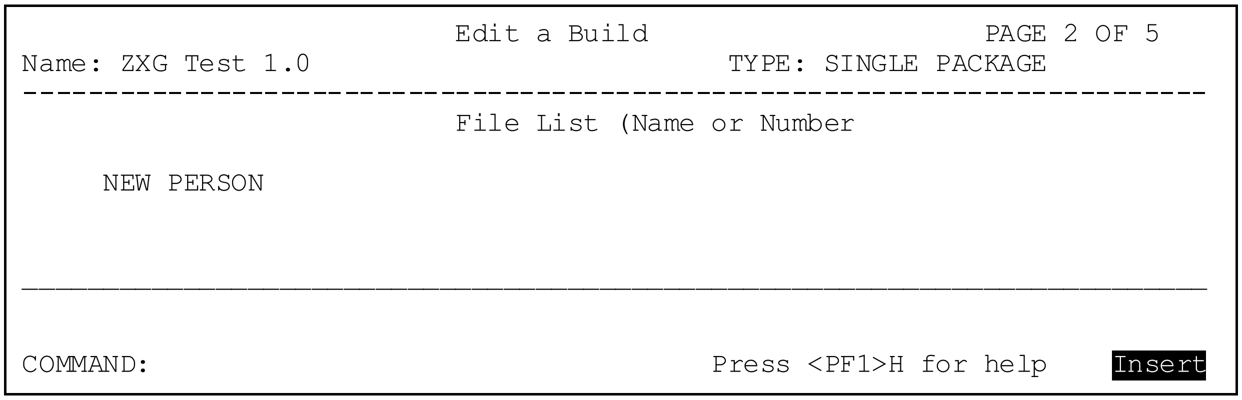
With KIDS, you can specify on a file-by-file basis whether to send security codes. For each file, you can set SEND SECURITY CODE to either **YES** or **NO**.

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 do *not* already exist), however. Security codes for a file are *not* updated at the installing site if the file already exists.

NOTE: Use VA FileMan's FILESEC^DDMOD API to set the security access codes for an existing file.

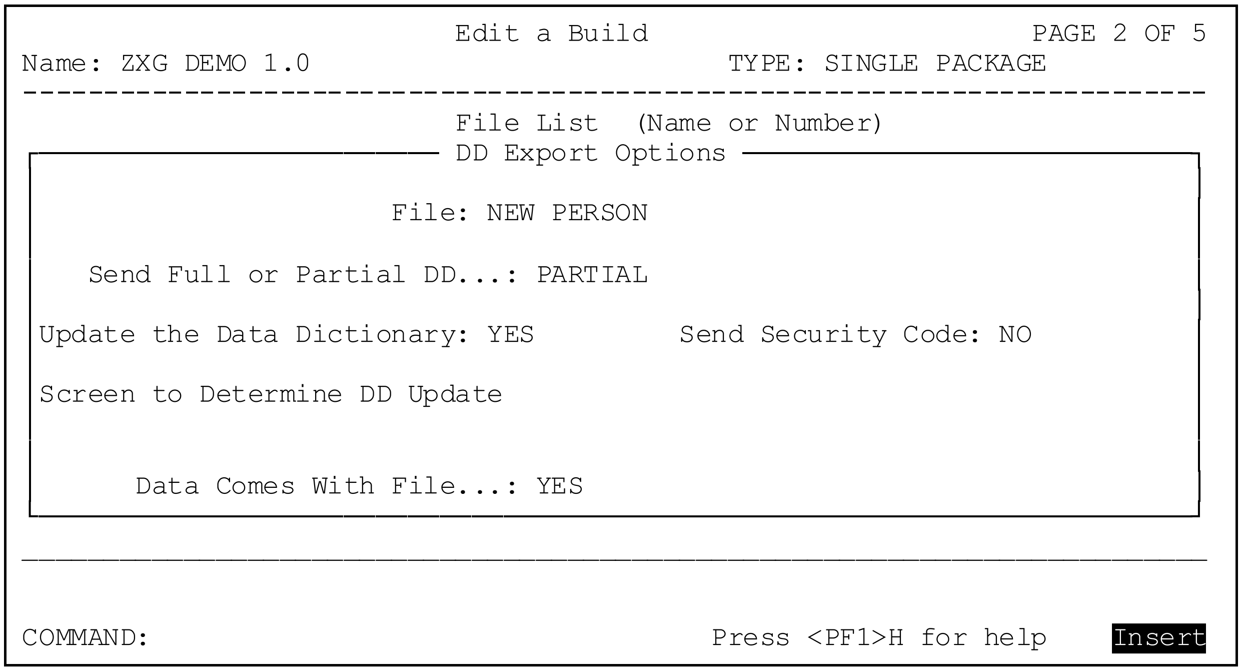
REF: For more information on the FILESEC^DDMOD API, see the "Database Server (DBS) API" section in the VA FileMan Developer's Guide located on the VDL at: https://www.va.gov/vdl/application.asp?appid=5

Figure 100: KIDS—Screen 2 of Edit a Build: Selecting Files



```
                               Edit a Build                          PAGE 2 OF 5
Name: ZXG Test 1.0                                       TYPE: SINGLE PACKAGE
--------------------------------------------------------------------------------
                               File List (Name or Number

     NEW PERSON



________________________________________________________________________________


COMMAND:                                          Press <PF1>H for help    Insert
```

Figure 101: KIDS—Data Dictionary and Data Settings



```
                            Edit a Build                        PAGE 2 OF 5
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
```

15.2.4.3.3 Sending Full or Partial Data Dictionaries

KIDS supports sending out either of the following:

- **Full Data Dictionaries (DD)**—Entire file definition.
- **Partial Data Dictionaries (DD)**—Specified fields in a file.

15.2.4.3.4 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.

15.2.4.3.5 Partial DD (Some Fields)

You can only send a partial DD if the file already exists at the site. If you answer **PARTIAL** at the "Send Full or Partial DD" prompt, KIDS lets you choose what data dictionary levels to export.

In the Data Dictionary Number popup window (Figure 103), you can select either one of the following types:

- **File Number**—Top level of the file.
- **Multiple**—Sub-data dictionary number (also known as a subfile). You can export any Multiple, no matter how deep (every Multiple's data dictionary number is selectable).

15.2.4.3.5.1 File Number Level

In the Field Number popup window (Figure 104), if you selected the file number type, you can select which fields to export at that data dictionary level:

- **If you do *not* specify *any* fields, *no* fields are sent**.
- **If you do specify fields, only the specified fields are sent.** You *cannot* choose any Multiples at this data dictionary level.

15.2.4.3.5.2 Multiple Level

In the Field Number popup window (Figure 104), if you selected the Multiple (sub-data dictionary number) type, you can select which fields to export at that sub-data dictionary level:

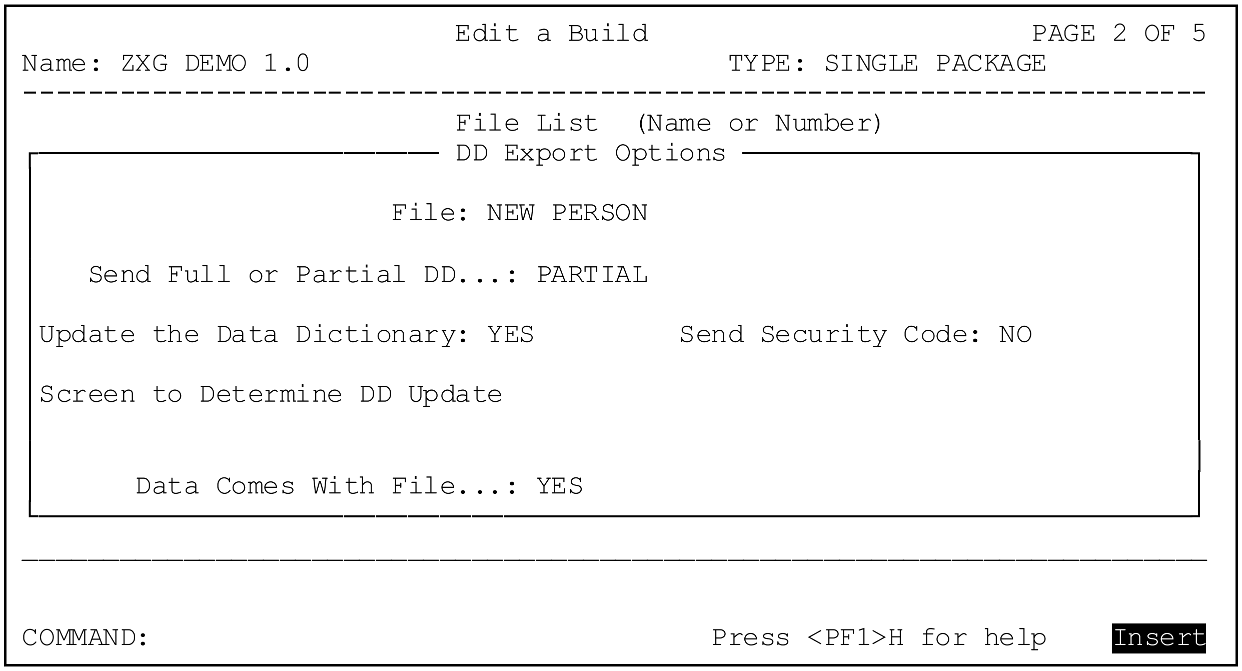
- **If you do *not* specify *any* fields, *all* fields are sent.** All fields at this level and their descendants are exported. You *must* do this if the Multiple is *new* at the site.
- **If you do specify fields, only the specified fields are sent.**

Unlike **DIFROM**, KIDS does *not* require sending the **.01** field of the file if you send a partial data dictionary.

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 is *not* installed.

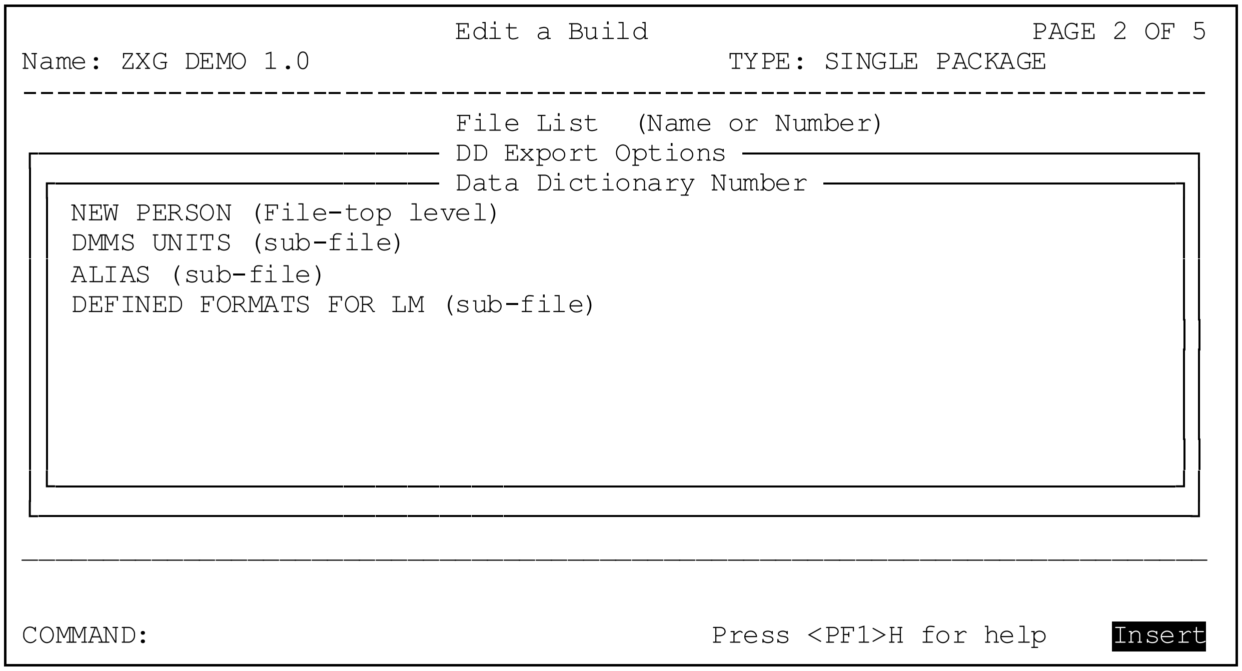
NOTE: Certain attributes (Identifiers, "**ID**" nodes, etc.) are considered file attributes (as opposed to field attributes), and so are sent only when you send a full DD. They are *not* sent with a partial DD.

Figure 102: KIDS—Data Dictionary Settings Screen—DD Export Options



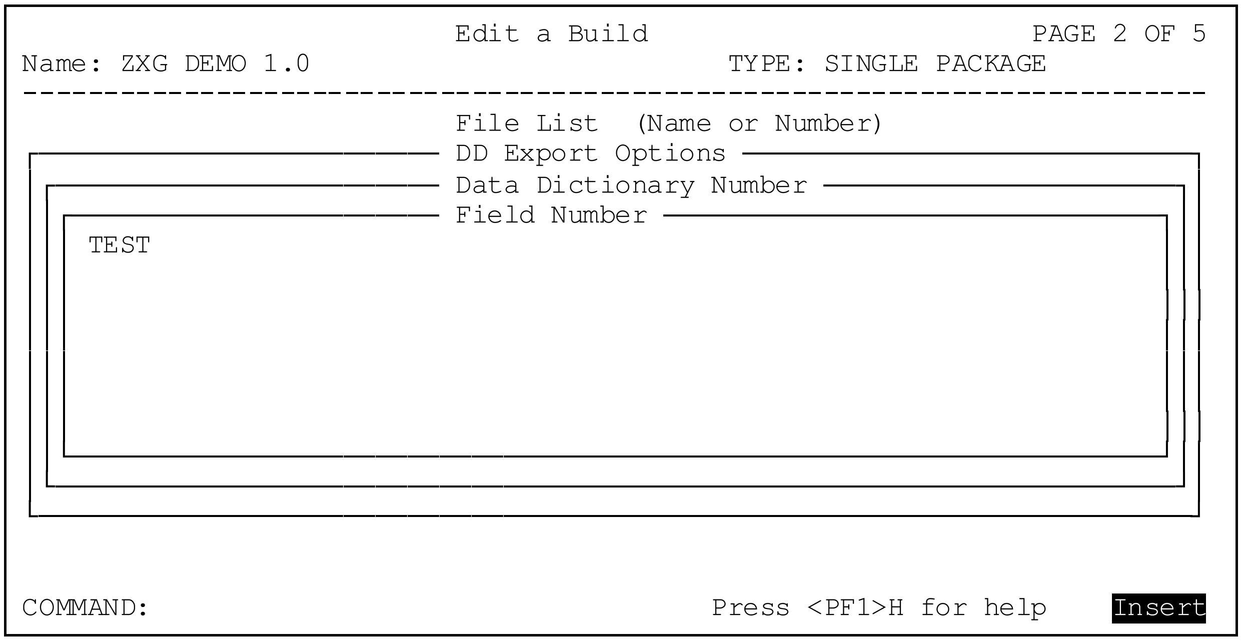
```
                           Edit a Build                          PAGE 2 OF 5
Name: ZXG DEMO 1.0                                  TYPE: SINGLE PACKAGE
--------------------------------------------------------------------------
                           File List  (Name or Number)
 ┌───────────────────────── 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 103: KIDS—Partial DD: Choosing DD Levels (Top Level and Multiple) to Send; Data Dictionary Number Level



```
                           Edit a Build                          PAGE 2 OF 5
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 104: KIDS—Partial DD: Choosing DD Levels (Top Level and Multiple) to Send; Field Number Level



```
                              Edit a Build                       PAGE 2 OF 5
Name: ZXG DEMO 1.0                                 TYPE: SINGLE PACKAGE
--------------------------------------------------------------------------
                              File List  (Name or Number)
┌─────────────────────────── DD Export Options ──────────────────────────┐
│┌────────────────────────── Data Dictionary Number ───────────────────┐│
││┌───────────────────────── Field Number ──────────────────────────┐││
│││ TEST                                                            │││
│││                                                                 │││
│││                                                                 │││
│││                                                                 │││
│││                                                                 │││
│││                                                                 │││
││└─────────────────────────────────────────────────────────────────┘││
│└─────────────────────────────────────────────────────────────────────┘│
└────────────────────────────────────────────────────────────────────────┘


COMMAND:                                          Press <PF1>H for help    Insert
```

15.2.4.3.6 Choosing What Data to Send with a File

When you send data, you can send all of the data in a file; however, 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**:

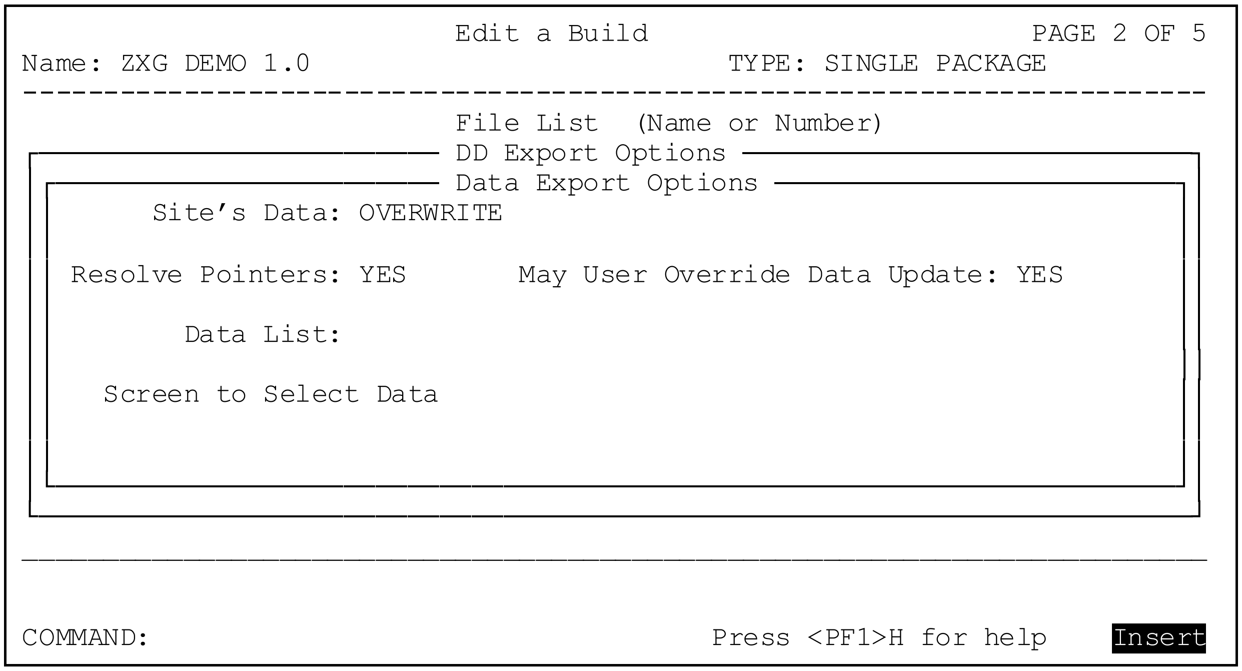
- If **$T** is set to **1**, the entry is sent.
- If **$T** is set to **0**, the entry is *not* sent.

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 are 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.

Figure 105: KIDS—Settings for Sending Data



```
                           Edit a Build                          PAGE 2 OF 5
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
```

15.2.4.3.7 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. Table 13 lists the four ways KIDS can install file entries at the receiving site:

Table 13: KIDS—Data Installation Actions

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. 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 are *not* part of the incoming data dictionary are preserved. 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. 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 are *not* in the incoming data dictionary are *not* restored.

You can specify different settings for separate files; within a file, however, all data *must* be installed in one of these four ways.

You can give the installing site the choice of overriding the data update. If you set May User Override Data Update to **YES**, the installing site has the choice of whether to bring in data that has been sent with this file. They are *not* given the choice of how to install data, however (add only if new file vs. merge vs. **OVERWRITE** vs. **REPLACE**). If you set this field to **NO**, the installing site cannot override bringing in data.

15.2.4.3.8 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 (#.7) file. KIDS uses this length [for example, **63** (default) or **99**] as the limit of characters to compare.

If a match (either by the **B** cross-reference or by the first piece of the **zero** node) is *not* found, the incoming entry is considered new and is added to the file. If a match or matches are found, two additional checks are made to determine whether any of the existing entries are a match.

KIDS next checks whether the IENs of any tentatively matched entries are related. If the file has a defined **.001** field, the IEN is a meaningful attribute of an entry. In this case, the IENs *must* match. If the input transform of the **.01** field contains **DINUM**, it operates the same way as a **.001** field. If the IEN is meaningful, and no match is found, the incoming entry is considered new and is added to the file.

If the possibility of a match remains after checking IENs, KIDS performs a final check based on identifiers.

A well-designed file uses one or more identifiers to act as key fields, so that each entry is unique with respect to name and identifiers. If identifiers exist on either the target file or the incoming data dictionary, KIDS checks the values of all such identifier fields. The value of each identifier field *must* be the same for the existing entry and the incoming entry to be considered a match. Only the internal value of the identifier field is checked (so if an identifier is a pointer field, problems could result). Only identifiers that have valid field numbers are used in this process.

If there is still more than one matching entry after checking **.01** fields, IENs, and identifiers, the lowest numbered entry in the site's file is considered a match for the incoming entry for the file. On the other hand, if no match is found after checking **.01** fields, IENs, and identifiers, the entry is considered new and is added to the file.

15.2.4.3.9 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 do *not* 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 *cannot* 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.

15.2.4.3.10 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* traditional cross-references and *all* new-style indexes with an ACTIVITY that contains an **I**, for *all* the records in the file. Only the **SET** logic is executed.

15.2.4.3.11 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 does *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.

15.2.4.4 Edit a Build: Components

In the third screen in the **Edit a Build** [XPD EDIT 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 (**-**).

To add an entry to the list when a similarly named entry already exists in the list, use the normal VA FileMan convention of surrounding the entry with quotes. For example, to add **ZZTK** to the list when **ZZTK1** already exists in the list, enter **"ZZTK"** in quotes.

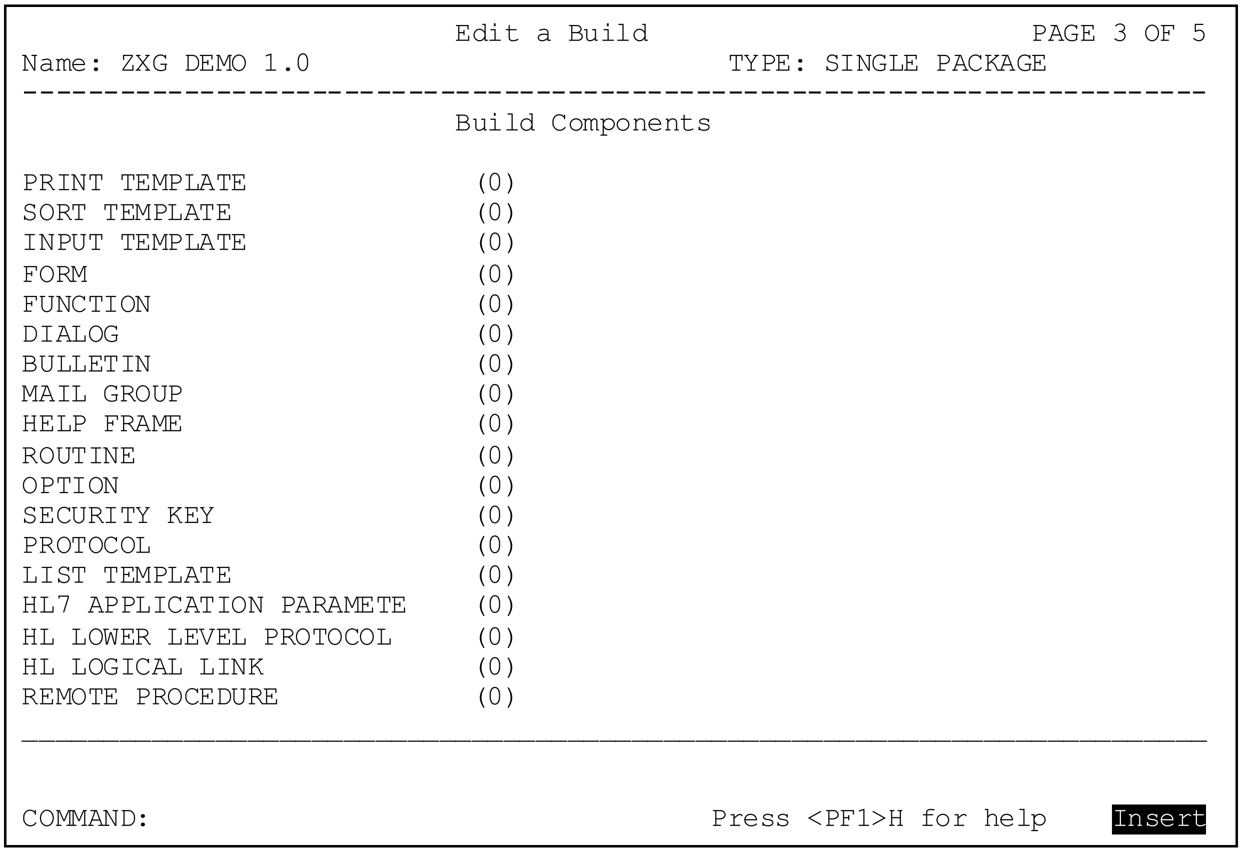
With most component types, the permissible installation actions are:

- **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, see the "Create a Build Using Namespace" section.

Figure 106: KIDS—Screen 3 of Edit a Build: Components



```
                           Edit a Build                       PAGE 3 OF 5
Name: ZXG DEMO 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
```

NOTE: This is an expanded view of this screen in order to show you all of the currently available component types. You have to scroll through the list in order to see all of the available types.

15.2.4.5 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 8.0, you can no longer send out an option with an attached scheduling frequency. Scheduling of options was moved out of the OPTION (#19) file and into the OPTION SCHEDULING (#19.2) file. One advantage to this is that a developer's scheduling settings no longer overwrites 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.

Table 14: KIDS—Option and Protocol Installation Actions

Option/Protocol Installation Action	Description
SEND TO SITE	Menu, option, or protocol is installed at the site; any existing version already at the site is completely purged beforehand, except those options that are currently marked as "Out of Order" (OoO). **NOTE:** The OUT OF ORDER MESSAGE field (aka OoO field) in the OPTION (#19) file is updated by KIDS during an install. When an option or protocol is sent, KIDS allows the site to disable them during the install. That means KIDS adds the OoO field at the beginning of the install and removes it at the end. In the case where the OoO already exists for an option, KIDS does nothing. Because of this, KIDS does *not* transport the OoO field. If a developer wants to add or change an OoO, they should use the OUT^XPDMENU(): Edit Option's Out of Order Message API during the post-install.
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 do *not* match incoming items are left as is. Any items that do match incoming items are completely replaced by the incoming items. The advantage with this action is that it preserves locally added items at the site. The disadvantage is that if you have removed items, the removed items are *not* purged at the site.
ATTACH TO MENU	Designates an option or protocol, *not* exported to the site, to be attached to a menu that is exported. This is

Option/Protocol Installation Action	Description
	used when a menu is sent by KIDS to a site and the developer wants the local option or protocol attached to the menu. The option or protocol is *not* exported to the site; instead, its name is sent and the local option or protocol becomes a sub-item on the menu that is sent.
DISABLE DURING INSTALL	Designates an option or protocol that is *not* exported to be disabled during the KIDS install process.

15.2.4.6 Edit a Build: Routines

Routine selection is done based on pointers to entries in the ROUTINE (#9.8) file, but this file is *not* automatically updated when programs are saved and deleted on an M system. So, before adding routines to a build entry, you should run KIDS' **Update Routine File** option. Be sure to update all the routines and routine namespaces that you 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***.

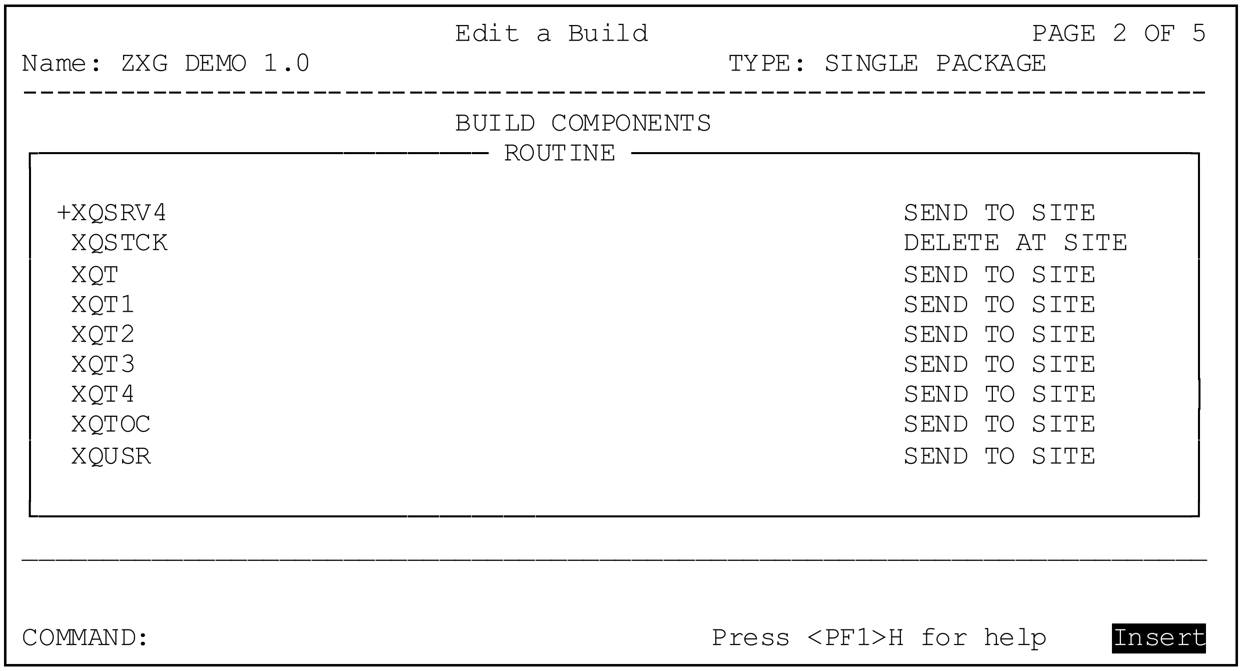
For each routine, you can choose one of two actions:

- **SEND TO SITE** (default)
- **DELETE AT SITE**

The default action is **SEND TO SITE**. If you choose **DELETE AT SITE**, the routine is 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 are installed (or deleted) across all CPUs the site selects. KIDS displays 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.

Figure 107: KIDS—Choosing Routines



```
                          Edit a Build                        PAGE 2 OF 5
Name: ZXG DEMO 1.0                                  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
```

15.2.4.7 Edit a Build: Dialog Entries (DIALOG [#.84] File)

VA FileMan supports the capability for other software applications to store their dialog in the VA FileMan DIALOG (#84) file. Some advantages to using the DIALOG (#.84) file for user interaction include:

- Separating user interaction from other program functionality. This is a helpful step for creating GUI interfaces.
- Reusing dialog. When dialog is stored in the DIALOG (#.84) file, it can be re-used.
- Easily generating software application error lists. If error lists are stored in DIALOG (#.84) 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 (#.84) file. 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 (#.84) file, see the *VA FileMan Developer's Guide*.

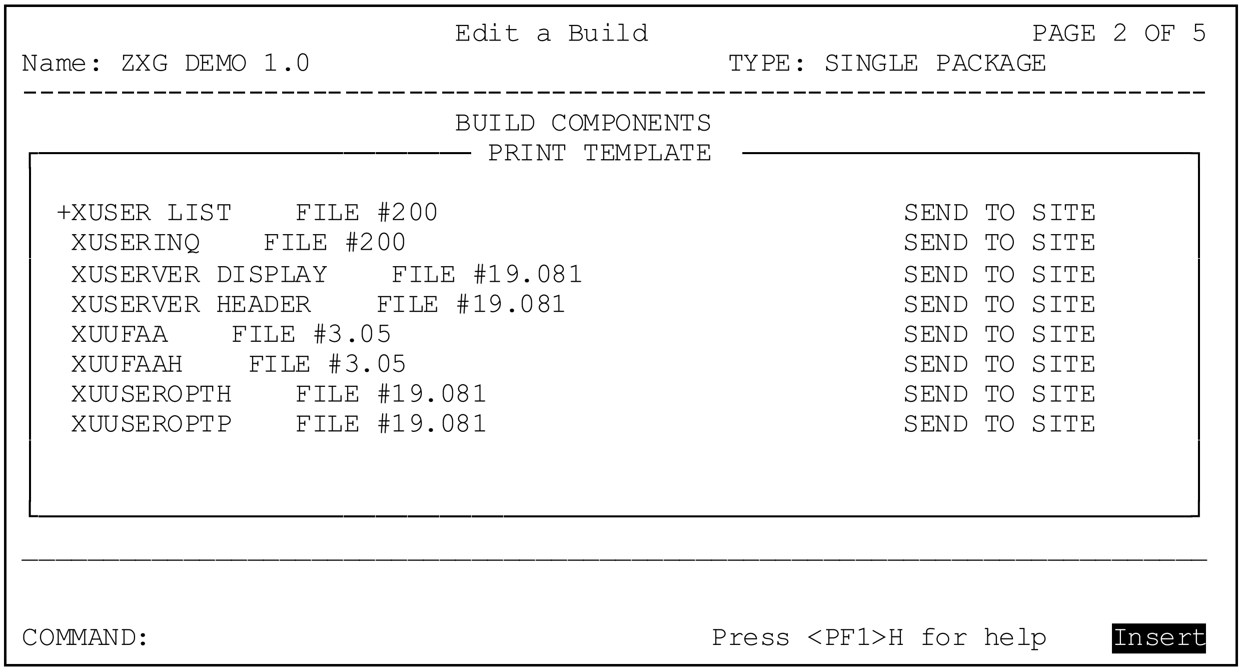
15.2.4.8 Edit a Build: Forms

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 have chosen the form.

15.2.4.9 Edit a Build: Templates

When you select 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).

Figure 108: KIDS—Selecting Templates



```
                          Edit a Build                          PAGE 2 OF 5
Name: ZXG DEMO 1.0                            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
```

15.2.5 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 a Distribution** [XPD TRANSPORT PACKAGE] 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.

Figure 109: KIDS—Transport a Distribution Option: Creating a Distribution Sample User Dialogue

```
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:
```

If you do *not* 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 have 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.

Figure 110: KIDS—Transport a Distribution Option: Sending via Network (PackMan Message) Sample User Dialogue

```
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 XUUSER,FIVE at REDACTED.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: XUUSER,FIVE     Last used MailMan: 04 Oct 96 15:28
  Select basket to send to: IN// <Enter>
And send to: <Enter>
```

15.2.5.1 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.

15.2.5.2 Multi-Package Builds

Multi-Package builds contain a list of other builds and lists their installation order. A Multi-Package build transports this list of builds (template or meta-build).

Figure 111: KIDS—Multi-package Builds Sample

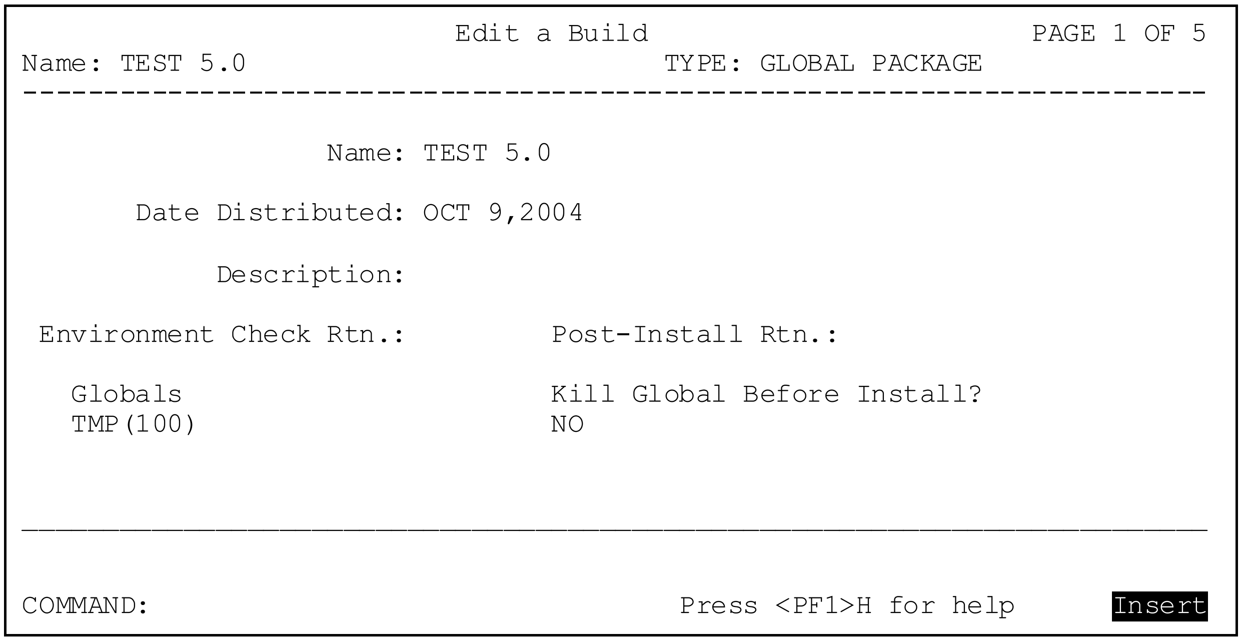
```
                           Edit a Build                       PAGE 1 OF 5
Name: TEST 3.0                         TYPE: MULTI-PACKAGE
---------------------------------------------------------------------------

                     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
```

15.2.5.3 Exporting Globals with KIDS

KIDS in Kernel 8.0 supports the installation of global distributions (distributions that export globals). KIDS 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 **KILL**ing 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, see the "KIDS: System Management—Installations" section in the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide*.

Figure 112: KIDS—Exporting Global Distributions Sample



```
                          Edit a Build                         PAGE 1 OF 5
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
```

15.2.6 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 did *not* 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.

15.3 Advanced Build Techniques

The previous sections in this section introduced KIDS from the developer's perspective, describing the basics of how to create build entries and how to transport distributions. This section describes advanced build techniques that developers can use when creating builds. The following subjects are covered:

- Environment Check Routine
- PRE-TRANSPORTATION ROUTINE (#900) Field
- Pre- and Post-Install Routines: Special Features
- 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

15.3.1 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 is 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 (Figure 67). Any routine that is specified is automatically sent by KIDS. You do *not* have to list the routine in the Build Components section (Figure 56).

15.3.1.1 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 is 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 do any of the following:

- Continue installing the software application.
- Abort installing the software application.
- 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(): Output a Message and BMES^XPDUTL(): Output a Message with Blank Line APIs, during the environment check routine you should use the ^DIR API for **READ**s and can use direct **WRITE**s.

15.3.1.2 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** [XPD LOAD DISTRIBUTION] option).

KIDS runs the environment check a second time when the user runs the **Install Package(s)** [XPD INSTALL BUILD] option to install the software applications in the loaded distribution.

The KIDS key variable **XPDENV** indicates in which phase (load or install) the environment check is running.

REF: For more information on **XPDENV**, see the "Key Variables during Environment Check" section.

15.3.1.3 Key Variables during Environment Check

Table 15Table 15 lists the key variables during the environment check (listed alphabetically):

Table 15: KIDS—Key Variables during the Environment Check (listed alphabetically)

Variable	Description
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.
XPDENV	The KIDS key variable **XPDENV** is available during the environment check only. It can have the following values: • **1—**The environment check is being run by the KIDS **Install Package(s)** option. • **0—**The environment check is being run by the KIDS **Load a Distribution** option. You can use **XPDENV** if, for example, there is a check that is valid to perform at install time, but *not* at load time.
XPDGREF	This variable contains the location of developer-defined information that is used during the install process. The developer can reference this information using the following syntax:

Variable	Description
	`S X=0,X=$O(@XPDGREF@(namespace,X))`
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 (#9.6) file entry, which is software application name, concatenated with a space, concatenated with version number.
XPDNM("SEQ")	The **XPDNM("SEQ")** variable is available during the pre- and post-install and environment check phases of a KIDS installation. **XPDNM("SEQ")** is set to one of the following values: • **Sequence Number—**If build is a patch and the National Patch Module (NPM) created a sequence number. • **NULL**. **NOTE:** This variable was released with Kernel Patch XU*8.0*559.
XPDNM("TST")	The **XPDNM("TST")** variable is available during the pre- and post-install and environment check phases of a KIDS installation. **XPDNM("TST")** is set to one of the following values: • **Test Number—**If build is a patch and the National Patch Module (NPM) created a test number. • **NULL**. **NOTE:** This variable was released with Kernel Patch XU*8.0*559.

15.3.1.4 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:

- $$VER^XPDUTL
- $$VERSION^XPDUTL

REF: For more on these APIs, see the "Application Programming Interface (API)" section in this section.

15.3.1.5 Telling KIDS to Skip Installing or Delete a Routine

During the environment check, you can tell KIDS to skip installing any routine, and change a routine's installation status to **DELETE AT SITE**.

For example, suppose you have one version of a routine for GT.M sites and one version for Caché sites. Based on the type of system your environment check finds, you can use the $$RTNUP^XPDUTL(): Update Routine Action function to tell KIDS which routines to skip installing.

REF: For more information on deleting environment check routines, see the "Key Parameters during Pre- and Post-Install Routines" section in this section.

15.3.1.6 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(): Verify Patch Installation function to have KIDS alert the user that a required patch is *not* installed on their system.

15.3.1.7 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 have chosen to abort the install.

Table 16 lists ways you can ask KIDS to continue or abort an installation, based on the conclusions of your environment check routine:

Table 16: KIDS—Actions Based on Environment Check Conclusions

KIDS Desired Action (Based on Environment Check Conclusions)	How to Tell KIDS to Take Action
OK to install this transport global.	(Take no action)
Do *not* install this transport global and **KILL** it from **^XTMP**.	`>S XPDQUIT =1`
Do *not* 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`

KIDS Desired Action (Based on Environment Check Conclusions)	**How to Tell KIDS to Take Action**
Abort all transport globals in distribution and **KILL** them from **^XTMP**.	`>S XPDABORT=1`
Abort all transport globals in distribution but leave them in **^XTMP**.	`>S XPDABORT=2`

NOTE: It is recommended that 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.

15.3.1.8 Controlling the Queuing 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 sees the following prompt and *not* be allowed to enter **Q**:

Figure 113: KIDS—Dialogue when the XPDNOQUE Variable is Set to Disable Queuing

```
Enter the Device you want to print the Install messages.
Enter a '^' to abort the install.

DEVICE: HOME//
```

15.3.1.9 Controlling the Disable Options/Protocols Prompt

By default, KIDS asks the following question during KIDS installations:

Figure 114: KIDS—"DISABLE" Default Prompt during Installations

```
Want to DISABLE Scheduled Options, Options, and Protocols? YES//
```

You can control the way this question is asked by defining the array **XPDDIQ("XPZ1")** during the environment check. The environment check runs 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):

Table 17: KIDS—Installation: XPDDIQ Array Sample

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**).

15.3.1.10 Controlling the Move Routines to Other CPUs Prompt

By default, KIDS asks the following question during KIDS installations:

Figure 115: KIDS—"MOVE routines" Default Prompt during Installations

```
Want to MOVE routines to other CPUs? NO//
```

You can control the way this question is asked by defining the array **XPDDIQ("XPZ2")** during the environment check. The environment check runs twice (once during load and once during installation), but setting this array only has an effect during the installation. Therefore, you may want to define the array only when **XPDENV=1**. You can use this array as follows (each node is optional):

Table 18: KIDS—Environment Check—XPDDIQ Array Sample

Array Node	Description
XPDDIQ("XPZ2")	(optional) Set to: • **Zero (0)—**Force answer to **NO**. • **1—**Force answer to **YES**. When this node is set, the question is *not* 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**).

Figure 116: KIDS—Environment Check Routine Sample

```
ZZUSER1     ;SFISC/REDACTED - 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","^%ZUA" W !,"Checking ",Y S
@(Y_"=$G("_Y_")")
     Q
```

15.3.2 PRE-TRANSPORTATION ROUTINE (#900) Field

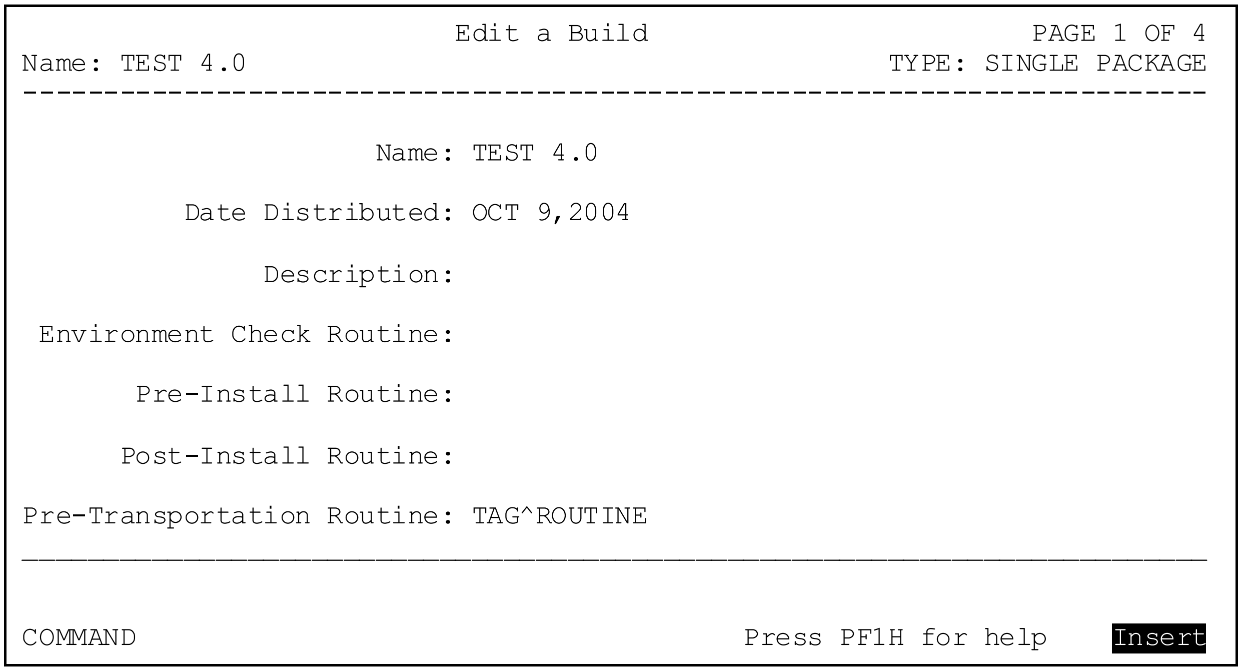
The PRE-TRANSPORTATION ROUTINE (#900) field in the BUILD (#9.6) file contains a [TAG^]ROUTINE that is run during the transportation process for the Build. This allows developers to populate the transport global using the **XPDGREF** variable.

Developers can put information in the KIDS Transport Global, which can be used by the Pre-install, Environment Check, and/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-install, Environment Check, and/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.

Figure 117: KIDS—PRE-TRANSPORTATION ROUTINE Field Sample



```
                           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
```

15.3.3 Pre- and Post-Install Routines: Special Features

KIDS, like DIFROM, lets you specify pre-install and post-install routines. Typically, the pre- and post-install routines are used to perform pre-install and post-install conversions. This section describes how to use pre- and post-install routines with KIDS installations.

Pre- and post-routines are optional; you 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 (Figure 117). Any routine that is specified is automatically sent by KIDS. You do *not* have to list the routine in the Build Components section (Figure 106).

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

Do *not* set up variables during the pre-install for use during the installation or the post-install, because these variables are 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.

15.3.3.1 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.

15.3.3.2 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. VA FileMan's $$GET1^DID can be used to retrieve the content of the node and PRD^DILFD API updates the node.

REF: For more information, see the *VA FileMan Developer's Guide.*

15.3.3.3 Key Parameters during Pre- and Post-Install Routines

Table 19: KIDS—Key Parameters during the Pre- and Post-install Routines

Parameter	Description
XPD NO_EPP_DELETE	If this parameter is set to **1**, KIDS does *not* delete any environment check Pre and Post routines, regardless if the Environment Check routine is marked as "**DELETE AT SITE**." By default, this parameter is set to **1** (do *not* delete), so support personnel are able to look at those routines for troubleshooting purposes.

15.3.3.4 Key Variables during Pre- and Post-Install Routines

Table 20: KIDS—Key Variables during the Pre- and Post-install Routines

Variable	Description
XPDNM	The **XPDNM** variable is available during the pre- and post-install and 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 (#9.6) file entry, which is software application name, concatenated with a space, concatenated with version number.
XPDNM("TST")	Released with Kernel Patch XU*8.0*559, the **XPDNM("TST")** variable is available during the pre- and post-install and environment check phases of a KIDS installation. **XPDNM("TST")** is set to one of the following values: • **Test Number—**If build is a patch and the National Patch Module (NPM) created a test number. • **NULL**.
XPDNM("SEQ")	Released with Kernel Patch XU*8.0*559, the **XPDNM("SEQ")** variable is available during the pre- and post-install and environment check phases of a KIDS installation. **XPDNM("SEQ")** is set to one of the following values: • **Sequence Number—**If build is a patch and the National Patch Module (NPM) created a sequence number. • **NULL**.
DIFROM	For the purpose of backward compatibility, the **DIFROM** variable 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.

Variable	Description
ZTQUEUED	If the **ZTQUEUED** variable is: • Present—You know that you are running as a queued installation. • *Not* present—You know that the installer chose to run the installation directly instead of queuing it.

15.3.3.5 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.

15.3.3.6 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, see the "Installation Progress" section in the "KIDS Systems Management Installations" section in the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide.*

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(%)
```

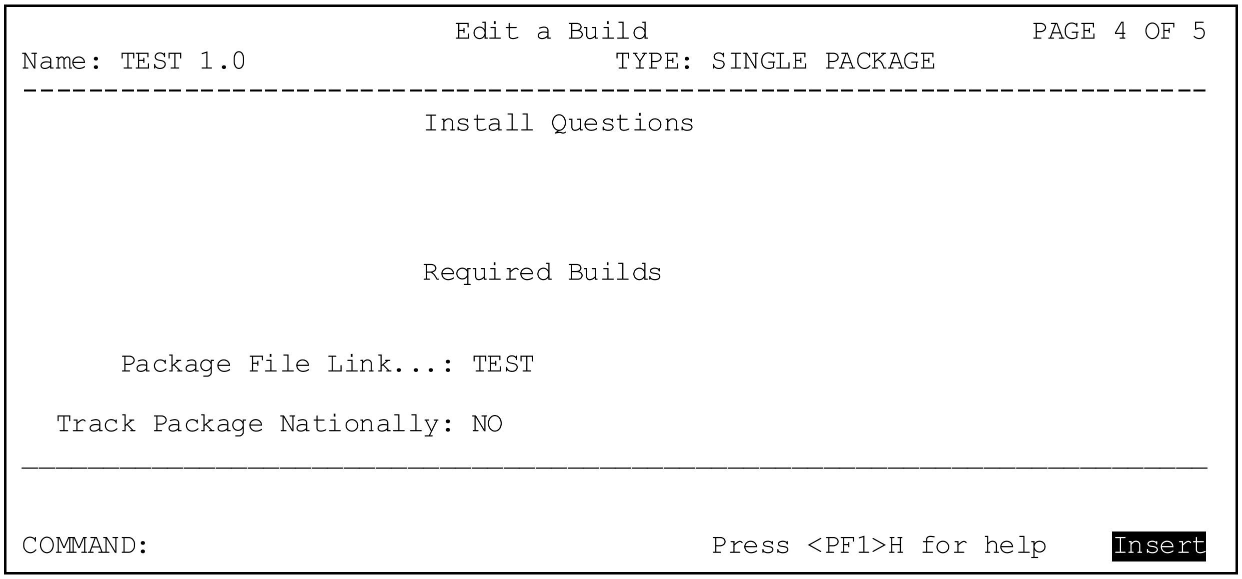
If you wish to display a status bar at various intervals throughout your Pre or Post-install routines, you should reset the status bar. To reset the status bar enter the following at the programmer prompt:

```
>SET XPDIDTOT=0
>D UPDATE^XPDID(0)
```

15.3.4 Edit a Build—Screen 4

Screen four of the **Edit a Build** [XPD EDIT BUILD] option is where you can set up the install questions, any required builds, PACKAGE (#9.4) file links, and tracking software application information for a build.

Figure 118: KIDS—Screen 4 of Edit a Build Sample



```
                            Edit a Build                       PAGE 4 OF 5
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
```

15.3.5 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 the **Edit a Build** [XPD EDIT 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:

Table 21: KIDS—DIR Input Values for KIDS Install Questions

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.

REF: For information on the purpose of these variables, permissible values for them, and which are required versus which are optional, see the *VA FileMan Developer's Guide*.

15.3.5.1 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—PRE***xxx***
- Post-Install Questions—POS***xxx***

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 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.

15.3.5.2 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 VA FileMan ^DIR call. The purpose of this line of M code is so that you can modify the **DIR** variables, 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 are loaded on the installing system.

15.3.5.3 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**).

15.3.5.4 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.

Figure 119: KIDS—Pre-install Question (Setting Up) Sample

```
                          Edit a Build                           PAGE 4 OF 5
┌──────────────────────── 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 120: KIDS—Appearance of Question during Installation

```
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//
```

15.3.5.5 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.

15.3.6 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 is an error during the pre- or post-install, and you use checkpoints, when the sites restart the installation, it resumes 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.

15.3.6.1 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 the $$NEWCP^XPDUTL(): Create Checkpoint function 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 in Table 22 (listed in alphabetic order):

Table 22: KIDS—Functions Using Checkpoints *with* Callbacks

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.)

15.3.6.2 Checkpoint Parameter Node

You can store how far you have progressed with a task you are performing in the callback by using a checkpoint parameter node. The $$UPCP^XPDUTL(): Update Checkpoint function updates the value of a checkpoint's parameter node; the $$PARCP^XPDUTL(): Get Checkpoint Parameter 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 are 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 the example in Figure 121 is PRE^ZZUSER2; the post-install API is POST^ZZUSER2.

Figure 121: KIDS—Using Checkpoints *with* Callbacks: Combined Pre- and Post-install Routine

```
ZZUSER2     ;TEST 1.0 PRE AND POST INSTALL
     ;;1.0
     ;build checkpoints for PRE
PRE  N %
     S %=$$NEWCP^XPDUTL("ZZUSER1","PRE1^ZZUSER2","C-")
     Q
PRE1 ;check terminal type file
     N DA,UPDATE,NAME
     ;quit if answer NO to question 1
     Q:'XPDQUES("PRE1")
     S UPDATE=XPDQUES("PRE2")
     ;write message to user about task
     D BMES^XPDUTL("Checking Terminal Type File")
     ;get parameter value to initialize NAME
     S NAME=$$PARCP^XPDUTL("ZZUSER1")
     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 %=$$UPCP^XPDUTL("ZZUSER1",NAME)
     Q
     ;build checkpoints for POST
POST N %
     S %=$$NEWCP^XPDUTL("ZZUSER1","POST1^ZZUSER2")
     S %=$$NEWCP^XPDUTL("ZZUSER2")
     Q
POST1       ;check version multiple
     N DA,VER,%
     ;quit if answer NO to question 1
     Q:'XPDQUES("POST1")
     ;write message to user about task
     D BMES^XPDUTL("Checking Package File")
     ;get parameter value to initialize DA
     S DA=+$$PARCP^XPDUTL("ZZUSER1")
     F  S DA=$O(^DIC(9.4,DA)) Q:'DA  D
     .S VER=+$$PARCP^XPDUTL("ZZUSER2")
     .F  S VER=$O(^DIC(9.4,DA,22,VER)) Q:'VER  D
     ..;here is where we could do something
     ..;update parameter VER
     ..S %=$$UPCP^XPDUTL("ZZUSER2",VER)
     .;update parameter DA
     .S %=$$UPCP^XPDUTL("ZZUSER1",DA),%=$$UPCP^XPDUTL("ZZUSER2",VER)
     Q
```

15.3.6.3 Checkpoints *without* Callbacks (Data Storage)

KIDS ignores checkpoints that do *not* have callback routines specified. The ability to create checkpoints without a callback routine is provided mainly as a facility for developers 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 ($$VERCP^XPDUTL(): Verify Checkpoint); if it does *not* exist, create it ($$NEWCP^XPDUTL(): Create Checkpoint).
2. Retrieve the current checkpoint parameter as the starting point if you want to ($$PARCP^XPDUTL(): Get Checkpoint Parameter); do the work for the checkpoint; update the parameter node if you want to ($$UPCP^XPDUTL(): Update Checkpoint).
3. Complete the checkpoint when the work is finished ($$COMCP^XPDUTL(): Complete Checkpoint).
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 in Table 23 (listed in alphabetic order):

Table 23: KIDS—Functions Using Checkpoints *without* Callbacks

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).

15.3.7 Required Builds

In the fourth screen of the Edit a Build [XPD EDIT BUILD] option, you can use the “Required Builds” section (i.e., REQUIRED BUILD [#11] 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 (#9.6) file for those software applications or patches *not* installed using KIDS. Include the name and version number in the BUILD (#9.6) file entry.

REF: For the action types available, see Table 24.

At the installing site, KIDS checks the PACKAGE (#9.4) file, VERSION (#22) Multiple field, and PATCH APPLICATION HISTORY (#9.49,1105) Multiple field to verify that the required build has been installed at that site.

Figure 122: KIDS—Required Builds Sample

```
                           Edit a Build                                PAGE 4 OF 5
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
```

Table 24: KIDS—Required Builds Installation Actions

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.

15.3.8 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 (#9.4) file. Use this link if you want to update the site's PACKAGE (#9.4) file when the software application you are creating is installed or if you want to use Kernel's Alpha/Beta Testing module. You can only link to a PACKAGE (#9.4) file entry that is the same name (minus the version number) as the build you are creating.

If you specify a PACKAGE (#9.4) file entry in the PACKAGE FILE LINK field, and the installing site does *not* have a matching entry in their PACKAGE (#9.4) file, KIDS creates a new entry in the installing site's PACKAGE (#9.4) file.

KIDS checks for duplicate version numbers and patch names when updating the PACKAGE (#9.4) file. When you link to an entry in the PACKAGE (#9.4) file, your installation automatically updates the VERSION (#22) Multiple field in the installing site's corresponding PACKAGE (#9.4) file entry. KIDS makes a new entry in the VERSION (#22) Multiple field for the version of the software application you are installing. KIDS fills in the following fields in the new VERSION entry:

- VERSION (#22; Multiple #9.49,.01)
- DATE DISTRIBUTED (#9.49,1)
- DATE INSTALLED AT THIS SITE (#9.49,2)
- INSTALLED BY (#9.49,3)
- DESCRIPTION OF ENHANCEMENTS (#9.49,41)

- PATCH APPLICATION HISTORY (#9.49,1105; Multiple #9.4901)
    - PATCH APPLICATION HISTORY (#9.4901,.01)
    - DATE APPLIED (#9.4901,.02)
    - APPLIED BY (#9.4901,.03)
    - DESCRIPTION (#9.4901,1)

KIDS saves patch names along with their sequence numbers in the PATCH APPLICATION HISTORY (#9.49,1105) Multiple field.

NOTE: This functionality was added with Kernel patch XU*8.0*30.

The Patch Application History sample (Figure 73) 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 field at the top level of the National PACKAGE (#9.4) file:

Table 25: KIDS—National PACKAGE File Field Updates

PACKAGE (#9.4) File Field Name	Description
AFFECTS RECORD MERGE (#20)	(Multiple) Select files that, if merged, affect this software application.

Beyond these fields, KIDS does *not* support maintaining any other information in the PACKAGE (#9.4) file.

Figure 123: KIDS—Patch Application History Sample

```
 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: XUUSER,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.
```

15.3.9 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 (#9.4) file 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 (#9.4) file.
- 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.

15.3.10 Alpha/Beta Tracking

Kernel provides a mechanism for tracking and monitoring installation and option usage during the alpha and beta testing phases of VistA software applications. This tool is primarily intended for application developers to use in monitoring the testing process at local test sites.

NOTE: In VA terminology, "**Alpha**" and "**Beta**" testing are defined as follows:

- **Alpha Testing**—VistA test software application that is running in a Test account.
- **Beta Testing**—VistA test software application that is running in a Production account.

Alpha/Beta Tracking provides the following services to both developers and system administrators:

- Notification when a new alpha or beta software version is installed at a site.
- Periodic option usage reports for alpha or beta options being tracked.
- Periodic listings of errors in the software's namespace that are currently in alpha or beta test at the site.

The Alpha/Beta Tracking of option usage is transparent to users. If the option counter is turned on, it records the number of times an option is invoked within the menu system when entered in the usual way via ^XUS. Options are *not* counted when navigated past in the course of menu jumping. Also, the counter is *not* set when entering the menu system with the developers ^XUP utility.

Alpha/Beta tracking data is stored in the following Multiples in the KERNEL SYSTEM PARAMETERS (#8989.3) file, which is stored in the **^XTV** global:

Table 26: Alpha/Beta Tracking—KERNEL SYSTEM PARAMETERS (#8989.3) File Field Setup for KIDS

Alpha/Beta Tracking Fields: KERNEL SYSTEM PARAMETERS(#8989.3) File	Description
ALPHA/BETA TEST PACKAGE (#32) Multiple	This field stores the list of software namespaces that are currently in alpha or beta test at the site.
ALPHA,BETA TEST OPTION (#33) Multiple	This field keeps a log of usage of the options associated with an alpha or beta test of VistA software based on the namespace indicated for the alpha or beta test software in the .ALPHA/BETA TEST PACKAGE (#32) Multiple field. This field

Alpha/Beta Tracking Fields: KERNEL SYSTEM PARAMETERS(#8989.3) File	Description
	stores pointers to entries in the OPTION (#19) file.

If there are any entries in these Multiples, the menu system's **XQABTST** variable is set and the options are tracked.

Each time any subsequent test software is loaded, the current alpha/beta data is sent to the data tracker (e.g., developer) and the alpha/beta data is purged from all Multiples.

15.3.10.1 Initiating Alpha/Beta Tracking

In order to initiate and setup Alpha/Beta Tracking at a test site, developers should perform the following procedures:

1. Create the build entry for the VistA software that is exported to sites.
2. Turn on Alpha/Beta Tracking—In the "Package File Link…" section in the fourth ScreenMan form of the build entry. Developers can turn on Alpha/Beta Tracking by entering **YES** at the "BUILD TRACK PACKAGE NATIONALLY:" prompt. ALPHA/BETA TESTING (#20) field in the BUILD (#9.6) file.
3. Edit THE BUILD file Entries—Highlight the software name and press the **<Enter>** key. KIDS places you in a ScreenMan form that lets you edit the following Alpha/Beta Tracking-related fields in the BUILD (#9.6) file:

Table 27: Alpha/Beta Tracking—BUILD (#9.6) File Field Setup for KIDS

Alpha/Beta Tracking Fields: BUILD (#9.6) File	Description
ALPHA/BETA TESTING (#20)	This field initiates Alpha/Beta Tracking. Developers should enter **YES** in this field to activate Alpha/Beta Tracking.
INSTALLATION MESSAGE (#21)	This field sends an installation message when the VistA software application is installed at a site. Developers should answer **YES** if you want the installation message sent to the mail group specified in the ADDRESS FOR USAGE REPORTING (#22) field in the BUILD (#9.6) file.
ADDRESS FOR USAGE REPORTING (#22)	This field should be set to the address of the VA MailMan mail group at the developer's domain. This mail group address is where installation and option usage messages are sent by the Alpha/Beta Tracking code. Also, the domain specified in the address is where server requests are sent from the sites to report errors.

Alpha/Beta Tracking Fields: BUILD (#9.6) File	Description
PACKAGE NAMESPACE OR PREFIX (#23) field	This field is where you identify the alpha/beta VistA software application namespaces to be tracked.

NOTE: At Alpha/Beta Tracking termination, these fields in the BUILD (#9.6) file need to remain populated so the software code knows where to send the final report.

4. Set up the server option at the development domain. This option *must* be set up correctly—In order to track errors at test sites, make sure that the **Handle Alpha/Beta Errors Logged at Sites** [XQAB ERROR LOG SERVER] server option resides at your development site, which should be the domain specified in the ADDRESS FOR USAGE REPORTING (#22) field in the BUILD (#9.6) file for the software build entry.

   This option processes server requests from the test sites, from the **Errors Logged in Alpha/Beta Test (QUEUED)** [XQAB ERROR LOG XMIT] option. The server stores the data from the requests into the XQAB ERRORS LOGGED (#8991.5) file.

   REF: For more information on the **Errors Logged in Alpha/Beta Test (Queued)** [XQAB ERROR LOG XMIT] option, see the "Error Tracking—Alpha/Beta Software Releases" section.

5. Schedule the **Errors Logged in Alpha/Beta Test (Queued)** [XQAB ERROR LOG XMIT] option to run at sites to gather errors and report these to the development server.

   REF: For more information on the **Errors Logged in the Alpha/Beta Test (Queued)** option, see the "Error Tracking—Alpha/Beta Software Releases" section.

6. Schedule the **Send Alpha/Beta Usage to Programmers** [XQAB AUTO SEND] option at the sites to send mail messages containing option usage.

   REF: For more information on the **Send Alpha/Beta Usage to Programmers** option, see the "Send Alpha/Beta Usage to Programmers Option" section.

15.3.10.2 Error Tracking—Alpha/Beta Software Releases

As well as tracking option usage and installations, Kernel also lets developers track errors that occur in the namespace of the alpha- or beta-tracked software. To report these errors to developers, the site should schedule the **Errors Logged in Alpha/Beta Test (QUEUED)** [XQAB ERROR LOG XMIT] option. This option *cannot* be run directly; it is located on the **ZTMQUEUEABLE OPTIONS** menu, which is *not* on any Kernel menu tree, as shown in Figure 124:

Figure 124: KIDS—Errors Logged in Alpha/Beta Test (QUEUED) Option

```
ZTMQUEUEABLE OPTIONS                                   [ZTMQUEUEABLE OPTIONS]
      Errors Logged in Alpha/Beta Test (QUEUED)        [XQAB ERROR LOG XMIT]
```

The **Errors Logged in Alpha/Beta Test (QUEUED)** [XQAB ERROR LOG XMIT] option identifies any errors associated with an application that is in either alpha or beta test. It collects error information and sends it to a server at the development domain. The developer may ask sites to schedule this option to run at a specified frequency, usually nightly. For example, developers can instruct 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 following information:

- 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

NOTE: 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.

15.3.10.3 Monitoring Alpha/Beta Tracking

There are a number of options available to sites used to monitor the progress of alpha or beta testing. These options are located on the **Alpha/Beta Test Option Usage Menu** [XQAB MENU], which is located on the **Operations Management** [XUSITEMGR] menu:

Figure 125: Alpha/Beta Test Option Usage Menu Options

```
Operations Management ...                                         [XUSITEMGR]
  Alpha/Beta Test Option Usage Menu ...                           [XQAB MENU]
    Actual Usage of Alpha/Beta Test Options      [XQAB ACTUAL OPTION USAGE]
    Low Usage Alpha/Beta Test Options            [XQAB LIST LOW USAGE OPTS]
    Print Alpha/Beta Errors (Date/Site/Num/Rou/Err)               [XQAB ERR
DATE/SITE/NUM/ROU/ERR]
    Send Alpha/Beta Usage to Programmers                    [XQAB AUTO SEND]
```

These options are described in the sections that follow.

15.3.10.3.1 Usage Report Options

To get usage reports during the software alpha/beta testing that is making use of the option counter, system administrators can review the tallies with the following options:

- **Actual Usage of Alpha/Beta Test Options** [XQAB ACTUAL OPTION USAGE]
- **Low Usage Alpha/Beta Test Options** [XQAB LIST LOW USAGE OPTS]

15.3.10.3.2 Actual Usage of Alpha/Beta Test Options Option

To get actual usage reports during the software alpha/beta testing that is making use of the option counter, system administrators can review the tallies with the **Actual Usage of Alpha/Beta Test Options** [XQAB ACTUAL OPTION USAGE] option. ADPACs may also be interested in being able to generate this information. Figure 126 shows a printout of the actual usage of options within the XU namespace:

Figure 126: Actual Usage of Alpha/Beta Test Options Option—Sample Option Usage Report

```
    OPTION USAGE SINCE 08-05-92

XUSERINQ                  I        44   User Inquiry
XUUSERDISP                R        49   Display User Characteristics
XUFILEACCESS              M        50   File Access Management
XUSERBLK                  R        51   Grant Access by Profile
XUTIME                    A        53   Time
XUHALT                    A        71   Halt
XUMAINT                   M        83   Menu Management
XUSITEMGR                 M        86   Operations Management
XUSEREDITSELF             R        87   Edit User Characteristics
XUSERTOOLS                M       129   User's Toolbox
XUSEREDIT                 A       175   Edit an Existing User
XUPROG                    M       191   Programmer Options
XUSER                     M       265   User Edit
XUPROGMODE                R       268   Programmer mode
```

15.3.10.3.3 Low Usage of Alpha/Beta Test Options Option

A similar report can be obtained of low usage options since the current version of the tracked software was installed, using the **Low Usage of Alpha/Beta Test Options** [XQAB LIST LOW USAGE OPTS] option.

15.3.10.3.4 Print Alpha/Beta Errors (Date/Site/Num/Rou/Err) Option

The **Print Alpha/Beta Errors (Date/Site/Num/Rou/Err)** [XQAB ERR DATE/SITE/NUM/ROU/ERR] option is used at the development domain, to print error information collected from sites. It does *not* report meaningful information when used at a site.

15.3.10.3.5 Send Alpha/Beta Usage to Programmers Option

At any time during software alpha/beta testing, system administrators can send an interim summary message back to the developers, with the **Send Alpha/Beta Usage to Programmers** [XQAB AUTO SEND] option.

To receive option usage reports, developers should instruct the sites to schedule this option to run at whatever frequency desired in order to receive option usage reports. It may be convenient to schedule this task to run, perhaps on a weekly basis; however, the developer may ask system administrators to schedule it to run at a different specified frequency. This option can also be run manually by the sites to send option usage information.

Mail messages are sent to the mail group and domain specified by the national application developer in the build entry for the ADDRESS FOR USAGE REPORTING (#22) field in the BUILD (#9.6) file when they exported the software.

NOTE: Developers/System Administrators, make sure that this mail group exists at the development domain!

15.3.10.4 Terminating Alpha/Beta Tracking

Alpha/Beta Tracking, once initiated for a VistA software application, *must* be turned off when the final version of the software application is released nationally (production). It is the developer's responsibility to *manually* stop Alpha/Beta Tracking, terminate the audit, and purge the data when appropriate prior to *national* release. However, system administrators can also terminate Alpha/Beta Tracking at the local level:

- **Local (Test) Software**—Developer or system administrators is responsible for terminating Alpha/Beta Tracking at the local site.
- **National (Production) Software**—Developers are responsible for terminating Alpha/Beta Tracking for software that is released nationally.

Information stored during Alpha/Beta Tracking is purged each time a subsequent test version of the software is installed. A final summary report of option usage is prepared and sent to the developer's mail group just before the purge.

15.3.10.4.1 Local (Test) Software Option Usage—Terminating Alpha/Beta Tracking

For *test* versions of the software application that is loaded locally (Test/Production accounts), it is the developer or system administrator's responsibility to stop Alpha/Beta Tracking, terminate the audit, and purge the data from the KERNEL SYSTEM PARAMETERS (#8989.3) file when appropriate. There is no Kernel option to purge locally collected option counts; purge the data via a global **KILL**. If a subsequent software version release is another *test* version, Alpha/Beta Tracking is automatically re-initiated and tracking counts are reset back to **zero**.

NOTE: If the ALPHA/BETA TESTING (#20) field is set to **YES**, any subsequent software version should be considered another test software version. If the ALPHA/BETA TESTING (#20) field is still set to **NO**, then the subsequent software version should be considered a production/release software version.

To *manually* stop Alpha/Beta Tracking at an individual site, developers or system administrators can use the **Enter/Edit Kernel Site Parameters** [XUSITEPARM] option located on the **Kernel Management Menu** [XUKERNEL] to remove the desired entries from the ALPHA/BETA TEST PACKAGE (#32) Multiple and ALPHA,BETA TEST OPTION (#33) Multiple field fields in the KERNEL SYSTEM PARAMETERS (#8989.3) file:

Figure 127: Enter/Edit Kernel Site Parameters—Sample User Dialogue

```
Select Kernel Management Menu Option: ENTER/EDIT KERNEL SITE PARAMETERS

Note: the TaskMan site parameters have been moved out of this file.
Use the Edit TaskMan Parameters option to edit those values.

DEFAULT # OF ATTEMPTS: 3// ^ALPHA BETA TEST PACKAGE
Select ALPHA/BETA TEST PACKAGE: ZZLOCAL// @
   SURE YOU WANT TO DELETE THE ENTIRE ALPHA,BETA TEST PACKAGE? Y
Select ALPHA/BETA TEST PACKAGE: <Enter>
Select ALPHA,BETA TEST OPTION: ZZSAMPLE// @
   SURE YOU WANT TO DELETE THE ENTIRE ALPHA,BETA TEST OPTION? Y
```

15.3.10.4.2 National (Production) Software Option Usage—Terminating Alpha/Beta Tracking

For the *final* version of the software application that is to be released nationally (production), it is the developer's responsibility to *manually* stop Alpha/Beta Tracking, terminate the audit, and purge the data from the local Test/Production accounts when appropriate *prior* to national release.

NOTE: For more information on how to terminate Alpha/Bea Tracking at local test sites, see the "Local (Test) Software Option Usage—Terminating Alpha/Beta Tracking" section in this section.

To *manually* stop Alpha/Beta Tracking of nationally released software, developers *must* enter **NO** in the ALPHA/BETA TESTING (#20) field in the BUILD (#9.6) file for the final build of the production software. When the sites install the build, Alpha/Beta Tracking is shut off.

15.4 Application Programming Interface (API)

Several APIs are available for developers 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 (#9.7) file and the output device.

15.4.1 UPDATE^XPDID(): Update Install Progress Bar

Reference Type:	Supported
Category:	KIDS
ICR #:	2172
Description:	The UPDATE^XPDID API updates the progress bar to show the percentage complete for the installation of the current number of items specified (i.e., **n** input parameter).
Format:	`UPDATE^XPDID(n)`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables:	**XPDIDTOT:**	(required) This variable is the total number of items that are being updated.
Input Parameters:	**n**:	(required) The current number of items being updated.
Output:	none.	

15.4.1.1 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:

Figure 128: UPDATE^XPDID API—Example

```
>Set XPDIDTOT=100
>F%=1:1:100 D CONVERT I'(%#10) D UPDATE^XPDID(%)
```

15.4.2 EN^XPDIJ(): Task Off KIDS Install

Reference Type:	Controlled Subscription	
Category:	KIDS	
ICR #:	2243	
Description:	The EN^XPDIJ 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 (#9.7) file.
Output:	none.	

15.4.3 $$PKGPAT^XPDIP(): Update Patch History

Reference Type:	Supported	
Category:	KIDS	
ICR #:	2067	
Description:	The $$PKGPAT^XPDIP extrinsic function updates the PATCH APPLICATION HISTORY (#9.49,1105) Multiple field of the VERSION (#22) Multiple field in the PACKAGE (#9.4) file. This function can be used during the Pre- or Post-Install routine.	
Format:	`$$PKGPAT^XPDIP(package_ien,version,.x)`	
Input Parameters:	**package_ien**:	(required) The software file entry Internal Entry Number (IEN) in the PACKAGE (#9.4) file.
	version:	(required) This is the software version number. The **version** number *must* contain a decimal (e.g., **8.0**).
	.x:	(required) This parameter is required.
Output:	returns:	Returns: `version ien^patch ien`

15.4.4 $$PKGVER^XPDIP(): Update Patch Version

Reference Type:	Supported	
Category:	KIDS	
ICR #:	2067	
Description:	The $$PKGVER^XPDIP extrinsic function updates the VERSION (#22) Multiple field in the PACKAGE (#9.4) file. This function can be used during the Pre- or Post-Install routine.	
Format:	`$$PKGPAT^XPDIP(package_ien,[.]version)`	
Input Parameters:	**package_ien**:	(required) The software file entry Internal Entry Number (IEN) in the PACKAGE (#9.4) file.
	[.]version:	(required) This can be either a string or an array. If it is an array, then it *must* be passed by reference. • **version** = version number^date distributed^date installed^installed by user ien • **version(1)** = closed global root of the location of the description [e.g., **^XTMP($J,""WP"")**]. All date values *must* be in internal VA FileMan date format. The **version** number *must* contain a decimal (e.g., **8.0**).
Output:	returns:	Returns: `version ien`

15.4.5 BMES^XPDUTL(): Output a Message with Blank Line

Reference Type:	Supported	
Category:	KIDS	
ICR #:	10141	
Description:	The BMES^XPDUTL API outputs a message string to the installation device during KIDS installations. A message is also recorded in the INSTALL (#9.7) file 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 does *not* 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.

15.4.6 $$COMCP^XPDUTL(): Complete Checkpoint

Reference Type: Supported

Category: KIDS

ICR #: 10141

Description: The $$COMCP^XPDUTL extrinsic function completes a checkpoint, in pre- or post-install routines during KIDS installations. Use this only to complete checkpoints that do *not* 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.

15.4.7 $$CURCP^XPDUTL(): Get Current Checkpoint Name/IEN

Reference Type: Supported

Category: KIDS

ICR #: 10141

Description: The $$CURCP^XPDUTL extrinsic function returns the name of the current checkpoint during KIDS installations. 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 returns 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 follows: • **Zero (0)**—To return checkpoint name. • **1**—To return checkpoint Internal Entry Number (IEN).
Output:	returns:	Returns: • **Checkpoint Name**—The current checkpoint name. • **NULL String**—If *not* currently in a checkpoint callback.

15.4.8 $$INSTALDT^XPDUTL(): Return All Install Dates/Times

Reference Type: Supported

Category: KIDS

ICR #: 10141

Description: The $$INSTALDT^XPDUT extrinsic function retrieves all dates/times that an install was performed for a given install name in the INSTALL (#9.7) file. It returns the results in an array.

NOTE: This API was released with Kernel Patch XU*8.0*491.

Format: `$$INSTALDT^XPDUTL(install,.result)`

Input Parameters:	**install**:	(required) Name of install in the INSTALL (#9.7) file.
	.result:	(required) Passed by reference, the name of the array to return values.
Output Parameters:	**.result**:	Returns the number of records in the result array: • result=number of records. • result(internal date/time)="TEST#^SEQ#" (Fields 61^62 from INSTALL [#9.7] file).

15.4.8.1 Example

Figure 129: $$INSTALDT^XPDUTL API—Example

```
>W $$INSTALDT^XPDUTL("XU*8.0*491", .RSLT)
1
>ZW RSLT
RSLT=1
RSLT(3080318.092151)="1^"
```

15.4.9 $$LAST^XPDUTL(): Last Software Patch

Reference Type:	Supported	
Category:	KIDS	
ICR #:	10141	
Description:	The $$LAST^XPDUTL extrinsic function returns the last patch and the date it was applied to the software. The patch also includes the Sequence # if the last patch was a released patch. **NOTE:** This API can be used outside of KIDS.	
Format:	`$$LAST^XPDUTL(x[,y][,z])`	
Input Parameters:	**x**:	(required) Software name or software namespace within quotes (e.g., "**KERNEL**" or "**XU**").
	y:	(optional) Full software version number with decimal point entered within quotes (e.g., "**8.0**"). The current version is assumed if this parameter is *not* supplied.
	z:	(optional) This parameter was added with Kernel Patch XU*8.0*559. If set to **1**, then only the last *released* patch information is returned.

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 #*nnn^yyymmdd***—Released patch; where "***nnn***" = patch number, "**Seq #*nnn***" = sequence number for released patch, and "***yyymmdd***" = date in VA FileMan format. • **-1**—If either the software or version does *not* exist or no patches have been applied.

15.4.9.1 Examples

15.4.9.1.1 Example 1

Figure 130: $$LAST^XPDUTL API—Example 1

```
>S X="KERNEL"

>S Y="8.0"

>W $$LAST^XPDUTL(X,Y)
543^3110503
```

15.4.9.1.2 Example 2

Figure 131: $$LAST^XPDUTL API—Example 2

```
>S X="KERNEL"

>S Y="8.0"

>S Z=1

>W $$LAST^XPDUTL(X,Y,Z)
431 SEQ #453^3110425.122831
```

15.4.9.1.3 Example 3

Figure 132: $$LAST^XPDUTL API—Example 3

```
>S X="KERNEL"

>S Y="9.0"

>S Z=1

>W $$LAST^XPDUTL(X,Y,Z)
-1
```

For this example, since there is no Kernel 9.0 the expected result is **-1**.

15.4.10 MES^XPDUTL(): Output a Message

Reference Type:	Supported	
Category:	KIDS	
ICR #:	10141	
Description:	The MES^XPDUTL API outputs a message string to the installation device during KIDS installations. A message is also recorded in INSTALL (#9.7) file 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.

15.4.11 $$NEWCP^XPDUTL(): Create Checkpoint

Reference Type:	Supported
Category:	KIDS
ICR #:	10141

Description:

The $$NEWCP^XPDUTL extrinsic function creates a checkpoint in pre- or post-install routines during KIDS installations. The checkpoint is stored in the INSTALL (#9.7) file.

Pre-and post-install checkpoints are stored separately, so you can use the same name for a pre- and post-install checkpoint if you wish:

- Checkpoints created with this function from the pre-install routine are pre-install checkpoints.
- Checkpoints created during the post-install routine are post-install checkpoints.

You can use $$NEWCP^XPDUTL to create a checkpoint with or without a 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 is *not* 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.

15.4.12 $$OPTDE^XPDUTL(): Disable/Enable an Option

Reference Type:	Supported	
Category:	KIDS	
ICR #:	10141	
Description:	The $$OPTDE^XPDUTL extrinsic function is used during KIDS installations in a Pre-Init or Post-Init routine. Use this function to disable or enable an option.	
Format:	`$$OPTDE^XPDUTL(name,action)`	
Input Parameters:	**name**:	(required) Option name.
	action:	(required) Set to: • **1**—Enable an option. • **0**—Disable an option.
Output:	returns:	Returns: • **1**—Success. • **0**—Failure.

15.4.12.1 Example

Figure 133: $$OPTDE^XPDUTL API—Example

```
I $$OPTDE^XPDUTL(“XMUSER”,0) W !,’Option Disabled.’
```

15.4.13 $$PARCP^XPDUTL(): Get Checkpoint Parameter

Reference Type:	Supported	
Category:	KIDS	
ICR #:	10141	
Description:	The $$PARCP^XPDUTL extrinsic function retrieves the current value of a checkpoint's stored parameter during KIDS installations. The parameter is stored in the INSTALL (#9.7) file. Use this API for checkpoints both with and without callbacks. Use the optional second parameter to retrieve a pre-install checkpoint's parameter during a post-install.	
Format:	`$$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.

15.4.14 $$PATCH^XPDUTL(): Verify Patch Installation

Reference Type:	Supported	
Category:	KIDS	
ICR #:	10141	
Description:	The $$PATCH^XPDUTL extrinsic function is used during KIDS installations, during the environment check only. Use this function to verify if a patch has been installed. You can check for patches with or without sequence numbers.	
Format:	`$$PATCH^XPDUTL(patch)`	
Input Parameters:	**patch**:	(required) Patch name. Patch name *must* include the full version number with the decimal point, such as XU***8.0***28.
Output:	returns:	Returns: • **1**—Specified patch was installed on the current system. • **0**—Specified patch was *not* installed on the current system.

15.4.14.1 Example

Checking for a patch installation. Enter the following at the programmer prompt:

Figure 134: $$PATCH^XPDUTL API—Example

```
>I '$$PATCH^XPDUTL("XU*8.0*28") W !,"You must install patch XU*8*28"
```

15.4.15 $$PKG^XPDUTL(): Parse Software Name from Build Name

Reference Type:	Supported	
Category:	KIDS	
ICR #:	10141	
Description:	The $$PKG^XPDUTL extrinsic function parses the name of a software application from a software application's build name. You can obtain the name of the build KIDS is installing from the KIDS key variable **XPDNM**, which is defined throughout a KIDS installation.	
Format:	`$$PKG^XPDUTL(buildname)`	
Input Parameters:	**buildname**:	Name of build (**.01** field of BUILD [#9.6] file).
Output:	returns:	Returns the software name.

15.4.16 $$PRODE^XPDUTL(): Disable/Enable a Protocol

Reference Type:	Supported	
Category:	KIDS	
ICR #:	10141	
Description:	The $$PRODE^XPDUTL extrinsic function is used in a Pre-Init or Post-Init routine during KIDS installations. Use this function to disable or enable a protocol.	
Format:	`$$PRODE^XPDUTL(name,action)`	
Input Parameters:	**name**:	(required) Protocol name.
	action:	(required) Enter one of the following values for this parameter: • **1**—Enable a protocol. • **2**—Disable a protocol.

Output:	returns:	Returns: • **1**—Success. • **0**—Failure.

15.4.17 $$RTNUP^XPDUTL(): Update Routine Action

Reference Type:	Supported
Category:	KIDS
ICR #:	10141
Description:	The $$RTNUP^XPDUTL extrinsic function updates the installation action for a routine during KIDS installations, during the environment check only.
Format:	`$$RTNUP^XPDUTL(routine,action)`

Input Parameters:	**routine**:	(required) Routine name.
	action:	(required) Enter one of the following values for this parameter: • **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.

15.4.18 $$UPCP^XPDUTL(): Update Checkpoint

Reference Type:	Supported
Category:	KIDS
ICR #:	10141
Description:	The $$UPCP^XPDUTL extrinsic function updates the parameter node of an existing checkpoint, in pre- or post-install routines during KIDS installations. The parameter node is stored in the INSTALL (#9.7) file. Use this API for checkpoints both with and without callbacks. During the pre-install, you can only update pre-install checkpoints; during the post-install, you can only update post-install checkpoints.
Format:	`$$UPCP^XPDUTL(name[,par_value])`

Input Parameters:	**name**:	(required) Checkpoint name.
	par_value:	(optional) Sets checkpoint parameter to this value.
Output:	returns:	Returns: • **Internal Entry Number (IEN)**—Successfully updated checkpoint. • **Zero (0)**—Error updating checkpoint.

15.4.19 $$VER^XPDUTL(): Parse Version from Build Name

Reference Type: Supported

Category: KIDS

ICR #: 10141

Description: The $$VER^XPDUTL 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 [#9.6] file).
Output:	returns:	Returns: • **Version**—The version of the build identified in the **buildname** input parameter. • **NULL**—If no match in the BUILD (#9.6) file.

15.4.20 $$VERCP^XPDUTL(): Verify Checkpoint

Reference Type: Supported

Category: KIDS

ICR #: 10141

Description: The $$VERCP^XPDUTL extrinsic function checks whether a given checkpoint exists and, if it exists, whether it has completed or *not* during KIDS installations.

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 does *not* exist.

15.4.21 $$VERSION^XPDUTL(): Package File Current Version

Reference Type:	Supported	
Category:	KIDS	
ICR #:	10141	
Description:	The $$VERSION^XPDUTL extrinsic function obtains the current version of a site's software application.	
Format:	`$$VERSION^XPDUTL(package_id)`	
Input Parameters:	**package_id**:	(required) Software application's name or namespace, from its entry in the PACKAGE (#9.4) file.
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 (#9.4) file. • **NULL**—If the software application is *not* matched.

16 Menu Manager: Developer Tools

16.1 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 section) to further enhance option creation and custom tailoring.

16.1.1 Option Types

Several different option types exist:

- Edit, Inquire, and Print are mainly used to access VA FileMan files.
- Action and Run Routine types are available for invoking M code.
- Menu types, as discussed earlier in this section, are used to group other options for presentation to the user at the select prompt.
- 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, see the "Server Options: Developer Tools" section in this section.

- 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, see the Computerized Patient Record System (CPRS) or Unwinder (XQOR) documentation.

16.1.2 Creating Options (Edit Options)

Figure 135: Menu Manager—Edit options [XUEDITOPT]

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

You can define options with the Edit Options template, available from the Menu Management menu. Depending on what type of option you are editing, the Edit Options template branches to the fields in the OPTION (#19) file appropriate for that option type.

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

For Edit type options, the **DI** fields presented correspond to the input variables for a VA FileMan ^DIE call. Likewise, inquire-type options correspond to VA FileMan ^DIQ calls, and print options to ^DIP calls.

REF: For a complete description of the meaning of the variables represented by each of the **DI** fields, see the *VA FileMan Developer's Guide*.

16.1.2.1 Options that Should be Regularly Scheduled

If an option should be regularly scheduled to run through TaskMan, you should set its SCHEDULING RECOMMENDED (#209) field in the OPTION (#19) file) to **YES**. Sites are *not* able to use Schedule/Unschedule Options to schedule an option unless this field is set to **YES** for the option.

16.2 Variables for Developer Use

The appearance and functioning of the menu system can be modified by developers by using several variables. The variables can be defined within applications, such as in an option's:

- **Entry Action**
- **Exit Action**
- **Header**

The following variables are described in the sections below:

- XQUIT: Quit the Option
- XQMM("A"): Menu Prompt
- XQMM("B"): Default Response

- XQMM("J"): The Phantom Jump
- XQMM("N"): No Menu Display

NOTE: 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**.

16.2.1 XQUIT: Quit the Option

The **XQUIT** variable can be set in an option's Entry Action to cause Menu Manager to quit and *not* invoke the option. The menu system does *not* run the option, either as a foreground job or background task, and does *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 developer chooses.

16.2.2 XQMM("A"): Menu Prompt

If **XQMM("A")** exists, the menu system uses it as the prompt instead of the normal "Select...option" menu prompt. The **XQMM("A")** variable is **KILL**ed 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 are displayed *before* the **XQMM("A")** prompt is presented. Unlike the phantom jump, prompts *must* be set singularly, and cannot be concatenated with a semicolon.

16.2.3 XQMM("B"): Default Response

If **XQMM("B")** is defined, the menu system uses it as the default response and is presented along with the usual two slashes (//). If the user accepts the default by pressing **<Enter>**, the default becomes 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 responds with two question marks (**??**), **KILL** both **XQMM("A")** and **XQMM("B")**, and display the standard menu prompt.

16.2.4 XQMM("J"): The Phantom Jump

The **XQMM("J")** 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 [#19] file) to which Menu Manager should jump. For example:

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

This jumps 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.

16.2.5 XQMM("N"): No Menu Display

The **XQMM("N")** variable can be used to suppress the AUTO MENU display of menu options for one menu cycle. **XQMM("N")** is then **KILL**ed and the display resumes as usual. **XQMM("N")** can be used in conjunction with **XQMM("A")** and **XQMM("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 is already suppressed. If **XQMM("J")** is present, then **XQMM("N")** is *not* **KILL**ed after the first cycle since the phantom jump is already inhibiting the display. In this case, **XQMM("N")** is **KILL**ed after the second cycle (the display of menus after the jump is completed). If several phantom jumps are chained together, **XQMM("N")** is *not* **KILL**ed until one cycle after the final jump unless code is added to explicitly **KILL** it between jumps.

16.3 Direct Mode Utilities

Several Menu Manager direct mode utilities are available for developers to use at the M prompt. They are *not* APIs and *cannot* be used in software application routines. These direct mode utilities are described below.

16.3.1 ^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
```

^XQ1 is also called by **^XUP**.

CAUTION: Developers 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, see the "Signon/Security: Developer Tools" section.

NOTE: While **D ^XQ1** is a direct mode utility, it is *not* a callable API.

16.4 Application Programming Interface (API)

Several APIs are available for developers to work with menu management. These APIs are described below.

16.4.1 $$ADD^XPDMENU(): Add Option to Menu

Reference Type:	Supported	
Category:	Menu Manager	
ICR #:	1157	
Description:	The $$ADD^XPDMENU extrinsic function adds an option as a new item to an existing menu.	
Format:	`$$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.

16.4.2 $$DELETE^XPDMENU(): Delete Menu Item

Reference Type:	Supported	
Category:	Menu Manager	
ICR #:	1157	
Description:	The $$DELETE^XPDMENU extrinsic function deletes an option from the Menu field of another option. It returns the following values: • **1**—If the function succeeded. • **0**—If it failed.	
Format:	`$$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.

16.4.3 $$LKOPT^XPDMENU(): Look Up Option IEN

Reference Type:	Supported	
Category:	Menu Manager	
ICR #:	1157	
Description:	The $$LKOPT^XPDMENU extrinsic function looks up an option's Internal Entry Number (IEN) using the "**B**" cross-reference.	
Format:	`$$LKOPT^XPDMENU(option)`	
Input Parameters:	**option**:	(required) The name of the option.
Output:	returns:	Returns the Internal Entry Number (IEN) of the input option in the OPTION (#19) file.

16.4.4 LOCK^XPDMENU(): Set LOCK Field in OPTION File

Reference Type:	Supported	
Category:	Menu Manager	
ICR #:	1157	
Description:	The LOCK^XPDMENU API sets the LOCK (#3) field in the OPTION (#19) file for a given option. **NOTE:** This API was released with Kernel Patch XU*8.0*672.	
Format:	`LOCK^XPDMENU(opt,txt)`	
Input Parameters:	**opt:**	(required) This is the name of the option you want to lock.
	txt:	(required) This is the security key name used to lock the option. It *must* match an entry in the SECURITY KEY (#19.1) file.
Output:	none.	Sets the LOCK (#3) field in the OPTION (#19) file for the **opt** input parameter.

16.4.5 OUT^XPDMENU(): Edit Option's Out of Order Message

Reference Type:	Supported	
Category:	Menu Manager	
ICR #:	1157	
Description:	The OUT^XPDMENU 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 a value in the OUT OF ORDER MESSAGE (#2) field in the OPTION (#19) file.
	text:	(required) Text of message to place in the option's OUT OF ORDER MESSAGE (#2) field. If this is *not* **NULL**, the text is stored in the option's OUT OF ORDER MESSAGE (#2) field and the option is placed out of order. If this parameter is passed as a **NULL** string, the current OUT OF ORDER MESSAGE (#2) field value is deleted, and the option is put back in order.

Output:	none.	

16.4.6 RENAME^XPDMENU(): Rename Option

Reference Type:	Supported	
Category:	Menu Manager	
ICR #:	1157	
Description:	The RENAME^XPDMENU API renames an existing option.	
Format:	`RENAME^XPDMENU(old,new)`	
Input Parameters:	**old**:	(required) Current option name (**.01** field of OPTION [#19] file entry). *Must* be an exact match.
	new:	(required) New name for option.
Output:	none.	

16.4.7 RLOCK^XPDMENU(): Set REVERSE/NEGATIVE Field in OPTION File

Reference Type:	Supported	
Category:	Menu Manager	
ICR #:	1157	
Description:	The RLOCK^XPDMENU API sets the REVERSE/NEGATIVE (#3.01) field in the OPTION (#19) file for a given option. **NOTE:** This API was released with Kernel Patch XU*8.0*672.	
Format:	`RLOCK^XPDMENU(opt,txt)`	
Input Parameters:	**opt:**	(required) This is the name of the option you want to reverse lock.
	txt:	(required) This is the security key name used to reverse lock the option. It *must* match an entry in the SECURITY KEY (#19.1) file.
Output:	none.	Reverses the lock on the REVERSE/NEGATIVE (#3.01) field in the OPTION (#19) file for the **opt** input parameter.

REF: For more information on reverse locks, see the “Using Security Keys with Reverse Locks” section in the “Menu Manager: System Management” section in the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide.*

16.4.8 $$TYPE^XPDMENU(): Get Option Type

Reference Type:	Supported	
Category:	Menu Manager	
ICR #:	1157	
Description:	The $$TYPE^XPDMENU extrinsic function returns the option’s TYPE (#4) field in the OPTION (#19) file.	
Format:	`$$TYPE^XPDMENU(option)`	
Input Parameters:	**option**:	(required) The name of the option.
Output:	returns:	Returns the one character TYPE (#4) field value of the input option in the OPTION (#19) file. For example:

- **A**—Action
- **E**—Edit
- **I**—Inquire
- **M**—Menu
- **P**—Print
- **R**—Run routine
- **O**—Protocol
- **Q**—Protocol Menu
- **X**—Extended Action
- **S**—Server
- **L**—Limited
- **C**—ScreenMan
- **W**—Window
- **Z**—Window Suite
- **B**—Broker (Client/Server)

16.4.9 $$ADD^XPDPROT(): Add Child Protocol to Parent Protocol

Reference Type:	Supported	
Category:	Menu Manager	
ICR #:	5567	
Description:	The $$ADD^XPDPROT extrinsic function adds a **child** input protocol to a **parent** input protocol ITEM (#10) Multiple field in the PROTOCOL (#101) file. **NOTE:** This API was released with Kernel Patch XU*8.0*547.	
Format:	`$$ADD^XPDPROT(parent,child[,mnemonic][,sequence])`	
Input Parameters:	**parent**:	(required) Name of the **parent** input protocol in the PROTOCOL (#101) file to which a **child** input protocol should be added.
	child:	(required) Name of the **child** input protocol being added to the **parent** input protocol in the PROTOCOL (#101) file.
	mnemonic:	(optional) The mnemonic value to be added to the MNEMONIC (#2) field in the ITEM (#10) Multiple field in the PROTOCOL (#101) file for the **child** in the **parent** protocol.
	sequence:	(optional) The sequence value to be added to the SEQUENCE (#3) field in the ITEM (#10) Multiple field in the PROTOCOL (#101) file for the **child** in the **parent** protocol.
Output:	returns:	Returns: • **1**—Success, **child** input protocol added to the **parent** input protocol ITEM (#10) Multiple field in the PROTOCOL (#101) file. • **0**—Failure, **child** input protocol *not* added to the **parent** input protocol ITEM (#10) Multiple field in the PROTOCOL (#101) file.

16.4.10 $$DELETE^XPDPROT(): Delete Child Protocol from Parent Protocol

Reference Type:	Supported	
Category:	Menu Manager	
ICR #:	5567	
Description:	The $$DELETE^XPDPROT extrinsic function deletes a **child** input protocol from a **parent** input protocol ITEM (#10) Multiple field in the PROTOCOL (#101) file. **NOTE:** This API was released with Kernel Patch XU*8.0*547.	
Format:	`$$DELETE^XPDPROT(parent,child)`	
Input Parameters:	**parent**:	(required) Name of the **parent** protocol in the PROTOCOL (#101) file from which a **child** protocol should be deleted.
	child:	(required) Name of the **child** protocol being deleted from the **parent** protocol in the PROTOCOL (#101) file.
Output:	returns:	Returns: • **1—Success**: The **child** input protocol deleted from the **parent** input protocol ITEM (#10) Multiple field in the PROTOCOL (#101) file. • **0—Failure**: The **child** input protocol *not* deleted from the **parent** input protocol ITEM (#10) Multiple field in the PROTOCOL (#101) file.

16.4.11 FIND^XPDPROT(): Find All Parents for a Protocol

Reference Type: Supported

Category: Menu Manager

ICR #: 5567

Description: The FIND^XPDPROT API finds all parents for a protocol in the PROTOCOL (#101) file and returns the list in the **RESULT** array:

- **RESULT(0)**=Number of parents found or **-1^error message**.
- **RESULT(IEN)**=Protocol name.

ℹ **NOTE:** This API was released with Kernel Patch XU*8.0*547.

Format:	`FIND^XPDPROT(.result,protocol)`	
Input Parameters:	**.result**:	(required) The array to return the results, passed by reference: • **RESULT(0)**=Number of parents found or **-1^error message**. • **RESULT(IEN)**=Protocol name.
	protocol:	(required) Name of the protocol in the PROTOCOL (#101) file for which to find the parents.
Output:	returns:	Returns the **RESULT** array:

```
RESULT(0)=Number of parents found or -
1^error message.
RESULT(ien)=Protocol name
```

16.4.12 $$LKPROT^XPDPROT(): Look Up Protocol IEN

Reference Type: Supported

Category: Menu Manager

ICR #: 5567

Description: The $$LKPROT^XPDPROT extrinsic function returns the internal entry number (IEN) of the input protocol from the PROTOCOL (#101) file.

ℹ **NOTE:** This API was released with Kernel Patch XU*8.0*547.

Format:	`$$LKPROT^XPDPROT(protocol)`	
Input Parameters:	**protocol**:	(required) Name of the protocol to look up in the PROTOCOL (#101) file.
Output:	returns:	Returns the internal entry number (IEN) of the input protocol in the PROTOCOL (#101) file.

16.4.13 OUT^XPDPROT(): Edit Protocol's Out of Order Message

Reference Type:	Supported	
Category:	Menu Manager	
ICR #:	5567	
Description:	The OUT^XPDPROT API creates or deletes an "Out of Order" message in the DISABLE (#2) field in the PROTOCOL (#101) file for the input protocol. **NOTE:** This API was released with Kernel Patch XU*8.0*547.	
Format:	`OUT^XPDPROT(protocol,text)`	
Input Parameters:	**protocol**:	(required) Name of the protocol in the PROTOCOL (#101) file to which the "Out of Order" text is assigned.
	text:	(required) Text value: • **Text**—Message text to place in the DISABLE (#2) field in the PROTOCOL (#101) file for the input protocol. • **NULL**—Delete any message text in the DISABLE (#2) field in the PROTOCOL (#101) file for the input protocol.
Output:	returns:	Returns: • **Text**—Updated message text in the DISABLE (#2) field in the PROTOCOL (#101) file for the input protocol. Marking the protocol "Out of Order." • **NULL**—Deleted message text in the DISABLE (#2) field in the PROTOCOL (#101) file for the input protocol.

16.4.14 RENAME^XPDPROT(): Rename Protocol

Reference Type:	Supported
Category:	Menu Manager
ICR #:	5567
Description:	The RENAME^XPDPROT API renames an existing protocol name. It updates the value in the NAME (#.01) field in the PROTOCOL (#101) file.

NOTE: This API was released with Kernel Patch XU*8.0*547.

Format: `RENAME^XPDPROT(old,new)`

Input Parameters:	**old**:	(required) Current (old) name of the protocol to be renamed in the PROTOCOL (#101) file.
	new:	(required) New name for the protocol.
Output:	returns:	Returns the updated NAME (#.01) field in the PROTOCOL (#101) file.

16.4.15 $$TYPE^XPDPROT(): Get Protocol Type

Reference Type:	Supported
Category:	Menu Manager
ICR #:	5567
Description:	The $$TYPE^XPDPROT extrinsic function returns the value of the TYPE (#4) field in the PROTOCOL (#101) file for the input protocol IEN.

NOTE: This API was released with Kernel Patch XU*8.0*547.

Format: `$$TYPE^XPDPROT(protocol_ien)`

Input Parameters:	**protocol_ien**:	(required) The protocol's internal entry number (IEN) in the PROTOCOL (#101) file.

Output:	returns:	Returns the one character TYPE (#4) field value in the PROTOCOL (#101) file for the input protocol IEN. For example:

- **A**—Action: Same as the **X** type, except any existing sub-items are *not* executed.
- **M**—Menu: Use this type for displaying and selecting items.
- **O**—Protocol: This value is strictly related to the Add orders function. It is the same as the **Q** type, except the protocol is the item selected. Protocols are directly executed when encountered.
- **Q**—Protocol Menu: This value is strictly related to the Add orders function. Use it for displaying and selecting orderable items during the add sequence. When this type of protocol is encountered OE/RR prompts the user with "Select PATIENT:," "LOCATION:," and "Provider:," and execute the transaction logic for the new orders screen.
- **L**—Limited Protocol: This value is strictly related to the Add orders function. It is the same as the **O** type, except any existing sub-items are *not* executed.
- **X**—Extended Action: Protocols of this type execute the entry action plus all sub-items.
- **D**—Dialog.
- **T**—Term.
- **E**—Event Driver.
- **S**—Subscriber.

16.4.16 NEXT^XQ92(): Restricted Times Check

Reference Type:	Supported	
Category:	Menu Manager	
ICR #:	10077	
Description:	The NEXT^XQ92 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 (#19) file.
Output:	**x**:	The date/time in VA FileMan format of the next unrestricted runtime when the option can run: • **Current Time**—If the option is able to run at the current time. • **NULL**—If the option is prohibited for the entire next week. It also issues a message regarding the time restriction.

16.4.17 $$ACCESS^XQCHK(): User Option Access Test

Reference Type:	Supported	
Category:	Menu Manager	
ICR #:	10078	
Description:	The $$ACCESS^XQCHK extrinsic function determines if a user has access to a particular option.	
Format:	`$$ACCESS^XQCHK(duz,option)`	
Input Parameters:	**duz**:	(required) The identification number of the user in question in the NEW PERSON (#200) file.
	option:	(required) The Internal Entry Number (IEN) or option name of the option in question in the OPTION (#19) file.
Output:	returns:	Returns: • **-1**—No such user in the NEW PERSON (#200) file. • **-2**—User terminated or has no Access code. • **-3**—No such option in the OPTION (#19) file.

- **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 the secondary menu tree **OpIEN**.

XQCODES can contain the following:

- **N**—No Primary Menu in the NEW PERSON (#200) file (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).

16.4.18 OP^XQCHK(): Current Option Check

Reference Type: Supported

Category: Menu Manager

ICR #: 10078

Description: The OP^XQCHK API returns the current option or protocol name and menu text in the first and second pieces of the **XQOPT** output variable. It looks for the local **XQORNOD** variable if defined or the local **XQY** variable; the internal number of the option. If **XQORNOD** is defined, it needs to be in the VARIABLE POINTER format:

```
XQORNOD=<internal number of the protocol>;<protocol
file>
```

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: OP^XQCHK *cannot* return option/protocol information if the job is a task that did *not* originate from an option.

Format: `OP^XQCHK`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables: **XQORNOD:** (optional) If this variable is defined, it should be in VARIABLE POINTER format. For example:

```
XQORNOD="1234;ORD(101,"
```

Output Variables: **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
```

16.4.18.1 Examples

16.4.18.1.1 Example 1

Figure 136: OP^XQCHK API—Example 1

```
>K XQORNOD D OP^XQCHK W !,XQOPT

>EVE^Systems Manager Menu
```

16.4.18.1.2 Example 2

Figure 137: OP^XQCHK API—Example 2

```
>S XQORNOD="445;ORD(101," D OP^XQCHK W !,XQOPT

>XU USER EVENT TERMINATE^Terminate User Event
```

16.4.18.1.3 Example 3

Figure 138: OP^XQCHK API—Example 3

```
>S XQORNOD="9;DIC(19," D OP^XQCHK W !,XQOPT

>EVE^Systems Manager Menu
```

16.4.18.1.4 Example 4

Figure 139: OP^XQCHK API—Example 4

```
>K XQORNOD,XQY,XQOPT D OP^XQCHK W !,XQOPT

>-1^Unknown
```

17 Lock Manager: Developer Tools

17.1 Application Programming Interface (API)—Housekeeping

When an application terminates, there may be housekeeping required. A prime example is the need to delete temporary data kept in the **^TMP** and **^XTMP** globals. An application that is terminated by the Lock Manager does *not* have the opportunity to do its own housecleaning, but the Lock Manager can do it for the application if it registers a housecleaning routine via the APIs described below.

17.1.1 CLEANUP^XULMU(): Execute the Housecleaning Stack

Reference Type:	Supported
Category:	Lock Manager
ICR #:	5832
Description:	The CLEANUP^XULMU API executes the housecleaning stack set by the process identified by **DOLLARJ**. Entries are executed in the first-in-first-out (FIFO) order, with the last entry added being the first to be executed, and **last** being the last entry executed. If the **last** parameter is *not* passed in, then the entire stack is executed.

NOTE: This API was released with Kernel Patch XU*8.0*608.

Format:	`CLEANUP^XULMU([last])`	
Input Parameters:	**last**:	(optional) This is the last entry that is executed. If *not* passed in, then the entire housecleaning stack is executed.
Output:	returns:	None.

17.1.1.1 Examples

17.1.1.1.1 Example 1

An application can execute the entire housecleaning stack with the code shown in Figure 140:

Figure 140: CLEANUP^XULMU API—Example 1

```
DO CLEANUP^XULMU
```

17.1.1.1.2 Example 2

If an application is called by another application, then the first application may have already placed entries of its own on the stack. So, the **last** parameter needs to be passed, with **last** being the first entry placed on the stack. It is the last entry executed, since that stack is executed in FIFO order.

Figure 141: CLEANUP^XULMU API—Example 2

```
DO CLEANUP^XULMU(last)
```

17.1.2 SETCLEAN^XULMU(): Register a Cleanup Routine

Reference Type: Supported

Category: Lock Manager

ICR #: 5832

Description: The SETCLEAN^XULMU API registers a cleanup routine that should be executed when the process is terminated by the Kernel Lock Manager. An entry is created on a stack kept for the process. The location is:

^XTMP("XULM CLEANUP_"_$J)

Where **$J** uniquely identifies the process.

A process can call SETCLEAN^XULMU repeatedly, and each time a new entry is placed on the stack.

CAUTION: Once an application calls SETCLEAN, upon exiting it *must* either execute its housecleaning stack or delete it using the APIs CLEAN or UNCLEAN.

NOTE: This API was released with Kernel Patch XU*8.0*608.

Format: `SETCLEAN^XULMU(rtn,.var)`

Input Parameters:

rtn:	(required) The routine to be executed when the process is terminated.
.var:	(required) An input array containing a list of variables that should be defined when the routine is executed. It is up to the application to ensure that all the required variables are defined when CLEANUP^XULMU is called.

Output:	returns:	Returns: An integer that identifies the entry created on the stack. The application needs to retain the value in order to either execute the entry on the housecleaning stack or to remove it.

17.1.2.1 Example

Suppose the application has a cleanup routine CLEANUP^XXAPP, and it needs to be executed with **DFN** defined with its present valued. The application would use this API as shown in Figure 142:

Figure 142: SETCLEAN^XULMU API—Example

```
N VAR,CLEANUP
S VAR("DFN")=DFN
S CLEANUP=$$SETCLEAN^XULMU("CLEANUP^XXAPP",.VAR)
```

The application's housekeeping stack would look like Figure 143:

Figure 143: SETCLEAN^XULMU API—Sample Stack

```
^XTMP("XULM CLEANUP",$J,1,"ROUTINE")="CLEANUP^XXAPP"
^XTMP("XULM CLEANUP",$J,1,"VARIABLES","DFN")=1000061
```

17.1.3 UNCLEAN^XULMU(): Remove Entries from the Housecleaning Stack

Reference Type: Supported

Category: Lock Manager

ICR #: 5832

Description: The UNCLEAN^XULMU API removes entries from the housecleaning stack set by calling the SETCLEAN^XULMU(): Register a Cleanup Routine API. Entries are removed in First-In-First-Out (FIFO) order. If the **last** input parameter is *not* passed in, then the entire stack is deleted; otherwise, just the entries back to **last** are removed.

NOTE: This API was released with Kernel Patch XU*8.0*608.

Format: `UNCLEAN^XULMU([last])`

Input Parameters:	**last**:	(optional) Identifies the last entry on the housekeeping stack to remove. Entries are removed in FIFO order. Therefore, the first entry removed is the

		last entry that was added, and the last entry removed is **last**. If *not* passed in, the entire housecleaning stack is deleted.
Output:	returns:	None.

17.1.3.1 Examples

17.1.3.1.1 Example 1

The example in Figure 144 would remove the entire housecleaning stack:

Figure 144: UNCLEAN^XULMU API—Example 1

```
DO UNCLEAN^XULMU
```

17.1.3.1.2 Example 2

If an application is called by another application, then the first application may have already placed entries of its own on the stack. So, the **last** input parameter needs to be passed, with **last** being the first entry placed on the stack. It is the last entry deleted, since that stack is executed in first-in-first-out (FIFO) order.

Figure 145: UNCLEAN^XULMU API—Example 2

```
DO UNCLEAN^XULMU(last)
```

17.2 Application Programming Interface (API)—Lock Dictionary

17.2.1 ADDPAT^XULMU(): Add Patient Identifiers for a Computable File Reference

Reference Type:	Supported	
Category:	Lock Manager	
ICR #:	5832	
Description:	The ADDPAT^XULMU API is very similar to the PAT^XULMU(): Get a Standard Set of Patient Identifiers API, except that it is used to *add* the patient identifiers for a computable file reference for a file that is *not* the PATIENT (#2) file. The computable file references can include additional identifiers. For example, a computable file reference for a billing file can contain the bill number as an identifier as well as the patient identifiers returned by the ADDPAT^XULMU API. **NOTE:** This API was released with Kernel Patch XU*8.0*608.	
Format:	`ADDPAT^XULMU(dfn)`	
Input Parameters:	**dfn**:	(required) The IEN of a record in the PATIENT (#2) file.

Output:	returns:	Returns: **ID(0)**: If *not* defined at the point the ADDPAT^XULMU API is called, it is initially set to **zero** (**0**). When the ADDPAT^XULMU API returns, the **ID(0)** is incremented by **4**.
		`ID(ID(0)+1)=<patient name>`
		`ID(ID(0)+2)=<patient sex>`
		`ID(ID(0)+3)=<patient date of birth>`
		`ID(ID(0)+4)=<patient Social Security Number>`

17.2.2 PAT^XULMU(): Get a Standard Set of Patient Identifiers

Reference Type:	Supported	
Category:	Lock Manager	
ICR #:	5832	
Description:	The PAT^XULMU API is for use within the M code for a computable file reference to the PATIENT (#2) file. It returns a standard set of patient identifiers.	

NOTE: This API was released with Kernel Patch XU*8.0*608.

Format:	`PAT^XULMU(dfn)`	
Input Parameters:	**dfn**:	(required) The IEN of a record in the PATIENT (#2) file.
Output:	returns:	Returns the following variables:

- **ID("IEN")**=DFN
- **ID(0)=4**
- **ID(1)**=<*patient name*>
- **ID(2)**=<*patient sex*>
- **ID(3)**=<*patient date of birth*>
- **ID(4)**=<*patient Social Security Number*>

17.2.2.1 Example

Assuming that **DFN** is a variable defined within the Lock template, then the M code for a computable file reference to the PATIENT (#2) file is shown in Figure 146:

Figure 146: PAT^XULMU API—Example

```
DO PAT^XULMU(DFN)
```

18 Miscellaneous: Developer Tools

18.1 Direct Mode Utilities

Several Kernel Toolkit direct mode utilities are available for developers to use at the M prompt, usually involving the **DO** command. They are *not* APIs and *cannot* be used in software application routines.

Many of the options on the **Programmer Options** menu can also be run as direct mode utilities. Some are *not* available as options, but only as direct mode utilities callable at the M prompt. Table 28 lists examples on how to run these utilities when working in Programmer mode.

Table 28: Miscellaneous Tools—Direct Mode Utilities

Direct Mode Utility	Description
`>D ^%G`	List the contents of a global to the screen.

18.2 Programmer Options Menu

Figure 147: Programmer Options Menu Options—Toolkit Miscellaneous Tools

```
SYSTEMS MANAGER MENU ...                                          [EVE]
  Programmer Options ... <locked with XUPROG>                  [XUPROG]
   KIDS Kernel Installation & Distribution System ...        [XPD MAIN]
            <locked with XUPROG>
   PG Programmer mode <locked with XUPROGMODE>             [XUPROGMODE]
      Calculate and Show Checksum Values               [XTSUMBLD-CHECK]
      Delete Unreferenced Options                 [XQ UNREF'D OPTIONS]
      Error Processing ...                                     [XUERRS]
      General Parameter Tools ...                     [XPAR MENU TOOLS]
      Global Block Count                                 [XU BLOCK COUNT]
      List Global <locked with XUPROGMODE>                     [XUPRGL]
      Routine Tools ...                              [XUPR-ROUTINE-TOOLS]
      Test an option not in your menu <locked with XUMGR>  [XT-OPTION TEST]
```

18.2.1 Delete Unreferenced Options

The **Delete Unreferenced Options** [XQ UNREF'D OPTIONS] option examines those options that are *not*:

- Located on any menu.
- Used as primary or secondary options.
- Tasked to run.

The user can then decide in each case whether to delete the unreferenced option.

18.2.2 Global Block Count Option

The **Global Block Count** [XU BLOCK COUNT] option can be used to count the number of data blocks in a global.

18.2.3 Listing Globals Option

The **List Global** [XUPRGL] option is found on the **Programmer Options** menu, locked with the XUPROG security key. This option is also locked with the XUPROGMODE security key as an extra level of security.

It can be used to list the contents of a global to the screen. It makes use of operating system-specific utilities such as **%G**, the Global Lister.

The option is locked with the XUPROGMODE security key

The corresponding direct mode utility can be used in programmer mode. For example:

```
>D ^%G (OS-specific)
```

18.2.4 Test an option not in your menu Option

Use the **Test an option *not* in your menu** [XT-OPTION TEST] option for in-house testing of options only. It allows the selection of an option from the OPTION (#19 file) and then executes it. This option is locked with the XUMGR security key.

CAUTION: No security checks are performed in the XT-OPTION TEST option; therefore, it should only be given to programmers.

REF: Kernel Toolkit Application Programming Interfaces (APIs) are documented in the "Toolkit: Developer Tools" section. Kernel and Kernel Toolkit APIs are also available in HTML format on the VA Intranet website.

18.3 ^%Z Editor

18.3.1 User Interface

The **^%Z** Editor (routine editor) is installed in the Manager account as the **^%Z** global by ZTMGRSET during installation. (It can also be installed with **D ^ZTEDIT**.) To use the editor, load the routine (it *must* pre-exist) and then **X ^%Z**. The example in Figure 148 creates a one-line routine in Caché and then calls the **^%Z** Editor.

Figure 148: Calling the ^%Z Editor—Sample User Entries

```
>ZR

>ZZTEST <Enter> ;ID/SITE;test routine;
>ZS ZZTEST


>ZL ZZTEST X ^%Z

%Z Editing: ZZTEST  Terminal type: C-VT100
Edit:
```

Enter **.F** (**dot-file**) at the edit prompt to change files. When saving with **dot-file**, an edit comment can be entered. This text is stored in the EDIT HISTORY (#23) Multiple field in the ROUTINE (#9.8) file as programmer documentation. Figure 149 shows how an entire routine can be displayed by entering the **ZP** print command followed by a space at the M prompt. **Dot-file** (**.File**) is then used to file. A **dot** is then used to exit. (The **dot** exit does *not* automatically file changes.)

Figure 149: ^%Z Editor—Displaying a Routine Using the ZP Command

```
>ZL ZZTEST X ^%Z

%Z Editing: ZZTEST  Terminal type: C-VT100
Edit: ZP<SPACE> <Enter>
ZZTEST   ;test routine

Length:  20 <Enter>  Line: ZZTEST
ZZTEST ;test routine
Edit: .Insert after: ZZTEST// <Enter>

Line:     ;NEXT LINE
Line:     Q
Line: <Enter>
Edit: .FILE ZZTEST
Edit comment:
  1> This text is stored in the Routine file's Edit History multiple.
<Enter>
  2> <Enter>
EDIT Option: <Enter>
Edit: . <Enter>
>
```

Routines are filed by the name used when loading, *not* by the first line tag. If a ROUTINE (#9.8) file exists, then the routine is added if *not* already there, and an entry is made of the date/time and **DUZ** of the user that filed it. When filing, the editor updates the third piece of the first line of the routine with the date/time.

When editing, a question mark (**?**) can be entered to provide help. The **dot** commands are listed first. They provide the usual break, join, insert, and remove functions. The **+n** method of selecting lines to edit is also noted. The line tag can be used along with a number (e.g., **TAG+3**) to reach a particular line. A minus sign (**-**) backs up lines. And the asterisk (*****) can be entered to reach the last line.

Figure 150: ^%Z Editor—Listing Edit Commands

```
>X ^%Z
Edit: ?
.ACTION menu              .BREAK line              .CHANGE every
.FILE routine             .INSERT after            .JOIN lines
.MOVE lines               .REMOVE lines            .SEARCH for
.TERMinal type            .XY change to/from replace-with
.  -TO EXIT THE EDITOR
""+n Absolute line n    +n To advance n lines   -n To backup n lines_
 use '*' to get last line

^NAME - to edit a GLOBAL node        *NAME - to edit a LOCAL variable
MUMPS command line (mumps command <space> or Z command <space>)
```

Help displays information about editing in line mode. A complete line is displayed and various keys can be used to navigate. The **<Spacebar>** moves forward by words, the period moves forward by characters, and the **<CTRL H>** command key sequence moves backwards by characters. Upon reaching the desired location, the **<Delete>** key can be used to remove characters. To enter characters, the character **E** *must* first be entered as an insert/delete toggle. Pressing the **<Enter>** key reverses the toggle and allows navigation. Pressing the **<Enter>** key again moves back to the beginning of the line.

Figure 151: ^%Z Editor—Line Mode Help Information

```
In the line mode,
Spacebar moves to the next space or comma. Dot to the next char.
'>' To move forward 80 char or to end of line.
Backspace to back up one char. E to enter new char's at the cursor.
CR to exit enter mode, return to start of line or EDIT prompt.
D to delete from the cursor to the next space or comma.
Delete (Rub) to delete the char under the cursor.
CTRL-R to restore line and start back at the beginning.
```

Replace mode editing can be invoked by entering **dot-XY** at the edit prompt. This method allows easy string substitution, as in VA FileMan's Line Editor. Entering a question mark at the next edit prompt displays the following help:

Figure 152: ^%Z Editor—Replace Mode Editing Help Information

```
In the replace/with mode,
SPECIAL <REPLACE> STRINGS:
  END    -to add to the END of a line
  ...    -to replace a line
  A...B  -to specify a string that begins with "A" and ends with "B"
  A...   -to specify a string that begins with "A" to the end of the line
CTRL-R to restore line.
```

The **ACTION** menu provides additional functions. Save and restore lines can be used to move lines within one routine or from one routine to another. To copy lines to another routine, first save the lines, then load and edit the other routine, and restore the lines. When patching a routine, the **ACTION** menu can be used to calculate checksums. Before filing changes, the new checksum can be displayed and compared with the patch report for verification of editing. Figure 85 shows how to reach the **ACTION** menu with **dot-A** (**.A**).

Figure 153: ACTION Menu—Sample User Entries

```
Edit: .A
Action: ?
Bytes in routine              Checksum                  Restore lines
Save lines                    Version #
Action: C
                       Checksum is 4971725
Action: <Enter>
Edit: <Enter>
```

Global nodes and local variables may also be edited with the **^%Z** Editor. Editing occurs directly, so the idea of filing does *not* apply. The editor *must* then be exited with a dot, *not* with a **dot-file**, since filing should *not* take place.

18.4 Application Programming Interface (API)

The following are miscellaneous APIs available for developers. These APIs are described below.

18.4.1 Progress Bar Emulator

The following APIs can be use d to emulate a KIDS Progress Bar outside of KIDS. To create the progress bar, you *must* first call the INIT^XPDID: Progress Bar Emulator: Initialize Device and Draw Box Borders API, and when you are finished, you *must* call the EXIT^XPDID(): Progress Bar Emulator: Restore Screen, Clean Up Variables, and Display Text API.

18.4.1.1 INIT^XPDID: Progress Bar Emulator: Initialize Device and Draw Box Borders

Reference Type: Supported

Category: Miscellaneous

ICR #: 2172

Description: The INIT^XPDID API initializes the device, draws the borders for the progress bar box, and draws the progress bar. When you are finished, you *must* call the EXIT^XPDID(): Progress Bar Emulator: Restore Screen, Clean Up Variables, and Display Text API.

Format: `INIT^XPDID`

Input Parameters: none.

Output: returns: Returns **XPDIDVT**:

- **1**—If output device supports graphics.
- **0**—If output device does *not* support graphics.

18.4.1.2 TITLE^XPDID(): Progress Bar Emulator: Display Title Text

Reference Type:	Supported	
Category:	Miscellaneous	
ICR #:	2172	
Description:	The TITLE^XPDID API displays the text in the **x** input parameter as a title at the top of the progress bar box.	
Format:	`TITLE^XPDID(x)`	
Input Parameters:	**x**:	(required) Title text to be displayed at the top of the box.
Output:	none.	

18.4.1.3 EXIT^XPDID(): Progress Bar Emulator: Restore Screen, Clean Up Variables, and Display Text

Reference Type:	Supported	
Category:	Miscellaneous	
ICR #:	2172	
Description:	The EXIT^XPDID 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.	

18.4.2 Lookup Utility

18.4.2.1 $$EN^XUA4A71(): Convert String to Soundex

Reference Type:	Supported	
Category:	Miscellaneous	
ICR #:	3178	
Description:	The $$EN^XUA4A71 extrinsic function converts a string into a numeric representation of the string, using soundex methods. Soundex represents the phonetic properties of a string; its chief feature is that it assigns similar strings the same soundex representation.	
Format:	`$$EN^XUA4A71(string)`	
Input Parameters:	**string**:	(required) String to convert into soundex form.

Output:	returns:	Returns the soundex version of the string.

18.4.3 Date Conversions and Calculations

18.4.3.1 ^XQDATE: Convert $H to VA FileMan Format (Obsolete)

NOTE: The ^XQDATE 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
ICR #:	10079
Description:	The ^XQDATE 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 this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variable s:	**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.

18.4.3.2 ^XUWORKDY: Workday Calculation (Obsolete)

NOTE: Calling the XUWORKDY routine from the top is obsolete. The ^XUWORKDY API was replaced by the $$EN^XUWORKDY API. This API is dependent on the HOLIDAY (#40.5) file being updated by the sites.

Reference Type: Supported

Category: Miscellaneous

ICR #: 10046

Description: To use the ^XUWORKDY APIs, you *must* make sure that HOLIDAY (#40.5) file is populated with each year's holidays for the workday calculation to work correctly. If it is *not* populated, you need to populate it yourself (Kernel distributes this file without data). Only enter holidays that fall on weekdays, however.

You can call the ^XUWORKDY routine to calculate the number of workdays between two dates (**X**, **X1**). It returns a positive value if **X<X1** and a negative value if **X>X1**. If either date is imprecisely specified, or if the **HOLIDAY** global is empty, then ^XUWORKDY returns a **NULL** string.

- 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
```

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables:	**X:**	(required) Starting date in VA FileMan internal format (e.g., 2850420).
	X1:	(required) Ending date in VA FileMan internal format (e.g., 2850707).
Output:	**X:**	The number of workdays in the interval.

18.4.3.3 Example

Figure 154: ^XUWORKDY API—Example

```
>S X=2850420,X1=2850707 D ^XUWORKDY W X

55
```

18.4.3.4 $$EN^XUWORKDY: Number of Workdays Calculation

NOTE: This API is dependent on the HOLIDAY (#40.5) file being updated by the sites.

Reference Type:	Supported
Category:	Miscellaneous
ICR #:	10046
Description:	To use the ^XUWORKDY APIs, you *must* make sure that HOLIDAY (#40.5) file is populated with each year's holidays for the workday calculation to work correctly. If it is *not* populated, you need to populate it yourself (Kernel distributes this file without data). Only enter holidays that fall on weekdays, however.

The $$EN^XUWORKDY extrinsic function calculates the number of workdays between two dates (**date1**, **date2**). It returns:

- **Positive Value**—If **date1<date2**.
- **Negative Value**—If **date1>date2**.

- **NULL String**—If either date is imprecisely specified, or if the **HOLIDAY** global is empty.

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, the return value is set equal to the total counted days minus the number of weekend days between the two Sundays [**-(%I\7*2)**].

Format:	`$$EN^XUWORKDY(date1,date2)`	
Input Parameters:	**date1**:	(required) Starting date in VA FileMan internal format (e.g., 2850420).
	date2:	(required) Ending date in VA FileMan internal format (e.g., 2850707).
Output:	returns:	Returns the number of workdays in the interval.

18.4.3.5 Example

Figure 155: $$EN^XUWORKDY API—Example

```
>W $$EN^XUWORKDY(3090102,3090108)
4
```

18.4.3.6 $$WORKDAY^XUWORKDY: Workday Validation

NOTE: This API is dependent on the HOLIDAY (#40.5) file being updated by the sites.

Reference Type:	Supported
Category:	Miscellaneous
ICR #:	10046

Description: To use the ^XUWORKDY APIs, you *must* make sure that HOLIDAY (#40.5) file is populated with each year's holidays for the workday calculation to work correctly. If it is *not* populated, you need to populate it yourself (Kernel distributes this file without data). Only enter holidays that fall on weekdays, however.

The $$WORKDAY^XUWORKDY extrinsic function returns:

- **1**—If the date submitted is a workday.
- **0**—If the date submitted is *not* a workday.

If the date is imprecisely specified, or if the **HOLIDAY** global is empty, then $$WORKDAY^XUWORKDY returns a **NULL** string.

Format: `$$WORKDAY^XUWORKDY(date)`

Input Parameters: **date**: (required) Starting date in VA FileMan internal format returns: (e.g., **2850420**).

Output: returns: Returns:

- **1**—Workday
- **0**—*Non*-Workday

18.4.3.7 Examples

18.4.3.7.1 Example 1

Figure 156 shows the return value when a workday in VA FileMan internal format is input:

Figure 156: $$WORKDAY^XUWORKDY API—Example 1

```
>W $$WORKDAY^XUWORKDY(3090102)
1
```

18.4.3.7.2 Example 2

Figure 157 shows the return value when a *non*-workday in VA FileMan internal format is input:

Figure 157: $$WORKDAY^XUWORKDY API—Example 2

```
>W $$WORKDAY^XUWORKDY(3090103)
0
```

18.4.3.8 $$WORKPLUS^XUWORKDY: Workday Offset Calculation

NOTE: This API is dependent on the HOLIDAY (#40.5) file being updated by the sites.

Reference Type:	Supported	
Category:	Miscellaneous	
ICR #:	10046	
Description:	To use the ^XUWORKDY APIs, you *must* make sure that HOLIDAY (#40.5) file is populated with each year's holidays for the workday calculation to work correctly. If it is *not* populated, you need to populate it yourself (Kernel distributes this file without data). Only enter holidays that fall on weekdays, however. The $$WORKPLUS^XUWORKDY extrinsic function returns the date that is **n** working days (i.e., offset) +/- of the input date. If the date is imprecisely specified, or if the **HOLIDAY** global is empty, then $$WORKPLUS^XUWORKDY returns a **NULL** string.	
Format:	`$$WORKPLUS^XUWORKDY(date,offset)`	
Input Parameters:	**date**:	(required) Starting date in VA FileMan internal format (e.g., **2850420**).
	offset:	(required) The number of days to offset.
Output:	returns:	Returns the date in VA FileMan internal format that is **n** working days (i.e., offset) +/- of the input date.

18.4.3.9 Example

Figure 158: $$WORKPLUS^XUWORKDY API—Example

```
>W $$WORKPLUS^XUWORKDY(3090108,3)
3090113
```

19 Name Standardization: Developer Tools

19.1 Application Programming Interface (API)

Several APIs are available for developers to work with name standardization. These APIs are described below.

19.1.1 $$BLDNAME^XLFNAME(): Build Name from Component Parts

Reference Type: Supported

Category: Name Standardization

ICR #: 3065

Description: The $$BLDNAME^XLFNAME 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 (#20) file, 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**). **REF:** For a description of the pruning algorithm, see the "Details" section.
Output:	returns:	Returns the name, truncated if necessary, in the following format: `Family_name,Given_name<space>Middle_name <space>Suffix(es)`

19.1.1.1 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.

19.1.1.2 Examples

19.1.1.2.1 Example 1

Suppose the MYNAME array contains the following elements:

```
MYNAME("FAMILY")="XUUSER"
MYNAME("GIVEN")="SIXTY"
MYNAME("MIDDLE")="K."
MYNAME("SUFFIX")="JR"
```

Calls to $$BLDNAME^XLFNAME returns the name as follows:

Figure 159: $$BLDNAME^XLFNAME API—Example 1: All Characters

```
>S X=$$BLDNAME^XLFNAME(.MYNAME)

>W X
XUUSER,SIXTY K JR
```

"Pruning" the name to 12 characters total:

Figure 160: $$BLDNAME^XLFNAME API—Example 1: Only 12 Characters

```
>S X=$$BLDNAME^XLFNAME(.MYNAME,12)

>W X
XUUSER,SI K
```

19.1.1.2.2 Example 2

If an entry in the NAME COMPONENTS (#20) file stores the components of a name stored in the NAME (#.01) field of record number 32 in the NEW PERSON (#200) file, and the data in the corresponding record in the NAME COMPONENTS (#20) file is:

```
FILE=200
FIELD=.01
IENS="32,"
GIVEN NAME="SIXTY"
MIDDLE NAME="K."
FAMILY NAME="XUUSER"
SUFFIX="JR"
```

You can set:

```
MYNAME("FILE")=200
MYNAME("FIELD")=.01
MYNAME("IENS")="32,"
```

Then call $$BLDNAME^XLFNAME as in Example 1:

Figure 161: $$BLDNAME^XLFNAME API—Example 2: All Characters

```
>S X=$$BLDNAME^XLFNAME(.MYNAME)

>W X
XUUSER,SIXTY K JR
```

“Pruning” the name to 12 characters total:

Figure 162: $$BLDNAME^XLFNAME API—Example 2: Only 12 Characters

```
>S X=$$BLDNAME^XLFNAME(.MYNAME,12)

>W X
XUUSER,SI K
```

19.1.2 $$CLEANC^XLFNAME(): Name Component Standardization Routine

Reference Type: Supported

Category: Name Standardization

ICR #: 3065

Description: The $$CLEANC^XLFNAME 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 uppercase.
  - 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.

19.1.2.1 Examples

19.1.2.1.1 Example 1

Standardize family (last) name:

Figure 163: $$CLEANC^XLFNAME API—Example 1

```
>Set X=$$CLEANC^XLFNAME("XUUSER-XU U SER","F")

>W X
XUUSER-XUUSER

>Set X=$$CLEANC^XLFNAME("XUUSER-XU U SER 2ND","F")

>W X
XUUSER-XUUSERII

>Set X=$$CLEANC^XLFNAME("XUUSER-XU U SER")

>W X
XUUSER-XU U SER

>Set X=$$CLEANC^XLFNAME("ST. USER","F")

>W X
STUSER
```

19.1.2.1.2 Example 2

Standardize other (*non*-family) name components:

Figure 164: $$CLEANC^XLFNAME API—Example 2

```
>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
```

19.1.3 $$FMNAME^XLFNAME(): Convert HL7 Formatted Name to Name

Reference Type:	Supported
Category:	Name Standardization
ICR #:	3065
Description:	The $$FMNAME^XLFNAME extrinsic function converts an HL7 formatted input **name** to a VistA formatted name.
Format:	`$$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 the "Output Parameters" section).
	flags:	(optional) Flags to controls processing. Possible values are: • **C**—Return name components in the **NAME** array (see the "Output Parameters" section). • **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)
```

19.1.3.1 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.

19.1.3.2 Examples

19.1.3.2.1 Example 1

Convert an HL7 formatted name to a VistA name:

Figure 165: $$FMNAME^XLFNAME API—Example 1

```
>S X=$$FMNAME^XLFNAME("XUUSER^SIXTY^K.^JR^MR.^PHD")

>W X
XUUSER,SIXTY K. JR

>S X=$$FMNAME^XLFNAME("XUUSER^SIXTY^K.^JR^MR.^PHD","S")

>W X
XUUSER,SIXTY K JR

>S X=$$FMNAME^XLFNAME("XUUSER^SIXTY^K.^JR^MR.^PHD","M")

>W X
Xuuser,Sixty K. Jr

>S X=$$FMNAME^XLFNAME("XUUSER^SIXTY^K.^JR^MR.^PHD","SL12")

>W X
XUUSER,SI K
```

19.1.3.2.2 Example 2

Convert an HL7 formatted name where the tilde character (~) is the delimiter to a standard name:

Figure 166: $$FMNAME^XLFNAME API—Example 2

```
>S X=$$FMNAME^XLFNAME("XUUSER~SIXTY~K.~JR~MR","S","~")

>W X
XUUSER,SIXTY K JR
```

19.1.3.2.3 Example 3

Convert an HL7 formatted name to a standard name, and return the components of that name in the MYNAME array:

Figure 167: $$FMNAME^XLFNAME API—Example 3: Converting an HL7 Formatted Name to a Standard Name, and Returning the Components in an Array

```
>S MYNAME="XUUSER^SIXTY^K.^JR^MR.^PHD"

>W $$FMNAME^XLFNAME(.MYNAME,"CS")
XUUSER,SIXTY K JR

>ZW MYNAME
MYNAME=XUUSER^SIXTY^K.^JR^MR.^PHD
MYNAME("DEGREE")=PHD
MYNAME("FAMILY")=XUUSER
MYNAME("GIVEN")=SIXTY
MYNAME("MIDDLE")=K.
MYNAME("PREFIX")=MR.
MYNAME("SUFFIX")=JR
```

19.1.4 $$HLNAME^XLFNAME(): Convert Name to HL7 Formatted Name

Reference Type: Supported

Category: Name Standardization

ICR #: 3065

Description: The $$HLNAME^XLFNAME 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 (#20) file, 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 (#20) file.

	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 is ^).
Output:	returns:	Returns the converted name in HL7 format.

19.1.4.1 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.

19.1.4.2 Examples

19.1.4.2.1 Example 1

Suppose the MYNAME array contains the following elements:

```
MYNAME("PREFIX")="MR."
MYNAME("GIVEN")="SIXTY"
MYNAME("MIDDLE")="K."
MYNAME("FAMILY")="XUUSER"
MYNAME("SUFFIX")="JR"
MYNAME("DEGREE")="PHD"
```

Then calls to the $$HLNAME^XLFNAME API returns the name, as shown in Figure 168:

Figure 168: $$HLNAME^XLFNAME API—Example 1

```
>S X=$$HLNAME^XLFNAME(.MYNAME)

>W X
XUUSER^SIXTY^K.^JR^MR.^PHD

>S X=$$HLNAME^XLFNAME(.MYNAME,"","~")

>W X
XUUSER~SIXTY~K.~JR~MR.~PHD

>S X=$$HLNAME^XLFNAME(.MYNAME,"S","~")

>W X
XUUSER~SIXTY~K~JR~MR~PHD

>S X=$$HLNAME^XLFNAME(.MYNAME,"L12S")

>W X
XUUSER^SI^K
```

19.1.4.2.2 Example 2

If an entry in the NAME COMPONENTS (#20) file stores the components of a name stored in the NAME (#.01) field of record number 32 in the NEW PERSON (#200) file, and the data in the corresponding record in the NAME COMPONENTS (#20) file is:

```
FILE = 200
FIELD = .01
IENS = "32,"
PREFIX = "MR."
GIVEN NAME = "SIXTY"
MIDDLE NAME = "K."
FAMILY NAME = "XUUSER"
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.

19.1.4.2.3 Example 3

Convert a name passed by value to HL7 format:

Figure 169: $$HLNAME^XLFNAME API—Example 3

```
>S X=$$HLNAME^XLFNAME("XUUSER,SIXTY HXXX II")

>W X
XUUSER^SIXTY^HXXX^II

>S X=$$HLNAME^XLFNAME("XUUSER,SIXTY HXXX II","S")

>W X
XUUSER^SIXTY^HXXX^II

>S X=$$HLNAME^XLFNAME("XUUSER,SIXTY HXXX II","SL10","~")

>W X
XUUSE~S~H
```

19.1.5 NAMECOMP^XLFNAME(): Component Parts from Standard Name

Reference Type:	Supported	
Category:	Name Standardization	
ICR #:	3065	
Description:	The NAMECOMP^XLFNAME 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 Parameters.")
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)
```

19.1.5.1 Example

In Figure 170, the **MYNAME** variable is set to the standard name. The NAMECOMP^XLFNAME call is made to return in the **MYNAME** array the component parts of that name:

Figure 170: NAMECOMP^XLFNAME API—Example

```
>S MYNAME="XUUSER-XUUSER,FORTY ONE S MD"
>D NAMECOMP^XLFNAME(.MYNAME)

>ZW MYNAME
MYNAME=XUUSER-XUUSER,FORTY ONE S MD
MYNAME("FAMILY")=XUUSER-XUUSER
MYNAME("GIVEN")=FORTY ONE
MYNAME("MIDDLE")=S
MYNAME("SUFFIX")=MD
```

19.1.6 $$NAMEFMT^XLFNAME(): Formatted Name from Name Components

Reference Type: Supported

Category: Name Standardization

ICR #: 3065

Description: The $$NAMEFMT^XLFNAME 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 (#20) file, 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 the "Details" section 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.

19.1.6.1 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.

19.1.6.2 Examples

19.1.6.2.1 Example 1

Suppose the MYNAME array contains the following elements:

```
MYNAME("PREFIX")="MR."
MYNAME("GIVEN")="SIXTY"
MYNAME("MIDDLE")="K."
MYNAME("FAMILY")="XUUSER"
MYNAME("SUFFIX")="JR"
MYNAME("DEGREE")="PHD"
```

Then calls to the $$NAMEFMT^XLFNAME API returns the name as follows:

Figure 171: $$NAMEFMT^XLFNAME API—Example 1

```
>S X=$$NAMEFMT^XLFNAME(.MYNAME,"F")

>W X
XUUSER SIXTY K. JR

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"F","C")

>W X
XUUSER,SIXTY K. JR

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"F","CS")

>W X
XUUSER,SIXTY K JR

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"F","CSD")

>W X
XUUSER,SIXTY K JR PHD

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"F","CDcXc")

>W X
XUUSER,SIXTY K., JR, PHD

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"F","CSL12")

>W X
XUUSER,SI K

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"F","CMD")

>W X
Xuuser,Sixty K. Jr PhD

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G")

>W X
SIXTY K. XUUSER JR

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G","D")

>W X
SIXTY K. XUUSER JR PHD

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G","Dc")

>W X
SIXTY K. XUUSER JR, PHD

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G","P")
```

```
>W X
MR. SIXTY K. XUUSER JR

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G","Xc")

>W X
SIXTY K. XUUSER, JR

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G","PDcXc")

>W X
MR. SIXTY K. XUUSER, JR, PHD

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G","PDcXcM")

>W X
Mr. Sixty K. Xuuser, Jr, PhD

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G","S")

>W X
SIXTY K XUUSER JR

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"G","SL12")

>W X
SI K XUUSER

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"O")

>W X
XUUSER

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"O","S")

>W X
XUUSER

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"O","M")

>W X
Xuuser

>S X=$$NAMEFMT^XLFNAME(.MYNAME,"O","L3")

>W X
XU
```

19.1.6.2.2 Example 2

If an entry in the NAME COMPONENTS (#20) file stores the components of a name stored in the NAME (#.01) field of record number 32 in the NEW PERSON (#200) file, and the data in the corresponding record in the NAME COMPONENTS (#20) file is:

```
FILE = 200
FIELD = .01
IENS = "32,"
PREFIX = "MR."
GIVEN NAME = "SIXTY"
MIDDLE NAME = "K."
FAMILY NAME = "XUUSER"
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.

19.1.7 STDNAME^XLFNAME(): Name Standardization Routine

Reference Type: Supported

Category: Name Standardization

ICR #: 3065

Description: The STDNAME^XLFNAME 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) 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 the "Output Parameters" section.)

flags: (optional) Flags to control processing. Possible values are:

- **C**—Return name components in the **NAME** array. (See the "Output Parameters" section.)
- **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**—Do *not* 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 the "Output Parameters" section.)

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, descendants 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 is *not* passed.
- **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 does *not* contain any vowels or **Y**. It is interpreted as a suffix.
    - Suffix is found between commas immediately after the Family Name.

19.1.7.1 Details

19.1.7.1.1 Standard Name

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.

19.1.7.1.2 Component Parts Name

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 carets (^) 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**, **ARNP**, **DO**, **PA**, and Roman numerals **I** through **X**.

NOTE: The **ARNP**, **DO**, and **PA** suffixes were added with Kernel Patch XU*8.0*535.

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.

19.1.7.2 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.

Figure 172: STDNAME^XLFNAME API—Example

```
>S MYNAME="XUUSER,FIFTY A. B. 2ND (TEST)"
>D STDNAME^XLFNAME(.MYNAME,"CP",.MYAUD)

>ZW MYNAME
MYNAME=XUUSER,FIFTY A B II
MYNAME("FAMILY")=XUUSER
MYNAME("GIVEN")=FIFTY A.
MYNAME("MIDDLE")=B.
MYNAME("SUFFIX")=2ND

>ZW MYAUD
MYAUD=XUUSER,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 **XUUSER,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.

19.1.8 DELCOMP^XLFNAME2(): Delete Name Components Entry

Reference Type:	Controlled Subscription
Category:	Name Standardization
ICR #:	3066
Description:	The DELCOMP^XLFNAME2 API deletes an entry in the NAME COMPONENTS (#20) file, and optionally, the value of the pointer in the source file that points to the name components entry.

NOTE: The DELCOMP^XLFNAME2 API is designed to be used in the **KILL** logic for the MUMPS cross-reference mentioned 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 (#20) file. Only if this parameter is passed is the value of this POINTER field deleted.
Output:	none.	Deletes record.

19.1.8.1 Example

Suppose that you have a NAME COMPONENTS (#20) file entry that contains the components of a name stored in File #1000, Record #132, Field #.01. The POINTER Field #1.1 of that File #1000 is a pointer to the NAME COMPONENTS (#20) file. To delete the entry in the NAME COMPONENTS (#20) file, and the value of the POINTER field, you can do the following:

Figure 173: DELCOMP^XLFNAME2 API—Example

```
>D DELCOMP^XLFNAME(1000,132,.01,1.1)
```

19.1.9 UPDCOMP^XLFNAME2(): Update Name Components Entry

Reference Type: Controlled Subscription

Category: Name Standardization

ICR #: 3066

Description: The UPDCOMP^XLFNAME2 API updates an entry in the NAME COMPONENTS (#20) file. 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 (#.11) file. The **ANAME** index is a MUMPS cross-reference on the NAME (#.01) field of the NEW PERSON (#200) file. If an entry’s NAME field is edited, the **ANAME** cross-reference updates the associated entry in the NAME COMPONENTS (#20) file.

NOTE: Existing MUMPS cross-references on the NAME COMPONENTS (#20) file 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 (#20) file 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 descendants [i.e., **$D(NAME)=1**], UPDCOMP^XLFNAME2 makes 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 (#20) file. Only if this parameter is passed is the value of this POINTER field updated with the entry number of the record in the NAME COMPONENTS (#20) file 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 is updated only if this value is different from the entry number of the record in the NAME COMPONENTS (#20) file that was added or edited.
Output:	returns:	Updated entry in the NAME COMPONENTS (#20) file.

19.1.9.1 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: XUUSER
- Given (first) name: FIFTY JXXX
- Middle name: A.
- Suffix: JR.

Field #1.1 is defined as a pointer to the NAME COMPONENTS (#20) file and has a value of 42, the IEN of a record in the NAME COMPONENTS (#20) file. To update the NAME COMPONENTS (#20) file with this name, you can do the following:

Figure 174: UPDCOMP^XLFNAME2 API—Example

```
>S MYNAME("FAMILY")="XUUSER"
>S MYNAME("GIVEN")="FIFTY JXXX"
>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 (#20) file 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 (#20) file updates the name in the Field #.01 of File #1000 to "XUUSER,FIFTY JXXX A JR" if it does *not* already contain that name.

20 National Provider Identifier (NPI): Developer Tools

20.1 Application Programming Interface (API)

The following are National Provider Identifier (NPI) APIs available for developers. These APIs are described below.

20.1.1 $$CHKDGT^XUSNPI(): Validate NPI Format

Reference Type:	Controlled Subscription	
Category:	National Provider Identifier (NPI)	
ICR #:	4532	
Description:	The $$CHKDGT^XUSNPI extrinsic function validates the format of a National Provider Identifier (NPI) number. It checks the following: • NPI is numeric. • Length of the Number (*must* be **10**-digits). • Check Digit is Valid. **NOTE:** This API was released with Kernel Patch XU*8.0*410.	
Format:	`$$CHKDGT^XUSNPI(xusnpi)`	
Input Parameters:	**xusnpi**:	(required) The **10**-digit National Provider Identifier (NPI) number to validate. No default.
Output:	returns:	Returns: • **1**—If check digit is valid. The NPI number *must* be **10**-digits long. • **0**—If check digit is *not* valid.

20.1.1.1 Examples

20.1.1.1.1 Example 1

Figure 175 shows the result when checking a valid NPI:

Figure 175: $$CHKDGT^XUSNPI API—Example 1

```
>W $$CHKDGT^XUSNPI(1234567893)
1
```

20.1.1.1.2 Example 2

Figure 176 shows the result when checking an invalid NPI (*not* **10** digits):

Figure 176: $$CHKDGT^XUSNPI API—Example 2

```
>W $$CHKDGT^XUSNPI(123456789)
0
```

20.1.1.1.3 Example 3

Figure 177 shows the result when checking an invalid NPI (*invalid* digit):

Figure 177: $$CHKDGT^XUSNPI API—Example 3

```
>W $$CHKDGT^XUSNPI(1234567892)
0
```

20.1.2 $$NPI^XUSNPI(): Get NPI from Files #200, #4, or #355.93

Reference Type: Controlled Subscription

Category: National Provider Identifier (NPI)

ICR #: 4532

Description: The $$NPI^XUSNPI extrinsic function retrieves the National Provider Identifier (NPI) and related utilities from any of the following files:

- NEW PERSON (#200)
- INSTITUTION (#4)
- IB NON/OTHER VA BILLING PROVIDER (#355.93)

NOTE: This API was released with Kernel Patch XU*8.0*410.

Format: `$$NPI^XUSNPI(xusqi,xusien[,xusdate])`

Input Parameters: **xusqi**: (required) The Qualified Identifier for the NPI. For example:

Individual_ID, Organization_ID, or Non_VA_Provider_ID

No default.

	xusien:	(required) The Internal Entry Number (IEN) from any of the following files: • NEW PERSON (#200) • INSTITUTION (#4) • IB NON/OTHER VA BILLING PROVIDER (#355.93) No default.
	xusdate:	(optional) A date of interest. Defaults to “**Today**”.
Output:	returns:	Returns any of the following strings: • **NPI^EffectiveDate^Status**—If National Provider Identifier (NPI) exists. • **0**—If NPI does *not* exist. • **-1^ErrorMessage**—If invalid **xusqi** or **xusien** input parameters.

20.1.2.1 Examples

20.1.2.1.1 Example 1

The example in Figure 178 uses the following file data:

- Individual_ID = **NEW PERSON (#200)** file
- NPI = **9876543213**
- EffectiveDate = **3061108.123651**
- Status = **Active**

Figure 178: $$NPI^XUSNPI API—Example 1

```
>W $$NPI^XUSNPI(“Individual_ID”,82)
9876543213^3061108.123651^Active
```

20.1.2.1.2 Example 2

The example in Figure 179 uses the following file data:

- Organization_ID = **INSTITUTION (#4)** file
- NPI = **1111111112**
- EffectiveDate = **3070122**
- Status = **Active**

Figure 179: $$NPI^XUSNPI API—Example 2

```
>W $$NPI^XUSNPI("Organization_ID",1)
1111111112^3070122^Active
```

20.1.2.1.3 Example 3

The example in Figure 180 uses the following file data:

- Non_VA_Provider_ID = **IB NON/OTHER VA BILLING PROVIDER (#355.93)** file
- NPI = **2222222228**
- EffectiveDate = **3070122**
- Status = **Active**

Figure 180: $$NPI^XUSNPI API—Example 3

```
>W $$NPI^XUSNPI("Non_VA_Provider_ID ",1)
2222222228 ^3070122^Active
```

20.1.3 $$QI^XUSNPI(): Get Provider Entities

Reference Type: Controlled Subscription

Category: National Provider Identifier (NPI)

ICR #: 4532

Description: The $$QI^XUSNPI extrinsic function retrieves all qualified provider entities for a National Provider Identifier (NPI) identifier.

NOTE: This API was released with Kernel Patch XU*8.0*410.

Format: `$$QI^XUSNPI(xusnpi)`

Input Parameters: **xusnpi**: (required) The National Provider Identifier (NPI) identifier. No default.

Output:	returns:	Returns either of the following strings: • **QualifiedIdentifier^IEN^EffectiveDate^Status**—National Provider Identifier (NPI) exists. If more than one record is found, they are separated by a semi-colon (**;**). • **0**—Qualified NPI does *not* exist.

20.1.3.1 Examples

20.1.3.1.1 Example 1

The example in Figure 181 uses the following file data:

- Individual_ID = **NEW PERSON (#200)** file
- IEN = **82**
- EffectiveDate = **3061108.123651**
- Status = **Active**

Figure 181: $$QI^XUSNPIAPI—Example 1

```
>W $$QI^XUSNPI(9876543213)
Individual_ID^82^3061108.123651^Active;
```

20.1.3.1.2 Example 2

The example in Figure 182 uses the following file data:

- Organization_ID = **INSTITUTION (#4)** file
- IEN = **1**
- EffectiveDate = **3070122**
- Status = **Active**

Figure 182: $$QI^XUSNPIAPI—Example 2

```
>W $$QI^XUSNPI(1111111112)
Organization_ID^1^3070122^Active;
```

20.1.3.1.3 Example 3

The example in Figure 183 uses the following file data:

- Non_VA_Provider_ID = **IB NON/OTHER VA BILLING PROVIDER (#355.93)** file
- IEN = **3**
- EffectiveDate = **3070122**
- Status = **Active**

Figure 183: $$QI^XUSNPI API—Example 3

```
>W $$QI^XUSNPI(2222222228)
Non_VA_Provider_ID^3^3070122^Active;
```

20.1.4 $$NPIUSED^XUSNPI1(): Returns an Error or Warning if an NPI is in Use

Reference Type: Controlled Subscription

Category: National Provider Identifier (NPI)

ICR #: 6888

Description: The $$NPIUSED^XUSNPI1 extrinsic function returns an error or warning if an NPI is in use.

Call this API from code where a new NPI is being added to a provider. It evaluates whether the NPI is currently or previously used by any entity on any of the following files:

- NEW PERSON (#200)
- INSTITUTION (4)
- IB NON/OTHER VA BILLING PROVIDER (#355.93)

If the API returns:

- **Error**—NPI should *not* be assigned to the provider.
- **Warning**—Warning should be displayed to the end user, but they should be allowed to add the NPI to the new provider.

NOTE: This API was released with Kernel Patch XU*8.0*480.

Format: `$$NPIUSED^XUSNPI1(xusnpi,xusqid,xusqil,xusrslt[,xusien])`

Input Parameters:	**xusnpi**:	(required) The NPI being checked. No default.
	xusqid:	(required) The Qualified Identifier for the NPI (e.g., "Individual_ID"). No default.
	xusqil:	(required) The delimited list of entities already using that NPI. No default. This is the output from $$QI^XUSNPI in the following format: `Qualified_Identifier^IEN^Effective_date/time^Active/Inactive;`
	xusien:	(optional) This input parameter is *only* set if this routine is being called from the Input transform of the NPI field in any of the following files: • NEW PERSON (#200) • INSTITUTION (4) • IB NON/OTHER VA BILLING PROVIDER (#355.93) It is set to the IEN of the entity being edited. No Default. **CAUTION: This input parameter should *only* be set if the routine is being called from an Input transform. It suppresses return of the error or warning message.**
Output Parameter:	**xusrslt**:	An array containing either an error or warning message (if any).
Output:	returns:	Returns: • **0 (Zero; No Error)**—If the NPI is *not* being used, or if the API is called from the Input transform and the NPI was previously used by the current user. • **1 (Error)**—If an error was found, an *error* message is returned in **xusrslt**.

- **2 (Warning)**—If the current file is the NEW PERSON (#200) or IB NON/OTHER VA BILLING PROVIDER (#355.93), and if a provider on the other file has the NPI, a *warning* message is returned in **xusrslt**.

  NOTE: A provider can be both a VA and a *non*-VA provider at the same time.

20.1.5 $$TAXIND^XUSTAX(): Get Taxonomy Code from File #200

Reference Type:	Controlled Subscription	
Category:	National Provider Identifier (NPI)	
ICR #:	4911	
Description:	The $$TAXIND^XUSTAX extrinsic function retrieves the taxonomy code for a given record in the NEW PERSON (#200) file. **NOTE:** This API was released with Kernel Patch XU*8.0*410.	
Format:	`$$TAXIND^XUSTAX(xuien)`	
Input Parameters:	**xuien**:	(required) This is the Internal Entry Number (IEN) of the record in the NEW PERSON (#200) file. No default.
Output:	returns:	Returns either of the following strings: • **TaxonomyX12Code^TaxonomyIEN**—Taxonomy exists. • ^—Taxonomy does *not* exist.

20.1.5.1 Example

The following example uses the following file data:

- Taxonomy **X12** code of the record in the NEW PERSON (#200) file = 2086S0105
- Taxonomy IEN from the PERSON CLASS (#8932.1) file = 900

Figure 184: $$TAXIND^XUSTAX API—Example

```
>W $$TAXIND^XUSTAX(82)
2086S0105X^900
```

20.1.6 $$TAXORG^XUSTAX(): Get Taxonomy Code from File #4

Reference Type:	Controlled Subscription	
Category:	National Provider Identifier (NPI)	
ICR #:	4911	
Description:	The $$TAXORG^XUSTAX extrinsic function retrieves the taxonomy code for a given record in the INSTITUTION (#4) file. **NOTE:** This API was released with Kernel Patch XU*8.0*410.	
Format:	`$$TAXORG^XUSTAX(xuien)`	
Input Parameters:	**xuien**:	(required) This is the Internal Entry Number (IEN) of the record in the INSTITUTION (#4) file. No default.
Output:	**r**eturns:	Returns either of the following strings: • **TaxonomyX12Code^TaxonomyIEN**—Taxonomy exists. • ^—Taxonomy does *not* exist.

20.1.6.1 Example

The following example uses the following file data:

- Taxonomy **X12** code of the record in the INSTITUTION (#4) file = 390200000X
- Taxonomy IEN from the PERSON CLASS (#8932.1) file = 144

Figure 185: $$TAXORG^XUSTAX API—Example

```
>W $$TAXORG^XUSTAX(2)
390200000X^144
```

21 Operating System (OS) Interface: Developer Tools

21.1 Overview

Kernel and Kernel Toolkit 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.

21.2 Direct Mode Utilities

21.2.1 >D ^%ZTBKC: Global Block Count

You can count the data blocks in a global using the **^%ZTBKC** direct mode utility. An entire global or a subscripted section can be measured, such as ^DIC or ^DIC(9.2). There is a corresponding option that can be used from the **Programmer Options** menu, called the **Global Block Count** [XU BLOCK COUNT] option.

REF: For more information on the XU BLOCK COUNT, see Section 28, "Miscellaneous Programmer Tools," in the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide.*

21.2.2 >D ^ZTMGRSET: Update ^%ZOSF Nodes

The **^ZTMGRSET** direct mode utility is only available from the manager's account. It is ordinarily run during Kernel installations to initialize Kernel in the manager's account. It can be used at a later time, however, to update an account's **^%ZOSF** nodes with new UCI and Volume Set information. The **^%ZOSF** nodes that **^ZTMGRSET** updates are:

- **^%ZOSF("MGR")**
- **^%ZOSF("PROD")**
- **^%ZOSF("VOL")**

An example of a use for re-running **^ZTMGRSET** would be when creating a new print, compute, file, or shadow server by copying an existing server's account. Although Kernel is already set up in the copied account, the new server's UCI and Volume Set **^%ZOSF** nodes would need to be updated from their old values to the values needed for the new server. Re-running **^ZTMGRSET** allows these values to be updated.

21.3 Application Programming Interface (API)

Several APIs are available for developers to work with the operating system. These APIs are described below.

21.3.1 $$CPUTIME^XLFSHAN: Return System and User CPU Time

Reference Type:	Supported	
Category:	Operating System Interface	
ICR #:	6157	
Description:	The $$CPUTIME^XLFSHAN extrinsic function returns two comma-delimited pieces: • "system" CPU time. • "user" CPU time (except on VMS where no separate times are available). **NOTE:** This API was released with Kernel Patch XU*8.0*657.	
Format:	`$$CPUTIME^XLFSHAN`	
Input Parameters:	none.	
Output:	returns:	The value returned is time measured as milliseconds of CPU time.

21.3.2 $$ETIMEMS^XLFSHAN(): Return Elapsed Time in Milliseconds

Reference Type:	Supported	
Category:	Operating System Interface	
ICR #:	6157	
Description:	The $$ETIMEMS^XLFSHAN extrinsic function calculates the elapsed time in milliseconds. It is intended to be used with $$CPUTIME^XLFSHAN API to evaluate the performance of a process. **NOTE:** This API was released with Kernel Patch XU*8.0*657.	
Format:	`$$ETIMEMS^XLFSHAN(start,end)`	
Input Parameters:	**start**:	(required) The starting CPU time; set by calling the $$CPUTIME^XLFSHAN API.

	end:	(required) The ending CPU time; set by calling the $$CPUTIME^XLFSHAN API.
Output:	returns:	The value returned is elapsed time measured as milliseconds of CPU time.

21.3.3 ^%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 29, however, are reference values. For example, to **WRITE** the operating system, use:

```
W ^%ZOSF("OS")
```

The nodes prefaced with two asterisks in Table 29 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**).

Table 29: ^%ZOSF API—Global Nodes

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.
EOT	Returns **Y = 1** if Magtape end-of-tape mark is detected.

Node	Description

**ERRTN	This node is set to the name of the routine that should be used to record errors. For most systems this is the KERNEL error recording routine (**%ZTER**):
	`>D @^%ZOSF("ERRTN")`
	To initially set the Error Trap:
	`>S X=^%ZOSF("ERRTN"),@^%ZOSF("TRAP")`

ETRP	Obsolete.
GD	Display the global directory.
GSEL	Returns the user's selection of globals as follows: `^UTILITY($J,"global name")` **NOTE:** This is only supported for Caché at this time.
JOBPARAM	When passed the job in **X**, returns the UCI for that job in **Y**. It determines whether the job is valid on the system.
LABOFF	Turn off echo to the **IO** device.
LOAD	Load routine **X** into **@(DIF_"XCNP,0)**".
LPC	Returns in **Y** the longitudinal parity check of the string in **X**.
MAGTAPE	Sets the **%MT** local variable to hold magtape functions. Issue the backspace command as follows: `>W @%MT("BS")` The full list of functions are: • **BS—**Back Space • **FS—**Forward Space • **WTM—WRITE** Tape Mark • **WB—WRITE** Block • **REW—**Rewind • **RB—READ** Block • **REL—READ** Label • **WHL—WRITE HDR** Label • **WEL—WRITE EOF** Label
MAXSIZ	For M/SQL-VAX only. Sets the partition size to **X**.
*MGR	Holds the name of the **MGR** account (UCI, Volume Set).
MTBOT	Returns **Y = 1** if the magtape is at **BOT**.
MTERR	Returns **Y = 1** if a magtape error is detected.
MTONLINE	Returns **Y = 1** if the magtape is online.
MTWPROT	Returns **Y = 1** if the magtape is **WRITE** Protected.
NBRK	Do *not* allow the user to break a routine.

Node	Description
NO-PASSALL	Sets device **$I** to interpret tabs, carriage returns, line feeds, or control characters (normal text mode).
NO-TYPE-AHEAD	Turn off the TYPE-AHEAD for the device **$I**.
*OS	In the first **^** piece, holds the type of MUMPS (e.g., Caché, VAX DSM, GT.M).
PASSALL	Sets device **$I** to pass all codes, allow tabs, carriage returns, and other control characters to be passed (binary transfer).
PRIINQ	Returns **Y** with the current priority of the job.
PRIORITY	Sets the priority of the job to **X** (**1** is low, **10** is high).
*PROD	Holds the name of the Production account (UCI, Volume Set).
PROGMODE	Returns **Y = 1** if the user is in Programmer mode.
RD	Displays the routine directory.
RESJOB	References the operating system routine for restoring a job.
RM	Sets the **$I** width to **X** characters. If **X=0**, then the line in set to no wrap.
RSEL	Returns the user's selection of routines as follows: `^UTILITY($J,"routine name")`
RSUM	Passes a routine name in **X**, and it returns the checksum in **Y**. Used by CHECK^XTSUMBLD. The second line and comments are *not* included in the total.
RSUM1	Passes a routine name in **X**, and it returns the checksum in **Y**. Used by CHECK1^XTSUMBLD. The second line and comments are *not* included in the total.
SAVE	Saves the code in **@(DIE_"XCN,0)")** as routine **X**.
SIZE	Returns **Y=size** (in bytes) of the current routine.
SS	Displays the system status.
TEST	Returns **$T = 1** if routine **X** exists.
TMK	Returns **Y = 1** if a tape mark was detected on the last **READ**.
TRAP	To set the Error Trap: `>S X="error routine",@^%ZOSF("TRAP")`
TRMOFF	Resets terminators to normal.
TRMON	Turns on all controls as terminators.
TRMRD	Returns in **Y** what terminated the last **READ**.
TYPE-AHEAD	Allow **TYPE-AHEAD** for the device **$I**.
UCI	Returns **Y** with the current account (UCI, Volume Set).

Node	Description
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**.

21.3.4 $$ACTJ^%ZOSV: Number of Active Jobs

Reference Type:	Supported
Category:	Operating System Interface
ICR #:	10097
Description:	The $$ACTJ^%ZOSV extrinsic function returns the number of active jobs in the scope of this process. It is the same as **^%ZOSF("ACTJ")**.
Format:	`$$ACTJ^%ZOSV`
Input Parameters:	none.
Output:	returns: Returns the number of active jobs.

21.3.5 $$AVJ^%ZOSV: Number of Available Jobs

Reference Type:	Supported
Category:	Operating System Interface
ICR #:	10097
Description:	The $$AVJ^%ZOSV extrinsic function returns a best effort on the number of available jobs (i.e., number of new jobs that could be started). It is the same as **^%ZOSF("AVJ")**.
Format:	`$$AVJ^%ZOSV`
Input Parameters:	none.
Output:	returns: Returns the number of available jobs.

21.3.6 DOLRO^%ZOSV: Display Local Variables

Reference Type:	Controlled Subscription
Category:	Operating System Interface
ICR #:	3883
Description:	The DOLRO^%ZOSV API saves 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 this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables:	**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.

21.3.6.1 Example

Figure 186: DOLRO^%ZOSV API—Example

```
>S X="^%ZTSK(ZTSKm.3," D DOLRO^%ZOSV
```

21.3.7 GETENV^%ZOSV: Current System Information

Reference Type:	Supported
Category:	Operating System Interface
ICR #:	10097
Description:	The GETENV^%ZOSV API returns environment information about the current system.
Format:	`GETENV^%ZOSV`
Input Parameters:	none.

Output Variables:	**Y:**	Returns a string in the following format: `UCI^VOL/DIR^NODE^BOX LOOKUP`

21.3.8 $$LGR^%ZOSV: Last Global Reference

Reference Type:	Supported	
Category:	Operating System Interface	
ICR #:	10097	
Description:	The $$LGR^%ZOSV extrinsic function returns the last global reference.	
Format:	`$$LGR^%ZOSV`	
Input Parameters:	none.	
Output:	returns:	Returns the string set to the last full global reference.

21.3.8.1 Example

Figure 187: $$LGR^%ZOSV API—Example

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

21.3.9 LOGRSRC^%ZOSV(): Record Resource Usage (RUM)

Reference Type: Supported

Category: Operating System Interface

ICR #: 10097

Description: The LOGRSRC^%ZOSV 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 VA Intranet Website.

21.3.10 $$OS^%ZOSV: Get Operating System Information

Reference Type: Supported

Category: Operating System Interface

ICR #: 10097

Description: The $$OS^%ZOSV 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).

21.3.10.1 Example

Figure 188: $$OS^%ZOSV API—Example

```
I ^%ZOSF("OS")["OpenM" S Y=$$OS^%ZOSV
```

21.3.11 SETENV^%ZOSV: Set VMS Process Name (Caché/OpenVMS Systems)

Reference Type: Supported

Category: Operating System Interface

ICR #: 10097

Description: The SETENV^%ZOSV 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 this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables:	**X:**	(required) This is a **1-15** character name to be given to the process at the VMS level.
Output:	none.	

21.3.12 SETNM^%ZOSV(): Set VMS Process Name (Caché/OpenVMS Systems)

Reference Type:	Supported	
Category:	Operating System Interface	
ICR #:	10097	
Description:	The SETNM^%ZOSV 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 Parameters:	**name**:	(required) This is a **1-15** character name to be given to the process at the VMS level.
Output:	none.	

21.3.13 T0^%ZOSV: Start RT Measure (Obsolete)

NOTE: The T0^%ZOSV API is obsolete as of the release of Kernel Toolkit patch XT*7.3*102 and Kernel Patch XU*8.0*425.

Reference Type: Supported

Category: Operating System Interface

ICR #: 10097

Description: The T0^%ZOSV 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 **IO**s between the **T0** start point and the **T1** stop point.

REF: For more information on RT measure, see the Resource Usage Monitor (RUM) documentation, located on the VDL at: http://www.va.gov/vdl/application.asp?appid=130

Format: `T0^%ZOSV`

Input Parameters: 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 **KILL**ed. Such a **KILL** would be appropriate if there is an exit path between the **T0** and **T1** checkpoints 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.

21.3.14 T1^%ZOSV: Stop RT Measure (Obsolete)

NOTE: The T1^%ZOSV API is obsolete as of the release of Kernel Toolkit patch XT*7.3*102 and Kernel Patch XU*8.0*425.

Reference Type:	Supported
Category:	Operating System Interface
ICR #:	10097
Description:	The T1^%ZOSV API stops RT Measure. This API logs the elapsed time into the **^%ZRTL** global (obsolete). 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, see the Resource Usage Monitor (RUM) documentation, located on the VDL at: http://www.va.gov/vdl/application.asp?appid=130

Format: `T1^%ZOSV`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

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 (obsolete), 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 (obsolete)

21.3.15 $$VERSION^%ZOSV(): Get OS Version Number or Name

Reference Type:	Supported	
Category:	Operating System Interface	
ICR #:	10097	
Description:	The $$VERSION^%ZOSV extrinsic function returns the operating system version number or name.	
Format:	`$$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 does *not* 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.

21.3.15.1 Examples

21.3.15.1.1 Example 1

Figure 189: $$VERSION^%ZOSV API—Example 1

```
>W $$VERSION^%ZOSV(1)

Cache for OpenVMS/ALPHA V7.x (Alpha)
```

21.3.15.1.2 Example 2

Figure 190: $$VERSION^%ZOSV API—Example 2

```
>W $$VERSION^%ZOSV

4.1.16
```

22 Security Keys: Developer Tools

22.1 Overview

As well as locking options, developers 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 security key; it restricts functionality within the Dequeue Task, Requeue Tasks, and Delete Tasks options.

22.2 Key Lookup

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

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

This is (and continues to be) a supported reference. The **^XUSEC** global is built by a cross-reference on the SECURITY KEY (#19.1) file.

22.3 Person Lookup

If a key is flagged for Person Lookup, a cross-reference on the NEW PERSON (#200) file is built and maintained to facilitate APIs. 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(0)="AEQ",D="AK.PROVIDER" D IX^DIC
```

22.4 Application Programming Interface (API)

Several APIs are available for developers to work with security keys. These APIs are described below.

22.4.1 DEL^XPDKEY(): Delete Security Key

Reference Type:	Supported
Category:	Security Keys
ICR #:	1367
Description:	The DEL^XPDKEY API deletes a security key from the SECURITY KEY (#19.1) file. All necessary indexing is performed to maintain the **^XUSEC** global. The security key is removed from all holders in the NEW PERSON (#200) file.
Format:	`DEL^XPDKEY(key_ien)`

Input Parameters:	**key_ien**:	(required) The internal entry number (IEN) of the security key to delete from the SECURITY KEY (#19.1) file.
Output:	none.	

22.4.1.1 Example

Figure 191: DEL^XPDKEY API—Example

```
>D DEL^XPDKEY(key_ien)
```

22.4.2 $$LKUP^XPDKEY(): Look Up Security Key Value

Reference Type: Supported

Category: Security Keys

ICR #: 1367

Description: The $$LKUP^XPDKEY 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.

22.4.2.1 Example

Figure 192: $$LKUP^XPDKEY API—Example

```
>S value=$$LKUP^XPDKEY(key_value)
```

22.4.3 $$RENAME^XPDKEY(): Rename Security Key

Reference Type:	Supported	
Category:	Security Keys	
ICR #:	1367	
Description:	The $$RENAME^XPDKEY extrinsic function renames a security key. All necessary indexing is performed to maintain the **^XUSEC** global.	
Format:	`$$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: • **1**—Success. • **0**—Failure.

22.4.4 OWNSKEY^XUSRB(): Verify Security Keys Assigned to a User

Reference Type: Supported

Category: Security Keys

ICR #: 3277

Description: The **XUS KEY CHECK** RPC uses the OWNSKEY^XUSRB API 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.

NOTE: 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:	**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.

22.4.4.1 Examples

22.4.4.1.1 Example 1

In Figure 193, the return array is named **ZZ** and the single security key to be checked is the XUPROG security key:

Figure 193: OWNSKEY^XUSRB API—Example 1

```
>K ZZ D OWNSKEY^XUSRB(.ZZ,"XUPROG") ZW ZZ
ZZ(0)=1
```

22.4.4.1.2 Example 2

In Figure 194, the return subscripted array is named **ZZ** and the input array of security keys to be checked is named **LST**:

Figure 194: OWNSKEY^XUSRB API—Example 2

```
>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
```

23 Server Options: Developer Tools

23.1 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, see Section 11 in the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide*.

REF: For more information on the developer API for processing server requests, see the *MailMan Developer's Guide*.

23.2 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 fields:

- ENTRY ACTION
- HEADER
- ROUTINE
- EXIT ACTION

Table 30 lists the key variables setup for server options:

Table 30: Key Variable Setup—Server Options

Variable	Description
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.

23.3 Appending Text to a Server Request Bulletin or Mailman Reply

Server options use bulletins and MailMan messages to communicate with the local system administrators 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 system administrators 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 Figure 195:

Figure 195: XQSERVER—Default Bulletin

```
Subj: Server request notice
From: <Postmaster>
-----------------------------------------------------------------

Dec. 21, 1989  3:08 PM

A request for execution of a server option was received.

Sender: <Child,Your@HOME.VA.GOV>
Option name: ZZUPDATECL
Subject: UPDATE CHRISTMAS LIST DATA BASE
Message #: 136771

Menu system Action: No error(s) detected by the menu system.

Please append these two lines of text
to the end of the bulletin XQSERVER.
```

You can use the same method to append text to MailMan messages.

23.4 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 you use a customized bulletin 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:

```
    is the server request bulletin XQSERVER
```

To avoid confusion, you should edit this line using the **Bulletin Edit** [XMEDITBUL] option to reflect the name of the new bulletin.

24 Signon/Security: Developer Tools

24.1 Overview

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 (respectively):

- **DUZ(2)**
- **DUZ("AG")**

24.2 Direct Mode Utilities

Several Signon/Security direct mode utilities are available for developers to use at the M prompt. They are *not* APIs and *cannot* be used in software application routines. These utilities allow developers to simulate ordinary user signon and yet work from Programmer mode to test code and diagnose errors. These direct mode utilities are described below.

24.2.1 ^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.
- **KILL**s and rebuilds **^XUTL("XQ",$J)**.
- **KILL**s **^UTILITY($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 displays 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.

24.2.2 ^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 developers 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 Error Trap and then calls the **^XUS** routine.

24.2.3 H^XUS: Programmer Halt

The following is an obsolete utility:

```
>D H^XUS
```

It simply transfers control to ^XUSCLEAN.

24.2.4 ^XUSCLEAN: Programmer Halt

Developers 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 (#3.081) file and **KILL**s the **$J** nodes in **^XUTL** and **^UTILITY**. It then performs a normal halt.

24.2.5 ^ZU: User Signon

The **ZU** routine sets the Error Trap and then calls **^XUS**:

```
>D ^ZU
```

User signons should be tied to **^ZU**.

24.3 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 protocol option that software applications can attach to and perform software application-specific tasks on user signon.

24.3.1 XU USER SIGN-ON: Package-Specific Signon Actions

Kernel 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 protocol option. Your action-type option should call your software application-specific user signon routine.

To attach your option to the XU USER SIGN-ON protocol 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 protocol option, which in turn executes any options that software applications have attached to XU USER SIGN-ON. No database Integration Control Registrations are required to attach to the XU USER SIGN-ON protocol 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 protocol option.

The **DUZ** variable is 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 (#200) file.

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.

24.3.1.1 Example

The option in Figure 196, when attached to the XU USER SIGN-ON protocol, outputs one line during signon:

Figure 196: XU USER SIGN-ON—Sample ZZTALK Protocol

```
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
```

24.4 XU USER START-UP Option

VistA software developers asked for the means to execute an action at VistA user signon, but *not* through the alert system. Added with Kernel Patch XU*8.0*593, the XU USER START-UP protocol option is used exclusively during a VistA user signon event. Items attached to this option are "**TYPE: action**" options in the OPTION (#19) file, which can be used for software-specific actions that prompt users for input upon VistA signon before their Primary Menu Option is displayed. Unlike the XU USER SIGN-ON protocol option, it can provide interactive prompting to users. It is *not* used for GUI signon. It is called from the **XQ12** routine.

24.4.1 XU USER START-UP: Application-specific Signon Actions

Kernel 8.0 introduced a method to support application-specific VistA (*non*-GUI) signon actions. Kernel Patch XU*8.0*593 exports the XU USER START-UP extended-action option. VistA applications that want Kernel to execute an application-specific user signon routine can do this by attaching their own option, of **TYPE: action**, to Kernel's XU USER START-UP protocol option. The action-type option should call the application-specific user signon routine.

To attach your option to the XU USER START-UP protocol option, perform the following procedure:

1. Make your option an item of the XU USER START-UP protocol.
2. Export your option with a KIDS action of **SEND**.
3. Export the XU USER START-UP option with a KIDS action of **USE AS LINK FOR MENU ITEMS**.

During signon, Kernel executes the XU USER START-UP protocol option before the user's Primary Menu Option is displayed, which in turn executes any options that applications have attached to XU USER START-UP. No database Integration Control Registrations are required to attach to the XU USER START-UP protocol option.

Since this option is only used for VistA signon sessions and *not* GUI signon, tasks requiring interaction are permitted. If you want a task to prevent a user from signing on, then the task should set the variable **XUSQUIT=1**.

The **DUZ** variable is 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 (#200) file.

Take care to make code efficient, since it is executed at every VistA signon. The following are examples of tasks you might want to accomplish during a VistA signon:

- Prompt the user to update their phone number in the NEW PERSON (#200) file.
- Block a user's access unless they electronically sign a security agreement.

24.4.1.1 Example:

The option in Figure 197, when attached to the XU USER SIGN-ON protocol, outputs one line during signon:

Figure 197: XU USER START-UP Option—Sample Signon Action-type Option

```
NAME: ZZXU593 SAMPLE OPTION
TYPE: action E ACTION PRESENT: YES
DESCRIPTION: PROMPT USER TO EDIT SIGNATURE BLOCK
ENTRY ACTION: D SAMPLE^XQ12
UPPERCASE MENU TEXT: SAMPLE OPTION
```

24.5 XU USER TERMINATE Option

Kernel 8.0 introduced a method to support software application-specific user termination actions. Kernel 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.

24.5.1 Discontinuation of USER TERMINATE ROUTINE

Kernel 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 the following fields of the software application's PACKAGE (#9.4) file entry:

- USER TERMINATE TAG (#200.1) field
- USER TERMINATE ROUTINE (#200.2) field

When Kernel 7.1 terminated a user, it executed the TAG^ROUTINE API stored in these fields, if any.

Kernel 8.0 continues to execute the API, if any, stored in a software application's PACKAGE (#9.4) file entry. However, Kernel 8.0 is the last version to support that method of software application-specific user termination routines.

24.5.2 Creating a Package-Specific User Termination Action

Beginning with Kernel 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 protocol 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 (#200) file of the user being terminated.

When terminating a user, Kernel executes the XU USER TERMINATE protocol option, which in turn executes any options attached to XU USER TERMINATE. No database Integration Control Registrations are required to attach to the XU USER TERMINATE protocol 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**.

24.6 Application Programming Interface (API)

Several APIs are available for developers to work with signon/security. These APIs are described below.

24.6.1 $$GET^XUPARAM(): Get Parameters

Reference Type:	Supported	
Category:	Signon/Security	
ICR #:	2542	
Description:	The $$GET^XUPARAM extrinsic function gets simple parameters from the KERNEL PARAMETERS (#8989.2) file that the site can edit.	
Format:	`$$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 (#8989.2) file and return the REPLACEMENT value or DEFAULT.
	style:	(optional) This input parameter controls the return value if the REPLACEMENT value or DEFAULT is empty.

Output:	returns:	Returns the REPLACEMENT value or DEFAULT.

24.6.2 $$KSP^XUPARAM(): Return Kernel Site Parameter

Reference Type: Supported

Category: Signon/Security

ICR #: 2541

Description: The $$KSP^XUPARAM extrinsic function retrieves a Kernel site parameter. The following parameters are currently supported:

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

Format: `$$KSP^XUPARAM(param)`

Input Parameters: **param**: (required) Site parameter to retrieve. Currently, the following values for **param** are supported:

- **INST**—Internal Entry Number (IEN) of the site's institution, in the site's INSTITUTION (#4) file.
- **SPOOL DOC**—MAX SPOOL DOCUMENTS PER USER (internal value) from the site's KERNEL SYSTEM PARAMETERS (#8989.3) file.
- **SPOOL LIFE**—MAX SPOOL DOCUMENT LIFE-SPAN (internal value) from the site's KERNEL SYSTEM PARAMETERS (#8989.3) file.
- **SPOOL LINE**—MAX SPOOL LINES PER USER (internal value) from site's KERNEL SYSTEM PARAMETERS (#8989.3) file.
- **WHERE**—Site's domain name (FREE TEXT value), from the site's DOMAIN (#4.2) file.

Output:	returns:	Returns the requested site parameter value.

24.6.2.1 Examples

24.6.2.1.1 Example 1

Figure 198: $$KSP^XUPARAM API—Example 1

```
>S A6ASITE=$$KSP^XUPARAM("WHERE")
```

24.6.2.1.2 Example 2

Figure 199: $$KSP^XUPARAM API—Example 2

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

24.6.3 $$LKUP^XUPARAM(): Look Up Parameters

Reference Type:	Supported	
Category:	Signon/Security	
ICR #:	2542	
Description:	The $$LKUP^XUPARAM extrinsic function looks up simple parameters from the KERNEL PARAMETERS (#8989.2) file that the site can edit.	
Format:	`$$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 (#8989.2) file and return the REPLACEMENT value or DEFAULT.
	style:	(optional) This input parameter controls the return value if the REPLACEMENT value or DEFAULT is empty.
Output:	returns:	Returns the REPLACEMENT value or DEFAULT.

24.6.4 SET^XUPARAM(): Set Parameters

Reference Type:	Supported	
Category:	Signon/Security	
ICR #:	2542	
Description:	The SET^XUPARAM API sets simple parameters in the KERNEL PARAMETERS (#8989.2) file.	
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 (#8989.2) file.
	style:	(optional) This input parameter controls the return value if the REPLACEMENT (#4) field value or DEFAULT (#3) field is empty.
Output:	none.	

24.6.5 $$PROD^XUPROD(): Production Vs. Test Account

Reference Type: Supported

Category: Signon/Security

ICR #: 4440

Description: The $$PROD^XUPROD extrinsic function 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** [XU SID ASK] option 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** [XU SID STARTUP] option 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.

NOTE: This API was released with Kernel Patch XU*8.0*284.

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.

24.6.6 H^XUS: Programmer Halt

Reference Type: Supported

Category: Signon/Security

ICR #: 10044

Description: The H^XUS API is the Programmer Halt.

Format: `H^XUS`

Input Parameters: none.

Output: none.

24.6.7 SET^XUS1A(): Output Message During Signon

Reference Type: Supported

Category: Signon/Security

ICR #: 3057

Description: The SET^XUS1A API performs any output during a software application-specific action 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 also contains 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.	

24.6.7.1 Details

As of Kernel 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, see the "XU USER SIGN-ON: Package-Specific Signon Actions" section.

24.6.8 AVHLPTXT^XUS2: Get Help Text

Reference Type:	Controlled Subscription	
Category:	Signon/Security	
ICR #:	4057	
Description:	The AVHLPTXT^XUS2 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.

24.6.9 $$CREATE^XUSAP: Create Application Proxy User

Reference Type:	Controlled Subscription
Category:	Signon/Security
ICR #:	4677
Description:	The $$CREATE^XUSAP 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.

CAUTION: If the user running this extrinsic function does *not* hold the XUMGR security key, it returns an error upon the filing of the Application Proxy as the User Class.

NOTE: This API was released with Kernel Patch XU*8.0*361.

Overview

The Application Proxy User is a special category of user account that is created in the NEW PERSON (#200) file and can run internal tasks or execute authorized Remote Procedure Calls (RPCs). The Application Proxy represents an application and *not* an end-user. The Application Proxy user account *must* adhere to the following criteria:

- The name added to the NEW PERSON (#200) file *must* be unique and *must* be namespaced in accordance with M Programming Standards and Conventions (SAC) Section 2.6, "Name Requirements."
- It *must* have a user class of "Application Proxy," as defined in the USER CLASS (#201) file and pointed to by the USER CLASS (#9.5) field in the NEW PERSON (#200) file.
- 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 Secondary menu option *must* be owned by the application that the Application Proxy represents, or the application *must* have an Integration Control Registration (ICR) with the option owner. The Secondary menu option contains a list of RPCs that the Application Proxy is authorized to call, as described in the "RPC Security" section in the *RPC Broker User Guide*.
- The RPCs that the menu options reference *must* have the APP PROXY ALLOWED (#.11) field in the REMOTE PROCEDURE (#8994) file set to **YES**. The RPCs *must* be owned by the application that the Application Proxy represents, or the application *must* have an ICR with the RPC owner.
- The use of an Application Proxy *must* be restricted to accessing *non*-protected data. Federal laws specify when an actual end-user *must* be represented when accessing Personally Identifiable Information (PII) and Protected Health Information (PHI). Information regarding user authentication, identity, auditing, and authorization can be found in:
    - *VA Information Security Handbook 6500 Appendix F*
    - National Institute of Standards and Technology (NIST) e-Authentication Guidelines (800-63-2)
    - Health Insurance Portability and Accountability Act of 1996 (HIPAA) federal law 45 CFR § 160 & § 164

Application Proxy Privacy and Auditing

Many VistA data interactions by human end-users *must* be represented with accurate and unambiguous user identity information, so that VistA audit mechanisms function as intended. Application Proxy user accounts do *not* identify the user and should be avoided, especially where the interaction is with PHI/PII data (regulated by federal law). The use of Application Proxy user accounts should be limited to background processes and machine-to-machine interactions.

Application Proxy Permission

Permission to use the $$CREATE^XUSAP API should be done early in the development process; as use of Application Proxy user accounts are reviewed by VA management due to security concerns.

Format: `$$CREATE^XUSAP(proxyusername[,filemanaccesscode][,options])`

Input Parameters:

proxyusername: (required) This is the name of the Application Proxy User (e.g., VPR,APPLICATION PROXY). 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, see the *VA FileMan Advanced User Manual*.

options: (optional) This is the name of a single option name (e.g., VPR APPLICATION PROXY) 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 the NEW PERSON (#200) file**—Successful; writes new Application Proxy User to the NEW PERSON (#200) file.
- **"0^Name In Use"**—Unsuccessful; Application Proxy User of that name already exists in the NEW PERSON (#200) file.

- **-1**—Unsuccessful due to either of the following:
    - Could *not* create Application Proxy User.
    - Error in call to UPDATE^DIE.

  NOTE: For more information on the UPDATE^DIE-related error, users should check **^TMP("DIERR",$J)**.

24.6.9.1 Examples

24.6.9.1.1 Application Proxy Example (Good)

Figure 200 shows a *successful* creation of an Application Proxy User:

Figure 200: $$CREATE^XUSAP API—Example

```
>IF $$CREATE^XUSAP("VPR,APPLICATION PROXY","","VPR APPLICATION PROXY")>0 W
!,"Proxy Created"

Proxy Created
```

Figure 201 is an example of an Application Proxy user account that is provisioned correctly:

Figure 201: Application Proxy Example (Good)

```
NAME: VPR,APPLICATION PROXY                 DATE ENTERED: SEP 01, 2011
  CREATOR: XUUSER,ONE
SECONDARY MENU OPTIONS: VPR APPLICATION PROXY
  TIMESTAMP: 62335,62903
User Class: APPLICATION PROXY               ISPRIMARY: Yes
```

The **Proxy User List** [XUSAP PROXY LIST] option lists the current Application Proxy user accounts, as shown in Figure 202:

Figure 202: Application Proxy Example (Good)—Displayed Using Proxy User List Option

```
PROXY USER LIST                              JAN 28,2016  09:44    PAGE 1
NAME                                 User Class          IsPrimary  Active
--------------------------------------------------------------------------


XOBVTESTER,APPLICATION PROXY         APPLICATION PROXY   Yes
ANRVAPPLICATION,PROXY USER           APPLICATION PROXY   Yes
VPFS,APPLICATION PROXY               APPLICATION PROXY   Yes
RADIOLOGY,OUTSIDE SERVICE            APPLICATION PROXY   Yes
LRLAB,HL                             APPLICATION PROXY   Yes
LRLAB,POC                            APPLICATION PROXY   Yes
TASKMAN,PROXY USER                   APPLICATION PROXY   Yes
CLINICAL,DEVICE PROXY SERVICE        APPLICATION PROXY   Yes
NHIN,APPLICATION PROXY               APPLICATION PROXY   Yes
EDPTRACKING,PROXY                    APPLICATION PROXY   Yes
KAAJEE,PROXY                         APPLICATION PROXY   Yes
VPR,APPLICATION PROXY                APPLICATION PROXY   Yes
AUTHORIZER,IB REG                    APPLICATION PROXY   Yes
HOWDY,BOT                            APPLICATION PROXY   Yes
LRLAB,TASKMAN                        APPLICATION PROXY   Yes
VIABAPPLICATIONPROXY,VIAB            APPLICATION PROXY   Yes
```

CAUTION: Some of the listed Application Proxy user accounts do *not* follow the rules for namespacing. There are other serious infractions in current applications using Application Proxy user accounts, which puts the VA in the position of violating federal privacy laws by accessing PHI/PII information. *VA Handbook 6500 Appendix F* lists VA System Security Controls that are applicable to Application Proxy user accounts as well as human end-users. An Application Proxy should *never* be used to circumvent VA System Security Controls.

24.6.9.1.2 Application Proxy Example (Bad)

Figure 203 is an example of an Application Proxy user account that is *not* provisioned correctly:

Figure 203: Application Proxy Example (Bad) (1 of 2)

```
NAME: TASKMAN,PROXY USER                FILE MANAGER ACCESS CODE: #
  DATE ENTERED: JUN 9,2009              CREATOR: LABTECH,FORTYEIGHT
  NAME COMPONENTS: 200
  SIGNATURE BLOCK PRINTED NAME: PROXY USER TASKMAN
  TIMESTAMP: 62362,53550
User Class: APPLICATION PROXY           ISPRIMARY: Yes
```

If provisioned correctly, the name "TASKMAN,PROXY USER" would be identified by the Kernel (XU) namespace, such as "XUTASKMAN,PROXY USER". This particular Application Proxy does *not* require access to any menu options or RPCs, so it does *not* contain a SECONDARY MENU OPTION.

Figure 204 is another example of an Application Proxy user account that is *not* provisioned correctly:

Figure 204: Application Proxy Example (Bad) (2 of 2)

```
NAME: CLINICAL,DEVICE PROXY SERVICE     DATE ENTERED: JUN 30,2010
  CREATOR: XUUSER,ONE
SECONDARY MENU OPTIONS: MD GUI MANAGER
SECONDARY MENU OPTIONS: MD GUI USER
  TIMESTAMP: 61907,71682
User Class: APPLICATION PROXY           ISPRIMARY: Yes
```

In this example, the SECONDARY MENU OPTIONs are in the Clinical Procedures (MD) namespace, so that if provisioned correctly, "CLINICAL,DEVICE PROXY SERVICE" would be more appropriately named "MDCLINICAL,DEVICE PROXY SERVICE".

24.6.10 KILL^XUSCLEAN: Clear all but Kernel Variables

Reference Type:	Supported
Category:	Signon/Security
ICR #:	10052

Description: The KILL^XUSCLEAN API clears the partition of all but key variables essential to Kernel. Application developers 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 **KILL**ing 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 (#1840) field in the OPTION (#19) file allows you to enter a comma-delimited list of variables to protect from being **KILL**ed 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 (#1840) field for the top-level menu in the software application. As long as a user does *not* exit the top-level menu of the software application's menu tree, the software-wide variable is protected from all calls to KILL^XUSCLEAN. "Up-arrow Jumps" (using a caret; ^) 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.

24.6.11 $$ADD^XUSERNEW(): Add New Users

Reference Type:	Supported	
Category:	Signon/Security	
ICR #:	10053	

Description: The $$ADD^XUSERNEW extrinsic function adds new entries to the NEW PERSON (#200) file. 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 (#200) file 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 (#8989.3) file.

NOTE: This API was modified with Kernel Patch XU*8.0*134.

To prompt for additional fields during this call, you pass a **DR** string containing the fields for which you wish to prompt as a parameter to this function. If the person adding the entry enters a caret (^) to exit out before filling in all the identifiers and requested fields, the entry is removed from the NEW PERSON (#200) file, and **-1** is 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, see 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, see the VA FileMan documentation.

24.6.11.1 Examples

24.6.11.1.1 Example 1

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

Figure 205: $$ADD^XUSERNEW API—Example 1: Adding a New User

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

Enter NEW PERSON's name (Family,Given Middle Suffix): XUUSER,TWO E
 Are you adding 'XUUSER,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: XUUSER// <Enter>
GIVEN (FIRST) NAME: TWO// <Enter>
MIDDLE NAME: E// <Enter>
SUFFIX: <Enter>
Now for the Identifiers.
INITIAL: TEX
SSN: 000222222
SEX: M <Enter>  MALE
>W X
1000118^XUUSER,TWO E^1
>
```

24.6.11.1.2 Example 2

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

Figure 206: $$ADD^XUSERNEW API—Example 2

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

24.6.11.1.3 Example 3

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

Figure 207: $$ADD^XUSERNEW API—Example 3

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

24.6.12 $$CHECKAV^XUSRB(): Check Access/Verify Codes

Reference Type:	Controlled Subscription	
Category:	Signon/Security	
ICR #:	2882	
Description:	The $$CHECKAV^XUSRB extrinsic function checks an Access/Verify code pair (delimited by a semi-colon) and returns whether or not it is a valid pair.	
Format:	`$$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.

24.6.12.1 Example

Figure 208: $$CHECKAV^XUSRB API—Example

```
>S X=$CHECKAV^XUSRB(<string>)
```

String = **Access code;Verify code**

24.6.13 CVC^XUSRB: VistALink—Change User's Verify Code

Reference Type:	Controlled Subscription	
Category:	Signon/Security	
ICR #:	4054	
Description:	The CVC^XUSRB API changes a VistALink user's Verify code.	
Format:	`CVC^XUSRB`	
Input Parameters:	none.	
Output Variables:	**DUZ**:	If **DUZ** is defined, you can consider the "change verify code" operation to have been successful.

24.6.14 $$INHIBIT^XUSRB: Check if Logons Inhibited

Reference Type:	Supported
Category:	Signon/Security
ICR #:	3277
Description:	The $$INHIBIT^XUSRB extrinsic function checks if logons have been inhibited.
Format:	$$INHIBIT^XUSRB
Input Parameters:	none.
Output:	none.

24.6.15 INTRO^XUSRB: VistALink—Get Introductory Text

Reference Type:	Controlled Subscription
Category:	Signon/Security
ICR #:	4054
Description:	The INTRO^XUSRB API retrieves 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.`

24.6.16 LOGOUT^XUSRB: VistALink—Log Out User from M

Reference Type:	Controlled Subscription
Category:	Signon/Security
ICR #:	4054
Description:	The LOGOUT^XUSRB API logs out a VistALink user from M.
Format:	`LOGOUT^XUSRB`
Input Parameters:	none.
Output:	none.

24.6.17 SETUP^XUSRB(): VistALink—Set Up User's Partition in M

Reference Type:	Controlled Subscription
Category:	Signon/Security
ICR #:	4054
Description:	The SETUP^XUSRB API sets up a VistALink user's partition in M prior to signon.
Format:	`SETUP^XUSRB(ret)`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

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.
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
```

24.6.18 VALIDAV^XUSRB(): VistALink—Validate User Credentials

Reference Type: Controlled Subscription

Category: Signon/Security

ICR #: 4054

Description: The VALIDAV^XUSRB 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: 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
```

24.6.19 $$DECRYP^XUSRB1(): Decrypt String

Reference Type: Supported

Category: Signon/Security

ICR #: 2241

Description: The $$DECRYP^XUSRB1 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.

24.6.20 $$ENCRYP^XUSRB1(): Encrypt String

Reference Type:	Supported	
Category:	Signon/Security	
ICR #:	2240	
Description:	The $$ENCRYP^XUSRB1 extrinsic function encrypts a string before transport to a Client system, where it is decrypted. This function performs encryption on the input string, returning the encrypted string.	
Format:	`$$ENCRYP^XUSRB1(string)`	
Input Parameters:	**string**:	(required) The input string to be encrypted.
Output:	returns:	Returns the encrypted string.

24.6.21 $$HANDLE^XUSRB4(): Return Unique Session ID String

Reference Type:	Supported	
Category:	Signon/Security	
ICR #:	4770	
Description:	The $$HANDLE^XUSRB4 extrinsic function returns a unique Caché cluster string for a VistA system for use by Health*e*Vet Desktop applications. **NOTE:** This API was released with Kernel Patch XU*8.0*395.	
Format:	`$$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 is 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 is cleaned up via the **XQ82** routine (i.e., **Clean old Job Nodes in XUTL** [XQ XUTL $J NODES] option).
Output:	returns:	Returns the unique Vista system Caché cluster string. The value generated includes the data entered in the **namespace** input parameter and **$J** and **$H**. If this value is already defined, a new value is generated.

24.6.21.1 Example

In Figure 209, you are creating a unique session ID for the RPC Broker namespace (i.e., "**XWB**"):

Figure 209: $$HANDLE^XUSRB4 API—Example

```
>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
```

24.6.22 ^XUVERIFY: Verify Access and Verify Codes

Reference Type:	Supported
Category:	Signon/Security
ICR #:	10051
Description:	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 onto the system.
Format:	`^XUVERIFY`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables: **%:** (required) If **%** equals:

- **A**—Check the Access code.
- **V**—Check the Verify code.
- **AV**—Check both the Access and Verify code.

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

Output Variables:	**%:**	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**—A caret (^) was entered.

24.6.23 $$CHECKAV^XUVERIFY(): Check Access/Verify Codes

Reference Type:	Supported	
Category:	Signon/Security	
ICR #:	10051	
Description:	The $$CHECKAV^XUVERIFY extrinsic function checks an Access/Verify code pair entered by the user (delimited by a semi-colon) and returns whether or not it is a valid pair.	
Format:	`$$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.

24.6.23.1 Example

Figure 210: $$CHECKAV^XUVERIFY API—Example

```
>S X=$CHECKAV^XUVERIFY(<string>)
```

String = **Access code;Verify code**

24.6.24 WITNESS^XUVERIFY(): Return IEN of Users with A/V Codes & Security Keys

Reference Type:	Controlled Subscription	
Category:	Signon/Security	
ICR #:	1513	
Description:	The WITNESS^XUVERIFY API returns the IEN of a user if he/she has: • Access code • Verify code • Security keys	
Format:	`WITNESS^XUVERIFY(prefix,keys)`	
Input Parameters:	**prefix**:	String to put before the Access/Verify code prompt.
	keys:	String of security keys the user *must* have.
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.

24.6.24.1 Example

Figure 211: WITNESS^XUVERIFY API—Example

```
>S Y=$$WITNESS^XUVERIFY("Cosign","XUMGR") W !,Y

Cosign ACCESS CODE: ********
Cosign VERIFY CODE: ********
2
```

24.6.25 GETPEER^%ZOSV: VistALink—Get IP Address for Current Session

Reference Type:	Controlled Subscription	
Category:	Signon/Security	
ICR #:	4056	
Description:	The GETPEER^%ZOSV 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.

25 Spooling: Developer Tools

25.1 Overview

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 is *not* accessible. The application should close the spool device by using the following:

```
>D ^%ZISC
```

Furthermore, queuing to the spooler requires that the application invoke ^%ZTLOAD with the proper variables defined.

The **ZTIO** input variable 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 variables 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, see the "Form Feeds" section in the "Special Device Issues" section.

Figure 212: 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  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 213: Spooling—Allowing Output to Go the Spooler (*without* Pre-defining ZTIO)

```
SAMPLE  ;SAMPLE ROUTINE
        ;
        S %ZIS="QM" D ^%ZIS G EXIT:POP
        I $D(IO("Q")) D  Q
        .S ZTRTN="DQ^SAMPLE",ZTDESC="Sample Test routine"
        .D ^%ZTLOAD D HOME^%ZIS K IO("Q") Q
DQ      U IO W !,"THIS IS YOUR REPORT"
        W !,"LINE 2"
        W !,"LINE 3"
        D ^%ZISC
EXIT    S:$D(ZTQUEUED) ZTREQ="@" K VAR1,VAR2,VAR3 Q
```

25.2 Application Programming Interface (API)

Several APIs are available for developers to work with spooling. These APIs are described below.

25.2.1 DSD^ZISPL: Delete Spool Data File Entry

Reference Type:	Controlled Subscription
Category:	Spooling
ICR #:	1092
Description:	The DSD^ZISPL API deletes SPOOL DATA (#3.519) file entry following transfer of data, to minimize consumption of data.
Format:	DSD^ZISPL
Input Parameters:	none.
Output:	none.

25.2.2 DSDOC^ZISPL: Delete Spool Document File Entry

Reference Type:	Controlled Subscription
Category:	Spooling
ICR #:	1092
Description:	The DSDOC^ZISPL API deletes the SPOOL DOCUMENT (#3.51) file entry following transfer of data, to minimize consumption of disk space.
Format:	`DSDOC^ZISPL`
Input Parameters:	none.
Output:	none.

26 TaskMan: Developer Tools

26.1 Overview

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.

Developers *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, see the "TaskMan System Management: Overview" section in the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide*.

26.2 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 is easier to write. These two parts are the queuer and the task:

- **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 (#19.2) file; TaskMan itself takes care of the queuing.
- **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.

Usually, both pieces of code should be planned together since they interact heavily.

26.2.1 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).

26.2.1.1 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, see the "^%ZTLOAD: Queue a Task" section.

26.2.1.2 Calling EN^XUTMDEVQ to Create Tasks

The EN^XUTMDEVQ API encapsulates the logic to handle both direct printing and queuing in a single call.

26.2.1.3 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 makes 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 (i.e., 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 is 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 do *not* want to create your own menu to be a parent of queueable 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 does *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 (#19) file entry provides a place for a permanent description of the task's purpose and behavior (the DESCRIPTION field).
- If the option is scheduled regularly, data can be passed to your task from the OPTION SCHEDULING file's (#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 queueable options, since options are usually set on their schedules shortly after installation.

26.2.2 Tasks

This section 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 (#19.2) file.

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.

26.2.2.1 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 VA FileMan format, of course).
- **DUZ(:** The entire **DUZ** array [except **DUZ("NEWCODE")**], as defined at the time of your call to the Program 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** all equal "".
- **ZTDESC:** This contains the free-text description of your task that you passed to the Program 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.
- **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 Program 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 calls 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 is open. The submanager even issues the **USE** command for you and after your task

completes, it properly closes the device for you. If you leave it open when you are finished with it, the submanager is 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 (#14.4) file.
- **Priority:** Your task begins 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**

26.2.2.2 Checking for Stop Requests

You should write tasks in such a way that your tasks honor stop requests. Since Kernel 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. Do *not* **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.

26.2.2.3 Purging the Task Record

According to the SAC, tasks have a responsibility to remove their own records from the TASKS (#14.4) file when they complete. This serves two purposes. First, it helps keep the TASKS (#14.4) file small, which makes TaskMan more efficient. Second, because any tasks that cause errors never reaches the final commands to delete the task's record, such tasks remain in the TASKS (#14.4) file after they complete. This greatly assists system management staff in identifying and troubleshooting problem tasks.

You have two methods to delete TASKS (#14.4) file 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.

26.2.2.4 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 is only 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)
```

26.2.2.5 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 runs. The purpose and format of each ^-piece roughly corresponds to the input variables of REQ^%ZTLOAD listed in Table 31:

  Table 31: TaskMan—ZTREQ Piece and Equivalent REQ^ZTLOAD Variable

ZTREQ Piece	Equivalent REQ^%ZTLOAD Variable
1	ZTDTH
2	ZTIO
3	ZTDESC
4^5	ZTRTN

  All of these ^-pieces in **ZTREQ** are optional; only set the pieces that affect parameters you want to change. 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. 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: Requeue a Task API. Remember, however, to include at least one caret (^) in **ZTREQ** to do this, since if **ZTREQ="@"** the task is 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.

26.2.2.6 Calling ^%ZTLOAD within a Task

Tasks can use all of the standard TaskMan API calls. There is no reason a task should *not* 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 need for the API call, but sometimes you 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.

26.2.2.7 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 (i.e., OPEN^%ZISUTL, USE^%ZISUTL, and CLOSE^%ZISUTL).

26.2.2.8 Long Running Tasks—Writing Two-step Tasks

A situation you should always consider is how to deal with jobs that 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 is eventually used to print that data sits 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 keeps 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:

- $$QQ^XUTMDEVQ(): Double Queue—Direct Queuing in a Single Call—This API creates the Gather and the Print tasks. The gather task is scheduled to run, while the print task is *not* scheduled.
- $$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.

26.2.2.9 Long Running Tasks—Using ^%ZIS

As an alternative to splitting the job into two separate tasks an interactive call can be made to ^%ZIS to allow the user to select the output device without opening it. The gather data portion of the job can then proceed without tying up the output device. When the job is ready to print it can open the output device using the variables that were saved when the ^%ZIS device selection call was made.

To allow for selection of the output device without actually opening it make sure the ^%ZIS input variable **%ZIS** contains **N**.

Some of the variables returned by the device selection call to ^%ZIS need to be saved for use when the device open call is made. These include:

- **IO**
- **IO("DOC")**
- **IOM**
- **ION**
- **IOSL**

If **IO("Q")** is **1** queuing has been selected and your code should handle that and take care of the queuing.

The code excerpt in Figure 214 shows the basic structure for allowing the user to select whether a job is queued or *not* and the output device to use.

Figure 214: TaskMan—Sample Code Allowing Users to Select whether a Job is Queued or Not and the Output Device to Use

```
         N POP,%ZIS
         S %ZIS="NQ"
         W !
         D ^%ZIS
         I POP G EXIT
         I ION=("HOST FILE SERVER")!(ION="P-MESSAGE-HFS") S SAVEHFIO=IO
         S
SAVEIOP=ION_";"_$G(IOST)_";"_$G(IO("DOC"))_";"_$G(IOM)_";"_$G(IOSL)
         ;
         I IO("Q") D  Q
         .;Queue the report.
         .;If ZTIO is not explicitly set to null then %ZTLOAD will open
         .;the device.
         . S ZTIO=""
            .
            .
            .
          . D ^%ZTLOAD
            .
            .
            .
         I 'IO("Q") D  Q
         .;Run the report.
            .
            .
            .
```

When it is time to print, the output device can be opened using the variables that were saved, as shown in Figure 215.

Figure 215: TaskMan—Sample Code Printing to a Device Using Saved Variables

```
          N IOP,POP,VDUZ,XMDUZ,XMQUIET,XMSUB,XMY,%ZIS
          ;Check for output to p-message. TaskMan will automatically copy
          ;^TMP(“XM-MESS”,$J) to the tasked job.
          I $D(^TMP(“XM-MESS”,$J)) D
          . S XMQUIET=1
          . S XMDUZ=$G(^TMP(“XM-MESS”,$J,“XMHOST”,“XMINSTR”,“FROM”))
          . I XMDUZ=“” S XMDUZ=^TMP(“XM-MESS”,$J,“XMHOST”,“XMDUZ”)
          . S XMSUB=^TMP(“XM-MESS”,$J,“XMHOST”,“XMSUB”)
          . S VDUZ=“”
          . F  S VDUZ=$O(^TMP(“XM-MESS”,$J,“XMY”,VDUZ)) Q:VDUZ=“”  S
XMY(VDUZ)=“”
          . I $D(XMY(DUZ)),$D(^TMP(“XM-MESS”,$J,“XMHOST”,“XMINSTR”,“SELF
BSKT”)
) S XMY(DUZ,0)=^TMP(“XM-MESS”,$J,“XMHOST”,“XMINSTR”,“SELF BSKT”)
          S IOP=SAVEIOP
          I $D(SAVEHFIO) S %ZIS(“HFSNAME”)=SAVEHFIO
          D ^%ZIS
          I POP G EXIT
          U IO
```

If **p-message** was selected then **^TMP(“XM-MESS”,$J)** is defined and contains all the information required to deliver the message. Setting **XMQUIET=1** stops interactive processing by MailMan. **XMDUZ** is the sender and **XMSUB** is the subject. The **VDUZ** loop is the list of people to which the user has chosen to send the message. Finally, the check for “**SELF BSKT**” is to determine if the user has selected a particular basket to which the message is to be delivered.

26.2.2.10 Using SYNC FLAGs to Control Sequences of Tasks

You can use **SYNC FLAG**s 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 do *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 FLAG**s 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 does *not* finish successfully, then all other tasks using the same **SYNC FLAG pair** waits.

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** variable 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 **KILL**ing the **ZTSTAT** variable or setting it to **0**. Otherwise, no subsequent tasks is 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 (#14.8) file.
3. If the **SYNC FLAG pair** does *not* exist in the TASK SYNC FLAG (#14.8) file, TaskMan creates an entry for the **SYNC FLAG pair** in the TASK SYNC FLAG (#14.8) file and starts the task.

   If, on the other hand, the **SYNC FLAG pair** already exists in the TASK SYNC FLAG (#14.8) file, then any task requiring the same **SYNC FLAG** has to wait until the corresponding entry in the TASK SYNC FLAG (#14.8) file is deleted.
4. If the task was able to start, the variable **ZTSTAT** is set to **1** in the running task.
   - To indicate success (e.g., that the series of tasks should continue), you *must* **KILL ZTSTAT** or set it to **zero**. In this case, when your task completes, the **SYNC FLAG pair** for that task is 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 (#14.8) file. This allows any future tasks in the series to run.
   - If, on the other hand, **ZTSTAT** is left with a positive value, the task is assumed to have had some kind of error. In this case, the value of **ZTSTAT** is saved in the STATUS field of the **SYNC FLAG pair** entry, and the entry in the TASK SYNC FLAG (#14.8) file is *not* deleted. Subsequent jobs in the series are prevented from running.

If the task errors out, the **SYNC FLAG pair** entry is also left in the TASK SYNC FLAG (#14.8) file, preventing subsequent jobs in the series from running. TaskMan puts a message in the STATUS field, saying that the task stopped due to an error.

26.3 Direct Mode Utilities

You can use TaskMan's direct mode utilities from both the Manager and Production UCIs. Developers *cannot* call them from applications, however.

26.3.1 >D ^ZTMB: Start TaskMan

The **^ZTMB** utility can be used to start TaskMan for the first time since system startup. As part of this startup, any tasks scheduled to begin at system startup are fired off.

26.3.2 >D RESTART^ZTMB: Restart TaskMan

The **RESTART^ZTMB** utility restarts TaskMan. **RESTART^ZTMB**, unlike **^ZTMB**, does *not* fire off the startup tasks and should be used whenever the startup tasks have already been initiated. The **Restart Task Manager** [XUTM RESTART] option uses this entry point.

26.3.3 >D ^ZTMCHK: Check TaskMan's Environment

The **^ZTMCHK** utility provides the same functionality as the **Check Taskman's Environment** [XUTM CHECK ENV] option but from Programmer mode.

26.3.4 >D RUN^ZTMKU: Remove Taskman from WAIT State Option

The **RUN^ZTMKU** utility provides the same functionality as the **Remove Taskman from WAIT State** [XUTM RUN] option but from Programmer mode.

26.3.5 >D STOP^ZTMKU: Stop Task Manager Option

The **STOP^ZTMKU** utility provides the same functionality as the **Stop Task Manager** [XUTM STOP] option but from Programmer mode.

26.3.6 >D WAIT^ZTMKU: Place Taskman in a WAIT State Option

The **WAIT^ZTMKU** utility provides the same functionality as the **Place Taskman in a WAIT State** [XUTM WAIT] option but from Programmer mode.

26.3.7 >D ^ZTMON: Monitor TaskMan Option

The **^ZTMON** utility provides the same functionality as the **Monitor Taskman** [XUTM ZTMON] option but from Programmer mode.

26.4 Application Programming Interface (API)

Several APIs are available for developers to work with TaskMan. These APIs are described below.

26.4.1 TOUCH^XUSCLEAN: Notify Kernel of Tasks that Run 7 Days or Longer

Reference Type:	Supported
Category:	TaskMan
ICR #:	10052

Description: The TOUCH^XUSCLEAN API notifies Kernel of any tasks that run **7** days or longer. If a task appears to have been running longer than **7** days, Kernel assumes that it really is *not* running anymore and **KILL**s off its temp global and user stack.

If your task legitimately runs more than **7** days, your task should call the TOUCH^XUSCLEAN API once a day to notify Kernel. This API sets:

^XUTL("XQ",$J,"KEEPALIVE")=$H

- If Kernel sees this node, and **$H** is less than **7** days ago, Kernel leaves your task alone, unless it determines that your task is really dead.
- If **$H** is more than **7** days ago, Kernel assumes your task is dead and **KILL**s the temp global and user stack for that task.

There are no inputs or outputs to this API.

Format: `TOUCH^XUSCLEAN`

Input Parameters: none.

Output: none.

26.4.2 $$DEV^XUTMDEVQ(): Force Queuing—Ask for Device

Reference Type:	Supported
Category:	TaskMan
ICR #:	1519

Description: The $$DEV^XUTMDEVQ extrinsic function encapsulates the logic to handle direct (FORCED) queuing in a single call and ask users for a device.

NOTE: This API was released with Kernel Patch XU*8.0*275.

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 any of the following: • **"LABEL^ROUTINE"** • **"^ROUTINE"** • **"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 is used to **S ZTSAVE()**. It can be a string of variable names separated by "**;**". Each **;**-piece is 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**, and **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 queuing was successful.
Output:	returns:	Returns: • **0**—If run **ZTRTN** without queuing. • **-1**—If unsuccessful device call or failed the ^%ZTLOAD call.

26.4.2.1 Example

The example in Figure 216 is a job that consists of gathering information and then printing it. Assume that the gathering takes a few hours. You do *not* want the device that the user selects to be tied up for that time, so you divide the job into two tasks:

1. The first task gathers the information.
2. The second task prints it.

Use the following APIs:

1. $$DEV^XUTMDEVQ API—To select the device and queue up the print task.
2. $$NODEV^XUTMDEVQ(): Force Queuing—No Device Selection API—To schedule the gather task.
3. REQ^%ZTLOAD: Requeue a Task API—To schedule the print task when the gather task finishes.

NOTE: You can also use the $$REQQ^XUTMDEVQ(): Schedule Second Part of a Task API to schedule the print task.

Figure 216: $$DEV^XUTMDEVQ API—Example: Sample Code

```
ARHBQQ   ;SFVAMC/REDACTED - 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 queuing.
         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
```

26.4.3 EN^XUTMDEVQ(): Run a Task (Directly or Queued)

Reference Type:	Supported
Category:	TaskMan
ICR #:	1519

Description: The EN^XUTMDEVQ 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** variable is defined before calling EN^XUTMDEVQ, it has the same effect as it does if defined before a ^%ZIS call.

If the **ZTPRI** or **ZTKIL** variables are defined before calling EN^XUTMDEVQ, they has 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** is 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** is appended to it (to allow queuing).

Format: `EN^XUTMDEVQ(ztrtn,ztdesc,.ztsave[,.%zis][,retztsk])`

Input Parameters:

ztrtn: (required) The API that TaskMan will **DO** to start the task. You can specify it as any of the following:

- "**LABEL^ROUTINE**"
- "**^ROUTINE**"
- "**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: Standard Device Call 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** exists as an output variable. Otherwise, **ZTSK** does *not* exist as an output variable.
Output Variable:	**ZTSK:**	If a number is entered in the **retztsk** input parameter, the task number assigned to a task is returned.

26.4.3.1 Example

Figure 217: EN^XUTMDEVQ API—Sample Report

```
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
```

26.4.4 $$NODEV^XUTMDEVQ(): Force Queuing—No Device Selection

Reference Type:	Supported	
Category:	TaskMan	
ICR #:	1519	
Description:	The $$NODEV^XUTMDEVQ extrinsic function encapsulates the logic to handle direct (FORCED) queuing in a single call and does *not* ask users for a device. **NOTE:** This API was released with Kernel Patch XU*8.0*275.	
Format:	`$$NODEV^XUTMDEVQ(ztrtn[,ztdesc][,%var][,%voth][,%wr])`	
Input Parameters:	**ztrtn**:	(required) The API that TaskMan will **DO** to start the task (job). You can specify it as any of the following: • "LABEL^ROUTINE" • "^ROUTINE" • "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 is used to **S ZTSAVE()**. It can be a string of variable names separated by "**;**". Each **;**-piece is 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**, and **ZTSAVE**. For example: `%VOTH("ZTDTH")=$H`
	%wr:	(optional) If **%WR>0** then write text to the screen as to whether or *not* the queuing was successful.
Output:	returns:	Returns: • **> 0**—Successful; Task # (number of the job). • **-1**—Unsuccessful; If failed, the ^%ZTLOAD call.

26.4.4.1 Example

The example in Figure 218 is a job that consists of gathering information and then printing it. Assume that the gathering takes a few hours. You do *not* want the device that the user selects to be tied up for that time, so you divide the job into two tasks:

1. The first task gathers the information.
2. The second task prints it.

Use the following APIs:

1. $$DEV^XUTMDEVQ(): Force Queuing—Ask for Device API—To select the device and queue up the print task.
2. $$NODEV^XUTMDEVQ API—To schedule the gather task.
3. 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.

Figure 218: $$NODEV^XUTMDEVQ API—Sample Code

```
ARHBQQ   ;SFVAMC/REDACTED - 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 queuing.
         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
```

26.4.5 $$QQ^XUTMDEVQ(): Double Queue—Direct Queuing in a Single Call

Reference Type: Supported

Category: TaskMan

ICR #: 1519

Description: The $$QQ^XUTMDEVQ 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 takes 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.

NOTE: This API was released with Kernel Patches XU*8.0*275 and updated with Kernel Patch XU*8.0*389.

Format:

```
$$QQ^XUTMDEVQ(%rtn[,%desc][,%var1][,%voth1][,%zis][,iop][,%
wr],%rtn2[,%desc2][,%var2][,%voth2])
```

Input Parameters:

%rtn: (required) First task that TaskMan runs, 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 any of the following:

- "LABEL^ROUTINE"
- "^ROUTINE"
- "ROUTINE"

The [tag]^routine that TaskMan runs.

%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 is used to **SET ZTSAVE()**. It can be a string of variable names separated by "**;**". Each **;**-piece is 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**, and **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 is 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;** is added.
%wr:	(optional) If **%WR>0** then write text to the screen as to whether or *not* the queuing was successful.
%rtn2:	(required) Second task that TaskMan runs, usually a print process (i.e., print data).The API that TaskMan will **DO** to start the task. You can specify it as any of the following: • "LABEL^ROUTINE" • "^ROUTINE" • "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 is used to **S ZTSAVE()**. It can be a string of variable names separated by "**;**". Each **;**-piece is 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**, and **ZTSAVE**. For example: `%VOTH("ZTDTH")=$H` **NOTE:** If **%voth1(“ZTDTH”)** is passed, it is 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.

26.4.5.1 Example

The example in Figure 219 is a job that consists of gathering information and then printing it. Assume that the gathering takes a few hours. You do *not* want the device that the user selects to be tied up for that time, so you divide the job into two tasks. The first task gathers the information, and the second task prints it. Use the $$QQ^XUTMDEVQ API to select the device, schedule the gather task, and queue the print task. Use the $$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.

Figure 219: $$QQ^XUTMDEVQ API—Sample Code

```
ARHBQQ   ;SFVAMC/REDACTED - 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
```

26.4.6 $$REQQ^XUTMDEVQ(): Schedule Second Part of a Task

Reference Type:	Supported	
Category:	TaskMan	
ICR #:	1519	
Description:	The $$REQQ^XUTMDEVQ extrinsic function schedules the second task (i.e., print data) from the $$QQ^XUTMDEVQ(): Double Queue—Direct Queuing in a Single Call API. If it takes 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. **NOTE:** This API was released with Kernel Patch XU*8.0*389.	
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 Queuing—Ask for Device API.
Output:	returns:	Returns: • **1**—Successful. • **0**—Unsuccessful.

26.4.6.1 Example

Figure 220 is a job that consists of gathering information and then printing it. Assume that the gathering takes a few hours. You do *not* want the device that the user selects to be tied up for that time, so you divide the job into two tasks. The first task gathers the information, and the second task prints it. Use the $$QQ^XUTMDEVQ(): Double Queue—Direct Queuing in a Single Call API to select the device, schedule the gather task, and queue the print task. Use the $$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.

Figure 220: $$REQQ^XUTMDEVQ API—Sample code

```
ARHBQQ   ;SFVAMC/REDACTED - 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
```

26.4.7 DISP^XUTMOPT(): Display Option Schedule

Reference Type:	Supported
Category:	TaskMan
ICR #:	1472
Description:	The DISP^XUTMOPT API displays the schedule for an option.
Format:	`DISP^XUTMOPT(option_name)`

Input Parameters:	**option_name**:	(required) The name of the option from the OPTION (#19) file for which the TaskMan schedule is to be displayed.
Output:	returns:	Returns the TaskMan option schedule.

26.4.7.1 Example

Figure 221: DISP^XUTMOPT API—Example

```
>D DISP^XUTMOPT(option_name)
```

26.4.8 EDIT^XUTMOPT(): Edit an Option's Scheduling

Reference Type: Supported

Category: TaskMan

ICR #: 1472

Description: The EDIT^XUTMOPT API allows users to edit an option's scheduling in the OPTION SCHEDULING (#19.2) file.

Format: `EDIT^XUTMOPT(option_name)`

Input Parameters:	**option_name**:	(required) The name of the option from the OPTION (#19) file whose schedule the user is to be allowed to edit.
Output:	returns:	Returns the requested option in order to edit the schedule.

26.4.9 OPTSTAT^XUTMOPT(): Obtain Option Schedule

Reference Type: Supported

Category: TaskMan

ICR #: 1472

Description: The OPTSTAT^XUTMOPT API allows an application to find out when an 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 (#19) file upon which to return data.
	.root:	(required) This parameter is passed by reference. This is an array because the same task can be scheduled more than once.
Output:	**.root**:	Returns an array of data about the option in question.

26.4.9.1 Example

Figure 222: OPTSTAT^XUTMOPT API—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
queuing flag
```

26.4.10 RESCH^XUTMOPT(): Set Up Option Schedule

Reference Type:	Supported
Category:	TaskMan
ICR #:	1472
Description:	The RESCH^XUTMOPT 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 (#19) file 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 is set to **-1** if the option was *not* found.

26.4.11 EN^XUTMTP(): Display HL7 Task Information

Reference Type: Controlled Subscription

Category: TaskMan

ICR #: 3521

Description: The EN^XUTMTP API is displays 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:

Figure 223: EN^XUTMTP—Sample Display Information

```
261181: EN^HLCSLM, HL7 Link Manager. No
device. DEV,MOU.
From 12/31/2001 at 14:17,  By
XUUSER,THIRTY.
Started running 12/31/2001 at 14:17.
Job #: 562039155
```

26.4.12 ^%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
ICR #:	10063
Description:	The ^%ZTLOAD 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 this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variable:	**ZTRTN:**	(required) The API that TaskMan will **DO** to start the task. You can specify it as any of the following: • "**LABEL^ROUTINE**" • "**^ROUTINE**" • "**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 @ causes 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 are 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 the **IOP** variable in the ^%ZIS: Standard Device Call.

REF: For more information, see the “Device Handler: Developer Tools” section.

ZTUCI: (optional) UCI the task should use. The current UCI is used if **ZTUCI** is undefined.

ZTCPU: (optional) Volume Set:CPU. Specifies the name of the volume set and CPU on which the task should run. The volume set can be passed in the first **:**-piece, and the CPU in the second. Neither piece of information is required; either can be passed without the other. If the CPU alone is passed, it *must* still be preceded by a “**:**” (e.g., **:KDAISC6A1**). If the volume set is *not* passed, TaskMan runs the task on the volume set it came from or on a Print Server. If the CPU is *not* passed, TaskMan runs 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 Caché 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.

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")**—Set equal to **NULL** or to a value:
    - o If **NULL**, the current value of that variable is copied for the task.
    - o The variable is created with the value assigned [e.g., **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")**—Set to **NULL** to declare a set of nodes within an array to be input variables to the task [e.g., **ZTSAVE("^UTILITY($J,")**].
- **ZTSAVE("NAMESPACE*")**—Set to **NULL** to save all local variables in a certain namespace [e.g., **ZTSAVE("LR*")**].
- **ZTSAVE("*")**—Used to save all local variables. *Non*-namespaced variables (esp. **%**, **X**, **Y**, etc.) may or may *not* be saved. Saving individual variables is more efficient. **ZTSAVE** nodes are saved just as they are typed, so special variables like **$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 VA FileMan or **$HOROLOG** format. Use of this variable is *recommended* when **ZTDTH** equals @.

ZTSYNC: (optional) Name of a **SYNC FLAG**. Using **SYNC FLAG**s 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 FLAG**s, the task *must* be queued to a device of type resource (through the **ZTIO** variable).

REF: For complete information on how to use **SYNC FLAG**s, see the "Using SYNC FLAGs to Control Sequences of Tasks" section.

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 displays a message to the user indicating that the task has been canceled.

NOTE: ZTSK is *not* a system variable. It is **KILL**ed 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.

26.4.12.1 Interactive Use of ^%ZTLOAD

The VistA Standards and Conventions (SAC) 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 lets 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:

Figure 224: ^%ZTLOAD API—Print Queuer Sample Code

```
SELECT      ;select IO device for report
            S %ZIS="Q" D ^%ZIS
            I POP D CANCEL Q
            I $D(IO("Q")) D QUEUE Q
            D PRINT,^%ZISC Q
            ;
QUEUE      ;queue the report
            S ZTRTN="PRINT^ZZREPORT"
            S ZTDESC="ZZ Application Daily Report 1"
            S ZTSAVE("ZZRANGE")=""
            D ^%ZTLOAD
            I $D(ZTSK)[0 W !!?5,"Report canceled!"
            E  W !!?5,"Report queued!"
            D HOME^%ZIS Q
```

The code to set up the task after the call to ^%ZIS has four steps:

1. Set the ^%ZTLOAD input variables to define the task.
2. Call ^%ZTLOAD to queue the task.
3. Check **$DATA(ZTSK)#2** to find out whether a task was really queued and provides appropriate feedback.
4. Call HOME^%ZIS API to reset its **IO** variables.

NOTE: This queuer did *not* define the **ZTIO** variable. 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 developer 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.

26.4.12.2 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 the **ZTIO** variable 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.

26.4.12.3 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 runs the task, with no **IO** device.

For sequential ***non*-IO** tasks, queuers *must* set the **ZTIO** variable to the name of a resource type device. TaskMan then ensures 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 FLAG**s 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 FLAG**s, see the "Using SYNC FLAGs to Control Sequences of Tasks" section.

26.4.12.4 Example

The example in Figure 225 is a job that consists of gathering information and then printing it. Assume that the gathering takes a few hours. You do *not* want the device that the user selects to be tied up for that time, so you divide the job into two tasks:

1. The first task gathers the information.
2. The second task prints it.

Use the following APIs:

1. ^%ZIS: Standard Device Call API to select the device.
2. ^%ZTLOAD API to queue the print task.
3. ^%ZTLOAD API to schedule the gather task.
4. 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.

Figure 225: ^%ZTLOAD API—Sample Code

```
ARHBQQ   ;SFVAMC/REDACTED - 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
         ;I '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
```

26.4.12.5 Code Execution

Figure 226: ^%ZTLOAD API—Sample 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: XUUSER,ONE// <Enter>  XUUSER,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
```

26.4.12.6 Output

Figure 227: ^%ZTLOAD API—Sample Output

```
Subj: MY PRINT  [#28881111] 01/25/05@11:30  2 lines
From: POSTMASTER (Sender: XUUSER,ONE - COMPUTER SPECIALIST)  In 'IN'
basket.
Page 1  *New*
-------------------------------------------------------------------------

The secret message is: 'HI MOM!'

Enter message action (in IN basket): Ignore//
```

26.4.13 $$ASKSTOP^%ZTLOAD: Stop TaskMan Task

Reference Type:	Supported	
Category:	TaskMan	
ICR #:	10063	
Description:	The $$ASKSTOP^%ZTLOAD extrinsic function asks TaskMan to stop running a specified task. Also, it checks for the **ZTNAME** variable, and if defined, it uses it instead of **DUZ** to value the STOP FLAG (#59.1) field. **ZTNAME** is supported by applications calling this API to indicate the process that asked the task to stop.	
Format:	`$$ASKSTOP^%ZTLOAD(ztsk)`	
Input Parameters:	**ztsk**:	(required) Task number of the TaskMan task to be stopped.
Output:	returns:	Returns:

- **0**—"Busy". If it returns "Busy", it could mean that:
    - Task is locked.
    - Someone else is changing it.
    - TaskMan is starting to run it.

- **1**—"Task missing" or Task "Finished running". If it returns:
    - **"Task missing"**—It could mean that it was an incorrect input task number, but it is most likely that the task ran and was removed after running.
    - **"Finished running"**—It means that the task was finished running before the API request could go through, so the API could *not* stop an already finished task.

- **2**—"Asked to stop" or "Unscheduled". If it returns:
    - **"Asked to Stop"**—Task has started running and the stop flag has been set, so if the application checks ($$S^%ZTLOAD) it should stop.
    - "Unscheduled"—It was successful and the task is *not* scheduled any more.

26.4.14 DESC^%ZTLOAD(): Find Tasks with a Description

Reference Type:	Supported	
Category:	TaskMan	
ICR #:	10063	
Description:	The DESC^%ZTLOAD 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 tasks with the specified description.

26.4.15 DQ^%ZTLOAD: Unschedule a Task

Reference Type:	Supported
Category:	TaskMan
ICR #:	10063
Description:	The DQ^%ZTLOAD 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 (#14.4) file. 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 this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables:	**ZTSK:**	(required) The number of the task to unschedule. This task *must* currently be defined in the TASKS (#14.4) file or the call fails.
Output Variables:	**ZTSK(0):**	Returns: • **1**—Task was unscheduled successfully. • **0**—Task was *not* unscheduled successfully.

26.4.16 ISQED^%ZTLOAD: Return Task Status

Reference Type:	Supported
Category:	TaskMan
ICR #:	10063
Description:	The ISQED^%ZTLOAD API returns whether a task is currently pending. Pending means that the task is any of the following: • Scheduled. • Waiting for an **I/O** device. • Waiting for a volume set link. • 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 this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

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 the ^%ZTLOAD API **ZTCPU** input variable, this variable does *not* 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 (#14.5) file. • **LS**—The link to that volume set is *not* available. • **U**—An unexpected error arose (e.g., disk full, protection, etc.).
ZTSK("D"):	(sometimes returned) The date and time the task was scheduled to start, in **$HOROLOG** format. It is returned only if **ZTSK(0)** equals **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**.

26.4.17 $$JOB^%ZTLOAD(): Return a Job Number for a Task

Reference Type:	Supported	
Category:	TaskMan	
ICR #:	10063	
Description:	The $$JOB^%ZTLOAD 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.

26.4.18 KILL^%ZTLOAD: Delete a Task

Reference Type:	Supported
Category:	TaskMan
ICR #:	10063
Description:	The KILL^%ZTLOAD API deletes a task. When a task is deleted by KILL^%ZTLOAD, the task referenced by **ZTSK** is *not* defined in the volume set's task file. If the task was pending, it does *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 this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

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.

26.4.19 OPTION^%ZTLOAD(): Find Tasks for an Option

Reference Type:	Supported	
Category:	TaskMan	
ICR #:	10063	
Description:	The OPTION^%ZTLOAD 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 tasks for the specified option.

26.4.20 PCLEAR^%ZTLOAD(): Clear Persistent Flag for a Task

Reference Type:	Supported	
Category:	TaskMan	
ICR #:	10063	
Description:	The PCLEAR^%ZTLOAD 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.	

26.4.21 $$PSET^%ZTLOAD(): Set Task as Persistent

Reference Type:	Supported	
Category:	TaskMan	
ICR #:	10063	
Description:	The $$PSET^%ZTLOAD extrinsic function sets a TaskMan task as persistent (sets the persistent node). A task that is marked as persistent is restarted if TaskMan finds that the lock on **^%ZTSCH("TASK",tasknumber)** has been removed. This adds the requirement that the task only use incremental locks, that the entry in **^%ZTSK(task...** be left in place as this restarts the task, and that the task can be restarted from the data that is in the **^%ZTSK(task,...** global.	
Format:	`$$PSET^%ZTLOAD(ztsk)`	
Input Parameters:	**ztsk**:	(required) The TaskMan task number.
Output:	returns:	Returns: • **1**—Flag was set. • **0**—Flag was *not* set.

26.4.22 REQ^%ZTLOAD: Requeue a Task

Reference Type:	Supported
Category:	TaskMan
ICR #:	10063
Description:	The REQ^%ZTLOAD 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, developers 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 this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

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 is *not* 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 sets **IOP** in the ^%ZIS: Standard Device Call API, and can take all of **IOP**'s format specification strings.

If the **ZTIO** variable is set to **@**, the task is rescheduled for no **I/O** device.

If the **ZTIO** variable is set to **NULL** or it is *not* passed, the originally requested **I/O** device is used.

- **ZTIO("H")—**If *not* set, it is set to the value of the **IO("HFSIO")** variable in the ^%ZIS: Standard Device Call API.
- **ZTIO("P")—**If *not* set, it is set to the value of the **IOPAR** variable in the ^%ZIS: Standard Device Call API.

ZTRTN: (optional) The API TaskMan will **DO** to start the task. You can specify it as any of the following:

- "**LABEL^ROUTINE**"
- "**^ROUTINE**"
- "**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.

26.4.22.1 Example

The example in Figure 228 is a job that consists of gathering information and then printing it. Assume that the gathering takes a few hours. You do *not* want the device that the user selects to be tied up for that time, so divide the job into two tasks:

1. The first task gathers the information.
2. The second task prints it.

Use the ^%ZIS: Standard Device Call API to select the device, the ^%ZTLOAD: Queue a Task API to queue the print task and schedule the gather task. 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.

Figure 228: REQ^%ZTLOAD API—Sample Code

```
ARHBQQ   ;SFVAMC/REDACTED - 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
         ;I ‘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
```

26.4.22.2 Code Execution

Figure 229: ^%ZTLOAD API—Sample 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: XUUSER,ONE// <Enter>  XUUSER,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
```

26.4.22.3 Output

Figure 230: ^%ZTLOAD API—Sample Output

```
Subj: MY PRINT  [#28881111] 01/25/05@11:30  2 lines
From: POSTMASTER (Sender: XUUSER,ONE - COMPUTER SPECIALIST)  In 'IN'
basket.
Page 1  *New*
-------------------------------------------------------------------------

The secret message is: 'HI MOM!'

Enter message action (in IN basket): Ignore//
```

26.4.23 RTN^%ZTLOAD(): Find Tasks that Call a Routine

Reference Type:	Supported	
Category:	TaskMan	
ICR #:	10063	
Description:	The RTN^%ZTLOAD 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:	**list**:	Returns a list of TaskMan tasks that call the specified routine.

26.4.24 $$S^%ZTLOAD(): Check for Task Stop Request

Reference Type: Supported

Category: TaskMan

ICR #: 10063

Description: The $$S^%ZTLOAD extrinsic function is used within a task to determine if the task has been asked to stop. Using the $$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 alerts 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: • **1**—Creator of the task that has asked the task to stop. • **0**—For all other cases.

26.4.25 STAT^%ZTLOAD: Task Status

Reference Type:	Supported
Category:	TaskMan
ICR #:	10063

Description: The STAT^%ZTLOAD 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.
- Runs.

ZTSK(1) and **ZTSK(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 does *not* start in the future without outside intervention; this can be because it:

- Has already completed.
- Was never scheduled.
- 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 this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

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"**—Task does *not* exist on this volume set. • **ZTSK(1) = 1 and ZTSK(2) = "Active: Pending"**—Task is: o Scheduled. o Waiting for an **I/O** device. o Waiting for a volume set link. o Waiting for a partition in memory. • **ZTSK(1) = 2 and ZTSK(2) = "Active: Running"**—Task has started running. • **ZTSK(1) = 3 and ZTSK(2) = "Inactive: Finished"**—Task quit normally after running. • **ZTSK(1) = 4 and ZTSK(2) = "Inactive: Available"**—Task was created without being scheduled or was edited without being rescheduled.

- **ZTSK(1) = 5 and ZTSK(2) = “Inactive: Interrupted”**—Task was interrupted before it would have quit normally. Causes can include:
  - Bad data.
  - User intervention.
  - Hard error.
  - Many other possibilities.

26.4.26 $$TM^%ZTLOAD: Check if TaskMan is Running

Reference Type: Supported

Category: TaskMan

ICR #: 10063

Description: The $$TM^%ZTLOAD 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.

26.4.27 ZTSAVE^%ZTLOAD(): Build ZTSAVE Array

Reference Type: Supported

Category: TaskMan

ICR #: 10063

Description: The ZTSAVE^%ZTLOAD 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 first **KILL**s the **ZTSAVE** array.

Output: returns: Stores the string of input variables in the **ZTSAVE** array.

27 Toolkit: Developer Tools

Several tools and Application Programming Interfaces (APIs) are available for developers to work with Kernel Toolkit. This section describes these APIs by type.

27.1 Toolkit—Data Standardization

27.1.1 Overview

The API set in this section has been developed to support Data Standardization's effort to allow the mapping of one term to another term. Mapping of terms is done via the REPLACED BY VHA STANDARD TERM (#99.97) field and provides the high-level goals of the following:

- *Non*-standard terms inheriting standardized characteristics.
- Deprecating a term and replacing it with a new term.

The Data Standardization API set:

1. Maps one term to another term.
2. Obtains the term in which another term is mapped.
3. Extracts field values from the term in which another term is mapped.
4. Shows the mapping relationships that a term has with other terms.

Keywords:

- VHA Unique ID (VUID)
- Data Standardization
- Term
- Replacement Term

NOTE: This Data Standardization API set was released with Kernel Toolkit Patch XT*7.3*111.

27.1.2 Replacement Relationships

Use the replacement relationships in Figure 231 to map the Data Standardization API set in context. These APIs are documented in this section:

Figure 231: Toolkit—Replacement Relationships: Data Standardization

```
     A --> B --> C --> D      A is replaced by B     G is replaced by C
    ^ ^         ^ ^           B is replaced by C     H is replaced by C
    |  \        |  \          C is replaced by D     I is replaced by F
    |   \       |   \         D has no replacement   J is replaced by F
    |    \      |    \        E is replaced by A     K is replaced by H
    |     F     |     H       F is replaced by A     L is replaced by H
    |    ^ ^    |    ^ ^
    |   /   \   |   /   \
    E  I     J  G  K     L

   $$GETRPLC(B) would return C

   $$RPLCMNT(B) would return D

   $$RPLCVALS(J) would return the requested field values from entry D

   $$RPLCTRL(G) in both directions would return D and the output array
would
   be set as follows:

    OutArr("BY",A) = B                    OutArr("FOR",A,E) = ""
    OutArr("BY",B) = C                    OutArr("FOR",A,F) = ""
    OutArr("BY",C) = D                    OutArr("FOR",B,A) = ""
    OutArr("BY",D) = ""                   OutArr("FOR",C,B) = ""
    OutArr("BY",E) = A                    OutArr("FOR",C,G) = ""
    OutArr("BY",F) = A                    OutArr("FOR",C,H) = ""
    OutArr("BY",G) = C                    OutArr("FOR",D,C) = ""
    OutArr("BY",H) = C                    OutArr("FOR",F,I) = ""
    OutArr("BY",I) = F                    OutArr("FOR",F,J) = ""
    OutArr("BY",J) = F                    OutArr("FOR",H,K) = ""
    OutArr("BY",K) = H                    OutArr("FOR",H,L) = ""
    OutArr("BY",L) = H

   $$RPLCTRL(L) in the forward direction would return D and the output
array
   would be set as follows:

    OutArr("BY",C) = D                    OutArr("FOR",C,H) = ""
    OutArr("BY",D) = ""                   OutArr("FOR",D,C) = ""
    OutArr("BY",H) = C                    OutArr("FOR",H,L) = ""
    OutArr("BY",L) = H

   $$RPLCTRL(B) in the backward direction would return D and the output
array
   would be set as follows:

    OutArr("BY",A) = B                    OutArr("FOR",A,E) = ""
    OutArr("BY",E) = A                    OutArr("FOR",A,F) = ""
    OutArr("BY",F) = A                  OutArr("FOR",B,A) = ""
    OutArr("BY",I) = F                  OutArr("FOR",F,I) = ""
    OutArr("BY",J) = F                  OutArr("FOR",F,J) = ""

   $$RPLCLST(G) in both directions would return D and the output array
would
   be set as follows:

    OutArr(1) = G ^ 0                   OutArr("INDEX",A) = 8
    OutArr(2) = C ^ 0                   OutArr("INDEX",B) = 7
    OutArr(3) = D ^ 1                   OutArr("INDEX",C) = 2
    OutArr(4) = H ^ 0                   OutArr("INDEX",D) = 3
    OutArr(5) = K ^ 0                   OutArr("INDEX",E) = 9
    OutArr(6) = L ^ 0                   OutArr("INDEX",F) = 10
    OutArr(7) = B ^ 0                   OutArr("INDEX",G) = 1
    OutArr(8) = A ^ 0                   OutArr("INDEX",H) = 4
    OutArr(9) = E ^ 0                   OutArr("INDEX",I) = 11
    OutArr(10) = F ^ 0                  OutArr("INDEX",J) = 12
    OutArr(11) = I ^ 0                  OutArr("INDEX",K) = 5
    OutArr(12) = J ^ 0                  OutArr("INDEX",L) = 6

   $$RPLCLST(L) in the forward direction would return D and the output
array
   would be set as follows if the status history was also included:

    OutArr(1) = L ^ 0                   OutArr("INDEX",C) = 3
    OutArr(1,3080101.0954) = 0          OutArr("INDEX",D) = 4
    OutArr(2) = H ^ 0                   OutArr("INDEX",H) = 2
    OutArr(2,3080101.1308) = 1          OutArr("INDEX",L) = 1
    OutArr(2,3080105.09) = 0
    OutArr(3) = C ^ 0
    OutArr(3,3080105.0859) = 1
    OutArr(3,3080112.1722) = 0
    OutArr(4) = D ^ 1
    OutArr(4,3080112.1723) = 1

   $$RPLCLST(B) in the backward direction would return D and the output
array
   would be set as follows:

    OutArr(1) = A ^ 0                   OutArr("INDEX",A) = 1
    OutArr(2) = E ^ 0                   OutArr("INDEX",E) = 2
    OutArr(3) = F ^ 0                   OutArr("INDEX",F) = 3
    OutArr(4) = I ^ 0                   OutArr("INDEX",I) = 4
    OutArr(5) = J ^ 0                   OutArr("INDEX",J) = 5
```

27.1.3 Application Programming Interfaces (APIs)

27.1.4 $$GETRPLC^XTIDTRM(): Get Mapped Terms (Term/Concept)

Reference Type:	Supported	
Category:	Toolkit—Data Standardization	
ICR #:	5078	
Description:	The $$GETRPLC^XTIDTRM extrinsic function gets the REPLACED BY VHA STANDARD TERM (#99.97) field for a given entry. **REF:** For an overview of the Data Standardization API set, see Toolkit—Data Standardization APIs. For a chart mapping the Data Standardization API set in context, see Replacement Relationships.	
Format:	`$$GETRPLC^XTIDTRM(file,ien)`	
Input Parameters:	**file**:	(required) File number.
	ien:	(required) Internal Entry Number (IEN).
Output:	returns:	Returns the REPLACED BY VHA STANDARD TERM (#99.97) field for a given entry.

27.1.4.1 Example

The $$GETRPLC^XTIDTRM extrinsic function sets **X** to **IEN_";"_FileNumber** of entry that replaces the input entry:

Figure 232: $$GETRPLC^XTIDTRM API—Example

```
>S X=$$GETRPLC^XTIDTRM(file,ien)
```

NOTE:

- **NULL** is returned on error. This typically occurs when the input entry does *not* exist.
- If the input entry is *not* replaced by another term then a reference to the input term is returned.

27.1.5 $$RPLCLST^XTIDTRM(): Get Replacement Terms, w/Optional Status Date & History (Term/Concept)

Reference Type: Supported

Category: Toolkit—Data Standardization

ICR #: 5078

Description: The $$RPLCLST^XTIDTRM extrinsic function traverses the REPLACED BY VHA STANDARD TERM (#99.97) field forwards and backwards to find all terms that are replacement terms for the input entry and all terms for which the input entry is a replacement. This is recursively done so that each potential branch of replacement terms forwards and backwards is traversed.

Format:

```
$$RPLCLST^XTIDTRM(file,ien,drctn,statdate,stathst,outarr)
```

Input Parameters:

file: (required) File number.

ien: (required) Entry number.

drctn: (optional) Flags denoting which direction to follow the trail of replacement terms. Possible flag values are:

- **F (default)**—Follow the trail forwards.
- **B**—Follow the trail backwards.
- *****—Follow the trail in both directions (same as **FB**/**BF**).

statdate: (optional) VA FileMan date/time in which to return term's status. Defaults to current date/time.

stathst: (optional) Flag denoting if a term's full status history should be included in the output:

- **0 (default)**—No.
- **1**—Yes.

Input/Output

Parameters: **outarr**:

I: (required) Array to put trail of replacement terms into (closed root).

O: The output array contains the list terms to which the input entry is somehow related.

- OutArr(1..*n*) = Term ^ StatusCode (based on input StatDate).
- OutArr(1..*n*,StatusDateTime) = StatusCode on this date/time.
- This node is only returned if StatHst is set to **1** (**Yes**).
- OutArr("INDEX",Term) = 1..*n*.

Where:

- **Term** is in the format IEN;FileNumber.
- **StatusCode**:
  - **1**—Active.
  - **0**—Inactive.
- **StatusDateTime** is in VA FileMan format.

27.1.5.1 Example

The $$RPLCLST^XTIDTRM extrinsic function sets **X=IEN_";"_FileNumber** of the entry that ultimately replaces the input entry:

Figure 233: $$RPLCLST^XTIDTRM API—Example

```
>S X=$$RPLCLST^XTIDTRM(File,IEN,Drctn,StatDate,StatHst,OutArr)
```

NOTE:

- **NULL** is returned on error. This typically occurs when the input entry does *not* exist.
- If the input entry is *not* replaced by another term then a reference to the input term is returned.

27.1.6 $$RPLCMNT^XTIDTRM(): M One Term to Another (Term/Concept)

Reference Type:	Supported
Category:	Toolkit—Data Standardization
ICR #:	5078
Description:	The $$RPLCMNT^XTIDTRM extrinsic function recursively traverses the REPLACED BY VHA STANDARD TERM (#99.97) field until the final replacement term is reached.
Format:	`$$RPLCMNT^XTIDTRM(fle,ien)`

Input Parameters:	**file**:	(required) File number.
	ien:	(required) Internal Entry Number (IEN).
Output:	none.	

27.1.6.1 Example

The $$RPLCMNT^XTIDTRM extrinsic function sets **X** to **IEN_";"_FileNumber** of the entry that ultimately replaces the input entry:

Figure 234: $$RPLCMNT^XTIDTRM API—Example

```
>S X=$$RPLCMNT^XTIDTRM(file,ien)
```

NOTES:

- **NULL** is returned on error. This typically occurs when the input entry does *not* exist.
- If the input entry is *not* replaced by another term then a reference to the input term is returned.

27.1.7 $$RPLCTRL^XTIDTRM(): Get Replacement Trail, w/ Replaced "BY" & Replacement "FOR" Terms

Reference Type: Supported

Category: Toolkit—Data Standardization

ICR #: 5078

Description: The $$RPLCTRL^XTIDTRM extrinsic function traverses the REPLACED BY VHA STANDARD TERM (#99.97) field forwards and backwards to find all terms that are replacement terms for the input entry and all terms for which the input entry is a replacement. This is recursively done so that each potential branch of replacement terms forwards and backwards is traversed.

Format: `$$RPLCTRL^XTIDTRM(file,ien,drctn,outarr)`

Input Parameters:

file: (required) File number.

ien: (required) Internal Entry Number (IEN).

drctn: (optional) Flags denoting which direction to follow the trail of replacement terms. Possible flag values are:

- **F (default)—**Follow the trail forwards.
- **B—**Follow the trail backward.
- ***—**Follow the trail in both directions (same as FB/BF).

Input/Output Parameters:

outarr: **I:** (required) Array to put trail of replacement terms into (closed root).

O: The output array contains the trail of replacement terms.

- OutArr("BY",Term) = Replacement Term means: Entry "Term" is replaced BY entry "Replacement Term."
- OutArr("FOR",Replacement Term, Term) = "" means: Entry "Replacement Term" is a replacement FOR entry "Term."
- Term and Replacement Term is in the format IEN;FileNumber.

27.1.7.1 Example

The $$RPLCTRL^XTIDTRM extrinsic function sets **X** to **IEN_";"_FileNumber** of the entry that ultimately replaces the input entry:

Figure 235 $$RPLCTRL^XTIDTRM API—Example

```
>S X=$$RPLCTRL^XTIDTRM(file,ien,drctn,outarr)
```

NOTES:

- **NULL** is returned on error. This typically occurs when the input entry does *not* exist.
- If the input entry is *not* replaced by another term then a reference to the input term is returned.

27.1.8 $$RPLCVALS^XTIDTRM(): Get Field Values of Final Replacement Term (Term/Concept)

Reference Type:	Supported	
Category:	Toolkit—Data Standardization	
ICR #:	5078	
Description:	The $$RPLCVALS^XTIDTRM extrinsic function retrieves one or more fields of data from an entry's final replacement term. The REPLACED BY VHA STANDARD TERM (#99.97) field is recursively traversed until the final replacement term is reached. The requested fields of the final replacement term are returned. It effectively bundles $$RPLCMNT^XTIDTRM and GETS^DIQ into a single call.	
Format:	`$$RPLCVALS^XTIDTRM(file,ien,fields,flags,outarr)`	
Input Parameters:	**file**:	(required) File number.
	ien:	(required) Internal Entry Number (IEN).
	fields:	(required) Fields for which you wish to get values. **REF:** For detailed description, see the definition of the FIELD parameter in the GETS^DIQ API in the *VA FileMan Developer's Guide*.

	flags:	(required) Flags that control output format. **REF:** For detailed description, see the definition of the FLAGS parameter in the GETS^DIQ API in the *VA FileMan Developer's Guide*.
Input/Output:		
Parameters	**outarr**:	Input/Output: • **I:** (required) Array to put output field values into (closed root). • **O:** The output array is in **FDA** format. **REF:** For example output, see the GETS^DIQ API in the *VA FileMan Developer's Guide*.

27.1.8.1 Example

The $$RPLCVALS^XTIDTRM extrinsic function sets **X** to **IEN_";"_FileNumber** of the entry that ultimately replaces the input entry:

Figure 236: $$RPLCVALS^XTIDTRM API—Example

```
>S X=$$RPLCVALS^XTIDTRM(file,ien,fields,flags,outarr)
```

NOTES:

- **NULL** is returned on error. This typically occurs when the input entry does *not* exist.
- If an error occurs when extracting the requested fields from the final replacement term then a reference to the final replacement term is still returned and **outarr** is **KILL**ed.
- If the input entry is *not* replaced by another term then a reference to the input term is returned and **outarr( )** contains the field values for the input entry.

27.1.9 $$SETRPLC^XTIDTRM(): Set Replacement Terms (Term/Concept)

Reference Type:	Supported	
Category:	Toolkit—Data Standardization	
ICR #:	5078	
Description:	The $$SETRPLC^XTIDTRM extrinsic function sets the REPLACED BY VHA STANDARD TERM (#99.97) field.	
Format:	$$SETRPLC^XTIDTRM(file,ien,rplcmnt)	
Input Parameters:	**file**:	(required) File number.
	ien:	(required) Internal Entry Number (IEN).
	rplcmnt:	(required) Entry number of replacement term.
Output Variables:	**X:**	Results: • **1 (success)**—If pointer to replacement term stored. • **0 (failure)**—If unable to store pointer to replacement term.

27.1.9.1 Example

The $$SETRPLC^XTIDTRM extrinsic function sets **X** to **1** if pointer to replacement term stored (i.e., success) or **0** if Unable to store pointer to replacement term (i.e., failure):

Figure 237: $$SETRPLC^XTIDTRM API—Example

```
>S X=$$SETRPLC^XTIDTRM(File,IEN,Rplcmnt)
```

27.2 Toolkit—Duplicate Record Merge

27.2.1 Overview

A file in which entries need to be merged can be entered in the DUPLICATE RESOLUTION (#15.1) file. This requires adding the file as one that can be selected as the VARIABLE POINTER, and search criteria would usually need to be specified to assist in identifying potential duplicate pairs (although an option can be used by which selected pairs can be added directly to the DUPLICATE RECORD (#15) file as verified duplicates). Verified duplicate pairs may be approved for merging, and a merge process generated for those approved pairs. A DUPLICATE RECORD (#15) file entry also has handle files that are *not* associated as normal pointers identified in the PACKAGE (#9.4) file under the AFFECTS RECORD MERGE subfile with special processing routines.

CAUTION: If a file has related files that are *not* normal pointers, they should be handled only as entries in the duplicate record file and the Kernel Toolkit options used for merges involving the file.

The merge utility of Kernel Toolkit as revised by Kernel Toolkit Patch XT*7.3*23 provides an entry point that is available to developers for the merging of one or more pairs of records (a **FROM** record and a **TO** record) in a specified file. The merge process merges the data of the **FROM** record into that of the **TO** record and deletes the **FROM** record, restoring by a hard set only the **zero** node with the **.01** value on it until the merge process is completed (such that any references to that location via pointers does *not* error out). Any files that contain entries **DINUM**ed with the data pairs are then also merged (and any files that are related to them by **DINUM** as well). Any pointers that can be identified rapidly by cross-references are modified so that references for the **FROM** entry become references to the **TO** entry instead. Following this, any files that contain other pointers are searched entry by entry to test for pointers to a **FROM** entry, and when found are modified to reference the **TO** entry. This search for pointer values is the most time consuming part of the entire process and may take an extended period depending upon the number of files that *must* be searched, the number of entries in those files, and how many levels at which subfiles pointers may be located. Since the search through these files takes the same period of time independent of the number of pairs that are being merged, it is suggested that as many pairs as convenient be combined in one process. At the end of the conversion of these pointers, the **zero** node stubs are removed from the primary file and all related **DINUM**ed files.

The merge process is a single job that is tracked with frequent updates on location and status from start to finish. The job can be stopped at any time if necessary using TaskMan utilities (or in the event of a system crash, etc.) and restarted at the point of interruption at a later time.

27.2.1.1 Manner in which data is Merged

When a primary file or a **DINUM**ed files entries are merged, any top level (single value) fields that are present in the **FROM** entry that are *not* present in the **TO** entry is merged into the **TO** entries data. Any of these fields that contain cross-references are entered using a VA FileMan utility (FILE^DIE) so that the cross-references are fired. Other fields (those *without* cross-references) are directly set into the data global.

If a subfile entry (Multiple) exists in the **FROM** record that is *not* present in the **TO** record (as identified by the **.01** value), that entry is created with a VA FileMan utility (UPDATE^DIE) and the rest of the subfile merged over into the **TO** record and the cross-references within the subfile and any descendent subfiles run.

If a subfile entry (Multiple) exists in the **FROM** record and an identical **.01** value exists in the **TO** record, the subfile in the **FROM** record is searched for any descendent subfiles that are *not* present in the **TO** record subfile. If such a subfile is found it is merged into the subfile in the **TO** record and any cross-references in the merged subfile run.

For fields that are simple pointers to the primary file (or any other file **DINUM**ed to the primary file) the reference to the **FROM** record is changed to a reference to the **TO** record. If the field contains a cross-reference this editing is performed using a VA FileMan Utility call (FILE^DIE), otherwise it is set directly into the global node.

27.3 Developing a File Merge Capability

This section provides developers with a set of instructions to follow in building a merge capability for a file. After a developer identifies a file that has a substantial number of duplicates and that the nature and use of the file warrants a merge utility, he/she then follows the steps outlined in this section in developing that merge capability.

For demonstration purposes, the rest of this section uses a specific example of developing a Patient Merge using the Duplicate Resolution Utilities.

27.3.1 Step 1

Notify the Kernel Toolkit developers of the perceived need for a duplicate checking/merge capability for a particular file. They will do the following:

1. Assists the developer in deciding whether there is indeed a need for a Duplicate Resolution Utility for this particular file.
2. Add the file to the **.01** and **.02** VARIABLE POINTER field definitions in the DUPLICATE RECORD (#15) file.
3. Notifies the application developer when the modified dictionary is to be released to the field.

27.3.2 Step 2

The developer needs to now communicate to the larger development community his/her intention to develop a merge capability for this file. All developers need to determine if the merging and deleting of records in this file affects their package in such a way that they need to have their own unique merge routine that deals with only their package's files. A developer usually has to write their own unique merge routine if any of the following conditions exist:

- Patient pointer field is defined as a NUMERIC or FREE TEXT field rather than a POINTER.
- Developer wants their end users to complete some task prior to the merge occurring.
- There are compound cross-references that include the patient pointer on another field but the cross-reference is *not* triggered by the changing of the patient pointer.
- Merge (Duplicate Resolution Utilities) does *not* do what the package developer desires.

27.3.3 Description of What Occurs during the Merge

The following is a brief description of what occurs during the merge process:

1. The base file (e.g., PATIENT file, #2) is checked to see if it exists.
2. The **PT** nodes (e.g., **^DD(2,0,"PT",**) are checked and any false positives are removed.
3. Creates a list of files and fields within those files that point to the file being merged (e.g., in this example the file being merged is the PATIENT file, #2).
4. If a file is pointing to the file being merged by its **.01** field, and if that **.01** field is **DINUM**, then all files/fields that point to that file are also gathered. The **DINUM** rule also applies to that file and any files pointing to it, to any depth.
5. Each file/field is checked and re-pointed/merged as follows:
    - If the field pointing is *not* a **.01** field, the **FROM** entry is changed to the **TO** entry.
    - If the field pointing is the **.01** field but *not* **DINUM**, the **FROM** entry is changed to the **TO** entry.

- Each pointing **.01 DINUM** field is handled as follows:
  - If the **.01 DINUM** field is at the file level, **^DIT0** is called to merge the **FROM** entry to the **TO** entry and then the **FROM** entry is deleted.

    ^DIT0 merges field by field but does *not* change any value in the **TO** entry. That means that **NULL** fields in the **TO** entry get the value from the same field in the **FROM** entry if it is *not* **NULL**, and valued fields in the **TO** entry remain the same.

    ^DIT0 also merges Multiples. If a Multiple entry in the **FROM** entry *cannot* be found in the **TO** entry, it is added to the **TO** entry. If a Multiple entry in the **FROM** entry can be found in the **TO** entry, then that Multiple entry is merged field by field.
  - If the **.01 DINUM** field is at the subfile level (in a Multiple), it is handled as follows:
    - If there is a **FROM** entry but no **TO** entry, the **FROM** entry is added to the **TO** entry, changing the **.01** field value in the process, and the **FROM** entry is deleted.
    - If there is a **FROM** entry and also a **TO** entry, the **FROM** entry is deleted and the **TO** entry remains unchanged.

If it is determined that a developer *must* have their own unique merge that deals with their files, they *must* make the appropriate entries in the PACKAGE (#9.4) file. If they have to have some sort of action taken by end-users prior to the merging of the records, they *must* update the MERGE PACKAGES (#1101) Multiple field in the DUPLICATE RECORD (#15) file for that pair of records.

27.3.4 Entries Needed in the PACKAGE (#9.4) File

In the PACKAGE (#9.4) file make entries in the following fields:

- AFFECTS RECORD MERGE (#20) field
- NAME (#.01) field—Enter the file affected (e.g., PATIENT [#2] file)
- NAME OF MERGE ROUTINE (#9.402,3) field—Enter the name of the merge routine, which is executed via indirection by Duplicate Resolution Utilities.

  If you leave this field blank but still place an entry in the PACKAGE (#9.4) file, Duplicate Resolution Utilities assumes that you have some sort of interactive merge process that your end-users *must* complete prior to the main merging of the two records. It also assumes that this interactive merge process is on a separate option within the developer's package options. The values of the two records being merged are placed in:
  - **^TMP("XDRMRGFR",$J,XDRMRG("FR"),**
  - **^TMP("XDRMRGTO",$J,XDRMRG("TO"),**

These should be referenced by the developer if they need any certain field values since the values might have been changed prior to the execution of their merge routine.

- RECORD HAS PACKAGE DATA (#9.402,4) field—Enter a string of M executable code that is passed the variable **XDRMRG("FR")** (the **FROM** record IEN) and set **XDRZ** to **0**. The code should set **XDRZ=1** if **XDRMRG("FR")** has data within your package files.

Remember to only make these entries in the PACKAGE (#9.4) file if the normal merge does *not* suffice for your package. If you have an entry in the PACKAGE (#9.4) file, the repointing and merging as described above does *not* take place for those files within your Package entry.

At the completion of your interactive merge process, the developer *must* set the STATUS (#15.01101,.02) field of the MERGE PACKAGES (#1101) Multiple field for their package in the DUPLICATE RECORD (#15) file entry to **Ready**. This *must* be done using VA FileMan, because of the trigger that is on the STATUS field. Once all of the MERGE PACKAGE entries have a STATUS of **Ready**, the main merging of the two records can occur.

27.3.5 Step 3

The developer needs to add an entry in the DUPLICATE RESOLUTION (#15.1) file for the file being built. The following fields need to be updated in the DUPLICATE RESOLUTION (#15.1) file and data should be entered by the developer:

- .01 FILE TO BE CHECKED (required)
- .06 CROSS-REF FOR NEW SEARCH (optional)
- .09 CANDIDATE COLLECTION ROUTINE (required)
- .11 DUPLICATE MANAGER MAIL GROUP (optional)
- .15 POTENTIAL DUPLICATE THRESHOLD% (required)
- .16 VERIFIED DUPLICATE MAIL GROUP (optional)
- .17 VERIFIED DUPLICATE MSG ROUTINE (optional)
- .18 VERIFIED DUPLICATE THRESHOLD% (optional)
- .25 MERGE STYLE (required)
- .26 DELETE FROM ENTRY (optional)
- .27 PRE-MERGE ROUTINE (optional)
- .28 POST-MERGE ROUTINE (optional)
- .29 MERGE MAIL GROUP (optional)
- .31 MERGE MSG ROUTINE (optional)
- .33 MERGE DIRECTION INP TRANSFORM (optional)

- 1100 DUPLICATE TESTS (required)
    - .01 DUPLICATE TEST (required)
    - .02 ORDER OF TEST (required)
    - .03 DUPLICATE TEST ROUTINE (required)
    - .04 FILE FOR INFORMATION (optional)
    - .05 FIELD TO BE CHECKED (required)
    - .06 SUCCESSFUL MATCH WEIGHT (required)
    - .07 UNSUCCESSFUL MATCH WEIGHT (required)

- 1200 DINUM FILES FOR MERGE (optional)
    - .01 DINUM FILES FOR MERGE (optional)

27.3.5.1 Explanation of Fields in Logical Order of Entry

Selected fields are explained in the logical order of entry versus strict numeric field order as follows:

27.3.5.1.1 .01 FILE TO BE CHECKED

Enter the file for which the developer wants to check and merge duplicates. You can only enter files that are also defined in the **.01** VARIABLE POINTER field of the DUPLICATE RECORD (#15) file. If the file you are interested in is *not* there, contact the Kernel Toolkit team for coordination.

27.3.5.1.2 .09 CANDIDATE COLLECTION ROUTINE

This field is updated with the name of the routine that the Duplicate Resolution Utilities executes to generate the list of potential duplicate candidates. The list of candidates is passed back to the merge shell in **^TMP("XDRD",$J,file number**. For example, if this is a patient merge utility, the candidate collection routine might pass back, to the merge shell, all patients who have the same last name as the record being processed, the same DOB as the record being processed, or who have the same or similar Social Security Number (SSN). This candidate collection routine is used to minimize the number of records the merge shell has to process in determining potential duplicates.

REF: For an example of a Candidate Collection routine, see the "Candidate Collection Routine for Patient Merge Example" section.

Selecting Fields to Compare in Candidate Collection:

- The developer needs to give this considerable thought as selecting wrong fields for candidate collection results in missed or many false potential duplicate candidates.
- The most important characteristic that a field should have is the probability of containing data. If a SSN field exists in a file but the field is rarely filled in, it would *not* be a good field from which to build candidates.
- Since selection of candidates deals with minimizing the set of records to test further, look at the whole file initially. It becomes desirable for the field to have a cross reference.
- Uniqueness of a field is also important. If all records contain one of two possible values (e.g., Male or Female), it makes little sense for you to select all records that are the same value as the record compared. However, such a field can be useful later in performing individual tests.
- One final point to keep in mind is, if you finally come up with very few fields to collect candidates on, you may need to be very liberal in the comparison. Furthermore, you might want to make more than one pass through the same field with different comparison logic, hoping to find additional records that you missed initially.

27.3.5.1.3 1100 DUPLICATE TESTS

The developer *must* identify data items/fields to be used to assist in determining if a pair of records are duplicates. These items/fields *must* be single valued fields (i.e., data in Multiple fields is *not* supported), as follows:

27.3.5.1.3.1 .01 DUPLICATE TEST

This is a free text name for the test (e.g., Name, SSN, and DOB).

27.3.5.1.3.2 .02 ORDER OF TEST

Enter in the numeric value of the order you want the tests executed.

27.3.5.1.3.3 .03 DUPLICATE TEST ROUTINE

Enter the name of the routine that is called to do the actual comparison of the two records for a specific field.

REF: For examples of duplicate test routines, see the "Duplicate Test Routine Examples" section.

Table 32: .03 DUPLICATE TEST ROUTINE—Variables Passed to the Test Routine

Variable	Value
XDRCD	IEN of Record **1**.
XDRCD2	IEN of Record **2**.
XDRFL	File number being checked
XDRDTEST(XDRDTO)	**Zero** node of the test entry from the DUPLICATE RESOLUTION (#15.1) file
XDRDCD(XDRFL,XDRCD,field number,"I")	Internal data value for this field for Record **1**.
XDRDCD2(XDRFL,XDRCD2,field number,"I")	Internal data value for this field for Record **2**.
XDRD("test score")	**0**; This variable is used to pass the test score back to **XDRDUP**.

The successful maximum score can be obtained from the following:

```
$P(XDRDTEST(XDRDTO),U,6)
```

The unsuccessful score can be obtained from the following:

```
$P(XDRDTEST(XDRDTO),U,7)
```

Within the duplicate test routine, the developer can assign the entire successful match weight if both records' data is exactly the same, or he can assign a percentage of the match score if the data is similar, but *not* exactly the same. For example, if Record **1** has a NAME of XUPATIENT,ONE-TWO and Record **2** has a NAME of XUPATIENT,ONE and the successful match weight for NAME is **50** points, this pair might be assigned **90%** of the total **50** points. The developers have to go through trial and error methods of changing and calculating the percent of the total match score that is assigned.

REF: For examples of duplicate test routines, see the "Duplicate Test Routine Examples" section.

27.3.5.1.3.4 .04 FILE FOR INFORMATION

If the field that is being tested is *not* in the base file being checked, the developer *must* enter the file where the information is stored. For example, in the Indian Health Service (IHS) Patient Merge, the TRIBE OF MEMBERSHIP is a field used for a duplicate test, and this data field is stored in the IHS PATIENT (#2) file. If no entry is made in this field, the Merge (Duplicate Resolution Utilities) assumes the base file.

27.3.5.1.3.5 .05 FIELD TO BE CHECKED

This field contains the field number of the data being used for this test. The developer *must* be aware that Multiple fields *cannot* be used for duplicate tests.

27.3.5.1.3.6 .06 SUCCESSFUL MATCH WEIGHT

This is the score or total number of points assigned when a match is made on the data item being checked. This score can be anywhere from **0** to **99**. The development team needs to determine the level of confidence associated with each test. The higher confidence fields would be assigned a greater successful match score than the lower confidence fields. For example, in a Patient Merge, if NAME matches exactly, a total of **60** points might be given, but if SEX or TRIBE OF MEMBERSHIP match exactly only **10** points is given. The total number of points between all the tests does *not* have to equal **100**. The calculations to determine whether or *not* the pair is a potential duplicate is based on a percentage of the total possible score. If a data item is missing, it does *not* figure in the denominator in calculating the percentage.

27.3.5.1.3.7 .07 UNSUCCESSFUL MATCH WEIGHT

This is the score or total number of points assigned when the data items for the two records being checked do *not* match. This score is normally a negative number. For example, if the DOB for the two records is different, a score of **-40** might be assigned. This score can be anywhere from **0** to **-99**. The development team needs to determine the level of confidence associated with each test. The higher confidence fields would be assigned a greater negative unsuccessful match score than the lower confidence fields.

27.3.5.1.4 .15 POTENTIAL DUPLICATE THRESHOLD%

This is the possible percentage out of **100** after the accumulation of the test scores. If the final accumulated test score is equal to or greater than this percentage of the total possible points, the record pair is added to the DUPLICATE RECORD (#15) file as a potential duplicate pair. The percentage has to be experimented with to find the best percentage to use. It is *recommended* that the percentage be set low at first and gradually increased to find the best possible percentage, so that you do *not* have a large number of false negatives.

27.3.5.1.5 .25 MERGE STYLE

This determines whether or *not* the merge process is to be interactive or *not*. It is *highly recommended* that the merge be interactive. If it is interactive, the user is able to select fields from both the **FROM** and the **TO** (target) record. If *non*-interactive, all values are taken from the source record.

27.3.5.1.6 .11 DUPLICATE MANAGER MAIL GROUP

This field contains a pointer to the mail group that receives messages in cases when the duplicate checking process could *not* be started. Some examples of conditions that would generate bulletins include:

- Test routine is *not* present.
- No entry in the DUPLICATE RESOLUTION (#15.1) file for this field.
- Global root node in ^DIC is undefined.

27.3.5.1.7 .16 VERIFIED DUPLICATE MAIL GROUP

This field contains a pointer to the mail group that receives messages when a pair of records have been verified as duplicates. For example, in the case of a patient merge, there might be things that pharmacy or lab staff want to do before the two records are merged.

27.3.5.1.8 .17 VERIFIED DUPLICATE MSG ROUTINE

This field allows a software developer to send a customized bulletin notifying the Verified Duplicate Mail Group about verified duplicates. If nothing is entered, the Kernel Duplicate Resolution software sends a brief bulletin to the members of the mail group. This bulletin only provides the **.01** value and the **DFN** numbers of the two records. The Duplicate Resolution software passes the **XDRMFR** and **XDRMTO** routines and it is up to this routine to gather any other information it wants to send in the bulletin and also to send the bulletin to the Verified Duplicate Mail Group. A label entry point is allowed but you *must* use a hyphen (**-**) instead of the normal caret (**^**), such as **ENTRY POINT-**.

27.3.5.1.9 .29 MERGE MAIL GROUP

This field contains a pointer to the mail group that receives messages when a pair of records have been merged. Generally, this is the same mail group as the VERIFIED DUPLICATE MAIL GROUP (#.16). These recipients can examine the merged-to record to make sure that all data transferred from the merged-from record successfully.

27.3.5.1.10 .31 MERGE MSG ROUTINE

This field is allows a software developer to send a customized bulletin notifying the Merge Mail Group about merged duplicate pairs. If nothing is entered, the Kernel Duplicate Resolution software sends a brief bulletin to the members of the mail group. The Kernel Bulletin only provides the **.01** values and the **DFN**s of the two records. The Duplicate Resolution software passes the **XDRMFR** and **XDRMTO** routines and it is up to the routine to gather any information it wants to send in the bulletin and also to send the bulletin to the Merge Mail Group. A label entry point is allowed but you *must* use a hyphen (**-**) instead of the normal caret (**^**), such as **ENTRY POINT-ROUTINE**. This entry point is executed by the Duplicate Resolution software after transforming the **-** into a **^**.

Also, this routine might very well need to be different from the VERIFIED DUPLICATE MSG ROUTINE (#.17), because the information that users need to see after the merge is different from before.

27.3.5.1.11 .18 VERIFIED DUPLICATE THRESHOLD%

If this field contains a percentage from **0** to **100**, the Duplicate Resolution Utilities (**XDR** namespace) software automatically marks the two records as Verified Duplicates if the comparison score percentage is equal or greater to this value. This number, if entered, needs to be somewhat high, probably above **90%** (e.g., IHS does *not* use this field in the case of the patient merge, because they would like human determination if the two records are indeed duplicates).

27.3.6 Special Processing Routine Examples

27.3.6.1 Candidate Collection Routine for Patient Merge Example

Figure 238: Special Processing Routine Examples—Candidate Collection Routine for Patient Merge

```
DPTDCAN    ;IHS/OHPRD/REDACTED - GETS POSSIBLE DUPLICATE CANDIDATES
;09/16/93/ 08:19
     ;;1.0;DPTD;;
     ;
     ; Calls: EN^DIQ1
     ;
START      ;
     K ^TMP("XDRD",$J,XDRFL),DPTDCAN
     Q:$P(^DPT(XDRCD,0),U,19)
     D VALUE
     D NAME
     D SSN
     D DOB
END  D EOJ
     Q
     ;
VALUE      ;
     S DIC=2,DA=XDRCD,DIQ(0)="I",DIQ="DPTDCAN",DR=".01;.03;.09"
     D EN^DIQ1 K DIC,DA,DR,DIQ
     Q
     ;
NAME ;Get patients with the same last name and first initial
     G:DPTDCAN(XDRFL,XDRCD,.01,"I")']"" NAMEX
     S DPTDCAN("NAME")=DPTDCAN(XDRFL,XDRCD,.01,"I")
     S
DPTDCAN("LNAME&FI")=$P(DPTDCAN("NAME"),",",1)_","_$E($P(DPTDCAN("NAME"
),",",2),1)_"AAA"
     S DPTDCAN("BNAME")=DPTDCAN("LNAME&FI")
     F I=0:0 S DPTDCAN("BNAME")=$O(^DPT("B",DPTDCAN("BNAME")))
Q:DPTDCAN("BNA
ME")=""!(($P(DPTDCAN("NAME"),",",1)_","_$E($P(DPTDCAN("NAME"),",",2),1)
)'=($P(DPTDCAN("BNAME"),",",1)_","_$E($P(DPTDCAN("BNAME"),",",2),1)))
D
. S DPTDCAN("BNAMEDFN")=0 F  S DPTDCAN("BNAMEDFN")=$O(^DPT("B",DPTDCAN("
BNAME"),DPTDCAN("BNAMEDFN"))) Q:DPTDCAN("BNAMEDFN")=""  S:DPTDCAN("BNAM
EDFN")'=XDRCD ^TMP("XDRD",$J,XDRFL,DPTDCAN("BNAMEDFN"))="".
QNAMEX     Q
     ;
SSN  ;Get patients with same last four digits of ssn
     G:DPTDCAN(XDRFL,XDRCD,.09,"I")']"" SSNX
     S DPTDCAN("SSN")=DPTDCAN(XDRFL,XDRCD,.09,"I")
     S DPTDCAN("L4SSN")=$E(DPTDCAN("SSN"),6,9)
     S DPTDCAN("BL4SSN")=XDRCD
     F %=0:0 S
DPTDCAN("BL4SSN")=$O(^DPT("BS",DPTDCAN("L4SSN"),DPTDCAN("BL4SS
N"))) Q:'DPTDCAN("BL4SSN")  S ^TMP("XDRD",$J,XDRFL,DPTDCAN("BL4SSN"))=""
     ;
     ; Check SSNS with same first five digits
     ; Commented out the following line, is not specific enough for IHS
     ; but would be useful for the VA
     ;
     ;S
DPTDCAN("F5SSN")=$E(DPTDCAN("SSN"),1,5)_"0000",DPTDCAN("5SSN")=DPTDCA
N("F5SSN") D
     . F %=0:0 S DPTDCAN("5SSN")=$O(^DPT("SSN",DPTDCAN("5SSN")))
Q:DPTDCAN("5
SSN")'=+DPTDCAN("5SSN")!($E(DPTDCAN("5SSN"),1,5)'=$E(DPTDCAN("SSN"),1,5
))  S ^TMP("DPTDCAN",$J,XDRFL,$O(^DPT("SSN",DPTDCAN("5SSN"),"")))=""
     . Q
SSNX Q
     ;
DOB  ;Get patients with same date of birth
     G:DPTDCAN(XDRFL,XDRCD,.03,"I")']"" DOBX
     S DPTDCAN("DOB")=DPTDCAN(XDRFL,XDRCD,.03,"I")
     S DPTDCAN("BDOB")=XDRCD
     F %=0:0 S
DPTDCAN("BDOB")=$O(^DPT("ADOB",DPTDCAN("DOB"),DPTDCAN("BDOB"))
) Q:'DPTDCAN("BDOB")  S ^TMP("XDRD",$J,XDRFL,DPTDCAN("BDOB"))=""
     ;
     ;Transpose day of birth and get patients with same date of birth
     ;
     S
DPTDCAN("TDOB")=$E(DPTDCAN("DOB"),1,5)_$E(DPTDCAN("DOB"),7)_$E(DPTDCAN
("DOB"),6)
     S DPTDCAN("BDOB")=XDRCD
     F %=0:0 S
DPTDCAN("BDOB")=$O(^DPT("ADOB",DPTDCAN("TDOB"),DPTDCAN("BDOB")
)) Q:'DPTDCAN("BDOB")  S ^TMP("XDRD",$J,XDRFL,DPTDCAN("BDOB"))=""
DOBX Q
     ;
EOJ  ;
     K DPTDCAN,%
     Q
```

27.3.6.2 Duplicate Test Routine Examples

27.3.6.2.1 Name Test Routine for a Patient Merge Example

Figure 239: Special Processing Routine Examples—Name Test Routine for a Patient Merge

```
DPTDN      ;IHS/OHPRD/REDACTED;COMPARES NAMES; [ 06/08/92  12:14 PM ]
     ;;1.0;DPTD;;AUG 13, 1991
     ;
     ; Calls: SOU^DICM1
     ;
START      ;
     D INIT
     D NAME
     I $O(^DPT(XDRCD,.01,0)) D OTHER
END  D EOJ
     Q
     ;
EN   ; EP - Entry Point for any routines comparing names
     ;
     D INIT1
     D COMPARE
     D EOJ
     Q
     ;
INIT ;
     D EOJ
     S DPTDN("MATCH")=$P(XDRDTEST(XDRDTO),U,6)
     S DPTDN("NO MATCH")=$P(XDRDTEST(XDRDTO),U,7)
     S
DPTDN=$G(XDRCD(XDRFL,XDRCD,.01,"I")),DPTDN2=$G(XDRCD2(XDRFL,XDRCD2,.01
,"I"))
     ;
INIT1      S DPTDNL=$P(DPTDN,","),DPTDNF=$P($P(DPTDN,",",2),"
"),DPTDNFI=$E(DPTDNF)
     ,DPTDNM=$P($P(DPTDN,",",2)," ",2),DPTDNMI=$E(DPTDNM)
     ;
INIT2      S DPTDNL2=$P(DPTDN2,","),DPTDNF2=$P($P(DPTDN2,",",2),"
"),DPTDNFI2=$E(DP
TDNF2),DPTDNM2=$P($P(DPTDN2,",",2)," ",2),DPTDNMI2=$E(DPTDNM2)
     Q
     ;
NAME ;
     D COMPARE
     D:$O(^DPT(XDRCD2,.01,0)) OTHER2
     Q
     ;
OTHER      ;
     F DPTDNO=0:0 S DPTDNO=$O(^DPT(XDRCD,.01,DPTDNO)) Q:'DPTDNO  S
DPTDN=$P(^
DPT(XDRCD,.01,DPTDNO,0),U,1) S:'$D(DPTDN2)
DPTDN2=XDRCD2(XDRFL,XDRCD2,.01,"I") D INIT1,NAME
     Q
     ;
OTHER2     ;
     F DPTDNO2=0:0 S DPTDNO2=$O(^DPT(XDRCD2,.01,DPTDNO2)) Q:'DPTDNO2  S
DPTDN
2=$P(^DPT(XDRCD2,.01,DPTDNO2,0),U,1) D INIT2,COMPARE
     Q
     ;
COMPARE    ;
     S:'$D(DPTDN("TEST SCORE")) DPTDN("TEST SCORE")=DPTDN("NO MATCH")
     I DPTDN=DPTDN2 S DPTDN("TEST SCORE2")=DPTDN("MATCH") G COMPAREX

       I DPTDNF=DPTDNF2,DPTDNL=DPTDNL2 S DPTDN("TEST
SCORE2")=DPTDN("MATCH")*.8
  G COMPAREX
     I DPTDNFI=DPTDNFI2,DPTDNL=DPTDNL2 S DPTDN("TEST
SCORE2")=DPTDN("MATCH")*
.6 G COMPAREX
     I DPTDNL=DPTDNL2 S DPTDN("TEST SCORE2")=DPTDN("MATCH")*.4 G COMPAREX
     S X=DPTDNL D SOU^DICM1 S DPTDNLS=X S X=DPTDNL2 D SOU^DICM1 S
DPTDNL2S=X
     S X=DPTDNF D SOU^DICM1 S DPTDNFS=X S X=DPTDNF2 D SOU^DICM1 S
DPTDNF2S=X
     I DPTDNLS=DPTDNL2S,DPTDNFS=DPTDNF2S S DPTDN("TEST
SCORE2")=DPTDN("MATCH"
)*.6 G COMPAREX
     I DPTDNFS=DPTDNF2S S DPTDN("TEST SCORE2")=DPTDN("MATCH")*.2 G COMPAREX
     S DPTDN("TEST SCORE2")=DPTDN("NO MATCH")
COMPAREX   ;
     S:DPTDN("TEST SCORE2")>(DPTDN("TEST SCORE")) DPTDN("TEST
SCORE")=DPTDN("
TEST SCORE2")
     K X,DPTDNLS,DPTDNL2S,DPTDNFS,DPTDNF2S,DPTDN("TEST SCORE2")
     Q
     ;
EOJ ;
     S:$D(DPTDN("TEST SCORE")) XDRD("TEST SCORE")=DPTDN("TEST SCORE")
     K DPTDN,DPTDN2,DPTDNF,DPTDNF2,DPTDNL,DPTDNL2,DPTDNM,DPTDNM2
     K DPTDNMI,DPTDNMI2,DPTDNFI,DPTDNFI2,DPTDNO,DPTDNO2
     Q
```

27.3.6.2.2 Date of Birth test Routine for a Patient Merge Example

Figure 240: Special Processing Routine Examples—Date of Birth Test Routine for a Patient Merge

```
DPTDOB    ;IHS/OHPRD/REDACTED;COMPARES DATE OF BIRTHS; [ 06/08/92  12:10
PM ]
     ;;1.0;DPTD;;AUG 13, 1991
START     ;
     D INIT
EN   ; EP - Entry point for comparing dates
     D COMPARE
END  D EOJ
     Q
     ;
INIT ;
     K DPTDOB,DPTDOB2
     S
DPTDOB=$G(XDRCD(XDRFL,XDRCD,.03,"I")),DPTDOB2=$G(XDRCD2(XDRFL,XDRCD2,.
03,"I"))
     S DPTDOB("MATCH")=$P(XDRDTEST(XDRDTO),U,6)
     S DPTDOB("NO MATCH")=$P(XDRDTEST(XDRDTO),U,7)
     Q
     ;
COMPARE   ;
     I DPTDOB']""!(DPTDOB2']"") G COMPAREX
     I DPTDOB=DPTDOB2 S XDRD("TEST SCORE")=DPTDOB("MATCH") G COMPAREX
     S DPTDOB("CNT")=0
     F DPTDOBI=1:1:7 Q:DPTDOB("CNT")>2  I
$E(DPTDOB,DPTDOBI)'=$E(DPTDOB2,DPTD
OBI) S DPTDOB("CNT")=DPTDOB("CNT")+1
     K DPTDOBI
     S XDRD("TEST SCORE")=$S(DPTDOB("CNT")>2:DPTDOB("NO
MATCH"),1:(DPTDOB("MA
TCH")*.8))
COMPAREX  Q
     ;
EOJ  ;
     K DPTDOB,DPTDOB2
     Q
```

27.3.7 Application Programming Interfaces (APIs)

27.3.8 EN^XDRMERG(): Merge File Entries

Reference Type: Supported

Category: Toolkit—Duplicate Record Merge

ICR #: 2365

Description: The EN^XDRMERG API provides for merging of one or more pairs of records in a specified file. This API takes two (**2**) arguments:

- File number (a numeric value).
- Closed reference to the location where the program finds an array with subscripts indicating the record pairs to be merged (a text value).

Format: `EN^XDRMERG(file,arraynam)`

Input Parameters: **file**: (required) Specifies the file number of the file in which the indicated entries are to be merged.

Input/Output

Parameter: **arraynam**: (required) This parameter contains the name of the array as a closed root under which the subscripts indicating the **FROM** and **TO** entries are found. The data can have either two or four subscripts descendent from the array, which is passed.

REF: For examples of its usage, see the "Overview" section.

27.3.8.1 Examples

The following command would result in record pairs specified as subscripts in the array **MYLOC** to be merged in a hypothetical file #999000014:

```
D EN^XDRMERG(999000014,"MYLOC")
```

The array MYLOC might have been set up prior to this call in the following manner (or any equivalent way) where the subscripts represent the internal entry numbers of the **FROM** and **TO** records, respectively.

```
S MYLOC(147,286)="",MYLOC(182,347)="",MYLOC(2047,192)=""
S MYLOC(837,492)="",MYLOC(298,299)=""
```

This would result in five record pairs being merged with record **147** (the **FROM** record) being merged into record **286** (the **TO** record), record **182** being merged into record **347**, etc., to record

298 being merged into **299**. Merges using the two subscript format occurs without a specific record of the entries prior to the merge (The internal entry numbers merged would be recorded under the file number in XDR REPOINTED ENTRY [#15.3] file) An alternative is a four subscript format for the data array that uses VARIABLE POINTER formats for the **FROM** and **TO** records as the third and fourth subscripts. If the merge is performed with this four subscript array, then a pre-merge image of the data of both the **FROM** and **TO** records in the primary file and all other merged files (those related by **DINUM**) and information on all single value pointer values modified is stored in the MERGE IMAGE (#15.4) file.

For the sample data above [assuming that the global root for the hypothetical File #999000014 is ^DIZ(999000014,] the four subscript array might be generated using the following code:

Figure 241: EN^XDRMERG API—Example

```
S MYROOT=";DIZ(99900014,"  <--- note the leading ^ is omitted
S MYLOC(147,286,147_MYROOT,286_MYROOT)=""
S MYLOC(182,347,182_MYROOT,347_MYROOT)=""
S MYLOC(2047,192,2047_MYROOT,192_MYROOT)=""
S MYLOC(837,492,837_MYROOT,492_MYROOT)=""
S MYLOC(298,299,298_MYROOT,299_MYROOT)=""
;
D EN^XDRMERG(99900014,"MYLOC")
```

Exclusion of Multiple Pairs For a Record—To insure that there are no unanticipated problems due to relationships between a specific record in multiple merges, prior to actually merging any data the various **FROM** and **TO** records included in the process are examined, and if one record is involved in more than one merge, all except the first pair of records involving that one are excluded from the merge. If any pairs are excluded for this reason, a mail message is generated to the individual responsible for the merge process as indicated by the **DUZ**.

If the following entries were included in the **MYLOC** array:

```
MYLOC(128,247)
MYLOC(128,536) and
MYLOC(247,128)
```

Only the first of these entries (based on the numeric sorting of the array) would be permitted to remain in the merge process, while the other two pairs would be omitted). And although it may seem unlikely that someone would indicate that a record should be merged into two different locations, while another location should be merged into one that was merged away, if the pairs are selected automatically and checks are *not* included to prohibit such behavior, they show up. That is why the merge process does *not* include more than one pair with a specific record in it.

27.3.8.2 Problems Related To Data Entry While Merging

The Merge Process has been designed to combine data associated with the two records in the manner described above. On occasion, however, there are problems that cause VA FileMan to reject the data that is being entered. This may happen for a number of reasons. Some examples that have been observed include:

- Clinics that had been changed so they no longer were indicated as Clinics (so they would *not* add to the number that people had to browse through to select a clinic), but were rejected since the input transform checked that they be clinics.
- Pointer values that no longer had a valid value in the pointed to file (dangling pointers).
- Fields that have input transforms that prohibit data entry.

It is possible to use a validity checker on your data prior to initiating the actual merge process (this is the action taken by merges working from the Potential Duplicate file). The data pairs are processed in a manner similar to the actual merge, so only that data in any of the files that would be merged and for which the data would be entered using VA FileMan utilities for the specific pair are checked to insure they pass the input transform. Any problems noted are incorporated into a mail message for resolution prior to attempting to merge the pair again, and the pair is removed from the data array that was passed in. Pairs that pass through this checking should *not* encounter any data problems while being merged.

27.3.9 RESTART^XDRMERG(): Restart Merge

Reference Type:	Supported	
Category:	Toolkit—Duplicate Record Merge	
ICR #:	2365	
Description:	The RESTART^XDRMERG API restarts a merge that has been stopped. The information necessary for restarting can be viewed using the CHKLOCAL^XDRMERG2 API (see LOCAL MERGE STATUS).	
Format:	`RESTART^XDRMERG(file,arraynam,phase,currfile,currien)`	
Input Parameters:	**file**:	(required) Specifies the file number of the file in which the indicated entries are to be merged.
	arraynam:	(required) This parameter contains the name of the array as a closed root under which the subscripts indicating the **FROM** and **TO** entries are found. The data can have either two or four subscripts descendent from the array, which is passed. **REF:** For examples of its usage, see the "Overview" section in the "Toolkit—Duplicate Record Merge" section.

phase:	(required) This parameter indicates the phase of the merge process in which the merge should be restarted. The value is a number in the range of **1** to **3**, with no decimal places: • **Phase 1** is usually quite short and is the merge of the specified entries in the primary file. • **Phase 2** is the merging of entries in files that are DINUMed to the primary file and changing pointers that can be identified from cross-references. • **Phase 3** is finding pointer values by searching each entry in a file. This is usually the longest phase of the merge process.
currfile:	(required) This is the current file number on which the merge process is operating.
currien:	(required) This is the current internal entry number in the file on which the merge process is operating.

Output: none.

27.3.10 SAVEMERG^XDRMERGB(): Save Image of Existing and Merged Data

Reference Type: Controlled Subscription

Category: Toolkit—Duplicate Record Merge

ICR #: 2338

Description: During special processing related to the Patient Merge, the **IBAXDR** routine needs to call the SAVEMERG^XDRMERGB API. The SAVEMERG^XDRMERGB API saves the file image of an entry involved in the merge process when only one of the entries (the entry being merged or the entry being merged into) is present in the **filenum** input parameter. Normally, the merge process would handle when it can identify a **FROM** or a **TO** entry that is *not* present based on the DINUMed values. For **filenum**, however, the internal entry numbers are determined from the "**B**"-cross-reference, and missing entries need to be handled separately.

This API acts to save an image of the currently existing data for the merge entry and merged into entry in the MERGE IMAGE (#15.4) file.

Format: `SAVEMERG^XDRMERGB(filenum,ienfrom[,iento])`

Input Parameters:	**filenum**:	(required) This is the file number for the file that is being merged and for which the images are to be saved.
	ienfrom:	(required) The internal entry number of the **FROM** entry (the entry being merged into another entry).
	iento:	(optional) The internal entry number of the **TO** entry (the entry into which the entry is being merged).
Output:	results:	Stored image.

27.4 Toolkit—HTTP Client

27.4.1 Overview

The Kernel Toolkit Hypertext Transfer Protocol (HTTP) Client Helper software release adds a new tool in a set of Infrastructure software tools that developers can use. HTTP is a fast and reliable way for an application to collect data from another source. Kernel Toolkit patch XT*7.3*123 allows VistA to t into this information and retrieve Web data.

NOTE: Kernel Toolkit patch XT*7.3*138 adds support for IPv6, HTTP/1.1, and HTTPS.

This code was originally developed by another VistA application that had a pressing need for this capability. The Kernel Toolkit development team is providing and maintaining it as generic tool so that other developers may use its functionality for their needs. For example:

- KIDS: Uses it to get the checksums from FORUM of patches that are sent in a Host File System (HFS) file.
- Pharmacy: Uses it to request the printing of FDA data sheets.

NOTE: XTHC* routines are part of the HTTP Client Helper application for developers.

27.4.2 Application Programming Interfaces (APIs)

27.4.3 $$GETURL^XTHC10: Return URL Data Using HTTP

Reference Type:	Supported
Category:	Toolkit—HTTP Client Helper
ICR #:	5553

Description: The $$GETURL^XTHC10 extrinsic function is a Hypertext Transfer Protocol (HTTP)/1.1 client that can request a Web page from another system and pass the returned data to the calling routine.

It can make both **GET** and **POST** requests.

It is the main API and returns in **xt8rdat** the returned data from the website.

NOTE: XTHC* routines are part of the HTTP Client Helper application for developers.

NOTE: This API was released with Kernel Toolkit patch XT*7.3*123.

NOTE: This API is **IPv6** compliant as of Kernel Toolkit patch XT*7.3*138.

Format:

```
$$GETURL^XTHC10(url[,xt8flg],xt8rdat,.xt8rhdr[,xt8sdat][,.xt8
shdr][,.xt8meth])
```

Input Parameters:

url:	(required) This is the Universal Resource Locator (URL) to access (**http://host:port/path**). It could be as simple as "**www.va.gov**".
xt8flg:	(optional) Request timeout. Default is **5** seconds.
xt8sdat:	(optional) Closed root of a variable containing the body of the request message. Data should be formatted as described in the **xt8rdat** parameter.

NOTE: If this parameter is defined (i.e., *not* empty) and the referenced array contains data, then the **POST** request is generated; otherwise, the **GET** request is sent.

.xt8shdr: (optional) Reference to a local variable containing header values, which is added to the request. For example:

```
XT8SHDR("CONTENT-TYPE")="text/html"
```

.xt8meth: (optional) Flag to indicate the request method:

- **GET**—Default if **xt8sdat** contains no data.
- **POST**—Default if **xt8sdat** contains data.
- **HEAD**
- **PUT**
- **OPTIONS**
- **DELETE**
- **TRACE**

Output / Output Parameters:

xt8rdat: (required) Closed root of the variable where the message body is returned. Data is stored in consecutive nodes (numbers starting from **1**). If a line is longer than **245** characters, only **245** characters are stored in the corresponding node. After that, overflow sub-nodes are created. For example:

```
@XT8DATA@(1)="<html>"
 @XT8DATA@(2)="<head><title>VistA</title></head>"
 @XT8DATA@(3)="<body>"
 @XT8DATA@(4)="<p>"
 @XT8DATA@(5)="Beginning of a very long line"
 @XT8DATA@(5,1)="Continuation #1 of the long
line"
 @XT8DATA@(5,2)="Continuation #2 of the long
line"
 @XT8DATA@(5,...)=...
 @XT8DATA@(6)="</p>"
```

.xt8rhdr: (required) Reference to a local variable where the parsed headers are returned. Header names are converted to uppercase; the values are left "as is". The root node contains the status line. For example:

```
XT8HDR="HTTP/1.1 200 OK"
XT8HDR("ACCEPT-RANGES")="bytes"
XT8HDR("CONNECTION")="close"
XT8HDR("CONTENT-LENGTH")="16402"
```

```
XT8HDR("CONTENT-TYPE")="text/html; charset=UTF-8"
XT8HDR("DATE")="Thu, 25 Jun 2015 14:43:01 GMT"
XT8HDR("ETAG")="a93a2-4012-5180156550680"
XT8HDR("LAST-MODIFIED")="Mon, 08 Jun 2015
13:08:26 GMT"
XT8HDR("SERVER")="Apache/2.2.15 (CentOS)"
```

Output: returns: Returns:

- **Success**: **HTTP_Status_Code^Description**

  Common HTTP status codes returned:

  Table 33: $$GETURL^XTHC10—Common HTTP Status Codes Returned

Status Code	Description
200	OK
301	Moved Permanently
400	Bad Request
401	Unauthorized
404	Not Found
407	Proxy Authentication Required
408	Request Time-out
500	Internal Server Error
505	HTTP Version *not* supported

- **Fail: -1^Error Descriptor**

  Additional error information can be found in the VistA error trap or **^XTER** in programmer mode.

  REF: For more details, visit the HTTP Frequently Asked Questions (FAQ) website at: http://www.faqs.org/rfcs/rfc1945.html or the Internet Engineering Task Force (IETF) sites at: http://www.ietf.org/rfc/rfc2616.txt (HTTP/1.1) and http://www.ietf.org/rfc/rfc2617.txt (HTTP Authentication).

27.4.4 $$ENCODE^XTHCURL: Encodes a Query String

Reference Type: Supported

Category: Toolkit—HTTP Client Helper

ICR #: 5554

Description: The $$ENCODE^XTHCURL extrinsic function encodes the query string. The $$MAKEURL^XTHCURL: Creates a URL from Components API uses this extrinsic function.

NOTE: This API was released with Kernel Toolkit patch XT*7.3*123.

NOTE: XTHC* routines are part of the HTTP Client Helper application for developers.

Format: `$$ENCODEURL^XTHCURL(str)`

Input Parameters: **str**: (required) String of data to be encoded.

Output: returns: Returns:

- **Success:** Encoded query string.
- **Fail:** -1^String *not* defined (if missing **str** parameter).

27.4.4.1 Example

Figure 242: $$ENCODE^XTHCURL API—Example

```
W $$ENCODE^XTHCURL("123+main+st.,Anycity,CA")
123%2Bmain%2Bst.%2CAnycity%2CCA
```

27.4.5 $$MAKEURL^XTHCURL: Creates a URL from Components

Reference Type:	Supported	
Category:	Toolkit—HTTP Client Helper	
ICR #:	5554	
Description:	The $$MAKEURL^XTHCURL extrinsic function creates a URL from components. **NOTE: XTHC*** routines are part of the HTTP Client Helper application for developers. **NOTE:** This API is **IPv6** compliant as of Kernel Toolkit patch XT*7.3*138. **NOTE:** This API was released with Kernel Toolkit patch XT*7.3*123.	
Format:	`$$MAKEURL^XTHCURL(host[,port][,path][,.query])`	
Input Parameters:	**host**:	(required) The Fully Qualified Domain Name (FQDN) or Internet Protocol (IP) address of the system to which it connects.
	port:	(optional) The port to use. Default is: • **HTTP**—Port **REDACTED**. • **HTTPS**—Port **REDACTED**.
	path:	(optional) The path to the Web page on the called server.
	.query:	(optional) An array of query parameters.
Output:	returns:	Returns: • **Success:** Normalized path (see Example). • **Fail: -1^Host** *not* defined (if missing **host** parameter).

27.4.5.1 Example

Figure 243: $$MAKEURL^XTHCURL API—Example

```
S host="http://www.map.com"
S path="api/staticmap"
S query("center")="main+st.,Anycity,CA"
S query("sensor")="false"
W $$MAKEURL^XTHCURL(host,,path,.query)

http://www.map.com/api/staticmap?center=main%2Bst.%2CAnycity%2CCA&sensor=f
alse
```

27.4.6 $$PARSEURL^XTHCURL: Parses a URL

Reference Type:	Supported
Category:	Toolkit—HTTP Client Helper
ICR #:	5554
Description:	The $$PARSEURL^XTHCURL extrinsic function parses a URL using into host, port, and path (path includes query string).

NOTE: XTHC* routines are part of the HTTP Client Helper application for developers.

NOTE: This API was released with Kernel Toolkit patch XT*7.3*123.

NOTE: This API is **IPv6** compliant as of Kernel Toolkit patch XT*7.3*138.

Format: `$$PARSEURL^XTHCURL(url,.host,.port,.path)`

Input Parameters:	**url**:	(required) Reference to variable where host name is to be returned.
Output Parameters:	**host**:	(required) Input URL.
	port:	(required) Reference to variable where port is to be returned.
	.path:	(required) Reference to variable where path string is to be returned.

Output:	returns:	Returns: • **Success: 0** • **Fail: -1^Error Description**

27.4.6.1 Example

Figure 244: $$PARSEURL^XTHCURL API—Example

```
D PARSEURL^XTHCURL("http://REDACTED.va.gov:REDACTED/tpl/PKG",.ZH,.ZP,.ZA)
W ZH,!,ZP,!,ZA

REDACTED.va.gov
REDACTED
/tpl/PKG
```

27.4.7 $$DECODE^XTHCUTL: Decodes a String

Reference Type: Supported

Category: Toolkit—HTTP Client Helper

ICR #: 5555

Description: The $$DECODE^XTHCUTL extrinsic function is used with the HTTP/1.1 Client. It decodes one string replacing the following:

- **&lt;** with <
- **&gt;** with >
- **&amp;** with **&**
- **&nbsp;** with **" "** (a space)
- **&os;** with **'**
- **&quot;** with **"**
- **&#65;** with **A**

NOTE: XTHC* routines are part of the HTTP Client Helper application for developers.

NOTE: This API was released with Kernel Toolkit patch XT*7.3*123.

Format: `$$DECODE^XTHCUTL(str)`

Input Parameters:	**str**:	(required) String to be decoded.
Output:	returns:	Returns: • **Success:** Decoded string. • **Fail:** -1^String *not* defined (if missing **str** parameter).

27.4.7.1 Example

Figure 245: $$DECODE^XTHCUTL API—Example

```
$$DECODE^XTHCUTL("123%2Bmain%2Bst.%2CAnytown%2CCA")
 123%2Bmain%2Bst.%2CAnytown%2CCA
```

27.5 Toolkit—KERMIT APIs

27.5.1 RFILE^XTKERM4: Add Entries to Kermit Holding File

Reference Type:	Supported
Category:	Toolkit—KERMIT
ICR #:	2075
Description:	The RFILE^XTKERM4 API allows access to the KERMIT HOLDING (#8980) file and the API that adds entries to it, RFILE^XTKERM4. The "**AOK**" cross-reference of the KERMIT HOLDING (#8980) file can be checked to see if the user has an entry in the KERMIT HOLDING (#8980) file. If *not*, RFILE^XTKERM4 can be called to add an entry to the file.

NOTE: A call to RFILE^XTKERM4 allows a user to add or select an entry in the KERMIT HOLDING (#8980) file.

Format: `RFILE^XTKERM4`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Output Variables:	**XTKDIC:**	This variable returns the global root and is a calling variable used by calls to RECEIVE^XTKERMIT: Load a File into the Host or SEND^XTKERMIT: Send Data from Host APIs.
	XTMODE:	This variable is returned. It is used as input to calls to RECEIVE^XTKERMIT: Load a File into the Host or SEND^XTKERMIT: Send Data from Host APIs.

27.5.2 RECEIVE^XTKERMIT: Load a File into the Host

Reference Type:	Supported
Category:	Toolkit—KERMIT
ICR #:	10095
Description:	The RECEIVE^XTKERMIT API loads a file into the host.
Format:	`RECEIVE^XTKERMIT`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Variables to call from outside of Kermit:	**XTKDIC:**	(required) Set **XTKDIC** to VA FileMan type global root.
	DWLC:	(required) Set **DWLC** to last current data node. Return **DWLC** to last data node, **XTKDIC** is **KILL**ed.
	TIREF:	(optional) Set **XTKMODE** as follows to send/receive: • **0**—Send/Receive in **IMAGE** mode (no conversion). • **1**—Send/Receive in **DATA** mode (just convert control character). • **2**—Send/Receive as **TEXT** (VA FileMan word-processing). Text mode sends a carriage return (**CR**) after each global node; makes a new global node for each **CR** received. **XTKMODE** set to **2** would be normal for most VistA applications.

27.5.3 SEND^XTKERMIT: Send Data from Host

Reference Type:	Supported
Category:	Toolkit—KERMIT
ICR #:	10095
Description:	The SEND^XTKERMIT API sends data from the host.
Format:	`SEND^XTKERMIT`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Variables to call from outside of KERMIT:	**XTKDIC:**	(required) Set **XTKDIC** to VA FileMan type global root.
	DWLC:	(required) Set **DWLC** to last current data node. Return **DWLC** to last data node; **XTKDIC** is **KILL**ed.
	TIREF:	(optional) Set **XTKMODE** as follows to send/receive: • **0**—Send/Receive in **IMAGE** mode (no conversion). • **1**—Send/Receive in **DATA** mode (just convert control character). • **2**—Send/Receive as **TEXT** (VA FileMan word-processing). Text mode sends a carriage return (**CR**) after each global node; makes a new global node for each **CR** received. **XTKMODE** set to **2** would be normal for most VistA applications.

27.6 Toolkit—Multi-Term Look-Up (MTLU) APIs

27.6.1 How to Override

If files are fully configured for the special Multi-Term Look-Up, all standard VA FileMan lookups invoke MTLU. The following procedures can be taken to override MTLU:

- Users can enter an accent grave (`) as a prefix to request a lookup by the Internal Entry Number (IEN).
- Users can enter a tilde (~) as a prefix to force a standard VA FileMan lookup.

NOTE: In the event that a search produces no matches, MTLU continues with a standard VA FileMan search by default.

- Developers can override MTLU by setting the variable **XTLKUT=""** prior to referencing the file and **KILL**ing it upon exit, or set **DIC(0)** to include **I**:

```
S DIC=81,DIC(0)="AEMQI",X="" D ^DIC
```

27.6.2 Application Programming Interfaces (APIs)

27.6.2.1 MTLU and VA FileMan Supported Calls

Developers can perform any supported VA FileMan calls on files fully configured for MTLU.

The preferred method of performing lookups from Programmer mode is to add the target file to the LOCAL LOOKUP (#8984.4) file and call LKUP^XTLKMGR. However, Multi-Term Look-Ups can be performed on any VA FileMan file, even if it has *not* been configured for use by MTLU. Using the developer API, the lookup can be performed using any index contained within the file, such as a VA FileMan **KWIC** cross-reference.

Entry Point: XTLKKWL

Required Input Variables:

(XTLKGBL, XTLKKSCH("GBL")): This is the global root (same as DIC).

XTLKKSCH("DSPLY"): This variable displays the routine. For example:

```
DGEN^XTLKKWLD
```

XTLKKSCH("INDEX"): Cross-reference selected by the developer for performing a multi-term lookup.

XTLKX: This is the user input.

Optional Input

Variables:	**XTLKSAY:**	This variable equals **1** or **0**. If **XTLKSAY = 1**, MTLU displays details during the lookup. **NOTE:** The purpose of **XTLKSAY** is to control the degree of output to the screen, *not* the amount of "file information" displayed.
	XTLKHLP:	Executable code to display custom help.

27.6.2.2 Kernel Toolkit Enhanced APIs

Programmer calls to MTLU-configured files return all standard VA FileMan variables (i.e., **Y**, **DTOUT**, **DUOUT**, **DIROUT**, and **DIRUT**).

The programmer's API for performing a lookup has been enhanced functionally, simplified, and converted to a procedure call.

Procedure calls provide full, *non*-interactive management of the following MTLU control files: LOCAL KEYWORD (#8984.1), LOCAL SHORTCUT (#8984.2), LOCAL SYNONYM (#8984.3), and LOCAL LOOKUP (#8984.4).

All procedure calls are contained in the routine **^XTLKMGR**.

Errors are returned in the **XTLKER()** array. **KILL** this array *before* calling any of these new procedure calls, and check the array after returning from the calls. All calls require that the target file be defined in the LOCAL LOOKUP (#8984.4) file. If removing an entry from the LOCAL LOOKUP (#8984.4) file, all shortcuts, synonyms, and keywords associated with that file *must* be deleted first.

27.6.3 XTLKKWL^XTLKKWL: Perform Supported VA FileMan Calls on Files Configured for MTLU

Reference Type:	Supported
Category:	Toolkit—Multi-Term Look-Up (MTLU)
ICR #:	10122
Description:	The XTLKKWL^XTLKKWL API lets developers perform any supported VA FileMan calls on files configured for MTLU. To ignore the special lookup routine, XTLKDICL, be sure that **DIC(0)** includes an **I**. Alternatively, multi-term lookups can be performed on any VA FileMan file, even if it has *not* been configured for primary use by MTLU. Using the API, the lookup can be performed using any index contained within the file, such as a VA FileMan Key Word In Context (**KWIC**) cross-reference.
Format:	`XTLKKWL^XTLKKWL`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Variables:	**(XTLKGBL, XTLKKSCH("GBL")):** (required) This is the global root (same as **DIC**).
XTLKKSCH("DSPLY"):	(required) This variable displays the routine. For example: `DGEN^XTLKKWLD`
XTLKKSCH("INDEX"):	(required) Cross-reference selected by the developer for performing a MTLU.
XTLKX:	(required) This is the user input.
XTLKSAY:	(optional) **XTLKSAY** values: • **1**—MTLU displays details during the lookup. • **0**. **NOTE:** The purpose of **XTLKSAY** variable is to control the degree of output to the screen, *not* the amount of "file information" displayed.
XTLKHLP:	(optional) **XTLKHLP**=Executable code to display custom help.

27.6.4 DK^XTLKMGR(): Delete Keywords from the Local Keyword File

Reference Type:	Supported
Category:	Toolkit—Multi-Term Look-Up (MTLU)
ICR #:	10153
Description:	The DK^XTLKMGR API deletes keywords from the LOCAL KEYWORD (#8984.1) file.
Format:	`DK^XTLKMGR(xtlk1,xtlk2)`

Input Parameters:	**xtlk1**:	(required) File name.
	xtlk2:	(required) Leave this parameter undefined to delete *all* keywords for a given target file, or pass in an array for *selected* keywords.
Output:	none.	

27.6.5 DLL^XTLKMGR(): Delete an Entry from the Local Lookup File

Reference Type: Supported

Category: Toolkit—Multi-Term Look-Up (MTLU)

ICR #: 10153

Description: The DLL^XTLKMGR API deletes an entry from the LOCAL LOOKUP (#8984.4) file.

Format: `DLL^XTLKMGR(xtlk1)`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Parameters:	**xtlk1**:	(required) The associated file name or number.
Output Variables:	**XTLKER(1,FILENAME):**	File is *not* in the LOCAL LOOKUP (#8984.4) file.
	XTLKER:	Entries exist for keywords, shortcuts, or synonyms for the associated file. These *must* be deleted first.

27.6.6 DSH^XTLKMGR(): Delete Shortcuts from the Local Shortcut File

Reference Type: Supported

Category: Toolkit—Multi-Term Look-Up (MTLU)

ICR #: 10153

Description: The DSH^XTLKMGR API deletes shortcuts from the LOCAL SHORTCUT (#8984.2) file.

Format: `DSH^XTLKMGR(xtlk1,xtlk2)`

Input Parameters:	**xtlk1**:	(required) File name.
	xtlk2:	(required) Leave this parameter undefined to delete all shortcuts for a given target file or pass in an array for selected shortcuts.

Output: none.

27.6.7 DSY^XTLKMGR(): Delete Synonyms from the Local Synonym File

Reference Type: Supported

Category: Toolkit—Multi-Term Look-Up (MTLU)

ICR #: 10153

Description: The DSY^XTLKMGR API deletes synonyms from the LOCAL SYNONYM (#8984.3) file.

Format: `DSY^XTLKMGR(xtlk1,xtlk2)`

Input Parameters:	**xtlk1**:	(required) File name.
	xtlk2:	(required) Leave this parameter undefined to delete *all* synonyms for a given target file, or pass in an array for *selected* synonyms.

Output: none.

27.6.8 K^XTLKMGR(): Add Keywords to the Local Keyword File

Reference Type: Supported

Category: Toolkit—Multi-Term Look-Up (MTLU)

ICR #: 10153

Description: The K^XTLKMGR API adds Keywords to the LOCAL KEYWORD (#8984.1) file.

Format: `K^XTLKMGR(xtlk1,xtlk2,xtlk3)`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Parameters:	**xtlk1**:	(required) Associated file.
	xtlk2:	(required) Code in the associated file.
	xtlk3:	(required) Keyword.
Output Variables:	**XTLKER(1,FILENAME):**	File *not* defined in the LOCAL LOOKUP (#8984.4) file.
	XTLKER(2,CODE):	The code is *not* in the associated file.
	XTLKER(3,SYNONYM):	The keyword could *not* be added.

27.6.9 L^XTLKMGR(): Define a File in the Local Lookup File

Reference Type:	Supported
Category:	Toolkit—Multi-Term Look-Up (MTLU)
ICR #:	10153
Description:	The L^XTLKMGR API defines a file in the LOCAL LOOKUP file (8984.4). Adding the target file here does *not* automatically place the special lookup routine, **^XTLKDICL**, in the file's Data Dictionary. Since use of this routine is at the discretion of the developer, it should be manually added via the Edit File option under VA FileMan's Utilities Menu.

REF: For information on the **Edit File** option, see the "Utility Functions" section in the *VA FileMan User Manual*.

Format: `L^XTLKMGR(xtlk1[,xtlk2],xtlk3,xtlk4)`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Parameters:	**xtlk1**:	(required) File name or number.
	xtlk2:	(optional) Application-specific display protocol.
	xtlk3:	(required) **MTLU** index to use for lookups.
	xtlk4:	(required) Variable pointer prefix.
Output Variables:	**XTLKER(1,FILENAME):**	File could *not* be added.

The following are examples (index and prefix can differ from actual implementation):

- For the ICD DIAGNOSIS (#80) file:
```
>K XTLKER
>D L^XTLKMGR(80,"DSPLYD^XTLKKWLD","AIHS","D")
```

- For the ICD OPERATION/PROCEDURE (#80.1) file:
```
>K XTLKER
>D L^XTLKMGR(80.1,"DSPLYO^XTLKKWLD","KWIC","O")
```

27.6.10 LKUP^XTLKMGR(): General Lookup Facility for MTLU

Reference Type:	Supported
Category:	Toolkit—Multi-Term Look-Up (MTLU)
ICR #:	10153
Description:	The LKUP^XTLKMGR API adds terms and synonyms to the LOCAL SYNONYM (#8984.3) file.
Format:	`LKUP^XTLKMGR(fil,xtlkx[,xtlksay][,xtlkhlp][,xtlkmore])`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Parameters: **fil**: (required) Target file (*must* be defined in the LOCAL LOOKUP (#8984.4) file.

xtlkx: (required) Word or phrase to use in lookup.

xtlksay: (optional) Set to any of the following:

- **1 (default)** or **""**—Full screen (normal) display.
- **-1**—Prevents screen display.
- **0**—Minimize screen display.

NOTE: The purpose of **xtlksay** is to control the degree of output to the screen, *not* the amount of "file information" displayed.

If screen displays are turned off, MTLU matches can be processed by checking the count in **^TMP("XTLKHITS",$J)**.

^TMP("XTLKHITS",$J,count)=IEN of the entry in the target file. **^TMP("XTLKHITS")** should be killed after processing.

	xtlkhlp:	(optional) The lookup was successful.
	xtlkmore:	(optional) Set to **1** to continue with VA FileMan search (**default=1**).
Output Variables:	**Y=-1:**	File *not* defined in the LOCAL LOOKUP (#8984.4) file.
	Y=N^S:	**N** is the internal entry number (IEN) of the entry in the file and **S** is the value of the **.01** field for that entry.
	Y=N^S^1:	**N** and **S** are defined as above and the **1** indicates that this entry has just been added to the file.

27.6.10.1 Examples

27.6.10.1.1 Example 1

Figure 246: LKUP^XTLKMGR API—Example 1: Standard Lookup; Single Term Entered

```
VAH,MTL>D LKUP^XTLKMGR(80,"MALIG")
( MALIG/MALIGNANT )
.........................................................................
.........................................................................
..........................................

The following 443 matches were found:

   1: 140.1 (MAL NEO LOWER VERMILION)
       MALIGNANT NEOPLASM OF LOWER LIP, VERMILION BORDER

   2: 140.3 (MAL NEO UPPER LIP, INNER)
       MALIGNANT NEOPLASM OF UPPER LIP, INNER ASPECT

   3: 140.4 (MAL NEO LOWER LIP, INNER)
       MALIGNANT NEOPLASM OF LOWER LIP, INNER ASPECT

   4: 140.5 (MAL NEO LIP, INNER NOS)
       MALIGNANT NEOPLASM OF LIP, UNSPECIFIED, INNER ASPECT

   5: 140.6 (MAL NEO LIP, COMMISSURE)
       MALIGNANT NEOPLASM OF COMMISSURE OF LIP

Press <RET> or Select 1-5: ^
...Nothing selected. Attempting Fileman lookup.
```

NOTE: Pressing the **<Enter>** key continues listing the MTLU matches. If no selection is made, MTLU initiates a standard VA FileMan lookup (using all available cross-references).

27.6.10.1.2 Example 2

Figure 247: LKUP^XTLKMGR API—Example 2: Standard Lookup; Multiple Terms Entered

```
VAH,MTL>D LKUP^XTLKMGR(80,"MALIGNANCY OF THE LIP")


(LIP/LIPIDOSES/LIPODYSTROPHY/LIPOID/LIPOMA/LIPOPROTEIN/LIPOTROPIC/LIPS
MALIGNAN/MALIGNANT)

The following words were not used in this search:
     OF
     THE
............

The following 12 matches were found:

   1: 140.1 (MAL NEO LOWER VERMILION)
       MALIGNANT NEOPLASM OF LOWER LIP, VERMILION BORDER

   2: 140.3 (MAL NEO UPPER LIP, INNER)
       MALIGNANT NEOPLASM OF UPPER LIP, INNER ASPECT

   3: 140.4 (MAL NEO LOWER LIP, INNER)
       MALIGNANT NEOPLASM OF LOWER LIP, INNER ASPECT

   4: 140.5 (MAL NEO LIP, INNER NOS)
       MALIGNANT NEOPLASM OF LIP, UNSPECIFIED, INNER ASPECT

   5: 140.6 (MAL NEO LIP, COMMISSURE)
       MALIGNANT NEOPLASM OF COMMISSURE OF LIP


Press <RET> or Select 1-5: ^
...Nothing selected. Attempting Fileman lookup. ??
```

27.6.10.1.3 Example 3

Figure 248: LKUP^XTLKMGR API—Example 3: Display Minimized by Setting the 3rd Parameter = 0

```
VAH,MTL>S XTLKX="MALIGNANCY OF THE LIP"

VAH,MTL>D LKUP^XTLKMGR(80,XTLKX,0)

The following 12 matches were found:

   1: 140.1 (MAL NEO LOWER VERMILION)
       MALIGNANT NEOPLASM OF LOWER LIP, VERMILION BORDER

   2: 140.3 (MAL NEO UPPER LIP, INNER)
       MALIGNANT NEOPLASM OF UPPER LIP, INNER ASPECT

   3: 140.4 (MAL NEO LOWER LIP, INNER)
       MALIGNANT NEOPLASM OF LOWER LIP, INNER ASPECT

   4: 140.5 (MAL NEO LIP, INNER NOS)
       MALIGNANT NEOPLASM OF LIP, UNSPECIFIED, INNER ASPECT

   5: 140.6 (MAL NEO LIP, COMMISSURE)
       MALIGNANT NEOPLASM OF COMMISSURE OF LIP


Press <RET> or Select 1-5: ^ <Enter> ??
VAH,MTL>
```

27.6.10.1.4 Example 4

Figure 249: LKUP^XTLKMGR API—Example 4: MTLU with Screen Display Turned Off

```
VAH,MTL>D LKUP^XTLKMGR(80,XTLKX,-1)


VAH,MTL>D ^%G

Global ^TMP("XTLKHITS",$J
        TMP("XTLKHITS",$J
^TMP("XTLKHITS",591795907) = 12
^TMP("XTLKHITS",591795907,1)  = 167

^TMP("XTLKHITS",591795907,2)  = 168
^TMP("XTLKHITS",591795907,3)  = 169
^TMP("XTLKHITS",591795907,4)  = 170
^TMP("XTLKHITS",591795907,5)  = 171
^TMP("XTLKHITS",591795907,6)  = 172
^TMP("XTLKHITS",591795907,7)  = 173
^TMP("XTLKHITS",591795907,8)  = 220
^TMP("XTLKHITS",591795907,9)  = 221
^TMP("XTLKHITS",591795907,10)  = 8595
^TMP("XTLKHITS",591795907,11)  = 8623
^TMP("XTLKHITS",591795907,12)  = 8624
```

27.6.11 SH^XTLKMGR(): Add Shortcuts to the Local Shortcut File

Reference Type:	Supported
Category:	Toolkit—Multi-Term Look-Up (MTLU)
ICR #:	10153
Description:	The SH^XTLKMGR API adds Shortcuts to the LOCAL SHORTCUT (#8984.2) file.
Format:	`SH^XTLKMGR(xtlk1,xtlk2,xtlk3)`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Parameters:	**xtlk1**:	(required) Associated file.
	xtlk2:	(required) Code in the associated file.
	xtlk3:	(required) Shortcut (word or phrase).
Output Variables:	**XTLKER(1,FILENAME):**	File *not* defined in the LOCAL LOOKUP (#8984.4) file.
	XTLKER(2,CODE):	The code is *not* in the associated file.
	XTLKER(3,SHORTCUT):	The shortcut could *not* be added.

27.6.12 SY^XTLKMGR(): Add Terms and Synonyms to the Local Synonym File

Reference Type:	Supported
Category:	Toolkit—Multi-Term Look-Up (MTLU)
ICR #:	10153
Description:	The SY^XTLKMGR API adds Terms and Synonyms to the LOCAL SYNONYM (#8984.3) file.
Format:	`SY^XTLKMGR(xtlk1,xtlk2,xtlk3)`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Parameters:	**xtlk1**:	(required) Associated file.
	xtlk2:	(required) Term.
	xtlk3:	(required) Synonym (or optional array for multiple synonyms per term).

NOTE: Use one-dimensional arrays wherever supported in ^XTLKMGR as in the following example:

```
SYN(1)=<first synonym>
SYN(2)=<second synonym>
SYN(3)=<third synonym>
>D SY^ROUTINE(XTLK1,XTLK2,.SYN)
```

Output Variables: **XTLKER(1,FILENAME):** File *not* defined in the LOCAL LOOKUP (#8984.4) file.

XTLKER(2,TERM): The term could *not* be added.

XTLKER(3,SYNONYM): The synonym could *not* be added.

27.7 Toolkit—M Unit Utility

27.7.1 Overview

M Unit is a utility (tool) that permits a series of tests to be written to address specific tags or entry points within a project and act to verify that the return results are as expected for that code. Kernel Toolkit patch XT*7.3*81 provides the M Unit code, but was never released to production. It is available to M developers upon request from the Kernel development team.

If run routinely any time that the project is modified, the tests indicate whether the intended function has been modified inadvertently, or whether the modification has had unexpected effects on other functionality within the project. The set of unit tests for a project should run rapidly (usually within a matter of seconds) and with minimal disruption for developers. Another function of unit tests is that they indicate what the intended software was written to do. This can be especially useful when new developers start working with the software or a programmer returns to a project after a prolonged period. Ensuring that well-designed unit tests are created for each project; therefore, it does the following:

- Assists development.
- Enhances maintainability.
- Improves end-user confidence in the deployed software.

NOTE: None of the Application Programming Interfaces (APIs), extrinsic functions, or sections of code in the M Unit are callable from outside a unit test, but are all part of a unit test. M UNIT is a self-contained application.

27.7.2 Introduction to M Unit Testing

A Unit Test framework permits small tests to be written to verify that the code under examination is doing what you expect it to do. Generally, the tests are performed on the smaller blocks of the application, and do *not* necessarily test all of the functionality within the application. These tests can be run frequently to validate that no errors have been introduced subsequently as changes are made in the code. The phrase "Test-Driven Development" is frequently used to indicate the strong use of unit testing during development; although, some think of it as equivalent to "Test First Development", in which the tests for code are written prior to writing the code. In "Test First Development", the test should initially fail (since nothing has been written) and then pass after the code has been written.

For client side languages, Junit (for Java), DUnit (for Delphi), NUnit and HarnessIt (for dotNet) all provide Unit Test frameworks. The ^XTMUNIT and ^XTMUNIT1 routines provide the same capabilities for unit testing M code. The tests are console-based (i.e., command line text, *not* windows).

For those who have problems keeping track of routine names for unit testing and with which application they are associated, the MUNIT TEST GROUP (#8992.8) file can be used to maintain groups of unit test routines with the **MUnit Test Group edit** [XTMUNIT GROUP EDIT] option. These unit tests can be run using either of the following:

- Menu Option: **Run MUnit Tests from Test Groups** [XTMUNIT GROUP RUN] option.
- Direct Mode Utility: **D RUNSET^XTMUNIT(setname)**.

While the order of processing within M Unit tests can be fairly constant, or at least appear to be so, it is preferable to have the unit tests independent of the order in which they are run. Having dependencies between tests can result in problems if the order were to change or if changes are made in the test being depended upon.

27.7.3 M Unit Test Definitions

Supported References in ^XTMUNIT are:

- EN^XTMUNIT(): Run Unit Tests
- CHKEQ^XTMUNIT: Check Two Values for Equivalence
- CHKLEAKS^XTMUNIT(): Check for Variable Leaks
- CHKTF^XTMUNIT(): Test Conditional Values
- FAIL^XTMUNIT(): Generate an Error Message
- $$ISUTEST^XTMUNIT: Evaluate if Unit Test is Running
- RUNSET^XTMUNIT (Direct Mode Utility)
- SUCCEED^XTMUNIT: Increment Test Counter

27.7.4 Getting Started

If you are going to modify sections of your code, it is best to create a unit test for those areas that you want to work. Then, the unit tests can be run as changes are made to ensure that nothing unexpected has changed. For modifications, the unit tests are then written to reflect the new expected behavior and used to ensure that it is what is expected.

A sample unit test can be found in the ^XTMZZUT1 routine.

27.7.5 Application Programming Interfaces (APIs)

NOTE: None of the Application Programming Interfaces (APIs), extrinsic functions, or sections of code in the M Unit are callable from outside a unit test, but are all part of a unit test. M UNIT is a self-contained application.

27.7.6 EN^XTMUNIT(): Run Unit Tests

Reference Type: N/A; Not callable from outside a unit test.

Category: Toolkit—M Unit Utility

ICR #: N/A

Description: The EN^XTMUNIT API runs unit tests. It is typically the first command within a suite of unit test routines, so that the entire suite of tests (multiple routines) can be run by executing the first routine of the suite. For example, the unit tests for testing M Unit can be run by:

```
>D ^XTMZZUT1"
```

The EN^XTMUNIT API starts the unit testing process.

Format:

```
D EN^XTMUNIT(rouname,[verbose,][break])
```

Input Parameters:

rouname: (required) provides the name of the routine where the testing should be started. That routine *must* have at least one test entry point (and possibly more) either specified as follows:

- In the lines immediately following the **XTENT** tag as the third semi-colon piece on the line.

  Or:

- It can have tags with **@TEST** as the first text of the comment for the tag line.

verbose: (optional) If it evaluates to **True** (e.g., **1**), it turns on verbose mode, which lists each individual test being run as well as its result.

break: (optional) If it evaluates to **True**, it causes the M Unit test process to terminate upon a failure or error instead of continuing until all tests have been evaluated.

Output:	returns:	Results of the unit tests.

The following sections of code in the XTMUNIT routine are additional test entry points added by the developer; however, they are *not* callable by the developer from inside or outside of the routine:

- STARTUP
- SHUTDOWN
- SETUP
- TEARDOWN
- XTENT: List Unit Test Entry Points
- XTROU: List of Routines Containing Additional Tests

27.7.6.1 STARTUP

This section of code in the **XTMUNIT** routine runs *before* anything else. It is useful for setting up an environment or variable values that are common to all of the tests.

27.7.6.2 SHUTDOWN

This section of code in the **XTMUNIT** routine runs *after* everything else. It is useful for shutting down an environment or clearing variable values that are common to all of the tests. It can also be used for cleaning up global or file entries that are left as a result of testing.

27.7.6.3 SETUP

This section of code in the **XTMUNIT** routine runs *before* every test. It is useful for resetting an environment or variable values that are used by the tests.

27.7.6.4 TEARDOWN

This section of code in the **XTMUNIT** routine runs *after* every test. It is useful for cleaning up an environment or variable values that are used by the tests.

27.7.6.5 XTENT: List Unit Test Entry Points

This section of code in the **XTMUNIT** routine is used to store information required by the EN^XTMUNIT API to run a unit test. It provides a list of unit test entry points. Each entry describes a group of tests.

Figure 250: XTENT: List Unit Test Entry Points

```
;;T4;Entry point using XTMENT
;;T5;Error count check
```

27.7.6.6 XTROU: List of Routines Containing Additional Tests

This section of code in the **XTMUNIT** routine is used to store information required by the EN^XTMUNIT API to run a unit test. It provides a list of routines containing additional tests. It extends a suite of tests beyond the limits of a single routine.

Figure 251: XTROU: List of Routines Containing Additional Tests

```
;;XTMZZUT2;
;;XTMZZUT3;
```

27.7.7 CHKEQ^XTMUNIT: Check Two Values for Equivalence

Reference Type:	Not callable from outside a unit test.	
Category:	Toolkit—M Unit Utility	
ICR #:	N/A	
Description:	The CHKEQ^XTMUNIT API runs a test that checks two values for equivalence.	
Format:	`D CHKEQ^XTMUNIT(expect,actual,msg)`	
Input Parameters:	**expect**:	(required) The expected value.
	actual:	(required) The actual value.
	msg:	(required) The error message to be generated if the result of the test is **False** (*not* equal).
Output:	returns:	Returns: • **A period or "dot"**—If the result of the test is **True**. • **The <*expected value*>, the <*actual value*>, and the error message "msg"**—If the result of the test is **False**.

27.7.8 CHKLEAKS^XTMUNIT(): Check for Variable Leaks

Reference Type: Not callable from outside a unit test

Category: Toolkit—M Unit Utility

ICR #: N/A

Description: The CHKLEAKS^XTMUNIT API runs a test that can be used within:

- Unit tests.
- Standalone test for variable leaks; those created within called code that are allowed to leak into the calling environment, unintentionally.

Format:

```
D CHKLEAKS^XTMUNIT(code,testloc,.nameinpt)
```

Input Parameters:

code: (required) Contains a command to be executed in the test for leaks. For example:

```
S X=$$NOW^XLFDT()
```

testloc: (required) Indicates the location under test. For example:

```
$$NOW^XLFDT() leak test
```

Or simply:

```
$$NOW^XLFDT
```

.nameinpt: (required) This parameter is passed by reference, and is an array that contains a list of all variables that the user is passing in and/or expects to be present when the code is finished. The variable **X** would be in the latter category, since it would then be present. The input is in the form of an array:

```
NAMEINPT("VARNAME")="VARVALUE"
```

Where:

- **VARNAME**—Name of a variable.
- **VARVALUE**—Value that is to be assigned to the variable *before* the contents of the **code** input parameter is to be executed.

Output:	returns:	Returns: • **Inside a unit test environment**—When run in a unit test environment, variables that are present after the contents of the **code** input parameter is executed that were *not* included in **NAMEINPT** array as variables, are listed as failures. • **Outside a unit test environment**—When called outside of a unit test environment; any leaked variables are listed on the current device.

27.7.9 CHKTF^XTMUNIT(): Test Conditional Values

Reference Type: Not callable from outside a unit test.

Category: Toolkit—M Unit Utility

ICR #: N/A

Description: The CHKTF^XTMUNIT API runs a test that checks conditional values (**True** or **False**).

Format: `D CHKTF^XTMUNIT(val,msg)`

Input Parameters:	**val**:	(required) The conditional value to be tested.
	msg:	(required) The error message to be generated if the result of the test is **False**.
Output:	returns:	Returns: • **A period or "dot"**—If the result of the test is **True**. • **An error message**—If the result of the test is **False**.

27.7.10 FAIL^XTMUNIT(): Generate an Error Message

Reference Type: Not callable from outside a unit test

Category: Toolkit—M Unit Utility

ICR #: N/A

Description: The FAIL^XTMUNIT API runs a test that simply generates an error message This command is useful for more complex unit tests that are built within the unit test routine itself.

Format: `D FAIL^XTMUNIT(msg)`

Input Parameters:	**msg**:	(required) The text of the error message.
Output:	returns:	Returns the error message.

27.7.11 $$ISUTEST^XTMUNIT: Evaluate if Unit Test is Running

Reference Type: Not callable from outside a unit test

Category: Toolkit—M Unit Utility

ICR #: N/A

Description: The $$ISUTEST^XTMUNIT extrinsic function is used to evaluate if a unit test is currently running. If a test is running, it returns a value of **1**; otherwise, it returns a value of **zero** (**0**). This can be used to select code to be run based on whether it is currently being tested (or something else that calls it is being tested).

Format:

```
S X=$$ISUTEST^XTMUNIT
```

Input Parameters: none.

Output: returns: Returns:

- **1**—If a test is running.
- **Zero (0)**—If a test is *not* running.

27.7.12 SUCCEED^XTMUNIT: Increment Test Counter

Reference Type: Not callable from outside a unit test

Category: Toolkit—M Unit Utility

ICR #: N/A

Description: The SUCCEED^XTMUNIT API runs a test command that increments the test counter; writes a "**dot**" to the screen for activity, which indicates a successful test. This command is useful for indicating a successful test within a more complex unit test built within the unit test routine itself, and is the counterpart to the FAIL^XTMUNIT(): Generate an Error Message API.

Format:

```
D SUCCEED^XTMUNIT
```

Input Parameters: none.

Output:	returns:	Increments test counter; writes a period or "**dot**" to the screen for activity, which indicates a successful test.

27.7.13 Sample M Unit Utility Output

Figure 252 is an example of the output from running a suite of unit tests to test M Unit:

Figure 252: Sample Output from the M Unit Test Tool—Verbose

```
VISTA>D ^XTMZZUT1

T1 - - Make sure Start-up Ran.-------------------------------------  [OK]
T2 - - Make sure Set-up runs.--------------------------------------  [OK]
T3 - - Make sure Teardown runs.------------------------------------  [OK]
T4 - Entry point using XTMENT--------------------------------------  [OK]
T5 - Error count check
T5^XTMZZUT1 - Error count check - This is an intentional failure.
.
T5^XTMZZUT1 - Error count check - Intentionally throwing a failure
.-----------------------------------------------------------------  [FAIL]
T6 - Succeed Entry Point...----------------------------------------  [OK]
T7 - Make sure we write to principal even though we are on another
device[OK]
T8 - If IO starts with another device, write to that device as if it's the
principal device---------------------------------------------------  [OK]
T11 - An @TEST Entry point in Another Routine invoked through XTROU
offsets.[OK]
T12 - An XTENT offset entry point in Another Routine invoked through XTROU
offse
ts.----------------------------------------------------------------  [OK]
MAIN - - Test coverage calculations--------------------------------  [OK]
NEWSTYLE - identify new style test indicator functionality.---------  [OK]
OLDSTYLE -  identify old style test indicator functionality..-------  [OK]
OLDSTYL1 -  identify old style test indicator 2.--------------------  [OK]
BADCHKEQ -  CHKEQ should fail on unequal value
BADCHKEQ^XTMZZUT5 -  CHKEQ should fail on unequal value - <4> vs <3> - SET
UNEQUAL ON PURPOSE - SHOULD FAIL----------------------------------  [FAIL]
BADCHKTF -  CHKTF should fail on false value
BADCHKTF^XTMZZUT5 -  CHKTF should fail on false value - SET FALSE (0) ON
PURPOSE - SHOULD FAIL---------------------------------------------  [FAIL]
BADERROR -  throws an error on purpose
BADERROR^XTMZZUT5 -  throws an error on purpose - Error:
<UNDEFINED>BADERROR+6^X
TMZZUT5 *Q
------------------------------------------------------------------  [FAIL]
CALLFAIL -  called FAIL to test it
CALLFAIL^XTMZZUT5 -  called FAIL to test it - Called FAIL to test it
------------------------------------------------------------------  [FAIL]
LEAKSOK - check leaks should be ok---------------------------------  [OK]
LEAKSBAD - check leaks with leak
LEAKSBAD^XTMZZUT5 - check leaks with leak - LEAKSBAD TEST - X NOT
SPECIFIED VARIABLE LEAK: X----------------------------------------  [FAIL]
NVLDARG1 - check invalid arg in CHKEQ
NVLDARG1^XTMZZUT5 - check invalid arg in CHKEQ - NO VALUES INPUT TO
CHKEQ^XTU - no evaluation possible--------------------------------  [FAIL]
ISUTEST - check ISUTEST inside unit test.--------------------------  [OK]
CHKCMDLN - check command line processing of XTMZZUT5---------------  [OK]
CHKGUI - check GUI processing of XTMZZUT5--------------------------  [OK]
```

```
CKGUISET - check list of tests returned by GUISET------------------  [OK]
NEWSTYLE - test return of valid new style or @TEST indicators...----  [OK]

Ran 5 Routines, 26 Entry Tags
Checked 25 tests, with 7 failures and encountered 1 error.
```

27.8 Toolkit—Parameter Tools

27.8.1 Overview

Parameter Tools is a generic method of handling parameter definitions, assignments, and retrieval. A parameter may be defined for various entities where an entity is the level at which you want to allow the parameter defined (e.g., package level, system level, division level, location level, user level, etc.). A developer can then determine in which order the values assigned to given entities are interpreted.

REF: Integration Control Registration (ICR) #2263 defines the various callable entry points in the XPAR routine.

ICR #2336 defines the various callable entry points in the XPAREDIT routine.

REF: For more information on parameter tools and files, see the "Parameter Tools" section in the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide*.

27.8.2 Definitions

The following are some basic definitions used by Parameter Tools:

- Entity
- Parameter
- Instance
- Value
- Parameter Template

27.8.2.1 Entity

An entity is a level at which you can define a parameter. The entities allowed are stored in the PARAMETER ENTITY (#8989.518) file. Table 34 lists the allowable entities at the time this utility was released:

Table 34: Parameter Tool—Parameter Entity Levels

Entity Prefix	Message	Points to File
PKG	Package	PACKAGE (#9.4)
SYS	System	DOMAIN (#4.2)
DIV	Division	INSTITUTION (#4)

Entity Prefix	Message	Points to File
SRV	Service	SERVICE/SECTION (#49)
LOC	Location	HOSPITAL LOCATION (#44)
TEA	Team	TEAM (#404.51)
CLS	Class	USR CLASS (#8930)
USR	User	NEW PERSON (#200)
BED	Room-Bed	ROOM-BED (#405.4)
OTL	Team (OE/RR)	OE/RR LIST (#100.21)
DEV	Device	DEVICE (#3.5)

NOTE: Entries are maintained via Kernel Toolkit patches. Entries existing in the file at the time it is referenced are considered supported.

27.8.2.2 Parameter

A parameter is the actual name under which values are stored. The name of the parameter *must* be namespaced and it *must* be unique. Parameters can be defined to store the typical package parameter data (e.g., the default add order screen in OE/RR), but they can also be used to store GUI application screen settings a user has selected (e.g., font or window width). When a parameter is defined, the entities that can set that parameter are also defined. The definition of parameters is stored in the PARAMETER DEFINITION (#8989.51) file.

27.8.2.3 Instance

Most parameters set instance to 1. Instances are used when more than one value may be assigned to a given entity/parameter combination. An example of this would be lab collection times at a division. A single division may have multiple collection times. Each collection time would be assigned a unique instance.

27.8.2.4 Value

A value may be assigned to every parameter for the entities allowed in the parameter definition. Values are stored in the PARAMETERS (#8989.5) file.

27.8.2.5 Parameter Template

A parameter template is similar to an input template. It contains a list of parameters that can be entered through an input session (e.g., option). Templates are stored in the PARAMETER TEMPLATE (#8989.52) File. Entries in this file *must* also be namespaced.

27.8.3 Application Programming Interfaces (APIs)

It's *not* possible to directly add, edit, or delete entries in in the PARAMETERS [#8989.5] file . The only way to do this is programmatically through the APIs described in this section.

27.8.4 ADD^XPAR(): Add Parameter Value

Reference Type:	Supported
Category:	Toolkit—Parameter Tools
ICR #:	2263
Description:	The ADD^XPAR API adds a new parameter value as an entry to the PARAMETERS (#8989.5) file if the Entity/Parameter/Instance combination does *not* already exist.

REF: For descriptive information about the elements and how they are used in the callable entry points into XPAR, see the "Definitions" section.

Format: `ADD^XPAR(entity,parameter[,instance],value[,.error])`

Input / Output Parameters	See EN^XPAR	For the definition of the input and output parameters used in this API, see the EN^XPAR(): Add, Change, Delete Parameters API.

27.8.4.1 Example

Figure 253: ADD^XPAR API—Example

```
>D ADD^XPAR("PKG.KERNEL","XPAR TEST FREE TEXT",,"Today Good",.ERROR)
```

27.8.5 CHG^XPAR(): Change Parameter Value

Reference Type:	Supported
Category:	Toolkit—Parameter Tools
ICR #:	2263
Description:	The CHG^XPAR API changes the value assigned to an existing parameter if the Entity/Parameter/Instance combination already exists.

REF: For descriptive information about the elements and how they are used in the callable entry points into XPAR, see the "Definitions" section.

Format: `CHG^XPAR(entity,parameter[,instance],value[,.error])`

Input / Output Parameters:

See EN^XPAR	For the definition of the input and output parameters used in this API, see the EN^XPAR(): Add, Change, Delete Parameters API.

27.8.5.1 Example

Figure 254: CHG^XPAR API—Example

```
>D CHG^XPAR("PKG.KERNEL","XPAR TEST FREE TEXT",,"Tomorrow Hot",.ERROR)
```

27.8.6 DEL^XPAR(): Delete Parameter Value

Reference Type:	Supported
Category:	Toolkit—Parameter Tools
ICR #:	2263
Description:	The DEL^XPAR API deletes an existing parameter instance if the value assigned is an at-sign (@).

REF: For descriptive information about the elements and how they are used in the callable entry points into XPAR, see the "Definitions" section.

Format: `DEL^XPAR(entity,parameter[,instance][,.error])`

Input / Output

Parameters:	See EN^XPAR	For the definition of the input and output parameters used in this API, see the EN^XPAR(): Add, Change, Delete Parameters API.

27.8.6.1 Example

Figure 255: DEL^XPAR API—Example

```
>D DEL^XPAR("PKG.KERNEL","XPAR TEST FREE TEXT",),.ERROR) I ERROR>0 W !.ERROR
```

27.8.7 EN^XPAR(): Add, Change, Delete Parameters

Reference Type: Supported

Category: Toolkit—Parameter Tools

ICR #: 2263

Description: The EN^XPAR API performs any one of the following functions:

- Adds the value as a new entry to the PARAMETERS (#8989.5) file if the **Entity|Parameter|Instance** combination does *not* already exist.
- Changes the value assigned to the parameter in the PARAMETERS (#8989.5) file if the **Entity|Parameter|Instance** combination already exists.
- Deletes the parameter instance in the PARAMETERS (#8989.5) file if the value assigned is @.

REF: For descriptive information about the elements and how they are used in the callable entry points into **XPAR**, see the "Definitions" section.

Format: `EN^XPAR(entity,parameter[,instance],value[,.error])`

Input Parameters: **entity**:

(required) Entity can be set to the following:

- Internal VARIABLE POINTER (**nnn;GLO(123,**).
- External format of the VARIABLE POINTER using the three-character prefix (**prefix.entryname**).
- Prefix alone to set the parameter based on the current entity selected.

This works for the following entities:

- **USR**—Uses current value of **DUZ**.
- **DIV**—Uses current value of **DUZ(2)**.
- **SYS**—Uses system (domain).
- **PKG**—Uses the package to which the parameter belongs.

parameter:

(required) Can be passed in external or internal format. Identifies the name or internal entry number (IEN) of the parameter as defined in the PARAMETER DEFINITION (#8989.51) file.

instance:

(optional) Defaults to **1** if *not* passed. Can be passed in external or internal format. Internal format requires that the value be preceded by the grave accent (`) character.

value:

(required) Can be passed in external or internal format. If using internal format for a POINTER type parameter, the **value** *must* be preceded by the grave accent (`) character.

If the **value** is being assigned to a WORD-PROCESSING parameter, the text can be passed in the subordinate nodes of **Value** [e.g., **Value(1,0)=Text**] and the variable "**Value**" itself can be defined as a title or description of the text.

Output Parameter: **.error**:

(optional) If used, *must* be passed in by reference. It returns any error condition that may occur:

- **0 (Zero)**—If no error occurs.
- **#^*errortext***—If an error does occur.

The # is the number in the VA FileMan DIALOG (#.84) file and the "***errortext***" describes the error.

27.8.7.1 Example

Figure 256: EN^XPAR API—Example

```
>D EN^XPAR("SYS","XPAR TEST FREE TEXT",0,"Good times",.ERROR)
>D EN^XPAR("SYS","XPAR TEST FREE TEXT",1,"to night",.ERROR)
```

27.8.8 ENVAL^XPAR(): Return All Parameter Instances

Reference Type:	Supported
Category:	Toolkit—Parameter Tools
ICR #:	2263
Description:	The ENVAL^XPAR API returns all parameter instances.

REF: For descriptive information about the elements and how they are used in the callable entry points into XPAR, see the "Definitions" section.

Format: `ENVAL^XPAR(.list,parameter,instance[,.error][,gbl])`

Input / Output Parameters:

.list: (required) If the **gbl** parameter is set to **1**, then the **.list** parameter becomes an input and holds the closed root of a global where the GETLST^XPAR(): Return All Instances of a Parameter API should put the output. For example:

```
$NA(^TMP($J,"XPAR"))
```

Input Parameters:	**parameter**:	(required) For a description of this parameter, see the EN^XPAR(): Add, Change, Delete Parameters API.
	instance:	(required) For a description of this parameter, see the EN^XPAR(): Add, Change, Delete Parameters API.
	gbl:	(optional) If this optional parameter is set to **1**, then the **.list** parameter *must* be set before the call to the closed global root where the return data should be put. For example: `S LIST=$NA(^TMP($J))` `ENVAL^XPAR(LIST,par,inst,.error,1`
Output Parameters:	**.error**:	(optional) For a description of this parameter, see the EN^XPAR(): Add, Change, Delete Parameters API.

27.8.9 $$GET^XPAR(): Return an Instance of a Parameter

Reference Type: Supported

Category: Toolkit—Parameter Tools

ICR #: 2263

Description: The $$GET^XPAR extrinsic function retrieves the value of a parameter. The value is returned from this call in the format defined by the **format** input parameter.

REF: For descriptive information about the elements and how they are used in the callable entry points into **XPAR**, see the "Definitions" section.

Format:

```
$$GET^XPAR(entity,parameter,instance[,format])
```

Input Parameters:	**entity**:	(required) Entity is defined as the single entity or group of entities you want to look at in order to retrieve the value. Entities may be passed in internal or external format [e.g., LOC.PULMONARY or LOC.'57 or 57;SC(]. The list of entities in this variable may be defined as follows: • A single entity to look at (e.g., LOC.PULMONARY). • The word **ALL** that tells the utility to look for values assigned to the parameter using the entity precedence defined in the PARAMETER DEFINITION (#8989.51) file. • A list of entities you want to search (e.g., "USR^LOC^SYS^PKG"). The list is searched from left to right with the first value found returned. • Items **2** or **3** with specific entity values referenced such as: ○ **ALL^LOC.PULMONARY**—To look at the defined entity precedence, but when looking at location, only look at the PULMONARY location. ○ **USR^LOC.PULMONARY^SYS^PKG**—To look for values for all current user, PULMONARY location, system, or package).
	parameter:	(required) For a description of this parameter, see the EN^XPAR(): Add, Change, Delete Parameters API.
	instance:	(required) For a description of this parameter, see the EN^XPAR(): Add, Change, Delete Parameters API.
	format:	(optional) The **format** input parameter determines how the value is returned. It can be set to the following: • **I**—Internal; returns list(#) = "internal value". • **Q**—Quick; returns list(#) = "internal instance^internal value". Returns the value in the quickest manner (default if *not* specified). • **E**—External; returns list(#) = "external instance^external value".

- **B**—Both; returns both list(#,“N”) = “internal instance^external instance” and list(#,“V”) = “internal value^external value”.
- **N**—Returns list(#) = “internal value^external value”.

Output:	returns:	Returns the parameter value in the format defined by the **format** input parameter.

27.8.10 GETLST^XPAR(): Return All Instances of a Parameter

Reference Type: Supported

Category: Toolkit—Parameter Tools

ICR #: 2263

Description: The GETLST^XPAR API is similar to the ENVAL^XPAR(): Return All Parameter Instances API; however, it returns *all* instances of a parameter.

REF: For descriptive information about the elements and how they are used in the callable entry points into **XPAR**, see the “Definitions” section.

Format: `GETLST^XPAR(.list,entity,parameter[,format][,.error][,gbl])`

Input/ Output Parameters:

.list: (required) The array passed as **.list** is returned with all of the possible values assigned to the parameter.

REF: To see how this data can be returned, see the **format** parameter description.

If the **gbl** parameter is set to **1**, then the **.list** parameter becomes an input and holds the closed root of a global where the GETLST^XPAR API should put the output [i.e., **$NA(^TMP($J,“XPAR”))**].

Input Parameters:

entity:	(required) For a description of this parameter, see the EN^XPAR(): Add, Change, Delete Parameters API.
parameter:	(required) For a description of this parameter, see the EN^XPAR(): Add, Change, Delete Parameters API.
format:	(optional) For a description of this parameter, see the $$GET^XPAR(): Return an Instance of a Parameter API.
gbl:	(optional) If this optional parameter is set to **1**. Then the **.list** parameter *must* be set before the call to the closed global root where the return data should be put. For example: `GETLST^XPAR($NA(^TMP($J)),ent,par,fmt,.error,1)`

Output Parameters: .error: (optional) For a description of this parameter, see the EN^XPAR(): Add, Change, Delete Parameters API.

27.8.10.1 Example

Figure 257: GETLST^XPAR API—Example

```
>D GETLST^XPAR(.LIST,"SYS","XPAR TEST MULTI FREE TEXT",,.ERROR)
```

27.8.11 GETWP^XPAR(): Return Word-Processing Text

Reference Type: Supported

Category: Toolkit—Parameter Tools

ICR #: 2263

Description: The GETWP^XPAR API returns word-processing text in the **returnedtext** parameter. The **returnedtext** parameter itself contains the value field, which is free text that may contain a title, description, etc. The word-processing text is returned in **returnedtext(#,0)**.

REF: For descriptive information about the elements and how they are used in the callable entry points into **XPAR**, see the "Definitions" section.

Format: `GETWP^XPAR(returnedtext,entity,parameter[,instance][,.error])`

Input / Output Parameters:	**.returnedtext**:	(required) This parameter is defined as the name of an array in which you want the text returned. The **.returnedtext** parameter is set to the title, description, etc. The actual word-processing text is returned in **returnedtext(#,0)**. For example: `>returnedtext="Select Notes Help"` `>returnedtext(1,0)="To select a progress note from the list, "` `>returnedtext(2,0)="click on the date/title of the note."`
Input Parameters:	**entity**:	(required) For a description of this parameter, see the EN^XPAR(): Add, Change, Delete Parameters API.
	parameter:	(required) For a description of this parameter, see the EN^XPAR(): Add, Change, Delete Parameters API.
	instance:	(optional) For a description of this parameter, see the EN^XPAR(): Add, Change, Delete Parameters API.
Output Parameters:	**.error**	(optional) For a description of this parameter, see the EN^XPAR(): Add, Change, Delete Parameters API.

27.8.11.1 Example

Figure 258: GETWP^XPAR API—Example

```
>D GETWP^XPAR(.X,"PKG","ORW HELP","lstNotes",.ERROR)
```

27.8.12 NDEL^XPAR(): Delete All Instances of a Parameter

Reference Type: Supported

Category: Toolkit—Parameter Tools

ICR #: 2263

Description: The NDEL^XPAR API deletes the value for all instances of a parameter for a given entity.

REF: For descriptive information about the elements and how they are used in the callable entry points into **XPAR**, see the "Definitions" section.

Forma t:	`NDEL^XPAR(entity,parameter[,.error])`	
Input Parameters:	**entity**:	(required) Entity can be set to the following: • Internal VARIABLE POINTER (**nnn;GLO(123,**). • External format of the VARIABLE POINTER using the three-character prefix (**prefix.entryname**). • Prefix alone to set the parameter based on the current entity selected. This works for the following entities: • **USR**—Uses current value of **DUZ**. • **DIV**—Uses current value of **DUZ(2)**. • **SYS**—Uses system (domain). • **PKG**—Uses the package to which the parameter belongs.
	parameter:	(required) Can be passed in external or internal format. Identifies the name or internal entry number (IEN) of the parameter as defined in the PARAMETER DEFINITION (#8989.51) file.
Output Parameter	**.error**:	(optional) If used, *must* be passed in by reference. It returns any error condition that may occur: • **0 (Zero)**—If no error occurs. • **#^errortext**—If an error does occur. The # is the number in the VA FileMan DIALOG (#.84) file and the "errortext" describes the error.

27.8.12.1 Example

Figure 259: NDEL^XPAR API—Example

```
>D NDEL^XPAR("SYS","XPAR TEST MULTI FREE TEXT",.ERROR)
```

27.8.13 PUT^XPAR(): Add/Update Parameter Instance

Reference Type:	Supported
Category:	Toolkit—Parameter Tools
ICR #:	2263
Description:	The PUT^XPAR API adds or updates a parameter instance and bypass the input transforms.

REF: For descriptive information about the elements and how they are used in the callable entry points into XPAR, see the "Definitions" section.

Format: `PUT^XPAR(entity,parameter[,instance],value[,.error])`

Input / Output Parameters:

See EN^XPAR	For the definition of the input and output parameters used in this API, see the EN^XPAR(): Add, Change, Delete Parameters API.

27.8.13.1 Example

Figure 260: PUT^XPAR API—Example

```
>D PUT^XPAR("SYS","XPAR TEST MULTI FREE TEXT",0,"Good times",.ERROR)
```

27.8.14 REP^XPAR(): Replace Instance Value

Reference Type:	Supported
Category:	Toolkit—Parameter Tools
ICR #:	2263
Description:	The REP^XPAR API replaces the value of an instance with another value.

REF: For descriptive information about the elements and how they are used in the callable entry points into **XPAR**, see the "Definitions" section.

Format: `REP^XPAR(entity,parameter,currentinstance,newinstance[,.error])`

Input Parameters:	**entity**:	(required) For a description of this parameter, see the EN^XPAR(): Add, Change, Delete Parameters API.
	parameter:	(required) For a description of this parameter, see the EN^XPAR(): Add, Change, Delete Parameters API.
	currentinstance:	(required) The instance for which the value is currently defined.
	newinstance:	(required) The instance to which you want to assign the value that is currently assigned to **currentinstance**.
Output Parameters:	**.error**:	(optional) For a description of this parameter, see the EN^XPAR(): Add, Change, Delete Parameters API.

27.8.15 BLDLST^XPAREDIT(): Return All Entities of a Parameter

Reference Type: Supported

Category: Toolkit—Parameter Tools

ICR #: 2336

Description: The BLDLST^XPAREDIT API returns in the array **list** all entities allowed for the input **parameter**.

Format: `BLDLST^XPAREDIT(.list,parameter)`

Input Parameters:	**.list**:	(required) Name of array to receive output.
	parameter:	(required) Internal Entry Number (IEN) of entry in the PARAMETER DEFINITION (#8989.51) file.
Output Parameters:	**.list**:	The array passed as **list** is returned with all of the possible values assigned to the parameter. Data is returned in the following format: `list(ent,inst)=val`

27.8.16 EDIT^XPAREDIT(): Edit Instance and Value of a Parameter

Reference Type: Supported

Category: Toolkit—Parameter Tools

ICR #: 2336

Description: The EDIT^XPAREDIT API interactively edits the instance (if multiple instances are allowed) and the value for a parameter associated with a given entity.

Format: `EDIT^XPAREDIT(entity,parameter)`

Input Parameters:	**entity**:	(required) Identifies the specific entity for which a parameter can be edited. The entity *must* be in VARIABLE POINTER format.
	parameter:	(required) Identifies the parameter that should be edited. Parameter should contain two pieces: `IEN^DisplayNameOfParameter`
Output:	results:	Returns parameter for Interactive edits.

27.8.17 EDITPAR^XPAREDIT(): Edit Single Parameter

Reference Type: Supported

Category: Toolkit—Parameter Tools

ICR #: 2336

Description: The EDITPAR^XPAREDIT API edits a single parameter.

Format: `EDITPAR^XPAREDIT(parameter)`

Input Parameters:	**parameter**:	(required) For a description of this parameter, see the EN^XPAR(): Add, Change, Delete Parameters API.
Output:	returns:	Returns requested parameter.

27.8.18 EN^XPAREDIT: Parameter Edit Prompt

Reference Type: Supported

Category: Toolkit—Parameter Tools

ICR #: 2336

Description: The EN^XPAREDIT API prompts the user for a parameter to edit. This is provided as a tool for developers and is *not intended for exported calls* as it allows editing of *any* parameter.

Format: `EN^XPAREDIT`

Input Parameters: none.

Output: none.

27.8.19 GETENT^XPAREDIT(): Prompt for Entity Based on Parameter

Reference Type:	Supported	
Category:	Toolkit—Parameter Tools	
ICR #:	2336	
Description:	The GETENT^XPAREDIT API interactively prompts for an entity, based on the definition of a parameter.	
Format:	`GETENT^XPAREDIT(.entity,parameter[,.onlyone?])`	
Input Parameters:	**.entity**:	(required) Returns the selected entity in VARIABLE POINTER format.
	parameter:	(required) Identifies the parameter that should be edited. Parameter should contain two pieces: `IEN^DisplayNameOfParameter`
Output:	**.onlyone?**:	(optional) Returns **1** if there is only one possible entity for the value. For example: • **1**—If the parameter can only be set for the system, **onlyone?**. • **0**—If the parameter could be set for any location, **onlyone?**.

27.8.20 GETPAR^XPAREDIT(): Select Parameter Definition File

Reference Type:	Supported
Category:	Toolkit—Parameter Tools
ICR #:	2336
Description:	The GETPAR^XPAREDIT API allows the user to select the PARAMETER DEFINITION (#8989.51) file entry.
Format:	`GETPAR^XPAREDIT(.variable)`

Make sure to perform the following steps before calling this API:

1. **NEW** all *non*-namespaced variables.
2. Set all input variables.
3. Call the API.

Input Parameters:	**.variable**:	(required) The name of the variable where data is returned.
Output Variables:	**.OUTPUTVALU:**	Returns the value **Y** in standard VA FileMan DIC lookup format.

27.8.21 TED^XPAREDIT(): Edit Template Parameters (No Dash Dividers)

Reference Type:	Supported	
Category:	Toolkit—Parameter Tools	
ICR #:	2336	
Description:	The TED^XPAREDIT API allows editing of parameters defined in a template. The parameters in the template are prompted in VA FileMan style—prompt by prompt. No dashed line dividers are displayed between each parameter. Since the dashed line headers are suppressed, it is important to define the VALUE TERM for each parameter in the template, as this is what prompts for the value.	
Format:	`TED^XPAREDIT(template[,reviewflags][,allentities])`	
Input Parameters:	**template**:	(required) The Internal Entry Number (IEN) or NAME of an entry in the PARAMETER TEMPLATE (#8989.52) file.
	reviewflags:	(optional) There are two flags (**A** and **B**) that can be used individually, together, or *not* at all: • **A**—Indicates that the new values for the parameters in the template are displayed *after* the prompting is done. • **B**—Indicates that the current values of the parameters are displayed *before* editing.
	allentities:	(optional) This is a VARIABLE POINTER that should be used as the entity for all parameters in the template. If left blank, prompting for the entity is done as defined in the PARAMETER TEMPLATE (#8989.52) file.
Output:	none.	

27.8.22 TEDH^XPAREDIT(): Edit Template Parameters (with Dash Dividers)

Reference Type:	Supported	
Category:	Toolkit—Parameter Tools	
ICR #:	2336	
Description:	The TEDH^XPAREDIT API is similar to the TED^XPAREDIT(): Edit Template Parameters (No Dash Dividers) API except that the dashed line headers *are* shown between each parameter. It allows editing of parameters defined in a template. The parameters in the template are prompted in VA FileMan style—prompt by prompt.	
Format:	`TEDH^XPAREDIT(template[,reviewflags][,allentities])`	
Input Parameters:	**template**:	(required) For a description of this parameter, see the TED^XPAREDIT(): Edit Template Parameters (No Dash Dividers) API.
	reviewflags:	(optional) For a description of this parameter, see the TED^XPAREDIT(): Edit Template Parameters (No Dash Dividers) API.
	allentities:	(optional) For a description of this parameter, see the TED^XPAREDIT(): Edit Template Parameters (No Dash Dividers) API.
Output:	none.	

27.9 Toolkit—VHA Unique ID (VUID) APIs

27.9.1 GETIREF^XTID(): Get IREF (Term/Concept)

Reference Type: Supported

Category: Toolkit—VHA Unique ID (VUID)

ICR #: 4631

Description: The GETIREF^XTID API searches and returns a list of terms/concepts for a given VHA Unique ID (VUID; i.e., **vuid** input parameter). Filtering of the list is applied when the following optional input parameters are defined:

- **file**
- **field**
- **master**

Format: `GETIREF^XTID([file][,field],vuid,array[,master])`

Input Parameters:

file: (optional) VistA file/subfile number where term/concept is defined.

- **Defined**—If defined, the search is limited to those term/concepts that exist in that file and have the VUID assigned to the **vuid** input parameter.
- **Not Defined**—If *not* defined, the search includes term/concepts that have the VUID assigned to the **vuid** input parameter and can exist in both file terms and in SET OF CODES terms.

field: (optional) Field number, in the **file** input parameter, where term/concept is defined.

- **Defined**—The search finds those terms/concepts that have the VUID assigned to the **vuid** input parameter and is limited to those terms/concepts that exist in the given file/field combination.
  - Entered as **.01**, it represents the terms defined in the file entered in the **file** input parameter.
  - Otherwise, the field number entered *must* be a SET OF CODES data type

field in the file entered in the **file** input parameter.

- **Not Defined**—The search finds those terms/concepts that have the VUID assigned to the **vuid** input parameter and is limited to those terms/concepts found in the file defined in the **file** input parameter.

	vuid:	(required) The VHA Unique ID (VUID) value, which is specified to limit the search.
	array:	(required) The name of the array (local or global) where results of the search is stored.
	master:	(optional) Flag to limit the search of terms based on the value of the MASTER ENTRY FOR VUID field. Returns: • **0**—Include all terms. • **1**—Include only those terms designated as MASTER ENTRY FOR VUID.
Output:	**array**:	Returns the given array populated as follows: • **@TARRAY** = *<list count>* • **@TARRAY@(<file#>,<field#>,<internalreference>)** = *<status info>* Where the *<status info>* is defined as "*<internal value>^<VA FileMan effective date/time>^<external value>^<master entry?>* " • **Empty Array**—Unpopulated array when no entries are found. • **Error Array**—When an error occurs, the array is populated as follows: **@TARRAY("ERROR")**="*<error message>*"

27.9.1.1 Examples

27.9.1.1.1 Example 1

Figure 261: GETIREF^XTID API—Example 1

```
>N array S array="MYARRAY"
>S file=16000009,field=.01,vuid=12343,master=0
>D GETIREF^XTID(file,field,vuid,array,master)
>ZW MYARRAY

MYARRAY=2
MYARRAY(16000009,.01,"1,")=1^3050202.153242^ACTIVE^0
MYARRAY(16000009,.01,"3,")=0^3050215.07584^INACTIVE^1
```

27.9.1.1.2 Example 2

When no entries are found, the named array is populated as shown in Figure 262:

Figure 262: GETIREF^XTID API—Example 2

```
>ZW MYARRAY

MYARRAY=0
```

27.9.1.1.3 Example 3

When an error occurs, the named array is populated as shoen in Figure 263:

Figure 263: GETIREF^XTID API—Example 3

```
>ZW MYARRAY
MYARRAY("ERROR")=<error message>
```

27.9.2 $$GETMASTR^XTID(): Get Master VUID Flag (Term/Concept)

Reference Type:	Supported
Category:	Toolkit—VHA Unique ID (VUID)
ICR #:	4631
Description:	The $$GETMASTR^XTID extrinsic function retrieves the value of the MASTER ENTRY FOR VUID field for a given term/concept reference.
Format:	`$$GETMASTR^XTID(file[,field],iref)`

Input Parameters:	**file**:	(required) VistA file/subfile number where term/concept is defined.
	field:	(optional) Field number in the **file** input parameter where term/concept is defined: • **Not Defined**—If *not* defined, this field defaults to the **.01** field number, and it represents terms defined in the **file** input parameter. • **Defined:** o Entered as **.01**, it represents the terms defined in the file entered in the **file** input parameter. o Otherwise, the field number entered *must* be a SET OF CODES data type field in the file entered in the **file** input parameter.

	iref:	(required) Internal reference for term/concept:
		• **File Entries**—This is an IENS. For example:
		iref="**5,**"
		• **SET OF CODES**—This is the internal value of the code. For example, any of the following:
		iref = **3**
		iref = "**f**"
		iref = "**M**"

Output:	returns:	Returns results of operation as follows: • **Successful**—Internal value of the MASTER ENTRY FOR VUID field as follows: o **0**—NO. o **1**—YES. • **Unsuccessful**—^*<error message>*

27.9.2.1 Examples

27.9.2.1.1 Example 1

For terms defined in fields that are SET OF CODES:

Figure 264: $$GETMASTR^XTID API—Example 1: Terms Defined in Fields that are SET OF CODES

```
>S file=2,field=.02,iref="M"
>W $$GETMASTR^XTID(file,field,iref)
1
```

27.9.2.1.2 Example 2

For terms defined in a single file:

Figure 265: $$GETMASTR^XTID API—Example 2: Terms Defined in a Single File

```
>S file=16000009,field=.01,iref="3,"
>W $$GETMASTR^XTID(file,field,iref)
0
```

27.9.3 $$GETSTAT^XTID(): Get Status Information (Term/Concept)

Reference Type:	Supported
Category:	Toolkit—VHA Unique ID (VUID)
ICR #:	4631
Description:	The $$GETSTAT^XTID extrinsic function retrieves the status information for a given term/concept reference and a specified date/time.
Format:	`$$GETSTAT^XTID(file[,field],iref[,datetime])`

Input Parameters: **file**: (required) VistA file/subfile number where term/concept is defined.

field: (optional) Field number, in the **file** input parameter where term/concept is defined:

- **Not Defined—**If *not* defined, this field defaults to the **.01** field number, and it represents terms defined in the **file** input parameter.
- **Defined:**
    - Entered as **.01**; it represents the terms defined in the file entered in the **file** input parameter.
    - Otherwise, the field number entered *must* be a SET OF CODES data type field in the file entered in the **file** input parameter.

iref: (required) Internal reference for term/concept.

- **File entries—**This is an IENS. For example:
  iref = "**5,**"
- **SETS OF CODES—**This is the internal value of the code. For example, any of the following:
  iref = **3**
  iref = "**f**"
  iref = "**M**"

datetime: (optional) VA FileMan date/time. It defaults to **NOW**.

Output: returns: Returns results of operation as follows:

- **Successful—***<internal value>^<VA FileMan effective date/time>^<external value>*

  For example:

```
0^3050220.115720^INACTIVE
1^3050225.115711^ACTIVE
```

- **Unsuccessful—***^<error message>*

NOTE: The first piece is empty. This differentiates it from the successful case, where the first piece is either **0** or **1**.

27.9.3.1 Examples

27.9.3.1.1 Example 1

For terms defined in fields that are SET OF CODES:

Figure 266: $$GETSTAT^XTID API—Example 1: Terms Defined in Fields that are SET OF CODES

```
>S file=2,field=.02,iref="M",datetime=$$NOW^XLFDT
>W $$GETSTAT^XTID(file,field,iref,datetime)
1^3050121.154752^ACTIVE
```

27.9.3.1.2 Example 2

For terms defined in a single file:

Figure 267: $$GETSTAT^XTID API—Example 2: Terms Defined in a Single File

```
>S file=16000009,field=.01,iref="3,",datetime=""
>W $$GETSTAT^XTID(file,field,iref,datetime)
0^3050122.154755^INACTIVE
```

27.9.4 $$GETVUID^XTID(): Get VUID (Term/Concept)

Reference Type:	Supported
Category:	Toolkit—VHA Unique ID (VUID)
ICR #:	4631
Description:	The $$GETVUID^XTID extrinsic function retrieves the VHA Unique ID (VUID) for a given term/concept reference.
Format:	`$$GETVUID^XTID(file[,field],iref)`

Input Parameters:	**file**:	(required) VistA file/subfile number where term/concept is defined.
	field:	(optional) Field number in the **file** input parameter where term/concept is defined. • **Not Defined**—If *not* defined, this field defaults to the **.01** field number, and it represents terms defined in the file entered in the **file** input parameter. • **Defined:** o Entered as **.01**; it represents the terms defined in the file entered in the **file** input parameter. o Otherwise, the field number entered *must* be a SET OF CODES data type field in the file entered in the **file** input parameter.

	iref:	(required) Internal reference for term/concept:
		• **File Entries**—This is an IENS. For example:
		  iref="**5,**"
		• **SET OF CODES**—This is the internal value of the code. For example, any of the following:
		  iref = **3**
		  iref = "**f**"
		  iref = "**M**"

Output:	returns:	Returns results of operation as follows: • **Successful**—VHA Unique ID (VUID) • **Unsuccessful**—0^<*error message*>

27.9.4.1 Examples

27.9.4.1.1 Example 1

For terms defined in fields that are SET OF CODES:

Figure 268: $$GETVUID^XTID API—Example 1: Terms Defined in Fields that are SET OF CODES

```
>S file=2,field=.02,iref="M"
>W $$GETVUID^XTID(file,field,iref)
123456
```

27.9.4.1.2 Example 2

For terms defined in a single file:

Figure 269: $$GETVUID^XTID API—Example 2: Terms Defined in a Single File

```
>S file=16000009,field=.01,iref="3,"
>W $$GETVUID^XTID(file,field,iref)
123457
```

27.9.5 $$SCREEN^XTID(): Get Screening Condition (Term/Concept)

Reference Type: Supported

Category: Toolkit—VHA Unique ID (VUID)

ICR #: 4631

Description: The $$SCREEN^XTID extrinsic function retrieves the screening condition for a given term/concept reference and specified date/time. It returns whether or *not* a given entry should be screened out of selection lists. This API should *not* be used to determine if the given entry is active/inactive, since the API takes into consideration where in the standardization process the facility is. It returns the following values:

- **0**—If the given entry is selectable (i.e., "do *not* screen it out").
- **1**—If the entry is *not* selectable (i.e., "screen it out").

NOTE: This extrinsic function was released with Kernel Toolkit Patch XT*7.3*108.

Format: `$$SCREEN^XTID(file[,field],iref[,datetime][,.cached])`

Input Parameters: **file**: (required) VistA file/subfile number where term/concept is defined.

field: (optional) Field number, in the **file** input parameter where term/concept is defined.

- ***Not* Defined**—If *not* defined, this field defaults to the **.01** field number, and it represents terms defined in the file entered in the **file** input parameter.
- **Defined:**
    - Entered as **.01**; it represents the terms defined in the file entered in the **file** input parameter.
    - Otherwise, the field number entered *must* be a SET OF CODES data type field in the file entered in the **file** input parameter.

iref: (required) Internal reference for term/concept:

- **File entries**—This is an IENS. For example:

  iref = "**5,**"
- **SET OF CODES**—This is the internal value of the code. For example, any of the following:

  iref = **3**
  iref = "**f**"
  iref = "**M**"

datetime: (optional) VA FileMan date/time against which screening is checked. It defaults to **NOW**.

NOTE: If the value of the **datetime** parameter contains a date and no time, no entries are returned for the first day.

.cached: (optional) Flag to indicate caching. Used mainly when defining the **screen** parameter [e.g., **DIC("S")**] while searching large files. This improves the speed of the search.

NOTE: It *must* be **KILL**ed before initiating each search query (e.g., before calling the VA FileMan ^DIC API).

Output:	returns:	Returns the screening condition as follows: • **0**—When term/concept is selectable (i.e., do *not* screen it out). • **1**—When term/concept is *not* selectable (i.e., screen it out).

27.9.5.1 Examples

27.9.5.1.1 Example 1

For terms defined in fields that are SET OF CODES:

Figure 270: $$SCREEN^XTID API—Example 1: Terms Defined in Fields that are SET OF CODES

```
>S file=2,field=.02,iref="M",datetime=$$NOW^XLFDT
>W $$SCREEN^XTID(file,field,iref,datetime)
0
```

27.9.5.1.2 Example 2

For terms defined in a single file:

Figure 271: $$SCREEN^XTID API—Example 2: Terms Defined in a Single File

```
>S file=16000009,field=.01,iref="3,",datetime=""
>W $$SCREEN^XTID(file,field,iref,datetime)
0
```

27.9.5.1.3 Example 3

When searching a large file:

Figure 272: $$SCREEN^XTID API—Example 3

```
>S file=120.52,field=.01,datetime=""
>S SCREEN="I '$$SCREEN^XTID(file,field,Y_"","",datetime,.cached)"
>. . .
>K cached
>D LIST^DIC(file,,".01;99.99",,"*",,,,SCREEN,,"LIST","MSG")
>K cached
```

27.9.6 $$SETMASTR^XTID(): Set Master VUID Flag (Term/Concept)

Reference Type: Supported

Category: Toolkit—VHA Unique ID (VUID)

ICR #: 4631

Description: The $$SETMASTR^XTID extrinsic function stores (SETs) the value of the MASTER ENTRY FOR VUID field for a given term/concept reference. The MASTER ENTRY FOR VUID field distinguishes references that might be duplicates.

Format: `$$SETMASTR^XTID(file[,field],iref,mstrflag)`

Input Parameters:

file: (required) VistA file/subfile number where term/concept is defined.

field: (optional) Field number in the **file** input parameter where term/concept is defined.

- ***Not* Defined**—If *not* defined, this field defaults to the **.01** field number. It represents the terms defined in the file entered in the **file** input parameter.
- **Defined:**
  - Entered as **.01**; it represents the terms defined in the file entered in the **file** input parameter.
  - Otherwise, the field number entered *must* be a SET OF CODES data type field in the file entered in the **file** input parameter.

iref: (required) Internal reference for term/concept:

- **File Entries**—This is an IENS. For example:
  iref="**5,**"

- **SET OF CODES**—This is the internal value of the code. For example, any of the following:
  iref = **3**
  iref = "**f**"
  iref = "**M**"

mstrflag: (required) The internal value of the MASTER ENTRY FOR VUID field. Possible values are as follows:

- **0**—NO.
- **1**—YES.

Output: returns: Returns results of operation as follows:

- **Successful**—1
- **Unsuccessful**—0^<*error message*>

27.9.6.1 Examples

27.9.6.1.1 Example 1

For terms defined in fields that are SET OF CODES:

Figure 273: $$SETMASTR^XTID API—Example 1: Terms Defined in Fields that are SET OF CODES

```
>S file=2,field=.02,iref="M",mstrflag=0
>W $$SETMASTR^XTID(file,field,iref,mstrflag)
1
```

27.9.6.1.2 Example 2

For terms defined in a single file:

Figure 274: $$SETMASTR^XTID API—Example 2: Terms Defined in a Single File

```
>S file=16000009,field=.01,iref="3,",mstrflag=1
>W $$SETMASTR^XTID(file,field,iref,mstrflag)
1
```

27.9.6.1.3 Example 3

Figure 275: $$SETMASTR^XTID API—Example 3

```
>S file=16000009,field=.01,iref="6,",mstrflag=1
>W $$SETMASTR^XTID(file,field,iref,mstrflag)
0^pre-existing master entry
```

27.9.7 $$SETSTAT^XTID(): Set Status Information (Term/Concept)

Reference Type: Supported

Category: Toolkit—VHA Unique ID (VUID)

ICR #: 4631

Description: The $$SETSTAT^XTID extrinsic function stores (**SET**s) the status and effective date/time for the given term/concept.

Format: `$$SETSTAT^XTID(file[,field],iref,status[,datetime])`

Input Parameters:

file: (required) VistA file/subfile number where term/concept is defined.

field: (optional) Field number in the **file** input parameter where term/concept is defined.

- ***Not* Defined**—If *not* defined, this field defaults to the **.01** field number, and it represents terms defined in the file entered in the **file** input parameter.
- **Defined:**
    - Entered as **.01**; it represents the terms defined in the file entered in the **file** input parameter.
    - Otherwise, the field number entered *must* be a SET OF CODES data type

		field in the file entered in the **file** input parameter.

	iref:	(required) Internal reference for term/concept:
		• **File Entries**—This is an IENS. For example:
		iref = "**5,**"
		• **SET OF CODES**—This is the internal value of the code. For example, any of the following:
		iref = **3**
		iref = "**f**"
		iref = "**M**"

	status:	(required) The status internal value. Possible values are as follows: • **0**—INACTIVE. • **1**—ACTIVE.
	datetime:	(optional) VA FileMan date/time. It defaults to **NOW**.
Output:	returns:	Returns results of operation as follows: • **Successful**—1 • **Unsuccessful**—0^<*error message*>

27.9.7.1 Examples

27.9.7.1.1 Example 1

For terms defined in fields that are SET OF CODES:

Figure 276: $$SETSTAT^XTID API—Example 1: Terms Defined in Fields that are SET OF CODES

```
>S file=2,field=.02,iref="M",status=1,datetime=$$NOW^XLFDT
>W $$SETSTAT^XTID(file,field,iref,status,datetime)
1
```

27.9.7.1.2 Example 2

For terms defined in a single file:

Figure 277: $$SETSTAT^XTID API—Example 2: Terms Defined in a Single File

```
>S file=16000009,field=.01,iref="3,",status=1,datetime=$$NOW^XLFDT
>W $$SETSTAT^XTID(file,field,iref,status,datetime)
1
```

27.9.8 $$SETVUID^XTID(): Set VUID (Term/Concept)

Reference Type: Supported

Category: Toolkit—VHA Unique ID (VUID)

ICR #: 4631

Description: The $$SETVUID^XTID extrinsic function populates (**SET**s) the VHA Unique ID (VUID) for a given term/concept reference.

It also automatically sets the MASTER ENTRY FOR VUID field to distinguish references that might be duplicates:

- If this is the first reference assigned the VUID, it sets the MASTER ENTRY FOR VUID equal to **1**.
- If another entry already has the given VUID, it sets the MASTER ENTRY FOR VUID equal to **0**.

Format: `$$SETVUID^XTID(file[,field],iref,vuid)`

Input Parameters:

file: (required) VistA file/subfile number where term/concept is defined.

field: (optional) Field number in the **file** input parameter where term/concept is defined.

- ***Not* Defined**—If *not* defined, this field defaults to the **.01** field number, and it represents terms defined in the file entered in the **file** input parameter.
- **Defined:**
    - o Entered as **.01**; it represents the terms defined in the file entered in the **file** input parameter.
    - o Otherwise, the field number entered *must* be a SET OF CODES data type

field in the file entered in the **file** input parameter.

iref: (required) Internal reference for term/concept.

- **File Entries**—This is an IENS. For example:
  iref = "**5,**"
- SET OF CODES—This is the internal value of the code. For example, any of the following:
  iref = **3**
  iref = "**f**"
  iref = "**M**"

vuid: (required) The VHA Unique ID (VUID) to assign the given term/concept reference.

Output: returns: Returns results of operation as follows:

- **Successful**—1
- **Unsuccessful**—0^<*error message*>

27.9.8.1 Examples

27.9.8.1.1 Example 1

For terms defined in fields that are SET OF CODES:

Figure 278: $$SETVUID^XTID API—Example 1: Terms Defined in Fields that are SET OF CODES

```
>S file=2,field=.02,iref="M",vuid=123456
>W $$SETVUID^XTID(file,field,iref,vuid)
1
```

27.9.8.1.1.1 Example 2

For terms defined in a single file:

Figure 279: $$SETVUID^XTID API—Example 2: Terms Defined in a Single File

```
>S file=16000009,field=.01,iref="3,",vuid=123457
>W $$SETVUID^XTID(file,field,iref,vuid)
1
```

27.10 Toolkit—Routine Tools

Kernel Toolkit provides developer utilities for working with M routines and globals. This section describes the routine tools exported with Kernel Toolkit. These tools are useful to system administrators and VistA software developers.

27.10.1 Direct Mode Utilities

Several Kernel Toolkit direct mode utilities are available for developers to use at the M prompt, usually involving the **DO** command. They are *not* APIs and *cannot* be used in software application routines.

Table 35: Routine Tools—Direct Mode Utilities

Direct Mode Utility	Description
>**D ^XTFCR**	Generate a flow chart of an entire routine.
>**D ^XTFCE**	Generate a flow chart of the processing performed from a specified entry point to the termination of processing resulting from that entry point.
>**D ^%INDEX**	(obsolete) To run **%INDEX**.
>**D ^XINDEX**	To run **XINDEX**.
>**X ^%Z**	Invokes the **^%Z** Editor.
>**D ^XTRGRPE**	Edit a group of routines.
>**D ^XTVCHG**	Changes all occurrences of one variable to another.
>**D ^XTVNUM**	Update or set the version number into a set of routines.
>**D ^%ZTP1**	A summary listing of the first, and optionally the second, line of one or more routines can be obtained.
>**D ^%ZTPP**	Print a listing of entire routines.
>**D ^XTRCMP**	Compare two routines with different names and display the differences (using MailMan's PackMan compare utilities).

Direct Mode Utility	Description
`>D TAPE^XTRCMP`	Compares routines in a Host File Server (HFS) file to an installed routine and displays the differences. **NOTE:** While it is still called a "**TAPE**" compare, it is actually comparing a routine in an HFS file to an installed routine.
`>D ^%ZTRDEL`	Delete one or more routines.
`>D ^%RR` (OS-specific)	Loads routines from an external device, such as magtape.
`>D ^%RS` (OS-specific)	Output routines to an external device, such as a magtape.

27.10.2 Routine Tools Menu

Most of these tools are available as options on the **Routine Tools** [XUPR-ROUTINE-TOOLS] menu located on the **Programmer Options** [XUPROG] menu, which is locked with the XUPROG security key. Some subordinate menu options are locked with the XUPROGMODE or XUPROG security keys as an extra level of security.

Routines can be edited, analyzed by flow-charting, printed, compared, deleted, and moved by using an option or its corresponding direct mode utility.

The **Routine Tools** menu is shown in Figure 280:

Figure 280: Routine Tools—Menu Options

```
SYSTEMS MANAGER MENU ...                                              [EVE]
  Programmer Options ... <locked with XUPROG>                      [XUPROG]
   Routine Tools ...                                   [XUPR-ROUTINE-TOOLS]
     %Index of Routines                                           [XUINDEX]
     Compare local/national checksums report           [XU CHECKSUM REPORT]
     Compare routines on tape to disk                   [XUPR-RTN-TAPE-CMP]
     Compare two routines                              [XT-ROUTINE COMPARE]
     Delete Routines <locked with XUPROGMODE>                      [XTRDEL]
     Flow Chart Entire Routine                                     [XTFCR]
     Flow Chart from Entry Point                                   [XTFCE]
     Group Routine Edit <locked with XUPROGMODE>                 [XTRGRPE]
     Input routines <locked with XUPROG>                    [XUROUTINE IN]
     List Routines                                               [XUPRROU]
     Load/refresh checksum values into ROUTINE file      [XU CHECKSUM LOAD]
     Output routines                                       [XUROUTINE OUT]
     Routine Edit <locked with XUPROGMODE>                 [XUPR RTN EDIT]
     Routines by Patch Number                             [XUPR RTN PATCH]
     Variable changer <locked with XUPROGMODE>         [XT-VARIABLE CHANGER]
     Version Number Update <locked with XUPROGMODE>     [XT-VERSION NUMBER]
```

These options are documented in the sections that follow, grouped by routine type.

27.10.2.1 Analyzing Routines

27.10.2.1.1 %Index of Routines Option—XINDEX

The **%Index of Routines** [XUINDEX] option calls Kernel Toolkit's XINDEX utility (formerly known as **%INDEX** utility). XINDEX is a static analysis tool that plays the dual role of a VistA-aware cross-referencing tool and a code checker (or recognizer).

As of Kernel Toolkit patch XT*7.3*132, the **%Index of Routines** [XUINDEX] option allows users to check the contents of any of the following:

- **Routines**—XINDEX checks the specified routines (e.g., **XU***).
- **Builds**—XINDEX checks the contents of the specified build defined in the BUILD (#9.6) file. XINDEX checks all components of the build on the current system, which includes, routines, options, templates, data dictionaries, etc.
- **Installs**—XINDEX checks the contents of the specified install defined in the INSTALL (#9.7) file. XINDEX checks all components of the install that have temporarily been loaded into **^XTEMP** global, which includes, routines, options, templates, data dictionaries, etc.
- **Packages**—XINDEX checks the contents of the specified package defined in the PACKAGE (#9.4) file. XINDEX checks all components of the package on the current system, which includes, routines, options, templates, data dictionaries, etc.

Figure 281: %Index of Routines Option—Sample User Entries

```
Select Routine Tools Option: %INDEX <Enter> of Routines

                   V. A.  C R O S S  R E F E R E N C E R  7.3
                       [2008 VA Standards & Conventions]
                   UCI: KRN CPU: KRN    Dec 13, 2011@07:40:44

All Routines? No => NO

Routine: HLUOPT
Routine: <Enter>
1 routine


Select BUILD NAME: <Enter>
Select INSTALL NAME: <Enter>
Select PACKAGE NAME: <Enter>

Print more than compiled errors and warnings? YES// <Enter>

Print summary only? NO// <Enter>

Print routines? YES// <Enter>


Print (R)egular,(S)tructured or (B)oth?  R// <Enter>

Print errors and warnings with each routine? YES// <Enter>


Save parameters in ROUTINE file? NO// <Enter>

Index all called routines? NO// <Enter>
DEVICE: <Enter>  Telnet Terminal    Right Margin: 80// <Enter>

                   V. A.  C R O S S  R E F E R E N C E R  7.3
                       [2008 VA Standards & Conventions]
                   UCI: KRN CPU: KRN    Dec 13, 2011@07:40:44
Routines: 1  Faux Routines: 0

HLUOPT

--- CROSS REFERENCING ---

   Press return to continue: <Enter>



Compiled list of Errors and Warnings              Dec 13, 2011@07:40:44
page 1

HLUOPT   * *  69 Lines,  3758 Bytes, Checksum: B18177059



HOLD+4        W - Null line (no commands or comment).
```

```
--- Routine Detail   --- with REGULAR ROUTINE LISTING ---
   Press return to continue:
<Enter>

HLUOPT   * *  69 Lines,  3758 Bytes, Checksum: B18177059
                                                 Dec 13, 2011@07:40:44
page 2
              548 bytes in comments
HLUOPT   ;AISC/REDACTED-Main Menu for HL7 Module ;07/26/99  08:47
         ;;1.6;HEALTH LEVEL SEVEN;**57**;Oct 13, 1995
```

REF: For more information on the XINDEX utility, see the "XINDEX" section.

27.10.2.1.2 Flow Chart Entire Routine Option

The **Flow Chart Entire Routine** [XTFCR] option generates a flow chart, showing the processing performed within an entire routine.

The following corresponding direct mode utility can be used in programmer mode:

```
>D ^XTFCR
```

27.10.2.1.3 Flow Chart From Entry Point Option

The **Flow Chart from Entry Point** [XTFCE] option generates a flow chart of the processing performed from a specified entry point to its termination of processing. It also allows the user to expand the code in other routines or entry points referenced by **DO** or **GOTO** commands.

The following corresponding direct mode utility can be used in programmer mode:

```
>D ^XTFCE
```

27.10.2.2 Editing Routines

27.10.2.2.1 Group Routine Edit Option

The **Group Routine Edit** [XTRGRPE] option calls the **XTRGRPE** routine to edit a group of routines. Once several routines are identified, the Kernel Toolkit **^%Z** Editor is called. This option is locked with the XUPROGMODE security key.

The corresponding direct mode utility can be used in programmer mode as follows:

```
>D ^XTRGRPE
```

27.10.2.2.2 Routine Edit Option

The **Routine Edit** [XUPR RTN EDIT] option invokes the **^%Z** Editor. The **^%Z** Editor can be used to edit a group of routines with the **Group Routine Edit** [XTRGRPE] option. This allows developers at an external site (e.g., on the site manager's staff) to edit M routines. This option is locked with the XUPROGMODE security key.

The corresponding direct mode utility can be used in programmer mode as follows:

```
>X ^%Z
```

REF: For more information on the ^%Z Editor, see the "^%Z Editor" section in Section 17, "Miscellaneous: Developer Tools."

27.10.2.2.3 Routines by Patch Number Option

The **Routines by Patch Number** [XUPR RTN PATCH] option allows users to print routines associated with a patch. When prompted, enter a list of routines. The output is sorted by patch number.

27.10.2.2.4 Variable Changer Option

The **Variable Changer** [XT-VARIABLE CHANGER] option runs the **XTVCHG** routine, which changes all occurrences of one variable to another. This option is locked with the XUPROGMODE security key.

CAUTION: This option changes DOs and GOTOs also, but it does *not* change the target of the DOs and GOTOs. For example, if you request to change all occurrences of "TAG" to "TAGS", "DO TAG" would be changed to "DO TAGS". However, the actual Line Label called TAG would not be changed.

The corresponding direct mode utility can be used in programmer mode as follows:

```
>D ^XTVCHG
```

27.10.2.2.5 Version Number Update Option

The **Version Number Update** [XT-VERSION NUMBER] option updates version numbers of one or more routines. This option runs the XTVNUM routine to update or set the version number into a set of routines. This option is locked with the XUPROGMODE security key.

The corresponding direct mode utility can be used in programmer mode as follows:

```
>D ^XTVNUM
```

27.10.2.3 Printing Routines

27.10.2.3.1 List Routines Option

The **List Routines** [XUPRROU] option uses the **%ZTPP** utility to print a listing of entire routines.

The corresponding direct mode utility can be used in programmer mode as follows:

```
>D ^%ZTPP
```

27.10.2.4 Comparing Routines

27.10.2.4.1 Compare local/national checksums report Option

The **Compare local/national checksums report** [XU CHECKSUM REPORT] option compares checksums for routines to the values in the ROUTINE (#9.8) file. It produces a report listing routines that differ by the following criteria:

- Patch or version, where the version or patch may be correct but checksums are off
- Local routines being tracked
- Information is *not* on record for a patch (e.g., test patches)

Nationally released routine checksums are sent by Master File Updates to the local ROUTINE (#9.8) file automatically. Local sites may also record checksums in the CHECKSUM VALUE field in the ROUTINE (#9.8) file. To compare local routines that are being tracked, the CHECKSUM REPORT field should be set to "Local – report."

As of Kernel Patch XU*8.0*369, the integrity checking CHECK1^XTSUMBLD routine supports the **Compare local/national checksums report** [XU CHECKSUM REPORT] option.

As of Kernel Patch XU*8.0*393, KIDS was modified to send a message to a server on FORUM when a KIDS build is sent to a Host File Server (HFS) device. This message contains the checksums for the routines in the patch. The server on FORUM matches the message with a patch if the sending domain is authorized on FORUM. There is no longer a need for developers to manually include routine checksums (either CHECK^XTSUMBLD or CHECK1^XTSUMBLD routines) in the patch description. The patch module includes the before and after CHECK1^XTSUMBLD values in the Routine Information section at the end of the patch document.

With changes in the National Patch Module (NPM) on FORUM, when the patch is released the checksums for the routines are moved to the ROUTINE (#9.8) file on FORUM. The checksum "before" values come from the FORUM ROUTINE (#9.8) file and are considered the GOLD standard for released checksums. The local site's **Compare local/national checksums report** [XU CHECKSUM REPORT] option uses the FORUM ROUTINE (#9.8) file as its source to create reports showing any routines that do *not* match.

This patch also modified the KIDS BUILD (#9.6) file by adding the TRANSPORT BUILD NUMBER (#63) field used to store a build number that is incremented each time a build is made.

This build number is added to the second line of each routine in the 7th "**;**" piece. This makes it easy to tell if a site is running the current release during testing and afterword. The leading "B" found in the checksum tells the code what checksum API to use.

27.10.2.4.2 Compare Routines on Tape to Disk Option

The **Compare Routines on Tape to Disk** [XUPR-RTN-TAPE-CMP] option compares routines and displays the differences. This option reads a standard Caché %RO Host File Server (HFS) file and compares the routines on the HFS file with a routine with the same name in the current account.

The corresponding direct mode utility can be used in programmer mode as follows:

```
>D TE^XTRCMP
```

NOTE: While it is still called a "TAPE" compare, it is actually comparing a routine in a Host File Server (HFS) file to an installed routine.

27.10.2.4.3 Compare Two Routines Option

The **Compare Two Routines** [XT-ROUTINE COMPARE] option compares two routines with different names that are located in the same account and displays/prints the differences (using MailMan's PackMan compare utilities).

The corresponding direct mode utility can be used in programmer mode as follows:

```
>D ^XTRCMP
```

27.10.2.5 Deleting Routines

27.10.2.5.1 Delete Routines Option

The **Delete Routines** [XTRDEL] option can be used to delete one or more routine(s). The wildcard syntax can be used to delete a set, such as **ABC*** to delete all those routines beginning with the letters **ABC**. This option is locked with the XUPROGMODE security key.

The corresponding direct mode utility can be used in programmer mode as follows:

```
>D ^%ZTRDEL
```

27.10.2.6 Load and Save Routines

The Input Routines and Output Routines options can be used to move routines from one UCI to another. These make use of operating system-specific utilities such as %RR for routine restore and %RS for routine save.

27.10.2.6.1 Input Routines Option

The **Input Routines** [XUROUTINE IN] option loads routines from an external device. This option is locked with the XUPROG security key.

The corresponding direct mode utility can be used in programmer mode as follows:

```
>D ^%RR (OS-specific)
```

27.10.2.6.2 Output Routines Option

The **Output Routines** [XUROUTINE OUT] option outputs routines to an external device, such as a host file.

The corresponding direct mode utility can be used in programmer mode as follows:

```
>D ^%RS (OS-specific)
```

27.10.2.6.3 Load/refresh checksum values into ROUTINE file Option

The Load/refresh checksum values into ROUTINE file option [XU CHECKSUM LOAD] can be used to update the ROUTINE (#9.8) file with the latest checksum values from FORUM.

REF: Kernel Toolkit Application Programming Interfaces (APIs) are documented in the "Toolkit: Developer Tools" section. Kernel and Kernel Toolkit APIs are also available in HTML format on the VA Intranet Website.

27.11 Toolkit—Verification Tools

Kernel Toolkit provides an Application Programming Interface (API) that includes developer utilities for working with routines and globals. This section describes the verification tools exported with Kernel Toolkit that are useful to system administrators and developers for reviewing Veterans Health Information Systems and Technology Architecture (VistA) software.

Verification tools can be accessed through one of three methods:

- Direct Mode Utilities
- Programmer Options Menu
- Operations Management Menu

27.11.1 Direct Mode Utilities

Several Kernel Toolkit direct mode utilities are available for developers to use at the M prompt, usually involving the **DO** command. They are *not* APIs and *cannot* be used in software application routines. These direct mode utilities are described below by category.

The XINDEX utility can be used to check a routine or set of routines against standards such as the 1995 ANSI M Standard syntax and VA *Programming Standards and Conventions (SAC)*.

REF: For more information on the XINDEX utility, see the "%Index of Routines Option" section in the "Toolkit—Routine Tools" section in this section.

The corresponding direct mode utility can be used in Programmer mode:

```
>D ^XINDEX
```

Many of the options on the **Programmer Options** menu can also be run as direct mode utilities. Some are *not* available as options, but only as direct mode utilities callable at the M prompt.

Table 36 lists examples on how to run these utilities when working in Programmer mode.

Table 36: Verification Tools—Direct Mode Utilities

Direct Mode Utility	Description
`>D CHCKSUM^XTSUMBLD`	Check the checksum value of a routine at any given time. This direct mode utility allows the developer to choose from the old CHECK^XTSUMBLD checksum routine or the new and more accurate CHECK1^XTSUMBLD checksum routine. **REF:** For more information on the CHECK^XTSUMBLD and CHECK1^XTSUMBLD routines, see Sections 23 and 24 in the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide*.
`>D ^nsNTEG`	Check Integrity of namespace (ns) Package. For example, D ^XTNTEG compares the Kernel Toolkit namespace (XT) checksums with expected values.
`>D ONE^nsNTEG`	Check Integrity Routine in namespace (ns) Package.
`>D ^%ZTER`	Record an Error.

Direct Mode Utility	Description
`>D ^XTER`	Display Error Trap.
`>D ^XTERPUR`	Purge Error Log.
`>D ^%INDEX`	(obsolete) To run %INDEX.
`>D ^XINDEX`	To run XINDEX. XINDEX is similar to %INDEX but supports the most current M standard.

NOTE: For information on the options associated with the routines associated with these verification tools direct mode utilities, see the "Verification Tools" section in the "Toolkit" section in the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide*.

27.11.2 Verifier Tools Menu

The Verifier Tools Menu contains options that are available as tools for verification during program development. These options are located on the Verifier Tools Menu [XTV MENU], which is located on the Systems Manager Menu. These tools are useful for developers to:

- Record the text of the routines indicated in the file used to maintain changes in routines.
- Compare one or more current routines to previous versions.

The **Verifier Tools Menu** [XTV MENU] consists of the following options shown in Figure 282:

Figure 282: Verifier Tools—Menu Options

```
SYSTEMS MANAGER MENU ...                                        [EVE]
  Verifier Tools Menu ...                                  [XTV MENU]
     Update with current routines                       [XTVR UPDATE]
     Routine Compare - Current with Previous           [XTVR COMPARE]
```

27.11.2.1 Update with Current Routines Option

The **Update with Current Routines** [XTVR UPDATE] option records the text of the routines indicated in the file used to maintain changes in routines. Only the last version entered is kept intact; previous entries reflect only the changes in lines added or deleted to make the next version. This option records the current routine structure so that it can be compared with future versions of the routine using the **Routine Compare - Current with Previous** [XTVR COMPARE] option.

After editing the routine, the **Update with Current Routines** option can again be used to store changes. Rather than storing all minor changes, the user can choose to wait and use the **Update**

with Current Routines option only after extensive edits have been made. Lines are compared and changes, including inserted or deleted lines, are recorded. (Alteration of the routine's second line is usually insignificant and is ignored.) The **Update with Current Routines** option can be used whenever the developer would like a new "snapshot" of the routine. The XTV ROUTINE CHANGES (#8991) file holds each new snapshot as a new version. This filing method does *not*, however, alter the actual version number of the routine itself.

27.11.2.2 Routine Compare - Current with Previous Option

The **Routine Compare - Current with Previous** [XTVR COMPARE] option compares one or more current routines to previous versions. To use the routine compare utility, copies of the selected routines *must* first be stored in the XTV ROUTINE CHANGES (#8991) file, stored in the **^XTV(8991,** global. This is achieved by use of the **Update with Current Routines** [XTVR UPDATE] option on the Verifier Tools Menu. Routines can be specified one by one or as a group with the wildcard syntax (e.g., **XQ***). Any initialize routines are automatically excluded. Differences between the current version and the indicated number of prior versions are noted. The user is prompted for the number of previous versions from which to begin the listing. An entire history or just a brief display of recent modifications can be obtained.

27.11.3 Programmer Options Menu

The **Programmer Options** [XUPROG] menu comprised of the following options:

Figure 283: Programmer Options—Menu options: Toolkit Verification Tools

```
SYSTEMS MANAGER MENU ...                                            [EVE]
  Programmer Options ...                                         [XUPROG]
    **> Locked with XUPROG
   KIDS Kernel Installation & Distribution System ...          [XPD MAIN]
           **> Locked with XUPROG
   PG   Programmer mode                                      [XUPROGMODE]
           **> Locked with XUPROGMODE
        Calculate and Show Checksum Values             [XTSUMBLD-CHECK]
        Delete Unreferenced Options                 [XQ UNREF'D OPTIONS]
        Error Processing ...                                     [XUERRS]
        General Parameter Tools ...                     [XPAR MENU TOOLS]
        Global Block Count                               [XU BLOCK COUNT]
        List Global                                             [XUPRGL]
           **> Locked with XUPROGMODE
        Routine Tools ...                           [XUPR-ROUTINE-TOOLS]
        Test an option not in your menu                  [XT-OPTION TEST]
           **> Locked with XUMGR
```

Tools found on the **Programmer Options** menu that can be of use for verification purposes include:

- **Calculate and Show Checksum Values** [XTSUMBLD-CHECK]
- **Error Processing** [XUERRS]

These options are described in the sections that follow.

27.11.3.1 Calculate and Show Checksum Values Option

The **Calculate and Show Checksum Values** [XTSUMBLD-CHECK] option gives developers the ability to check the value of a routine at any given time. It does *not* regenerate **NTEG** routines and can safely be used anytime.

This option calls the CHCKSUM^XTSUMBLD direct mode utility to calculate and show the checksum value for one or more routines in the current account. This value is referenced in the Patch Module description for routine patches.

NOTE: Kernel Toolkit patch XT*7.3*94, deployed the CHECK1^XTSUMBLD routine and the new logic Checksum: %^ZOSF("RSUM1"). Kernel Toolkit patch XT*7.3*100 included the CHECK1^XTSUMBLD routine into the **Calculate and Show Checksum Values** [XTSUMBLD-CHECK] option.

The CHECK1^XTSUMBLD routine is more accurate than the old integrity checking utility (CHECK^XTSUMBLD). CHECK1^XTSUMBLD. It determines the current checksums for selected routine(s), the functionality of which is shown as follows:

- Any comment line with a single semi-colon is presumed to be followed by comments and only the line tag is included.
- Line 2 is excluded from the count.
- The total value of the routine is determined (excluding exceptions noted above) by multiplying the ASCII value of each character by its position on the line and position of the line in the routine being checked.

The corresponding direct mode utility can be used in programmer mode:

```
>D CHCKSUM^XTSUMBLD
```

NOTE: The integrity checking utility CHCKSUM^XTSUMBLD supports the **Compare local/national checksums report** [XU CHECKSUM REPORT] option, as released with Kernel Patch XU*8.0*369.

NOTE: The modification, CHECK1^XTSUMBLD, to the integrity checking utility CHCKSUM^XTSUMBLD fixes the problem in which the old checksum output is the same checksum value, even if some lines were swapped within a routine.

27.11.3.2 Error Processing—Kernel Error Trapping and Reporting

Technical personnel who have entered programmer mode with D ^XUP, might choose to record an error encountered with D ^%ZTER. The error log can be displayed with D ^XTER, or with the corresponding option. Also, the error log can be purged with D ^XTERPUR. Errors can also be purged from within the menu system with an option that is locked with the XUPROGMODE security key.

The corresponding direct mode utilities can be used in programmer mode as follows:

- Record an Error:
  ```
  >D ^%ZTER
  ```
- Display Error Trap:
  ```
  >D ^XTER
  ```
- Purge Error Log:
  ```
  >D ^XTERPUR
  ```

REF: For more information on Error Processing, see Section 13, "Error Processing," in the *Kernel 8.0 & Kernel Toolkit 7.3 Systems Management Guide*.

27.12 XINDEX

Kernel Toolkit's XINDEX utility (formerly known as %INDEX utility) is a static analysis tool that plays the dual role of a VistA-aware cross-referencing tool and a code checker (or recognizer). As of Kernel Toolkit patch XT*7.3*132, XINDEX creates a cross-referenced list of global references and routines invoked by selecting any of the following:

- **Routines**—XINDEX checks the specified routines (e.g., **XU***).
- **Builds**—XINDEX checks the contents of the specified build defined in the BUILD (#9.6) file. XINDEX checks all components of the build on the current system, which includes, routines, options, templates, data dictionaries, etc.
- **Installs**—XINDEX checks the contents of the specified install defined in the INSTALL (#9.7) file. XINDEX checks all components of the install that have temporarily been loaded into **^XTEMP** global, which includes, routines, options, templates, data dictionaries, etc.
- **Packages**—XINDEX checks the contents of the specified package defined in the PACKAGE (#9.4) file. XINDEX checks all components of the package on the current system, which includes, routines, options, templates, data dictionaries, etc.

Use XINDEX to verify parts of a software application in the VistA environment that contain M code, including the following:

- Routines
- Options
- Compiled Templates
- Data Dictionaries (DD)
- Functions

XINDEX provides greater analysis capability than other syntax analysis tools that operate at the routine level only. As a *static* analysis tool, however, XINDEX has a *fundamental* limitation of the types of errors that it is able to catch and report. XINDEX is only able to look at the written structure of M code. It *cannot* look at dynamic aspects, such as the run-time symbol table or flow of control when it is modified by conditional branching (e.g., through post-conditionals or argument indirection). XINDEX is also generally conservative, at times preferring to report false positives rather than ignore potential problems. When analyzing XINDEX output, you *must* take all of this into consideration.

VistA applications are required to follow a set of Standards and Conventions (SAC) as set by the VA's Standards and Conventions Committee (SACC), which are defined as follows:

- **Standard**—Requirement that *must* be adhered to.
- **Convention**—Rule that *should* be followed.

VistA protects many of its abstractions via convention, even when those conventions are requirements. XINDEX checks that the MUMPS (M) routine code conforms to the 1995 ANSI M Standard and *VA Programming Standards and Conventions (SAC)*. XINDEX considers all SAC prohibitions as an error. XINDEX checks SAC requirements, because conformance to the SAC is essential to the proper function of VistA.

VistA is comprised of a number of software packages (defined by namespace), which can be further divided into the following two basic groups:

- **Applications**—VistA client applications or application modules (e.g., Pharmacy, Laboratory, Patient Care Encounter [PCE]).
- **Infrastructure Applications**—Collection of Infrastructure packages that implement the basic programming and runtime VistA framework. For example:
    - **Kernel/Kernel Toolkit**—Provides a portable system interface, a common execution environment, and essential services such as signon and security.
    - **MailMan**—Provides VistA email functionality.
    - **VA FileMan**—Provides database functionality built on top of the M global subsystem integrated with the VistA security model.

It is important to recognize that the rules for VistA infrastructure packages (particularly Kernel and VA FileMan) are different from other VistA applications. Code used in infrastructure packages to implement a system interface *must* be able to use implementation-specific code. Accordingly, Kernel (and sometimes VA FileMan) has standing exemptions from many of the requirements of the SAC. Thus, XINDEX sometimes reports errors and standards violations for allowed constructs.

REF: For more information on the Standards and Conventions Committee (SACC) and Standards and Conventions (SAC) documentation, see the SACC VA Intranet website.

27.12.1 Types of XINDEX Findings

XINDEX reports its findings under the following general categories of codes (error flags):

Table 37: XINDEX—Types of Findings (Category Codes or Flags)

Category Code/Other	Description
F	**Fatal M Errors (Hard MUMPS Error)**—These are unrecoverable errors that cause a program to fail if the commands are executed. It is possible, however, that these types of errors might exist in routines that run correctly. The error occurs (or may occur, depending on the underlying implementation) only when the errant commands are executed. ℹ **REF:** For a description and sample code analysis on errors in this category, see Section 26.13.3.1, "Fatal M Errors (Hard MUMPS Error)."
W	**Warning Violation Errors (According to VA Conventions)**—These are potential problems that are *not* necessarily fatal errors but most likely indicate an error. They require careful implementation. ℹ **REF:** For a description and sample code analysis on errors in this category, see Section 27.12.3.2, "Warning Violation Errors (According to VA Conventions)."
S	**Standards Violation Errors (According to VA Standards)**—These are issues that do *not* pertain to the M language *per se*, but rather the requirements of the VA Standards and Conventions (SAC). Issues flagged as Standards Violations can still be syntactically correct M code that follows the portability guidelines, but does *not* follow the more stringent requirements set forth in the SAC. ℹ **REF:** For a description and sample code analysis on errors in this category, see Section 26.13.3.3, "Standards Violation Errors (According to VA Standards)."
I	**Informational Errors**—These issues are *not* necessarily errors but still require attention, because they could indicate potential problems. ℹ **REF:** For a description and sample code analysis on errors in this category, see Section 26.13.3.4, "Informational."
Manual Check	**Marked Items Errors (Manual Check)**—These issues only apply if a line contains **$TEXT ($T)**. XINDEX records the location and prints it out under the "Marked Items" sub-header on the XINDEX report. ℹ **REF:** For a description on errors in this category, see Section 26.13.3.5, "Marked Items Errors (Manual Check."

Table 38 lists the current error conditions (messages) that the XINDEX utility flags. XINDEX retrieves and displays the messages from the XINDX1 routine.

NOTE: Any updates (e.g., add, modify, or delete messages) made to the list of XINDEX messages are based on changes to the XINDEX utility via subsequent Kernel Toolkit patches.

Table 38: XINDEX—List of Error Conditions (Messages) Flagged: Grouped by Category and Listed Alphabetically); Messages are Stored in XINDX1 Routine

Message Displayed (click on link for more detail)
Category: Fatal M Errors (Hard MUMPS Error)
F—Bad Number
F—Bad WRITE syntax
F—Block structure mismatch
F—Call to missing label 'label' in this routine
F—Call to this *label/routine* (MISSING LABEL)
F—Command missing an argument
F—Error in pattern code
F—FOR Command followed by only one space
F—FOR Command did not contain '='
F—General Syntax Error
F—GO or DO mismatch from block structure (M45)
F—Invalid or wrong number of arguments to a function
F—Label is not valid
F—Missing argument to a command post-conditional
F—Non-standard (Undefined) 'Z' command
F—Quoted string not followed by a separator
F—Reference to routine '^*routine name*'. That isn't in this UCI
F—UNDEFINED COMMAND (rest of line not checked) **NOTE:** Developers *must* manually check these errors.
F—Undefined Function
F—Undefined Special Variable
F—Unmatched Parenthesis

Message Displayed (click on link for more detail)
F—Unmatched Quotation Marks
F—Unrecognized argument in SET command
Category: Warning Violation Errors (According to VA Conventions)
W—Blank(s) at end of line
W—Duplicate label, (M57) (M standard error)
W—First line label NOT routine name
W—Invalid global variable name
W—Invalid local variable name
W—Line contains a CONTROL (non-graphic) character
W—Null line (no commands or comment)
Category: Standards Violation Errors (According to VA Standards)
S—$View function used
S—Access to SSVN's restricted to Kernel
S—Break command used
S—Extended reference
S—First line of routine violates the SAC
S—2nd line of routine violates the SAC
S—Patch number 'nnn' missing from second line
S—'HALT' command should be invoked through 'G ^XUSCLEAN'
S—Kill of a protected variable (*variable name*)
S—Kill of an unsubscripted global
S—Unargumented Kill
S—Exclusive Kill
S—Exclusive or Unargumented NEW command
S—LABEL+OFFSET syntax
S—Line is longer than 245 bytes
S—Lock missing Timeout
S—Lower/Mixed case Variable name used
S—Lowercase command(s) used in line
S—Non-Incremental Lock
S—Non-standard $Z function used

Message Displayed (click on link for more detail)
S—Non-standard $Z special variable used
S—'OPEN' command should be invoked through ^%ZIS
S—'Close' command should be invoked through 'D ^%ZISC'
S—Read command doesn't have a timeout
S—Routine code exceeds SACC maximum size of 15000 (*nnnnn*)
S—Routine exceeds SACC maximum size of 20000 (*nnnnn*)
S—Set to a '%' global
S—Should use 'TASKMAN' instead of 'JOB' command
S—View command used
S—Violates VA programming standards
Category: Informational Errors
I—QUIT Command followed by only one space
I—Star or pound READ used

27.12.2 Running the XINDEX Utility

CAUTION: When running XINDEX to review an entire software application, it is best to queue the report for an off-peak time, since processing is intensive.

Use either of the following methods to call the XINDEX utility:

- **Direct Mode Utility:**

```
>D ^XINDEX
```

  REF: For examples using the Direct Mode Utility, see "Examples."

- **Option**—Use the **%Index of Routines** [XUINDEX] option located on the on the **Routine Tools** [XUPR-ROUTINE-TOOLS] menu located on the **Programmer Options** [XUPROG] menu, which is locked with the XUPROG security key.

  REF: For more information on the **%Index of Routines** option, see the "%Index of Routines Option—XINDEX" section.

27.12.2.1 Examples

27.12.2.1.1 Example 1

Specifying a Routine Name Only:

Figure 284: XINDEX—Direct Mode Utilities Sample User Entries: Specifying a Routine Name Only

```
KRN>D ^XINDEX

                   V. A.  C R O S S  R E F E R E N C E R  7.3
                       [2008 VA Standards & Conventions]
                   UCI: KRN CPU: KRN    Jan 12, 2012@14:47:16

All Routines? No => <Enter> No

Routine: XDRMAIN
Routine: <Enter>
1 routine

Select BUILD NAME: <Enter>
Select INSTALL NAME: <Enter>
Select PACKAGE NAME: <Enter>

Print more than compiled errors and warnings? YES// <Enter>

Print summary only? NO// <Enter>

Print routines? YES// <Enter>


Print (R)egular,(S)tructured or (B)oth?  R// <Enter>

Print errors and warnings with each routine? YES// <Enter>


Save parameters in ROUTINE file? NO// <Enter>

Index all called routines? NO// <Enter>
DEVICE: ;P-OTHER <Enter>  Telnet Terminal    Right Margin: 255// 80


                   V. A.  C R O S S  R E F E R E N C E R  7.3
                       [2008 VA Standards & Conventions]
                   UCI: KRN CPU: KRN    Jan 12, 2012@14:47:16
Routines: 1  Faux Routines: 0

XDRMAIN

--- CROSS REFERENCING ---

Compiled list of Errors and Warnings              Jan 12, 2012@14:47:16
page 1
No errors or warnings to report
```

```
--- Routine Detail   --- with REGULAR ROUTINE LISTING ---

XDRMAIN  * *  80 Lines,  3431 Bytes, Checksum: B16902409
                                                   Jan 12, 2012@14:47:16
page 2
              104 bytes in comments
XDRMAIN  ;SF-IRMFO/IHS/OHPRD/REDACTED - MAIN DRIVER FOR DUPLICATE MERGE
SOFTWARE;
          [ 08/13/92  09:50 AM ]
         ;;7.3;TOOLKIT;**23**;r 25, 1995
         ;;
START    ;
         S XDRMAINI="MERGE" D ^XDRMAINI G:XDRQFLG END
         F XDRMI1=0:0 S XDRMPAIR=$O(@XDRM("GL")) Q:'XDRMPAIR!(XDRQFLG)  S
XDRMPD
          A="^VA(15,""OT"",""_""""_$P(XDRGL,U,2)_""""_",XDRMPAIR,0)" S
XDRMPDA=
          $O(@XDRMPDA) D MAIN D:'$D(XDRM("NOTALK")) ASK
END      D EOJ
         Q
         ;
MAIN     ;
         S XDRMCD=$P(XDRMPAIR,U,1),XDRMCD2=$P(XDRMPAIR,U,2)
         S XDRMRG("LCK")="+" D LOCK^XDRU1 K XDRMRG("LCK") I $D(XDRMLOCK) G
MAINX
         I '$D(XDRM("NOVERIFY")) S XDRMRG=0 D ^XDRMVFY G:'XDRMRG!(XDRQFLG)
MAINX
         S
(XDRMRG("FR"),XDRMAIN("FR"))=$S($P(^VA(15,XDRMPDA,0),U,4)=2:XDRMCD2,1
          :XDRMCD)

.
.
.
```

27.12.2.1.2 Example 2

Specifying a Build Name:

Figure 285: XINDEX—Direct Mode Utilities Sample User Entries: Specifying a Build Name

```
>D ^XINDEX

                   V. A.  C R O S S  R E F E R E N C E R  7.3
                       [2008 VA Standards & Conventions]
                   UCI: KRN CPU: KRN    Jan 12, 2012@14:47:16

All Routines? No => <Enter>  No

Routine: <Enter>
0 routines


Select BUILD NAME: XT*7.3*102 <Enter>  TOOLKIT
Include the compiled template routines: N// <Enter>

Print more than compiled errors and warnings? YES// <Enter>

Print summary only? NO// <Enter>

Print routines? YES// <Enter>


Print (R)egular,(S)tructured or (B)oth?  R// <Enter>
Print the DDs, Functions, and Options? YES// <Enter>

Print errors and warnings with each routine? YES// <Enter>


Save parameters in ROUTINE file? NO// <Enter>
Index all called routines? NO// <Enter>
DEVICE: ;P-OTHER <Enter>  Telnet Terminal    Right Margin: 255// 80


                   V. A.  C R O S S  R E F E R E N C E R  7.3
                       [2008 VA Standards & Conventions]
                   UCI: KRN CPU: KRN    Jan 12, 2012@14:43:02

The BUILD file Data Dictionaries are being processed.

The option and function files are being processed.


Routines are being processed.
Routines: 1  Faux Routines: 0

XTPOST

--- CROSS REFERENCING ---


Compiled list of Errors and Warnings              Jan 12, 2012@14:59:51
page 1

XTPOST   * *  106 Lines,  3234 Bytes, Checksum: B14328994
```

```
          ;;8.0;KERNEL;**102**;Jul 10, 1995
   XTPOST+1      S - 2nd line of routine violates the SAC.
          .S $P(^%ZRTL(3.091,0),U)="RESPONSE TIME"
   CHECK+34      S - Set to a '%' global.
          .S $P(^%ZRTL(3.091,0),U,2)="3.091P"
   CHECK+35      S - Set to a '%' global.
          .S $P(^%ZRTL(3.092,0),U)="RT DATE_UCI,VOL"
   CHECK+38      S - Set to a '%' global.
          .S $P(^%ZRTL(3.092,0),U,2)="3.092"
   CHECK+39      S - Set to a '%' global.
          .S $P(^%ZRTL(3.094,0),U)="RT RAWDATA"
   CHECK+42      S - Set to a '%' global.
          .S $P(^%ZRTL(3.094,0),U,2)="3.094D"
   CHECK+43      S - Set to a '%' global.

--- Routine Detail   --- with REGULAR ROUTINE LISTING ---
.
.
.
```

27.12.2.1.3 Example 3

Specifying a Package Name:

Figure 286: XINDEX—Direct Mode Utilities Sample User Entries: Specifying a Package Name

```
KRN>D ^XINDEX


                    V. A.  C R O S S  R E F E R E N C E R  7.3
                         [2008 VA Standards & Conventions]
                    UCI: KRN CPU: KRN    Jan 12, 2012@15:01:53

All Routines? No => <Enter> No

Routine: XDRMAIN
Routine: <Enter>
1 routine<Enter>

Select BUILD NAME: <Enter>
Select INSTALL NAME: <Enter>
Select PACKAGE NAME: KERNEL <Enter>    XU

Include the compiled template routines: N// <Enter>

Print more than compiled errors and warnings? YES// <Enter>

Print summary only? NO// <Enter>

Print routines? YES// <Enter>


Print (R)egular,(S)tructured or (B)oth?  R// <Enter>

Print the DDs, Functions, and Options? YES// <Enter>

Print errors and warnings with each routine? YES// <Enter>

Save parameters in ROUTINE file? NO// <Enter>

Index all called routines? NO// <Enter>
DEVICE: ;P-OTHER <Enter>  Telnet Terminal    Right Margin: 255// 80


                    V. A.  C R O S S  R E F E R E N C E R  7.3
                         [2008 VA Standards & Conventions]
                    UCI: KRN CPU: KRN    Jan 12, 2012@15:01:53

The package file Data Dictionaries are being processed.

The option and function files are being processed.


Routines are being processed.
Routines: 1  Faux Routines: 2
```

```
XDRMAIN

          Data Dictionaries
|func          |opt

--- CROSS REFERENCING ---


Compiled list of Errors and Warnings              Jan 12, 2012@15:01:53
page 1

|opt     * *  974 Lines,  35949 Bytes, Checksum:
         I '$P(^VA(200,DO,0),U,11),$P(^(0),U,4)="@"!($N(^("FOF",0))>0)
   161+4       F - Undefined Function.
   589+2       F - Reference to routine '^XUCSPRG'. That isn't in this
UCI.

--- Routine Detail   --- with REGULAR ROUTINE LISTING ---
.
.
.
```

27.12.3 Analysis of XINDEX Error Findings by Category

27.12.3.1 Fatal M Errors (Hard MUMPS Error)

These are unrecoverable errors that cause a program to fail if the commands are executed. It is possible, however, that these types of errors might exist in routines that run correctly. The error occurs (or may occur, depending on the underlying implementation) only when the errant commands are executed.

27.12.3.1.1 F—Bad Number

XINDEX can only check static numbers in code. It does *not* check the boundaries of the number, only that it is a legitimate number and *not* a string.

27.12.3.1.2 F—Bad WRITE syntax

This error is usually a **WRITE** argument misuse. The most common occurrence is due to a missing comma after the argument.

27.12.3.1.3 F—Block structure mismatch

These are potentially one of the most serious types of errors, and may lead to fatal runtime exceptions. However, examination of a number of routines indicates that a significant number of these errors are empty **DO** blocks. These are still potential logic errors, but do *not* cause runtime exceptions under Caché. The **DO** command, Section 8.2.3 of the standard, does *not* seem to have a provision for empty blocks, so this is an error.

Figure 287 is a code extract from ENGET^DGRUGMFU is an example of this type of error:

Figure 287: F - Block structure mismatch—Sample Code Error

```
ENGET() ;DETERMINE DIVISION TO GET SUBSCRIBERS
 ;
N I,J,X
       F I=1:1 X HLNEXT Q:HLQUIT'>0  D
       .S X(I)=HLNODE,J=0
       ..F  S J=$O(HLNODE(J)) Q:'J  S X(I,J)=HLNODE(J)
```

Because there is no **DO** command before the double dot syntax, that line is never executed.

27.12.3.1.4 F—Call to missing label '*label*' in this routine

In this case, reference is made to a label inside a routine that is *not* (or no longer) present. There could be many reasons for this. The most likely candidate being removal of code that is no longer used.

27.12.3.1.5 F—Call to this *label/routine* (MISSING LABEL)

This is the complementary situation in which code calls a label/routine that is no longer present on the system. Again, there are a number of reasons why this might occur, including typographical errors and removal of code that is no longer used.

27.12.3.1.6 F—Command missing an argument

This is another syntax type error. Most M command arguments are optional. This error is usually associated with the **WRITE** argument tab character, which is the question mark (**?**). It *must* be followed by an integer or variable.

27.12.3.1.7 F—Error in pattern code

XINDEX checks that only the seven pattern codes (i.e., **ACELNPU**) of the 1995 M Standard are used. They also can be lowercase (i.e., **acelnpu**). The seven pattern codes are defined as:

- **A**—Alphabetic
- **C**—Control
- **E**—Every Character
- **L**—Lowercase

- **N**—Numeric
- **P**—Punctuation
- **U**—Uppercase

27.12.3.1.8 F—FOR Command followed by only one space

This error is only for the argumentless **FOR** command. It *must* be followed by two spaces.

27.12.3.1.9 F—FOR Command did not contain '='

XINDEX checks that if the **FOR** command has an argument, it *must* set a variable.

27.12.3.1.10 F—General Syntax Error

This error indicates a construct that is *not* valid M syntax and is otherwise unrecognized. Almost any malformed code is possible here.

27.12.3.1.11 F—GO or DO mismatch from block structure (M45)

This is another error that has to do with the **dot** syntax used to create anonymous blocks in standard M. Typically, a **GOTO** that jumps from one stack level to another would generate this type of error.

Figure 288: F—GO or DO mismatch from block structure (M45)—Sample Code Error

```
TEST         ;test routine
             F I=1:1 D
             . S X=1,Y=Z
             .I Y>0 G QUIT^TESTA
             .S Z=0
```

In this example, the code is trying to **GO** out of the **DO** block to another routine.

27.12.3.1.12 F—Invalid or wrong number of arguments to a function

This error involves calling functions with the wrong number of arguments, or with invalid argument syntax.

27.12.3.1.13 F—Label is not valid

M allows the arguments to commands (e.g., **DO**) to be specified indirectly (i.e., via the **@** syntax). What is *not* standard, however, is to use indirection just to specify the *label* in a label^routine combination.

The following code extract from **EN+6^MXMLPRSE** is invalid:

Figure 289: F - Label is not Valid—Sample Code Error

```
F  Q:EOD  D READ,EPOS,@ST^MXMLPRS0:'EOD
```

27.12.3.1.14 F—Missing argument to a command post-conditional

Most M commands allow a post condition, which is designated by a colon and followed by the argument. This error occurs if the argument is missing.

27.12.3.1.15 F—Non-standard (Undefined) 'Z' command

XINDEX flags all uses of **Z** commands. Vendor-specific commands use the **Z** prefix. The SAC restricts the use of such commands to Kernel. You may occasionally see other packages make use of these commands, but in these cases, an exemption is required.

27.12.3.1.16 F—Quoted string not followed by a separator

XINDEX checks that anywhere a quoted string is used, it *must* stand alone or have a separator after it.

27.12.3.1.17 F—Reference to routine '^*routine name*'. That isn't in this UCI

These errors flag references to routines that are *not* present on the system.

27.12.3.1.18 F—UNDEFINED COMMAND (rest of line not checked)

This is a syntax error. It requires a manual check of the line/routine.

27.12.3.1.19 F—Undefined Function

Checks that a function is part of the M standard.

27.12.3.1.20 F—Undefined Special Variable

This is essentially the same as the "F - Undefined Function" error. The only difference is that in M special variables are built-in functions that take no arguments.

27.12.3.1.21 F—Unmatched Parenthesis

This is a syntax error. XINDEX checks that the static code has matching parenthesis. It does have problems when indirection is used, which are evaluated during execution.

27.12.3.1.22 F—Unmatched Quotation Marks

This is a syntax error. XINDEX checks that the static code has matching quotation marks. It does have problems when indirection is used, which are evaluated during execution.

27.12.3.1.23 F—Unrecognized argument in SET command

XINDEX checks the syntax of the **SET** statement. It does have problems when indirection is used, which are evaluated during execution.

27.12.3.2 Warning Violation Errors (According to VA Conventions)

These are potential problems that are *not* necessarily fatal errors but most likely indicate an error. They require careful implementation.

27.12.3.2.1 W—Blank(s) at end of line

Standard M has very specific whitespace requirements. Some text editors create extra whitespace that is caught by XINDEX.

27.12.3.2.2 W—Duplicate label, (M57)

This is an M standard error. During execution, the first occurrence of the label is executed.

27.12.3.2.3 W—First line label NOT routine name

The first line of VistA routines is required to be a label that is the same as the routine name.

27.12.3.2.4 W—Invalid global variable name

Checks that the global name is uppercase and *not* longer than **eight** characters.

27.12.3.2.5 W—Invalid local variable name

XINDEX checks that the local variable name is uppercase and *not* longer than **sixteen** characters.

27.12.3.2.6 W—Line contains a CONTROL (non-graphic) character

The only *non*-graphic characters permitted in VistA routines are whitespace.

27.12.3.2.7 W—Null line (no commands or comment)

Every line in an M routine *must* contain at least **one** character. The most common single character is the semi-colon (**;**), which denotes a comment.

27.12.3.3 Standards Violation Errors (According to VA Standards)

These are issues that do *not* pertain to the M language *per se*, but rather the requirements of the VA Standards and Conventions (SAC). Issues flagged as Standards Violations can still be syntactically correct M code that follows the portability guidelines, but does *not* follow the more stringent requirements set forth in the SAC.

27.12.3.3.1 S—$View function used

The **$VIEW** function directly examines memory. The use of **$VIEW** is restricted to Kernel and VA FileMan.

27.12.3.3.2 S—Access to SSVN's restricted to Kernel

Structured System Variable Names (SSVNs) are a mechanism used to provide programmatic information to certain system information and are covered in Section 7.1.3 of the M language standard. The use of SSVNs is restricted to Kernel.

Common SSVNs include the following:

- **^$ROUTINE**
- **^$JOB**
- **^$LOCK**
- **^$GLOBAL**

27.12.3.3.3 S—Break command used

The **BREAK** command is prohibited except for Kernel.

If applications ever need to use **BREAK**, they should use **^%ZOSF("BRK")** and **^%ZOSF("NBRK")** instead.

27.12.3.3.4 S—Extended reference

In M, use extended references to refer to routines or globals outside the current environment (called a namespace in Caché). The use of extended references is restricted to Kernel.

27.12.3.3.5 S—First line of routine violates the SAC

Section 2.2.1 of the SAC specifies the format of the first line of a routine as follows:

> 2.2.1 The first line of a routine *must* be in the following format: **routine name<ls>;site/programmer<space>-<space>brief description [optional space];date [time is optional]**.
>
> ZZAA12 ;DALOI/XXX - Example Routine;2/13/07

NOTE: M editors frequently modify the first line of a routine.

27.12.3.3.6 S—2nd line of routine violates the SAC

In VistA, the second line of routines records the following information:

- Package/Application version number
- Package/Application name
- Patches ID numbers (if any applied)
- Original routine creation date & time
- Build number

Section 2.2.2 of the SAC specifies the second line format as follows:

> 2.2.2 The second line of a routine *must* be in the following format: **[LABEL-optional]<ls>;;version number; package name; **pm,...pn**; version date;Build n** where:
>
> ;;1.0;PACKAGE;**pm,…pn**;Feb 1, 2007;Build 1

27.12.3.3.7 S—Patch number '*nnn*' missing from second line

The list of patch numbers *must* fall between the set of asterisks (**) and be separated by commas as shown in Section 2.2.2 of the SAC (see Section 26.13.3.3.6).

27.12.3.3.8 S— 'HALT' command should be invoked through 'G ^XUSCLEAN'

The **HALT** command causes a program to exit; this is *not* a common requirement in VistA. If for some reason a routine needs to halt, you *must* first perform certain housekeeping tasks. Kernel provides an API to cleanly halt a program. Application programs *cannot* use the **HALT** command.

Anomaly

This reported error message is out of date; applications should use H^XUS (see Section 2.4.3 of the SAC).

27.12.3.3.9 S—Kill of a protected variable (*variable name*)

Kernel makes use of certain local variables to maintain a standard environment for processes. Applications *cannot* **KILL** the following variables:

- **DT**
- **DTIME**
- **DUZ**
- **IOST**
- **IOM**
- **U**

27.12.3.3.10 S—Kill of an unsubscripted global

The SAC specifies that unsubscripted globals shall be **KILL**ed:

> 2.3.2.3 The **KILL**ing of unsubscripted globals is prohibited and should be protected. (Special instruction to the site is required to enable the **KILL**ing of an unsubscripted global. Application developers *must* document when calls to EN^DIU2 are made to delete files stored in unsubscripted globals).

27.12.3.3.11 S—Unargumented Kill

Kernel maintains a set of local variables that *cannot* be **SET** or **KILL**ed. The unargumented **KILL** is prohibited except for Kernel.

27.12.3.3.12 S—Exclusive Kill

The use of the exclusive **KILL** is prohibited except for Kernel.

27.12.3.3.13 S—Exclusive or Unargumented NEW command

The exclusive **NEW** command is the same as the exclusive **KILL** and is restricted except for Kernel.

27.12.3.3.14 S—LABEL+OFFSET syntax

The only situation in which application routines are allowed to use the **LABEL+OFFSET** syntax to refer to lines of code is when using **$TEXT** to retrieve data lines. For example, it *cannot* be used in conjunction with a **DO** or **GOTO** command.

27.12.3.3.15 S—Line is longer than 245 bytes

Lines of code *cannot* be longer than **245** bytes.

27.12.3.3.16 S—Lock missing Timeout

In M, a **LOCK** command may include a timeout. If the specified timeout period expires before obtaining the lock, the **LOCK** command fails. In VistA, application programs are required to specify a timeout when using this command. If for some reason it is necessary to use a **LOCK** with no timeout (e.g., to manage collaborating processes), an exemption is required.

NOTE: Kernel can use locks *without* a timeout. Kernel can also use *non*-incremental and unargumented locks.

27.12.3.3.17 S—Lower/Mixed case Variable name used

The rules regarding variable case have been relaxed somewhat in the most recent revision of the SAC. The relevant sections are:

> 2.2.5 The line body *must* contain at least **1** printable character, *must not* exceed **245** characters in length, and *must* contain only the ASCII characters values **32-126**. Line labels, global variable names, system variables, SSVNs, etc. *must* be uppercase.
>
> 2.3.1.1 Local variable names may *not* exceed **sixteen** characters. Namespaced variables may *not* contain lowercase characters. Variables local to a routine, subroutine or **DoDot** may be any case. Any variable containing lowercase characters *must* be **NEW**ed at the beginning of the routine, subroutine or **DoDot**.

27.12.3.3.18 S—Lowercase command(s) used in line

All M commands *must* be uppercase. They can be spelled out or abbreviated to the first character.

27.12.3.3.19 S—Non-Incremental Lock

M allows locks to be one of the following types:

- **Incremental**—Allows a process to maintain multiple locks on the same resource and release them one at a time.
- ***Non*-Incremental**—Either a process obtains the lock or the command fails.

Application programs are required to use the incremental form of the **LOCK** command.

NOTE: This restriction does *not* apply to Kernel.

27.12.3.3.20 S—Non-standard $Z function used

M implementations may provide special functions with names beginning with **$Z**. These are platform dependent. Application programs *cannot* use them.

NOTE: This restriction does *not* apply to Kernel.

27.12.3.3.21 S—Non-standard $Z special variable used

M implementations may provide special variables with names beginning with **$Z**. These are platform dependent. Application programs *cannot* use them.

NOTE: This restriction does *not* apply to Kernel.

27.12.3.3.22 S—'OPEN' command should be invoked through ^%ZIS

Applications *cannot* directly use the **OPEN** and **CLOSE** commands. Instead, they *must* use the Kernel Device Handler.

NOTE: This restriction does *not* apply to Kernel, MailMan, and VA FileMan. See the noted exemptions in Section 2.4.8.1 of the SAC.

27.12.3.3.22.1 Anomaly

This error is a bit misleading, because there are now several APIs other than ^%ZIS that can be used. This includes:

- ^%ZISH
- ^%ZISUTL
- ^%ZISTCP

Regardless, applications *must* use one of the **^%ZIS*** APIs and *cannot* use **OPEN** directly.

REF: For more details of the **CLOSE** command, see the "S—'Close' command should be invoked through 'D ^%ZISC'" section.

27.12.3.3.23 S—'Close' command should be invoked through 'D ^%ZISC'

Kernel's Device Handler encapsulates certain **I/O**-related commands (e.g., **OPEN** and **CLOSE**) and provides a common device abstraction used by VistA applications. Applications are required to use the Device Handler.

At one time, devices were always opened using **D ^%ZIS** and closed using **D ^%ZISC**, but that is no longer true. Kernel provides some additional APIs:

- ^%ZISH for working with host files (that is, operating system files).
- ^%ZISUTL to make working with multiple devices easier.
- ^%ZISTCP for TCP connections.

If a device is opened using OPEN^%ZISUTL, it *must* be closed with CLOSE^%ZISUTL. Do *not* close the device through the **CLOSE** command.

27.12.3.3.24 S—Read command doesn't have a timeout

Application programs *must* provide a timeout (usually the variable **DTIME**) when using the **READ** command. In fact, it is good practice for applications to *not* use **READ** at all, but use the VA FileMan ^%DIR API (commonly known as the "Response Reader"); though, this is *not* a requirement. It is, however, a requirement to use a timeout.

In addition, if a timeout exceeds **300** seconds, you *must* document that fact in the package technical manual.

If for some reason this is inappropriate, an exemption is required.

27.12.3.3.25 S—Routine code exceeds SACC maximum size of 15000 (*nnnnn*)

The maximum routine size for M code and **;;** comments (comments beginning with double semi-colons are considered code) is set to **15K** characters in a routine.

NOTE: An additional **5K** characters in a routine is available for regular comments (i.e., comments beginning with a single semi-colon).

27.12.3.3.26 S—Routine exceeds SACC maximum size of 20000 (*nnnnn*)

The maximum routine size as determined by **^%ZOSF("SIZE")** is set to **20K** for all characters in a routine.

27.12.3.3.27 S—Set to a '%' global

Application programs *cannot* modify globals with names beginning with **%**.

NOTE: This restriction does *not* apply to Kernel.

27.12.3.3.28 S—Should use ‘TASKMAN’ instead of ‘JOB’ command

This is a requirement. Application programs *cannot* start background processes with the **JOB** command, but *must* use one of the APIs provided by TaskMan.

NOTE: This restriction does *not* apply to Kernel.

27.12.3.3.29 S—View command used

The **VIEW** command modifies memory or disk buffers. Use of this command is restricted to Kernel and VA FileMan.

REF: For more details about **VIEW** and **$VIEW**, see the “S—$View function used” section.

27.12.3.3.30 S—Violates VA programming standards

This is something of a catchall category and requires manual review for violations of VA programming standards.

27.12.3.4 Informational Errors

These issues are *not* necessarily errors but still require attention, because they could indicate potential problems.

27.12.3.4.1 I—QUIT Command followed by only one space

This is another whitespace issue. In standard M, a routine is terminated by a single **QUIT** command and a function returns a value with a **QUIT** followed by a single space and then an expression that evaluates to the value to be returned. When you encounter a **QUIT** followed by a space, it is most likely extra whitespace at the end of a line.

27.12.3.4.2 I—Star or pound READ used

In M, **READ** is normally a line-oriented command. However, there are two syntactic variations on the **READ** command where its use is inappropriate:

Figure 290: API—Star or pound READ used—Syntactic Variation (1 of 2)

```
READ *X
```

Reads a single character into **X**.

Figure 291: API—Star or pound READ used—Syntactic Variation (2 of 2)

```
READ X#100
```

Reads **100** contiguous characters (bytes on most M systems) into **X**. Use of so-called star and pound **READ**s was once disallowed, but is now permitted so long as applications follow other relevant standards.

27.12.3.5 Marked Items Errors (Manual Check)

You *must* manually check flagged references under Marked Items.

Currently, Marked Items only apply if a line contains **$TEXT ($T)**. XINDEX records the location of the **$T** code and prints it out under the "Marked Items" sub-header on the XINDEX report, since XINDEX does *not* check the references of a **$T**.

M uses the **$TEXT** function to retrieve lines from a routine, and routines sometimes incorporate data items that are retrieved in this fashion. Section 2.2.4 of the SAC describes the required format for lines referenced by **$TEXT**, which states (in part):

> 2.2.4.1 **LABEL+OFFSET** references will *not* be used except for **$TEXT** references.
>
> 2.2.4.2 Lines referenced by **$TEXT** for use other than to check for the existence of a routine or a line label in that routine *must* be in the following format: **[LABEL-optional]<ls>;;text** or M code.

In standard M, a semicolon (**;**) introduces comments. A double semicolon (**;;**) indicates that the comment should be preserved even if the routine is compiled. The **LABEL+OFFSET** syntax is required to prevent errors that could be introduced if lines are inserted ahead of the label. According to the SAC, if code uses **$T**, the reference *must* start with a double semicolon (**;;**).

28 Unwinder: Developer Tools

28.1 Application Programming Interface (API)

Several APIs are available for developers to work with Kernel Unwinder. These APIs are described below.

28.1.1 EN^XQOR(): Navigating Protocols

Reference Type:	Supported	
Category:	Unwinder	
ICR #:	10101	
Description:	The EN^XQOR 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,"` This 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 [#101] file).
Output:	none.	

28.1.2 EN1^XQOR(): Navigating Protocols

Reference Type:	Supported	
Category:	Unwinder	
ICR #:	10101	
Description:	The EN1^XQOR 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 6 processed protocols that were defined in the OPTION (#19) file.	
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,"` This 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 [#101] file).
Output:	none.	

28.1.3 MSG^XQOR(): Enable HL7 Messaging

Reference Type:	Supported	
Category:	Unwinder	
ICR #:	10101	
Description:	The MSG^XQOR API enables 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 are associated.
	.msgtext:	(required) The array containing the HL7 message.
Output:	none.	

28.1.4 EN^XQORM(): Menu Item Display and Selection

Reference Type: Supported

Category: Unwinder

ICR #: 10140

Description: The EN^XQORM 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.

28.1.5 XREF^XQORM(): Force Menu Recompile

Reference Type: Supported

Category: Unwinder

ICR #: 10140

Description: The XREF^XQORM 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) A VARIABLE POINTER to the protocol that should be recompiled.
Output:	returns:	Returns recompiled menu.

28.1.6 DISP^XQORM1(): Display Menu Selections From Help Code

Reference Type: Supported

Category: Unwinder

ICR #: 10102

Description: The DISP^XQORM1 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 a question mark (**?**).
Output:	returns:	Returns menu selections.

29 User: Developer Tools

29.1 Application Programming Interface (API)

Several APIs are available for developers to work with the user. These APIs are described below.

29.1.1 $$CODE2TXT^XUA4A72(): Get HCFA Text

Reference Type:	Supported	
Category:	User	
ICR #:	1625	
Description:	The $$CODE2TXT^XUA4A72 extrinsic function returns the three parts of the Health Care Financing Administration (HCFA) text from the PERSON CLASS (#8932.1) file based on passing in the Internal Entry Number (IEN) or the VA's **Vcode**.	
Format:	`$$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.

29.1.2 $$GET^XUA4A72(): Get Specialty and Subspecialty for a User

Reference Type: Supported

Category: User

ICR #: 1625

Description: The $$GET^XUA4A72 extrinsic function returns the following:

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).

NOTE: This API was released with Kernel Patch XU*8.0*27.

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 (#200) file.
	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.

29.1.3 $$IEN2CODE^XUA4A72(): Get VA Code

Reference Type: Supported

Category: User

ICR #: 1625

Description: The $$IEN2CODE^XUA4A72 extrinsic function returns the VA CODE from the PERSON CLASS (#8932.1) file 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 (#8932.1) file, an empty string is returned.

NOTE: This API was released with Kernel Patch XU*8.0*27.

Format:	`$$IEN2CODE^XUA4A72(ien)`	
Input Parameters:	**ien**:	(required) Internal Entry Number (IEN) in the PERSON CLASS (#8932.1) file.
Output:	returns:	Returns the VA CODE.

29.1.4 $$DTIME^XUP(): Reset DTIME for USER

Reference Type:	Supported	
Category:	User	
ICR #:	4409	

Description: The $$DTIME^XUP extrinsic function resets the **DTIME** variable for the user identified by the **duz** input parameter. This extrinsic function accepts two parameters:

- IEN or **DUZ** of the user in the NEW PERSON (#200) file.
- IEN of the device in the DEVICE (#3.5) file.

The return value should be assigned to the **DTIME** variable as shown in the examples. This **DTIME** variable is used on all timed **READ**s 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 (#200) file.

ios: (optional) The IEN of the device in the DEVICE (#3.5) file. 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 is based on the first available data found in the following fields/files (listed in search order):

1. TIMED READ (# OF SECONDS) (#200.1) field of the NEW PERSON (#200) file.
2. TIMED READ (# OF SECONDS) (#51.1) field of the DEVICE (#3.5) file.
3. DEFAULT TIMED READ (SECONDS) (#210) field of the KERNEL SYSTEM PARAMETERS (#8989.3) file.
4. (default) If *no* data is available in any of the three fields above, then the return value defaults to **300** seconds.

29.1.4.1 Examples

29.1.4.1.1 Example 1

Sending **DUZ** only, returns the value in the TIMED READ (# OF SECONDS) (#200.1) field in the NEW PERSON (#200) file:

Figure 292: $$DTIME^XUP API—Example 1

```
>S DTIME=$$DTIME^XUP(DUZ)

>W DTIME
1800
```

29.1.4.1.2 Example 2

Sending **DUZ** and **IOS**, returns the value in the TIMED READ (# OF SECONDS) (#200.1) field in the NEW PERSON (#200) file:

Figure 293: $$DTIME^XUP API—Example 2

```
>S DTIME=$$DTIME^XUP(DUZ,IOS)

>W DTIME
1800
```

29.1.4.1.3 Example 3

Sending **IOS** only, returns the value in the TIMED READ (# OF SECONDS) (#51.1) field in the DEVICE (#3.5) file:

Figure 294: $$DTIME^XUP API—Example 3

```
>S DTIME=$$DTIME^XUP(,IOS)

>W DTIME
500
```

29.1.4.1.4 Example 4

Not Sending **DUZ** or **IOS**, returns the value in the DEFAULT TIMED READ (SECONDS) (#210) field in the KERNEL SYSTEM PARAMETERS (#8989.3) file:

Figure 295: $$DTIME^XUP API—Example 4a

```
>S DTIME=$$DTIME^XUP(,)

>W DTIME
400
```

Or:

Figure 296: $$DTIME^XUP API—Example 4b

```
>S DTIME=$$DTIME^XUP()

>W DTIME
400
```

29.1.4.1.5 Example 5

Not Sending **DUZ** or **IOS** *and* no value is in DEFAULT TIMED READ (SECONDS) (#210) field in the KERNEL SYSTEM PARAMETERS (#8989.3) file:

Figure 297: $$DTIME^XUP API—Example 5

```
>S DTIME=$$DTIME^XUP()

>W DTIME
300
```

29.1.5 DUZ^XUP(): Set the DUZ Variable

Reference Type:	Controlled Subscription	
Category:	User	
ICR #:	4129	
Description:	The DUZ^XUP API sets the **DUZ** variable equal to the input parameter. Also, it sets up the components of the **DUZ** array. For example, some applications need the **DUZ** to be a *non*-human user in the NEW PERSON (#200) file, which is identified as an application-specific user name. This allows all filing and database activities appear to have been performed by a *non*-human user instead of by background processes (e.g., Postmaster). Also, when processing information initiated by HL7 messages, **DUZ** may *not* have been defined correctly, or *not* defined at all. This presents a challenge for any HL7-based application, because many VistA applications expect a correct **DUZ** array to be present. For example, when creating a Text Integration Utilities (TIU) note, **DUZ** is *not* part of the input parameter, but it is simply expected to be present. This API allows the application to set the **DUZ** so that it can file TIU notes correctly for providers as indicated by HL7 messages. Also, other applications expect the **DUZ** array to be present by default. It has been observed that failure to populate the correct provider's information in a **DUZ** array may result in a crash or somebody else's **DUZ** being used as a provider for something they did *not* request.	
Format:	`DUZ^XUP(da)`	
Input Parameters:	**da**:	(required) This is the internal entry number (IEN) of the user in the NEW PERSON (#200) file to which you want to set the **DUZ** value and **DUZ** array.
Output Variable:	**DUZ**:	Returns the **DUZ** variable equal to the input parameter. Also, it sets up the components of the **DUZ** array.

29.1.6 $$ACTIVE^XUSER(): Status Indicator

Reference Type:	Supported
Category:	User
ICR #:	2343
Description:	The $$ACTIVE^XUSER extrinsic function returns the active status indicator and latest signon information of a user in the NEW PERSON (#200) file.
Format:	`$$ACTIVE^XUSER(ien)`

Input Parameters:	**ien**:	(required) Internal Entry Number (IEN) of the user to be checked in the NEW PERSON (#200) file.
Output:	returns:	Returns any of the following codes: • **“”—NULL**, no user record found. • **0**—User *cannot* sign on. • **0^DISUSER**—User *cannot* 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.

29.1.6.1 Examples

29.1.6.1.1 Example 1

Figure 298 is an example of an Active User in the NEW PERSON (#200) file:

Figure 298: $$ACTIVE^XUSER API—Example 1

```
>S X=$$ACTIVE^XUSER(1529)

>WRITE X
1^ACTIVE^3030321.093756
```

29.1.6.1.2 Example 2

Figure 299 is an example of a Terminated User in the NEW PERSON (#200) file:

Figure 299: $$ACTIVE^XUSER API—Example 2

```
>S X=$$ACTIVE^XUSER(957)

>WRITE X
0^TERMINATED^2980504
```

29.1.6.1.3 Example 3

Figure 300 is an example of a User with no record in the NEW PERSON (#200) file, returns a **NULL** string:

Figure 300: $$ACTIVE^XUSER API—Example 3

```
>S X=$$ACTIVE^XUSER(999999999)

>W X

>
```

29.1.6.1.4 Example 4

Figure 301 is an example of a User in the NEW PERSON (#200) file with the DISUSER flag set:

Figure 301: $$ACTIVE^XUSER API—Example 4

```
>S X=$$ACTIVE^XUSER(111)

>W X
0^DISUSER
```

29.1.7 $$DEA^XUSER()—Get User's DEA Number

Reference Type:	Supported
Category:	User: DEA ePCS Utility
ICR #:	2343
Description:	The $$DEA^XUSER extrinsic function returns a user's DEA number, if it exists in the DEA# (#53.2) Multiple field in the NEW PERSON (#200) file. If the DEA# (#53.2) field value is **NULL**, the value returned depends on the optional **flag** input parameter.

NOTE: Fee Basis and C&A providers only return DEA# or **NULL**.

NOTE: This API was originally requested as part of the Public Key Infrastructure (PKI) Project. This API was updated with Kernel Patch XU*8.0*580, which was created in support of the Drug Enforcement Agency (DEA) e-Prescribing of Controlled Substances (ePCS) Utility. This utility uses Public Key Infrastructure (PKI) and meets the requirements proposed by the DEA Interim Final Rule (IFR) for Electronic Prescriptions for Controlled Substances effective as of June 1, 2010.

Format:	`$$DEA^XUSER([flag],ien[,date])`	
Input Parameters:	**flag**:	(optional) This flag controls what is returned when the user does *not* have a value in the DEA# (#53.2) field of the NEW PERSON (#200) file. If the **flag** is: • **NULL** or **0**—This routine checks to see if the user has values in the VA# (#53.3) field of the NEW PERSON (#200) file and the (new) FACILITY DEA NUMBER (#52) field of the INSTITUTION (#4) file. If values are found in both of those fields, this routine returns the following: **FACILITY DEA NUMBER (#52) field_"-"_VA# (#53.3) field** • **1**—This routine checks to see if the user has a value in the VA# (#53.3) field of the NEW PERSON (#200) file. If a value is found in that field, this routine returns that field value. Otherwise, this routine returns an empty string.
	ien:	(required) This is the NEW PERSON (#200) file IEN for the entry to be checked.
	date:	(optional) The date to be checked against the DEA# Expiration Date instead of default **DT** (today's date).
Output:	returns:	Returns the DEA#: DEA# (#53.2) field value or the value returned based on the (optional) **flag** input parameter.

29.1.7.1 Examples

29.1.7.1.1 Example 1

The following are the data values for this example:

- DEA# (#53.2) field = **AB1234567**.
- FACILITY DEA NUMBER (#52) field = **VA7654321**.
- VA# (#53.3) field = **789**.

If the **flag** input parameter is **NULL** or **0**, this API would return **AB1234567**.

If the **flag** input parameter is **1**, this API would return **AB1234567**.

29.1.7.1.2 Example 2

The following are the data values for this example:

- DEA# (#53.2) field = **NULL**.
- FACILITY DEA NUMBER (#52) field = **VA7654321**.
- VA# (#53.3) field = **789**.

If the **flag** input parameter is **NULL** or **0**, this API would return **VA7654321-789**.

If the **flag** input parameter is **1**, this API would return **789**.

29.1.7.1.3 Example 3

The following are the data values for this example:

- DEA# (#53.2) field = **NULL**.
- FACILITY DEA NUMBER (#52) field = **VA7654321**.
- VA# (#53.3) field = **NULL**.

If the **flag** input parameter is **NULL** or **0**, this API would return **“”** (an empty string).

If the **flag** input parameter is **1**, this API would return **“”** (an empty string).

In both cases, it returns an empty string.

29.1.7.1.4 Example 4

The following are the data values for this example:

- DEA# (#53.2) field = **NULL**.
- FACILITY DEA NUMBER (#52) field = **VA7654321**.
- VA# (#53.3) field = **789**.
- PROVIDER TYPE (#53.6) field = **FEE BASIS** or **C&A**.

If the **flag** input parameter is **NULL** or **0**, this API would return **“”** (an empty string).

If the **flag** input parameter is **1**, this API would return **“”** (an empty string).

In both cases, it returns an empty string.

29.1.7.1.5 Example 5

The following are the data values for this example:

- DEA# (#53.2) field = **AB1234567**, but expired.
- FACILITY DEA NUMBER (#52) field = **VA7654321**.
- VA# (#53.3) field = **789**.
- PROVIDER TYPE (#53.6) field is *not* = **FEE BASIS** nor **C&A**.

If the **PSOEPCS EXPIRED DEA FAILOVER** XPAR parameter is set to **Yes**, this API would return **VA7654321-789**.

If the **PSOEPCS EXPIRED DEA FAILOVER** XPAR parameter is set to **No**, this API would return **NULL** ("").

29.1.7.1.6 Example 6

The following are the data values for this example:

- DEA# (#53.2) field = **AB1234567**.
- DEA EXPIRATION DATE = **3201105**.

If the **date** parameter “**3201104**" passed in is less than DEA EXPIRATION DATE, this API would return "**AB1234567**".

If the **date** parameter "**3201106**" passed in is greater than DEA EXPIRATION DATE, this API would return **NULL** ("").

29.1.8 $$DETOX^XUSER()—Get Detox/Maintenance ID Number

Reference Type:	Supported
Category:	User: DEA ePCS Utility
ICR #:	2343

Description: The $$DETOX^XUSER extrinsic function obtains the value stored in the DETOX/MAINTENANCE ID NUMBER (#53.11) field in the NEW PERSON (#200) file. It returns one of the following:

- **User's DETOX/MAINTENANCE ID number**—If it exists in the DETOX/MAINTENANCE ID NUMBER (#53.11) field of the NEW PERSON (#200) file.
- **NULL**—If DETOX/MAINTENANCE ID number is **NULL** or the DEA EXPERATION DATE (#747.44) field in the NEW PERSON (#200) file is unpopulated.
- **DEA EXPIRATION DATE (#747.44)**—This date is returned when the DETOX/MAINTENANCE ID number is valid but the DEA EXPIRATION DATE has expired.

NOTE: This API was released with Kernel Patch XU*8.0*580, which was created in support of the Drug Enforcement Agency (DEA) e-Prescribing of Controlled Substances (ePCS) Utility. This utility uses Public Key Infrastructure (PKI) and meets the requirements proposed by the DEA Interim Final Rule (IFR) for Electronic Prescriptions for Controlled Substances effective as of June 1, 2010.

Format:

```
$ $DETOX^XUSER(ien[,date])
```

Input Parameters:

ien:	(required) The IEN of the user in the NEW PERSON (#200) file.
date:	(optional) The date to be checked against the DEA# Expiration Date instead of default **DT** (today's date).

Output:	returns:	Returns: one of the following: • **User's DETOX/MAINTENANCE ID number**—If valid. • **NULL**—DETOX/MAINTENANCE ID number is **NULL** or the DEA EXPERATION DATE (#747.44) field in the NEW PERSON (#200) file is unpopulated. • **DEA EXPIRATION DATE (#747.44)**—When the DETOX/MAINTENANCE ID number is valid but the DEA EXPIRATION DATE has expired.

29.1.9 DIV4^XUSER(): Get User Divisions

Reference Type: Controlled Subscription

Category: User

ICR #: 2533

Description: The DIV4^XUSER API returns all divisions for a user. It returns:

- **1**—If the user has a Division entry in the NEW PERSON (#200) file. It indicates that the array of pointers to the Institution file has been defined.
- **0**—The array of pointers to the INSTITUTION (#4) file 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 (#200) file. 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 (#200) file. It indicates that the array of pointers to the Institution file has been defined.

  The array includes all IENs for the INSTITUTION (#4) file 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 (#200) file entry. The format is:

  ```
  ARRAY([^DIC(4 IEN])
  ```

- **0**—The array of pointers to the INSTITUTION (#4) file has *not* been defined.

29.1.9.1 Example

Figure 302: DIV4^XUSER API—Example

```
>S X=$$DIV4^XUSER(.ZZ,duz)
```

29.1.10 $$LOOKUP^XUSER(): New Person File Lookup

Reference Type: Supported

Category: User

ICR #: 2343

Description: The $$LOOKUP^XUSER extrinsic function does a user lookup on the NEW PERSON (#200) file screening out users that are terminated. You are first asked to enter a name of a user in the NEW PERSON (#200) file. By default, the function then asks if the correct user name was selected. For example:

```
Select NEW PERSON NAME: XUUSER,THREE
Is XUUSER,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 *not* allowed.

Format:	`$$LOOKUP^XUSER([""])`	
Input Parameters:	"":	(optional) This optional input parameter does the following: • **NULL (default)—**Do *not* suppress the NEW PERSON (#200) file name confirmation prompt for each entry selected. • **A—**Screen out terminated users. • **Q—**Suppress the NEW PERSON (#200) file name confirmation prompt for each entry selected. • **AQ—**Screen out terminated users and suppress the NEW PERSON (#200) file name confirmation prompt for each entry selected.
Output:	returns:	Returns the Internal Entry Number (IEN) and name of the user in the NEW PERSON (#200) file entered after the "Select NEW PERSON NAME:" prompt (IEN^NAME).

29.1.10.1 Examples

29.1.10.1.1 Example 1

Figure 303 is an example of a lookup of an active user when *not* passing in the optional **Q** parameter:

Figure 303: $$LOOKUP^XUSER API—Example 1: Showing Confirmation Prompt

```
>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: XUUSER,TWO E <Enter>      TX          COMPUTER
SPECIALIST
Is XUUSER,TWO E the one you want? YES// <Enter>

>W LRDOC
1529^XUUSER,TWO E
```

29.1.10.1.2 Example 2

Figure 304 is an example of a lookup of an active user when passing in the optional **Q** parameter:

Figure 304: $$LOOKUP^XUSER API—Example 2: Suppressing Confirmation Prompt

```
>S LRDOC=$$LOOKUP^XUSER("Q")

Select NEW PERSON NAME: XUUSER,TWO E <Enter>     TX           COMPUTER
SPECIALIST

>W LRDOC
1529^XUUSER,TWO E
```

29.1.10.1.3 Example 3

Figure 305 is an example of a lookup of a terminated user when passing in the optional **A** parameter:

Figure 305: $$LOOKUP^XUSER API—Example 3: Terminated User

```
>S LRDOC=$$LOOKUP^XUSER(“A”)

Select NEW PERSON NAME: XUUSER,EIGHT <Enter>       EX
                 This user was terminated on May 04, 1998
Select NEW PERSON NAME:
```

29.1.11 $$NAME^XUSER(): Get Name of User

Reference Type:	Supported	
Category:	User	
ICR #:	2343	
Description:	The $$NAME^XUSER extrinsic function returns the full name of the specified user in a mixed case displayable format. The user’s given name (i.e., First Last) is returned unless a second parameter of **F** is passed in to get the Family name (i.e., Last,First).	
Format:	`$$NAME^XUSER(ien[,format])`	
Input Parameters:	**ien**:	(required) Internal Entry Number (IEN) of the provider to be checked in the NEW PERSON (#200) file.
	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., “Xuuser,Two”). • **G (default)**—Given (e.g., “Two Xuuser”).
Output:	returns:	Returns user’s family or given name.

29.1.11.1 Examples

29.1.11.1.1 Example 1

Retrieving the user name in Given format:

Figure 306: $$NAME^XUSER API—Example 1

```
>S X=$$NAME^XUSER(1529)

>W X
Two E Xuuser
```

29.1.11.1.2 Example 2

Retrieving the user name in Family format:

Figure 307: $$NAME^XUSER API—Example 2

```
>S X=$$NAME^XUSER(1529,"F")

>W X
Xuuser,Two E.
```

29.1.12 $$PROVIDER^XUSER(): Providers in New Person File

Reference Type:	Supported
Category:	User
ICR #:	2343
Description:	The $$PROVIDER^XUSER extrinsic function indicates any provider in the NEW PERSON (#200) file. The definition of a provider is any entry in the NEW PERSON (#200) file that does *not* have a termination date. A second parameter can also be added to invoke other checks, such as whether or not to inclue visitors.

NOTE: This API was requested to be added by the Computerized Patient Record System (CPRS) Development Team.

Additional parameters may be added in the future in order to perform other tests/checks.

Format: `$$PROVIDER^XUSER(xuda,xuf)`

Input Parameters:	**xuda**:	(required) Internal Entry Number (IEN) of the provider to be checked in the NEW PERSON (#200) file.
	xuf:	(optional) Flag to control processing: • **0** or **Not Passed**—Do *not* include visitors. • **1**—Include visitors.
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.

29.1.12.1 Examples

29.1.12.1.1 Example 1

Figure 308 is an example of an Active Provider in the NEW PERSON (#200) file:

Figure 308: $$PROVIDER^XUSER API—Example 1

```
>S X=$$PROVIDER^XUSER(1529)

>WRITE X
1
```

29.1.12.1.2 Example 2

Figure 309 is an example of a Terminated Provider in the NEW PERSON (#200) file:

Figure 309: $$PROVIDER^XUSER API—Example 2

```
>S X=$$PROVIDER^XUSER(957)

>W X
0^TERMINATED^2980504
```

29.1.12.1.3 Example 3

Figure 310 is an example of a Provider with no record in the NEW PERSON (#200) file, returns a **NULL** string:

Figure 310: $$PROVIDER^XUSER API—Example 3

```
>S X=$$PROVIDER^XUSER(000999999)

>W X

>
```

29.1.13 $$SDEA^XUSER()—Check for Prescribing Privileges

Reference Type: Supported

Category: User: DEA ePCS Utility

ICR #: 2343

Description: The $$SDEA^XUSER extrinsic function uses the following "Privileges Algorithm" to check for prescribing privileges:

- Blank = never answered (Allow all schedules but system to send the following electronic message: "DEA credentials have *not* been populated, call TBD responsible person.")
- Any or all fields are answered = provide explicit set of permissions (that have been identified).
- If it is answered that Prescriber has No privileges for all schedules = remove DEA number or VA number from the NEW PERSON (#200) file.
- If Prescriber has been issued a DEA number, you have privileges.
- If the Prescriber has been issued a VA number, this is a presumption of privileges.

NOTE: Not all of these checks apply to documentation of *non*-VA medication.

REF: This API calls the $$DEA^XUSER()—Get User's DEA Number API.

NOTE: This API was released with Kernel Patch XU*8.0*580, which was created in support of the Drug Enforcement Agency (DEA) e-Prescribing of Controlled Substances (ePCS) Utility. This utility uses Public Key Infrastructure (PKI) and meets the requirements proposed by the DEA Interim Final Rule (IFR) for Electronic Prescriptions for Controlled Substances effective as of June 1, 2010.

Format: `$$SDEA^XUSER([fg,]ien,psdea[,date])`

Input Parameters:

fg: (optional) This flag is used for $$DEA^XUSER call, see the flag input parameter in the $$DEA^XUSER()—Get User's DEA Number API.

ien: (required) This is the NEW PERSON (#200) file IEN for the entry to be checked.

psdea: (required) This parameter is DEA schedule. DEA schedule is a **2-6** position field. It comes from the DRUG (#50) file in Pharmacy. This API uses this field to verify the provider is allowed to write orders for specific controlled substances. For example, if the schedule is **2A**, this indicates a controlled substance, schedule **2**.

Chart for all values:

- MANUFACTURED IN PHARMACY
- SCHEDULE 1 ITEM
- SCHEDULE 2 ITEM
- SCHEDULE 3 ITEM
- SCHEDULE 4 ITEM
- SCHEDULE 5 ITEM
- LEGEND ITEM:
    - **9**—OVER-THE-COUNTER
    - **L**—DEPRESSANTS AND STIMULANTS
    - **A**—NARCOTICS AND ALCOHOLS
    - **P**—DATED DRUGS
    - **I**—INVESTIGATIONAL DRUGS
    - **M**—BULK COMPOUND ITEMS
    - **C**—CONTROLLED SUBSTANCES - NON NARCOTIC
    - **R**—RESTRICTED ITEMS

- **S**—SUPPLY ITEMS
- **B**—ALLOW REFILL (SCH. 3, 4, 5 ONLY)
- **W**—NOT RENEWABLE
- **F**—NON REFILLABLE
- **E**—ELECTRONICALLY BILLABLE
- **N**—NUTRITIONAL SUPPLEMENT
- **U**—SENSITIVE DRUG

	date:	(optional) The date to be checked against the DEA# Expiration Date instead of default **DT** (today's date).
Output:	returns:	Returns: DEA# or **Facility DEA_"-"_user VA#** similar to the $$DEA^XUSER call.

- **1**—DEA# is **NULL** from the $$DEA^XUSER call.
- **2**—When all schedules equals **0**.
- **4^expiration date**—DEA# expiration date has expired. It checks if the DEA# and expiration date are *not* **NULL**. The expiration date is returned in external format.

29.1.14 $$VDEA^XUSER()—Check if User Can Sign Controlled Substance Orders

Reference Type: Supported

Category: User: DEA ePCS Utility

ICR #: 2343

Description: The $$VDEA^XUSER extrinsic function determines if a user in the NEW PERSON (#200) file is able to sign orders for controlled substances.

NOTE: This API was released with Kernel Patch XU*8.0*580, which was created in support of the Drug Enforcement Agency (DEA) e-Prescribing of Controlled Substances (ePCS) Utility. This utility uses Public Key Infrastructure (PKI) and meets the requirements proposed by the DEA Interim Final Rule (IFR) for Electronic Prescriptions for Controlled Substances effective as of June 1, 2010.

Format:	`$VDEA^XUSER(.return,ien)`	
Input Parameters:	**.return**:	(required) This is a reference to an array where the reasons why the user *cannot* sign orders for controlled substances and which DEA schedules the user can prescribe is returned. For example: RETURN("Is permitted to prescribe all schedules.")=""
	ien:	(required) This is the IEN of the user in the NEW PERSON (#200) file.
Output Parameters:	**.return**:	This array contains the reasons why the user *cannot* sign orders for controlled substances and which DEA schedules the user can prescribe. For example: RETURN("Is *not* permitted to prescribe any schedules.")=""
Output:	returns:	Returns: • **1**—If the user is able to sign orders for controlled substances. • **0**—If the user is *not* able to sign orders for controlled substances.

29.1.15 $$KCHK^XUSRB(): Check If User Holds Security Key

Reference Type:	Controlled Subscription	
Category:	User	
ICR #:	2120	
Description:	The $$KCHK^XUSRB extrinsic function checks to see if a user holds a given security key.	
Format:	`$$KCHK^XUSRB(key[,ien])`	
Input Parameters:	**key**:	(required) The name of the security key to be checked.
	ien:	(optional) Internal Entry Number (IEN). It defaults to **DUZ**.
Output:	returns:	Returns: • **1**—User holds security key. • **0**—User does *not* hold security key.

29.1.15.1 Examples

29.1.15.1.1 Example 1

Figure 311 illustrates the results when a user holds a security key input:

Figure 311: $$KCHK^XUSRB API—Example 1

```
>S X=$$KCHK^XUSRB("XUPROGMODE")

>W X
1
```

29.1.15.1.2 Example 2

Figure 312 illustrates the results when a user does *not* hold the security key input:

Figure 312: $$KCHK^XUSRB API—Example 2

```
>S X=$$KCHK^XUSRB("XUMGR")

>W X
0
```

29.1.15.1.3 Example 3

Figure 313 illustrates the results when checking if another user holds a security key input by including their IEN:

Figure 313: $$KCHK^XUSRB API—Example 3

```
>S X=$$KCHK^XUSRB("XUPROGMODE",30)

>W X
1
```

29.1.16 DIVGET^XUSRB2(): Get Divisions for Current User

Reference Type:	Controlled Subscription	
Category:	User	
ICR #:	4055	
Description:	The DIVGET^XUSRB2 API retrieves the list of divisions for the current user. (This was developed as a Broker Remote Procedure Call [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 is 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 #`

29.1.17 DIVSET^XUSRB2(): Set Division for Current User

Reference Type:	Controlled Subscription
Category:	User
ICR #:	4055
Description:	The DIVSET^XUSRB2 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 grave accent (`) an Internal Entry Number (IEN) is being passed and is processed as such.
Output:	**ret()**:	Returns a Boolean value in the subscripted output array:

- **True (*non*-zero)**—Division selection is considered successful.
- **False (zero)**—Division selection failed.

29.1.18 USERINFO^XUSRB2(): Get Demographics for Current User

Reference Type: Controlled Subscription

Category: User

ICR #: 4055

Description: The USERINFO^XUSRB2 API retrieves various user demographic information for the current user.

NOTE: 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:	**ret()**:	Returns a subscripted output array:

- **RET(1)**—User's name from the **.01** field of the NEW PERSON (#200) file.
- **RET(2)**—Concatenated user name from the NAME COMPONENTS(#20) file.
- **RE(3)**—Logged on division:

  ```
  ien^name^number
  ```
- **RET(4)**—User's title from the NEW PERSON (#200) file.
- **RET(5)**—User's service section from NEW PERSON (#200) file (external format).

- **RET(6)**—User’s language from the NEW PERSON (#200) file.
- **RET(7)**—User’s timeout.

30 XGF Function Library: Developer Tools

30.1 Overview

The XGF Function Library supports developers designing text-based applications. The functions in this library support cursor positioning, overlapping text windows, video attribute control, and keyboard escape processing, all in a text-mode environment.

If you intend to make simple interface enhancements for an existing text-mode application, then you may find the XGF Function Library useful. The XGF Function Library provides the following functionality:

- Text-mode overlapping windows.
- Text-mode cursor positioning by screen coordinate.
- Text-mode video attribute control (bold, blink, etc.).
- Keyboard reader using M escape processing (thereby making use of keystrokes like **<UP-ARROW>** ("↑"), **<DOWN-ARROW>** ("↓"), **<PREV>** ("←"), **<NEXT>** ("→"), etc.).

The XGF Function Library may *not* be appropriate if you need:

- A full graphical user interface (GUI) front end for your application.
- Support for ***non*-ANSI VT**-compatible display devices.

To use the XGF Function Library, your system *must* use an M implementation that complies with the 1995 ANSI M standard. At a minimum, the M implementation *must* support the features listed in Table 39 to use the XGF Function Library:

Table 39: XGF Function Library—Minimum M Implementation Features Required

Feature	Example
SET into **$EXTRACT**	`S X="this is a string",$E(X,1,4)="that"`
Reverse **$ORDER**	`S X=$O(^TMP(""),-1)`
Two argument **$GET**	`K Y S X=$G(Y,"DEFAULT")`
Skipping parameters	`D TAG^ROUTINE(,P2,,P4)`
$NAME	`W $NA(^TMP($J))`
SET $X and **$Y**	`S $X=10`

This XGF Function Library supports terminals that are **ANSI-compatible** and at least **VT100-compatible**. As a result, this software does *not* support **QUME QVT102/QVT102A** terminals.

REF: The XGF Function Library Application Programming Interfaces (APIs) are documented in the "XGF Function Library: Developer Tools" section. Kernel and Kernel Toolkit APIs are also available in HTML format on the VA Intranet Website.

30.2 Direct Mode Utilities

Several XGF Function Library direct mode utilities are available for developers to use at the M prompt. They are *not* APIs and *cannot* be used in software application routines. These direct mode utilities are described below.

30.2.1 ^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
```

Table 40: XGF Function Library—Demo Functional Division

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

30.3 Application Programming Interface (API)

Several APIs are available for developers to work with the XGF Function Library. These APIs are described below.

30.3.1 CHGA^XGF(): Screen Change Attributes

Reference Type:	Supported
Category:	XGF Function Library
ICR #:	3173

Description: The CHGA^XGF API changes individual video attributes for subsequent screen **WRITE**s.

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.

30.3.1.1 Examples

30.3.1.1.1 Example 1

To clear the screen in blinking, reverse video and high intensity, do the following:

Figure 314: CHGA^XGF API—Example 1

```
>D CHGA^XGF("R1B1I1"),CLEAR^XGF(0,0,23,79)
```

30.3.1.1.2 Example 2

To print Hello World, do the following:

Figure 315: CHGA^XGF API—Example 2

```
>D CHGA^XGF("I1"),SAY^XGF(,,"Hello ")
>D CHGA^XGF("U1"),SAY^XGF(,,"World")
```

30.3.1.1.3 Example 3

To draw the bottom of a small box, do the following:

Figure 316: CHGA^XGF API—Example 3

```
>D CHGA^XGF("G1")
>D SAY^XGF(,,IOBLC_IOHL_IOHL_IOBRC)
>D CHGA^XGF("G0")
```

30.3.2 CLEAN^XGF: Screen/Keyboard Exit and Cleanup

Reference Type: Supported

Category: XGF Function Library

ICR #: 3173

Description: The CLEAN^XGF API exits the **XGF** screen and keyboard environments. It does the following:

- Removes **XGF** screen and keyboard variables and tables.
- Turns all video attributes off.
- Turns echo on.
- Turns the cursor on.
- 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 **READ**s 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.

30.3.3 CLEAR^XGF(): Screen Clear Region

Reference Type:	Supported	
Category:	XGF Function Library	
ICR #:	3173	
Description:	The CLEAR^XGF 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.

30.3.3.1 Examples

30.3.3.1.1 Example 1

For example, to clear the entire screen, do the following:

Figure 317: CLEAR^XGF API—Example 1

```
>D CLEAR^XGF(0,0,23,79)
```

30.3.3.1.2 Example 2

To clear a rectangular region in the center of the screen, do the following:

Figure 318: CLEAR^XGF API—Example 2

```
>D CLEAR^XGF(5,20,15,60)
```

30.3.4 FRAME^XGF(): Screen Frame

Reference Type: Supported

Category: XGF Function Library

ICR #: 3173

Description: The FRAME^XGF 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.

30.3.4.1 Example

For example, to draw a box in the center of the screen, do the following:

Figure 319: FRAME^XGF API—Example

```
>D FRAME^XGF(5,20,15,60)
```

30.3.5 INITKB^XGF(): Keyboard Setup Only

Reference Type: Supported

Category: XGF Function Library

ICR #: 3173

Description: The INITKB^XGF 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.

30.3.6 IOXY^XGF(): Screen Cursor Placement

Reference Type:	Supported
Category:	XGF Function Library
ICR #:	3173

Description: The IOXY^XGF 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,column)`

Input Parameters:	**row**:	(required) Row position to which the cursor is moved.
	column:	(required) Column position to which the cursor is moved.
Output Variables:	**$X** and **$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.

30.3.6.1 Example

For example, to position the cursor at row **12**, column **39**, do the following:

Figure 320: IOXY^XGF API—Example

```
>D IOXY^XGF(12,39)
```

30.3.7 PREP^XGF(): Screen/Keyboard Setup

Reference Type: Supported

Category: XGF Function Library

ICR #: 3173

Description: The PREP^XGF API sets up the **XGF** screen and keyboard environments.

Before using any **XGF** screen functions, you *must* call the PREP^XGF API. PREP^XGF does the following:

- Sets up screen control variables and tables.
- Turns off all video attributes.
- Turns echo off.
- Turns the cursor off.
- Sets the keypad to application mode.
- 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.

NOTE: 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 Parameters: **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.

30.3.8 $$READ^XGF(): Read Using Escape Processing

Reference Type:	Supported
Category:	XGF Function Library
ICR #:	3173

Description: The $$READ^XGF extrinsic function provides a way to perform **READ**s using escape processing. **READ**s, when escape processing is turned on, are terminated by:

- **<UP-ARROW>** ("↑")
- **<DOWN-ARROW>** ("↓")
- **<PREV>** ("←")
- **<NEXT>** ("→")
- **<TAB>**
- 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 do *not* 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 **READ**s is:

Table 41: XGF Function Library—Mnemonics for Keys that Terminate READs

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**

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 Variables:

XGRT: Set to the mnemonic of the key that terminated the **READ**.

REF: For a list of possible values, see the list in Table 41 or the table in routine **XGKB**.

DTOUT: If defined, signifies that the **READ** timed out.

Output: returns: Returns the string **READ** from the user.

30.3.8.1 Examples

30.3.8.1.1 Example 1

To **READ** a name (with a maximum length of **30**) from input and display that name on the screen, do the following:

Figure 321: SAY^XGF API—Example 1: READ a Name

```
D INITKB^XGF("*")
W "Name: " S NM=$$READ^XGF(30)
D SAY^XGF(10,20,"Hello "_NM)
```

30.3.8.1.2 Example 2

To accept only **<Up-Arrow> ("↑")** or **<Down-Arrow> ("↓")** keys to exit a routine, do the following:

Figure 322: $$READ^XGF API—Example 2: Accept Only Up-Arrow ("↑") and Down-Arrow ("↓") Keys

```
;Only accept UP or DOWN arrow keys
F  S %=$$READ^XGF(1) Q:XGRT="UP"!(XGRT="DOWN")
```

NOTE: When you set up the **XGF** keyboard environment using INITKB^XGF rather than PREP^XGF, the keypad is *not* automatically set to application mode. For **READ**s 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:

```
X=$X W IOKPAM S $X=X
```

30.3.9 RESETKB^XGF: Exit XGF Keyboard

Reference Type:	Supported
Category:	XGF Function Library
ICR #:	3173
Description:	The RESETKB^XGF 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 **READ**s 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.

30.3.10 RESTORE^XGF(): Screen Restore

Reference Type:	Supported
Category:	XGF Function Library
ICR #:	3173
Description:	The RESTORE^XGF 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 Variables:	**$X** and **$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.

30.3.10.1 Example

To restore the screen contents saved to the local array SELECT to their original position, do the following:

Figure 323: RESTORE^XGF API—Example

```
>D RESTORE^XGF("SELECT")
```

30.3.11 SAVE^XGF(): Screen Save

Reference Type: Supported

Category: XGF Function Library

ICR #: 3173

Description: The SAVE^XGF 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 Variables: **$X** and **$Y**: Left unchanged.

REF: See also: CLEAR^XGF(): Screen Clear Region, RESTORE^XGF(): Screen Restore, and WIN^XGF(): Screen Text Window APIs.

30.3.11.1 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:

Figure 324: SAVE^XGF API—Example

```
>D SAVE^XGF(5,20,15,60,"SELECT")
```

30.3.12 SAY^XGF(): Screen String

Reference Type: Supported

Category: XGF Function Library

ICR #: 3173

Description: The SAY^XGF 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, see the SETA^XGF(): Screen Video Attributes API.

Format: `SAY^XGF([row][,column,]str[,atr])`

Input Parameters:

row:	(optional) Row position to start **WRITE**.
column:	(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, see the $$READ^XGF(): Read Using Escape Processing API.

Output Variables: **$X** and **$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.

30.3.12.1 Examples

30.3.12.1.1 Example 1

For example, to print "**Hello, World**" in the center of the screen, in the current video attribute, do the following:

Figure 325: SAY^XGF API—Example 1

```
>D SAY^XGF(11,35,"Hello World")
```

30.3.12.1.2 Example 2

To print "**ERROR!**" at (row,col) position (**$X+1,$Y+5**), in reverse and bold video attributes, do the following:

Figure 326: SAY^XGF API—Example 2

```
>D SAY^XGF("+","+5","ERROR!","R1B1")
```

30.3.12.1.3 Example 3

To print "**...**" at the current cursor position, in the current video attribute, do the following:

Figure 327: SAY^XGF API—Example 3

```
>D SAY^XGF(,,"...")
```

30.3.13 SAYU^XGF(): Screen String with Attributes

Reference Type:	Supported
Category:	XGF Function Library
ICR #:	3173
Description:	The SAYU^XGF 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][,column,]str[,atr])`

Input Parameters:	**row**:	(optional) Row position to start **WRITE**.
	column:	(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, see the $$READ^XGF(): Read Using Escape Processing API.
Output Variables:	**$X,$Y:**	Set to the position of the last character output.

REF: See also: IOXY^XGF(): Screen Cursor Placement and SAY^XGF(): Screen String APIs.

30.3.13.1 Example

For example, to print Save at row 5, column 10, do the following:

Figure 328: SAYU^XGF API—Example

```
>D SAYU^XGF(5,10,"&Save")
```

30.3.14 SETA^XGF(): Screen Video Attributes

Reference Type:	Supported	
Category:	XGF Function Library	
ICR #:	3173	

Description: The SETA^XGF 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 Variables:	**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.

30.3.14.1 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:

Figure 329: SETA^XGF API—Example

```
>D PREP^XGF S SAVEATR=XGCURATR
>D SOME^THING
>D SETA^XGF(SAVEATR)
```

30.3.15 WIN^XGF(): Screen Text Window

Reference Type:	Supported
Category:	XGF Function Library
ICR #:	3173

Description: The WIN^XGF 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.

Output Variables: $X and **$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.

30.3.15.1 Examples

30.3.15.1.1 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:

Figure 330: WIN^XGF API—Example 1

```
>D WIN^XGF(5,20,15,60,"SELECT")
```

30.3.15.1.2 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:

Figure 331: WIN^XGF API—Example 2

```
>D WIN^XGF(5,20,15,60,$NA(^TMP($J)))
```

31 XLF Function Library: Developer Tools

31.1 Overview

Several APIs are available for developers to work with the XLF Function Library. These APIs are described in the sections that follow.

The XLF Function Library provides the following functions:

- Bitwise Logic Functions—XLFSHAN
- CRC Functions—XLFCRC
- Date Functions—XLFDT
- Hyperbolic Trigonometric Functions—XLFHYPER
- Mathematical Functions—XLFMTH
- Measurement Functions—XLFMSMT
- String Functions—XLFSTR
- Utility Functions—XLFUTL
- IP Address Functions—XLFIPV
- JSON Conversion Functions—XLFJSON

31.2 Application Programming Interface (API)

31.3 Bitwise Logic Functions—XLFSHAN

These functions help process bitwise logic[1].

31.3.1 $$AND^XLFSHAN(): Bitwise Logical AND

Reference Type:	Supported
Category:	Bitwise Logic Functions
ICR #:	6157
Description:	The $$AND^XLFSHAN extrinsic function performs a bitwise logical **AND** of two **32** bit integers.

NOTE: This API was released with Kernel Patch XU*8.0*657.

[1] Wikipedia Definition for "Bitwise operation:" https://en.wikipedia.org/wiki/Bitwise_operation

Format:	`$$AND^XLFSHAN(x,y)`	
Input Parameters:	**x**:	(required) An integer of **32** bits or less.
	y:	(required) An integer of **32** bits or less.
Output:	returns:	Returns the bitwise logical **AND**.

31.3.1.1 Example

Figure 332: $$AND^XLFSHAN API—Example

```
>W $$AND^XLFSHAN(345,123)
89
```

31.3.2 $$OR^XLFSHAN(): Bitwise Logical OR

Reference Type: Supported

Category: Bitwise Logic Functions

ICR #: 6157

Description: The $$OR^XLFSHAN extrinsic function performs a bitwise logical **OR** of two **32** bit integers.

NOTE: This API was released with Kernel Patch XU*8.0*657.

Format:	`$$OR^XLFSHAN(x,y)`	
Input Parameters:	**x**:	(required) An integer of **32** bits or less.
	y:	(required) An integer of **32** bits or less.
Output:	returns:	Returns the bitwise logical **OR**.

31.3.2.1 Example

Figure 333: $$OR^XLFSHAN API—Example

```
>W $$OR^XLFSHAN(345,123)
379
```

31.3.3 $$XOR^XLFSHAN(): Bitwise Logical XOR

Reference Type:	Supported	
Category:	Bitwise Logic Functions	
ICR #:	6157	
Description:	The $$XOR^XLFSHAN extrinsic function performs a bitwise logical **XOR** of two **32** bit integers. **NOTE:** This API was released with Kernel Patch XU*8.0*657.	
Format:	`$$XOR^XLFSHAN(x,y)`	
Input Parameters:	**x**:	(required) An integer of **32** bits or less.
	y:	(required) An integer of **32** bits or less.
Output:	returns:	Returns the bitwise logical **XOR**.

31.3.3.1 Example

Figure 334: $$XOR^XLFSHAN API—Example

```
>W $$XOR^XLFSHAN(345,123)
290
```

31.4 CRC Functions—XLFCRC

These functions are provided to help process strings.

31.4.1 $$CRC16^XLFCRC(): Cyclic Redundancy Code 16

Reference Type: Supported

Category: CRC Functions

ICR #: 3156

Description: The $$CRC16^XLFCRC extrinsic function computes a Cyclic Redundancy Code (CRC) of the **8**-bit character string, using the following as the polynomial:

X^16 + X^15 + X^2 + 1

The optional **seed** input parameter can supply an initial value, which allows for running CRC calculations on multiple strings. If the **seed** input parameter is *not* specified, a default value of **zero** (**0**) is assumed. The **seed** value is limited to **0 <= seed <= 2^16**. The **function** value is 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.

31.4.1.1 Examples

31.4.1.1.1 Example 1

```
SET CRC=$$CRC16^XLFCRC(string)
```

A checksum can also be calculated over multiple strings.

Figure 335: $$CRC16^XLFCRC API—Example 1: Calculating a Checksum over Multiple Strings (1 of 2)

```
SET (I,C)=0
FOR  SET I=$ORDER(X(I)) QUIT:'I  DO
. SET C=$$CRC16^XLFCRC(X(I),C)
```

Or:

Figure 336: $$CRC16^XLFCRC API—Example 1: Calculating a Checksum over Multiple Strings (2 of 2)

```
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.

31.4.1.1.2 Example 2

Figure 337: $$CRC16^XLFCRC API—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.

31.4.2 $$CRC32^XLFCRC(): Cyclic Redundancy Code 32

Reference Type: Supported

Category: CRC Functions

ICR #: 3156

Description: The $$CRC32^XLFCRC extrinsic function computes a Cyclic Redundancy Code (CRC) of the **8**-bit character string, using the following as the polynomial:

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

The optional **seed** input parameter can supply an initial value, which allows for running CRC calculations on multiple strings. If the **seed** input parameter is *not* specified, a default value of **4,294,967,295 (2^32-1)** is assumed. The **seed** value is limited to **0 <= seed <= 2^32**. The **function** value is 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.

31.4.2.1 Examples

31.4.2.1.1 Example 1

```
SET CRC=$$CRC32^XLFCRC(string)
```

A checksum can also be calculated over multiple strings.

Figure 338: $$CRC32^XLFCRC API—Example 1: Calculating a Checksum over Multiple Strings (1 of 2)

```
SET (I,C)=0
FOR  SET I=$ORDER(X(I)) QUIT:'I  DO
. SET C=$$CRC32^XLFCRC(X(I),C)
```

Or:

Figure 339: $$CRC32^XLFCRC API—Example 1: Calculating a Checksum over Multiple Strings (2 of 2)

```
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.

31.4.2.1.2 Example 2

Figure 340: $$CRC32^XLFCRC API—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.

31.5 Date Functions—XLFDT

31.5.1 $$%H^XLFDT(): Convert Seconds to $H

Reference Type:	Supported	
Category:	Date Functions	
ICR #:	10103	
Description:	The $$%H^XLFDT extrinsic function converts the number of seconds input to a **$H** formatted date. It converts the output of the $$SEC^XLFDT(): Convert $H/VA FileMan date to Seconds API back to a **$H** value.	
Format:	`$$%H^XLFDT(seconds)`	
Input Parameters:	**seconds**:	(required) Input seconds.
Output:	returns:	Returns seconds in **$H** date format.

31.5.1.1 Example

Figure 341: $$%H^XLFDT API—Example

```
>S X=$$%H^XLFDT(5108536020)

>W X
59126,49620
```

31.5.2 $$DOW^XLFDT(): Day of Week

Reference Type:	Supported	
Category:	Date Functions	
ICR #:	10103	
Description:	The $$DOW^XLFDT extrinsic function returns the corresponding day of the week from a date in VA FileMan format.	
Format:	`$$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.

31.5.2.1 Examples

31.5.2.1.1 Example 1

Figure 342: $$DOW^XLFDT API—Example 1

```
>S X=$$DOW^XLFDT(2901231.111523)

>W X
Monday
```

31.5.2.1.2 Example 2

Figure 343: $$DOW^XLFDT API—Example 2

```
>S X=$$DOW^XLFDT(2901231.111523,1)

>W X
1
```

31.5.3 $$DT^XLFDT: Current Date (VA FileMan Date Format)

Reference Type:	Supported	
Category:	Date Functions	
ICR #:	10103	
Description:	The $$DT^XLFDT extrinsic function returns the current date in VA FileMan format.	
Format:	`$$DT^XLFDT`	
Input Parameters:	none.	
Output:	returns:	Returns the current date in VA FileMan format.

31.5.3.1 Example

Figure 344: $$DT^XLFDT API—Example

```
>S X=$$DT^XLFDT

>W X
3040126
```

31.5.4 $$FMADD^XLFDT(): VA FileMan Date Add

Reference Type:	Supported	
Category:	Date Functions	
ICR #:	10103	
Description:	The $$FMADD^XLFDT extrinsic function returns the result of adding days, hours, minutes, and seconds to a date in VA FileMan format.	
Format:	`$$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.

31.5.4.1 Example

Figure 345: $$FMADD^XLFDT API—Example

```
>S X=$$FMADD^XLFDT(2901231.01,2,2,20,15)

>W X
2910102.032015
```

31.5.5 $$FMDIFF^XLFDT(): VA FileMan Date Difference

Reference Type:	Supported
Category:	Date Functions
ICR #:	10103
Description:	The $$FMDIFF^XLFDT extrinsic function returns the difference between two VA FileMan format dates.
Format:	`$$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.

31.5.5.1 Examples

31.5.5.1.1 Example 1

Figure 346 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:

Figure 346: $$FMDIFF^XLFDT API—Example 1

```
>S X=$$FMDIFF^XLFDT(2901229,2901231.111523)

>W X
-2

>S X=$$FMDIFF^XLFDT(2901229,2901231.111523,1)

>W X
-2
```

31.5.5.1.2 Example 2

Figure 347 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:

Figure 347: $$FMDIFF^XLFDT API—Example 2

```
>S X=$$FMDIFF^XLFDT(2901231.111523,2901229.173404,2)

>W X
150079
```

31.5.5.1.3 Example 3

Figure 348 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:

Figure 348: $$FMDIFF^XLFDT API—Example 3

```
>S X=$$FMDIFF^XLFDT(2901231.024703,2901230.012301,3)

>W X
1 1:24:2
```

31.5.6 $$FMTE^XLFDT(): Convert VA FileMan Date to External Format

Reference Type:	Supported
Category:	Date Functions
ICR #:	10103
Description:	The $$FMTE^XLFDT extrinsic function converts a VA FileMan formatted input date to an external formatted date.
Format:	`$$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.

31.5.6.1 Examples

31.5.6.1.1 Example 1

Return the date in the following format: Standard VA FileMan date format.

Figure 349: $$FMTE^XLFDT API—Example 1: Standard VA FileMan Date Format

```
>S X=$$FMTE^XLFDT(2940629.105744,1)

>W X
Jun 29, 1994@10:57:44
```

31.5.6.1.2 Example 2

Return the date in the following format: Standard VA FileMan date format and include am/pm.

Figure 350: $$FMTE^XLFDT API—Example 2: Standard VA FileMan Date Format and Including am/pm

```
>S X=$$FMTE^XLFDT(2940629.1057,"1P")

>W X
Jun 29, 1994 10:57 am
```

31.5.6.1.3 Example 3

Return the date in the following format: **MM/DD/YY@HH:MM:SS**.

Figure 351: $$FMTE^XLFDT API—Example 3: MM/DD/YY@HH:MM:SS Format

```
>S X=$$FMTE^XLFDT(2940629.105744,2)

>W X
6/29/94@10:57:44
```

31.5.6.1.4 Example 4

Return the date in the following format: **MM/DD/YY@HH:MM**.

Figure 352: $$FMTE^XLFDT API—Example 4: MM/DD/YY@HH:MM Format

```
>S X=$$FMTE^XLFDT(2940629.105744,"2M")

>W X
6/29/94@10:57
```

31.5.6.1.5 Example 5

Return the date in the following format: **MM/DD/YY@HH:MM:SS** and include am/pm.

Figure 353: $$FMTE^XLFDT API—Example 5: MM/DD/YY@HH:MM:SS Format and Including am/pm

```
>S X=$$FMTE^XLFDT(2940629.105744,"2P")

>W X
6/29/94 10:57:44 am
```

31.5.6.1.6 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.

Figure 354: $$FMTE^XLFDT API—Example 6: MM/DD/YY@HH:MM:SS Format with Forced Seconds Displayed

```
>S X=$$FMTE^XLFDT(2940629.1057,"2S")

>W X
6/29/94@10:57:00
```

31.5.6.1.7 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.

Figure 355: $$FMTE^XLFDT API—Example 7: MM/DD/YY@HH:MM:SS Format Including Leading Spaces and with Forced Seconds Displayed

```
>S X=$$FMTE^XLFDT(2940629.1057,"2SF")

>W X
 6/29/94@10:57:00
```

31.5.6.1.8 Example 8

Return the date in the following format: **DD/MM/YY@HH:MM:SS** and include leading spaces.

Figure 356: $$FMTE^XLFDT API—Example 8: DD/MM/YY@HH:MM:SS Format Including Leading Spaces

```
>S X=$$FMTE^XLFDT(2940629.105744,"3F")

>W X
29/ 6/94@10:57:44
```

31.5.6.1.9 Example 9

Return the date in the following format: **YY/MM/DD**, ignore the time values entered and only display the date.

Figure 357: $$FMTE^XLFDT API—Example 9: YY/MM/DD Format Ignoring Time Values

```
>S X=$$FMTE^XLFDT(2940629.1057,"4D")

>W X
94/6/29
```

31.5.6.1.10 Example 10

To output a really short date/time try the following, convert **space** to **zero** and remove slash, as shown in Figure 358:

Figure 358: $$FMTE^XLFDT API—Example 10: Short Date/Time Format Converting Spaces to Zeroes and Removing Slashes

```
>S X=$TR($$FMTE^XLFDT(2940629.1057,“4F”),“ /”,“0”)

>W X
940629@10:57
```

31.5.6.1.11 Example 11

Return the date in the following format: **MM/DD/YYYY@HH:MM:SS**.

Figure 359: $$FMTE^XLFDT API—Example 11: MM/DD/YYYY@HH:MM:SS Format

```
>S X=$$FMTE^XLFDT(3000229.110520,5)

>W X
2/29/2000@11:05:20
```

31.5.6.1.12 Example 12

Return the date in the following format: **MM/DD/YYYY@HH:MM:SS** and include leading spaces.

Figure 360: $$FMTE^XLFDT API—Example 12: MM/DD/YYYY@HH:MM:SS Format Including Leading Spaces

```
>S X=$$FMTE^XLFDT(3000229.110520,“5F”)

>W X
 2/29/2000@11:05:20
```

31.5.6.1.13 Example 13

Return the date in the following format: **MM/DD/YYYY@HH:MM:SS**, forcing seconds.

Figure 361: $$FMTE^XLFDT API—Example 13: MM/DD/YYYY@HH:MM:SS Format Forcing Seconds

```
>S X=$$FMTE^XLFDT(3000229.1105,"5S")

>W X
2/29/2000@11:05:00
```

31.5.6.1.14 Example 14

Return the date in the following format: **MM/DD/YYYY HH:MM:SS**, include leading zeroes and am/pm.

Figure 362: $$FMTE^XLFDT API—Example 14: MM/DD/YYYY HH:MM:SS Format Including Leading Zeroes and am/pm

```
>S X=$$FMTE^XLFDT(3000229.110520,"5ZP")

>W X
02/29/2000 11:05:20 am
```

31.5.6.1.15 Example 15

Return the date in the following format: **DD/MM/YYYY@HH:MM:SS**, with leading spaces.

Figure 363: $$FMTE^XLFDT API—Example 15: DD/MM/YYYY@HH:MM:SS Format with Leading Spaces

```
>S X=$$FMTE^XLFDT(3000229.110520,"6F")

>W X
29/ 2/2000@11:05:20
```

31.5.6.1.16 Example 16

Return the date in the following format: **DD/MM/YYYY@HH:MM:SS**, with leading zeroes.

Figure 364: $$FMTE^XLFDT API—Example 16: DD/MM/YYYY@HH:MM:SS Format with Leading Zeroes

```
>S X=$$FMTE^XLFDT(3000229.1105,"6Z")

>W X
29/02/2000@11:05
```

31.5.6.1.17 Example 17

Return the date in the following format: **YYYY/MM/DD@HH:MM:SS**.

Figure 365: $$FMTE^XLFDT API—Example 17: YYYY/MM/DD@HH:MM:SS Format

```
>S X=$$FMTE^XLFDT(3000301.1105,7)

>W X
2000/3/1@11:05
```

31.5.6.1.18 Example 18

Return the date in the following format: **YYYY/MM/DD**, ignore the time values entered and only display the date.

Figure 366: $$FMTE^XLFDT API—Example 18: YYYY/MM/DD Format Ignoring Time Values

```
>S X=$$FMTE^XLFDT(3000301.1105,"7D")

>W X
2000/3/1
```

31.5.7 $$FMTH^XLFDT(): Convert VA FileMan Date to $H

Reference Type:	Supported
Category:	Date Functions
ICR #:	10103
Description:	The $$FMTH^XLFDT extrinsic function converts a VA FileMan formatted input date to a **$H** formatted date.
Format:	`$$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.

31.5.7.1 Examples

31.5.7.1.1 Example 1

Figure 367: $$FMTH^XLFDT API—Example 1

```
>S X=$$FMTH^XLFDT(2901231.111523)

>W X
54786,40523
```

31.5.7.1.2 Example 2

Figure 368: $$FMTH^XLFDT API—Example 2

```
>S X=$$FMTH^XLFDT(2901231.111523,1)

>W X
54786
```

31.5.8 $$FMTHL7^XLFDT(): Convert VA FileMan Date to HL7 Date

Reference Type:	Supported	
Category:	Date Functions	
ICR #:	10103	
Description:	The $$FMTHL7^XLFDT extrinsic function converts a VA FileMan formatted input date/time into a Health Level Seven (HL7) formatted date, including the time offset.	
Format:	`$$FMTHL7^XLFDT(fm_date_time)`	
Input Parameters:	**fm_date_time**:	(required) VA FileMan date.
Output:	returns:	Returns the converted date in HL7 format.

31.5.8.1 Example

Figure 369: $$FMTHL7^XLFDT API—Example

```
>S X=$$FMTHL7^XLFDT(3001127.1525)

>W X
200011271525-0800
```

31.5.9 $$HADD^XLFDT(): $H Add

Reference Type:	Supported	
Category:	Date Functions	
ICR #:	10103	
Description:	The $$HADD^XLFDT extrinsic function returns the result of adding days, hours, minutes, and seconds to a date in **$H** format.	
Format:	`$$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.

31.5.9.1 Example

Figure 370: $$HADD^XLFDT API—Example

```
>S X=$$HADD^XLFDT("54786,3600",2,2,20,15)

>W X
54788,12015
```

31.5.10 $$HDIFF^XLFDT(): $H Difference

Reference Type:	Supported	
Category:	Date Functions	
ICR #:	10103	
Description:	The $$HDIFF^XLFDT extrinsic function returns the difference between two **$H** formatted dates.	
Format:	`$$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.

31.5.10.1 Examples

31.5.10.1.1 Example 1

Return the **$H** difference in days.

Figure 371: $$HDIFF^XLFDT API—Example 1

```
>S X=$$HDIFF^XLFDT(“54789,40523”,“54786,25983”,1)

>W X
3
```

31.5.10.1.2 Example 2

Return the **$H** difference in seconds.

Figure 372: $$HDIFF^XLFDT API—Example 2

```
>S X=$$HDIFF^XLFDT("54789,40523","54786,25983",2)

>W X
273740
```

31.5.10.1.3 Example 3

Return the **$H** difference in days **hours:minutes:seconds** format (**DD HH:MM:SS**).

Figure 373: $$HDIFF^XLFDT API—Example 3

```
>S X=$$HDIFF^XLFDT("54789,40523","54786,25983",3)

>W X
3 4:02:20
```

31.5.11 $$HL7TFM^XLFDT(): Convert HL7 Date to VA FileMan Date

Reference Type:	Supported
Category:	Date Functions
ICR #:	10103
Description:	The $$HL7TFM^XLFDT extrinsic function converts an HL7 formatted input date/time into a VA FileMan formatted date/time.
Format:	`$$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, aka GMT, or Greenwich Mean Time) time offset from the MAILMAN TIME ZONE (#4.4) file. 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.

31.5.11.1 Examples

31.5.11.1.1 Example 1

To get date with no offset:

Figure 374: $$HL7TFM^XLFDT API—Example 1

```
>S X=$$HL7TFM^XLFDT("200011271525-0700")

>W X
3001127.1525
```

31.5.11.1.2 Example 2

To get UTC time offset:

Figure 375: $$HL7TFM^XLFDT API—Example 2

```
>S X=$$HL7TFM^XLFDT("200011271525-0700","U")

>W X
3001127.2225
```

31.5.11.1.3 Example 3

To get Local time in PST offset:

Figure 376: $$HL7TFM^XLFDT API—Example 3

```
>S X=$$HL7TFM^XLFDT("200011271525-0700","L")

>W X
3001127.1425
```

31.5.11.1.4 Example 4

To get Local time when only providing a time (no date) as the input parameter:

Figure 377: $$HL7TFM^XLFDT API—Example 4

```
>S X=$$HL7TFM^XLFDT("1525-0700","L",1)

>W X
.1525
```

31.5.12 $$HTE^XLFDT(): Convert $H to External Format

Reference Type:	Supported
Category:	Date Functions
ICR #:	10103
Description:	The $$HTE^XLFDT extrinsic function converts a **$H** formatted input date to an external formatted date.
Format:	`$$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 an **F** then output date with leading **spaces**.
- If **y** contains an **M** then output **HH:MM** only.
- 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 format of a **$H** date.

31.5.12.1 Examples

31.5.12.1.1 Example 1

Return the date in the following format: **Standard external format**.

Figure 378: $$HTE^XLFDT API—Example 1

```
>S X=$$HTE^XLFDT("54786,40523")

>W X
Dec 31, 1990@11:15:23
```

31.5.12.1.2 Example 2

Return the date in the following format: **MM/DD/YY@HH:MM:SS**.

Figure 379: $$HTE^XLFDT API—Example 2

```
>S X=$$HTE^XLFDT("54786,40523",2)

>W X
12/31/90@11:15:23
```

31.5.12.1.3 Example 3

Return the date in the following format: **MM/DD/YY@HH:MM:SS**, omitting the seconds.

Figure 380: $$HTE^XLFDT API—Example 3

```
>S X=$$HTE^XLFDT("57386,33723","2M")

>W X
2/12/98@09:22
```

31.5.12.1.4 Example 4

Return the date in the following format: **MM/DD/YYYY@HH:MM:SS**.

Figure 381: $$HTE^XLFDT API—Example 4

```
>S X=$$HTE^XLFDT("57351,27199",5)

>W X
1/8/1998@07:33:19
```

31.5.12.1.5 Example 5

Return the date in the following format: **DD/MM/YYYY@HH:MM:SS**.

Figure 382: $$HTE^XLFDT API—Example 5

```
>S X=$$HTE^XLFDT("57351,27199",6)

>W X
8/1/1998@07:33:19
```

31.5.12.1.6 Example 6

Return the date in the following format: **YYYY/MM/DD@HH:MM:SS**.

Figure 383: $$HTE^XLFDT API—Example 6

```
>S X=$$HTE^XLFDT("57351,27199",7)

>W X
1998/1/8@07:33:19
```

31.5.13 $$HTFM^XLFDT(): Convert $H to VA FileMan Date Format

Reference Type:	Supported	
Category:	Date Functions	
ICR #:	10103	
Description:	The $$HTFM^XLFDT extrinsic function converts a **$H** formatted input date to a VA FileMan formatted date.	
Format:	`$$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.

31.5.13.1 Examples

31.5.13.1.1 Example 1

Figure 384: $$HTFM^XLFDT API—Example 1

```
>S X=$$HTFM^XLFDT("54786,40523")

>W X
2901231.111523
```

31.5.13.1.2 Example 2

Figure 385: $$HTFM^XLFDT API—Example 2

```
>S X=$$HTFM^XLFDT("54786,40523",1)

>W X
2901231
```

31.5.14 $$NOW^XLFDT: Current Date and Time (VA FileMan Format)

Reference Type:	Supported	
Category:	Date Functions	
ICR #:	10103	
Description:	The $$NOW^XLFDT extrinsic function returns the current date and time in VA FileMan format.	
Format:	`$$NOW^XLFDT`	
Input Parameters:	none.	
Output:	returns:	Returns the current date and time in VA FileMan format.

31.5.14.1 Example

Figure 386: $$NOW^XLFDT API—Example

```
>S X=$$NOW^XLFDT

>W X
3040126.103044
```

31.5.15 $$SCH^XLFDT(): Next Scheduled Runtime

Reference Type:	Supported	
Category:	Date Functions	
ICR #:	10103	
Description:	The $$SCH^XLFDT extrinsic function returns the next run-time based on Schedule code.	
Format:	`$$SCH^XLFDT(schedule_string,base_date[,force_future_flag])`	
Input Parameters:	**schedule_string**:	(required) Interval to add to base_date, as follows:

- ***n*S**—Add ***n*** seconds to **base_date**.
- ***n*H**—Add ***n*** hours to **base_date**.
- ***n*D**—Add ***n*** days to **base_date**.
- ***n*M**—Add ***n*** months to **base_date**.
- **$H;$H;$H**—List of **$H** dates.

- ***n*M(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).
    - ***n*Day[@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.

31.5.15.1 Examples

31.5.15.1.1 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:

- Middle of the Month:

Figure 387: $$SCH^XLFDT API$$SCH^XLFDT API—Example 1: Middle of the Month

```
>S X=$$SCH^XLFDT("1M(15@2PM,L@6PM)",2931003)

>W X
2931015.14
```

- End of the Month:

Figure 388: $$SCH^XLFDT API$$SCH^XLFDT API—Example 1: End of the Month

```
>S X=$$SCH^XLFDT("1M(15@2PM,L@6PM)",X)

>W X
2931031.18
```

31.5.15.1.2 Example 2

To schedule something to run every month on the 15th of the month at 11:00 p.m. and on the last day of every month at 8:00 p.m., you would enter the following:

- Middle of the Month:

Figure 389: $$SCH^XLFDT API$$SCH^XLFDT API—Example 2: Middle of the Month

```
>S X=$$SCH^XLFDT("1M(15@11PM,L@8PM)",2931028)

>W X
2931031.2
```

- End of the Month:

Figure 390: $$SCH^XLFDT API$$SCH^XLFDT API—Example 2: End of the Month

```
>S X=$$SCH^XLFDT("1M(15@11PM,L@8PM)",X)

>W X
2931115.23
```

31.5.15.1.3 Example 3

To schedule something to run every 3 months on the last day of the month at 6:00 p.m., you would enter the following:

- Middle of the Month:

Figure 391: $$SCH^XLFDT API$$SCH^XLFDT API—Example 3: Middle of the Month

```
>S X=$$SCH^XLFDT("3M(L@6PM)",2930927)

>W X
2930930.18
```

- End of the Month:

Figure 392: $$SCH^XLFDT API$$SCH^XLFDT API—Example 3: End of the Month

```
>S X=$$SCH^XLFDT("3M(L@6PM)",X)

>W X
2931231.18
```

31.5.15.1.4 Example 4

The $$SCH^XLFDT API can return a date that is closer to the date the API is run if the user does *not* use the **force_future_flag** parameter and the **base_date** parameter is set to a date in the past. In this example, the **base_date** parameter is set to a date in the past, 11/17/2014 at 8:00, and the interval is set to find the date 2 months out on the second Monday of the month. The date that is returned is the date that the API was run, 1/12/15, which happens to be the second Monday of the month and two months out from the **base_date**.

Figure 393: $$SCH^XLFDT API—Example 4: Not Using Future flag

```
>S X=$$SCH^XLFDT("2M(2M@0800)",3141117.0800)

>W X
3150112.08
```

If using the **force_future_flag** parameter to the API, using the same interval as above, the API forces the return date to be a date in the future from the date the API is run.

Figure 394: $$SCH^XLFDT API—Example 4: Using Future Flag

```
>S X=$$SCH^XLFDT("2M(2M@0800)",3141117.0800,1)

>W X
3150309.08
```

NOTE: The **base_date** *must* be passed correctly. The **base_date** parameter is compared to the **schedule_string** parameter in the interval to return the correct output.

31.5.16 $$SEC^XLFDT(): Convert $H/VA FileMan date to Seconds

Reference Type:	Supported	
Category:	Date Functions	
ICR #:	10103	
Description:	The $$SEC^XLFDT extrinsic function converts a **$H** or VA FileMan formatted input date to the number of seconds. The input date can be entered as either a VA FileMan date or a **$H** date. If entered as a VA FileMan date, the date is first converted to **$H** via the $$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.

31.5.16.1 Examples

31.5.16.1.1 Example 1

Inputting a VA FileMan date/time:

Figure 395: $$SEC^XLFDT—Example 1

```
>S X=$$SEC^XLFDT(3021118.1347)

>W X
5108536020
```

31.5.16.1.2 Example 2

Inputting a **$H** date:

Figure 396: $$SEC^XLFDT—Example 2

```
>S X=$$SEC^XLFDT($H)

>W X
5146022146
```

31.5.17 $$TZ^XLFDT: Time Zone Offset (GMT)

Reference Type: Supported

Category: Date Functions

ICR #: 10103

Description: The $$TZ^XLFDT extrinsic function returns the Time Zone offset from Greenwich mean time (GMT) based on a pointer from the TIME ZONE (#1) field in the MAILMAN SITE PARAMETERS (#4.3) file to the MAILMAN TIME ZONE (#4.4) file.

The accuracy of this value is dependent on system administrators updating the TIME ZONE (#1) field in the MAILMAN SITE PARAMETERS (#4.3) file 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.

31.5.17.1 Example

For Pacific Daylight Savings Time (PDT), the offset from GMT is:

Figure 397: $$TZ^XLFDT—Example

```
>S X = $$TZ^XLFDT

>W X
-0700
```

31.5.18 $$WITHIN^XLFDT(): Checks Dates/Times within Schedule

Reference Type: Supported

Category: Date Functions

ICR #:

Description: The $$WITHIN^XLFDT 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**.

REF: For alternate values, see the $$SCH^XLFDT(): Next Scheduled Runtime API.

base_date: (required) VA FileMan date checked to determine if it is within the input **schedule_string**.

Output: returns: Returns whether or *not* a date/time is within a specified schedule string.

31.6 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.

31.6.1 $$ACOSH^XLFHYPER(): Hyperbolic Arc-Cosine

Reference Type:	Supported	
Category:	Hyperbolic Trigonometric Functions	
ICR #:	10144	
Description:	The $$ACOSH^XLFHYPER extrinsic function returns the hyperbolic arc cosine, with radians output.	
Format:	`$$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.

31.6.1.1 Example

Figure 398: $$ACOSH^XLFHYPER API—Example

```
>S X=$$ACOSH^XLFHYPER(3,12)

>W X
1.762747174
```

31.6.2 $$ACOTH^XLFHYPER(): Hyperbolic Arc-Cotangent

Reference Type:	Supported	
Category:	Hyperbolic Trigonometric Functions	
ICR #:	10144	
Description:	The $$ACOTH^XLFHYPER extrinsic function returns the hyperbolic arc cotangent, with radians output.	
Format:	`$$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.

31.6.2.1 Example

Figure 399: $$ACOTH^XLFHYPER API—Example

```
>S X=$$ACOTH^XLFHYPER(3,12)

>W X
.34657359025
```

31.6.3 $$ACSCH^XLFHYPER(): Hyperbolic Arc-Cosecant

Reference Type: Supported

Category: Hyperbolic Trigonometric Functions

ICR #: 10144

Description: The $$ACSCH^XLFHYPER 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.

31.6.3.1 Example

Figure 400: $$ACSCH^XLFHYPER API—Example

```
>S X=$$ACSCH^XLFHYPER(3,12)

>W X
.3274501502
```

31.6.4 $$ASECH^XLFHYPER(): Hyperbolic Arc-Secant

Reference Type:	Supported	
Category:	Hyperbolic Trigonometric Functions	
ICR #:	10144	
Description:	The $$ASECH^XLFHYPER extrinsic function returns the hyperbolic arc secant, with radians output.	
Format:	`$$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.

31.6.4.1 Example

Figure 401: $$ASECH^XLFHYPER API—Example

```
>S X=$$ASECH^XLFHYPER(.3,12)

>W X
1.8738202425
```

31.6.5 $$ASINH^XLFHYPER(): Hyperbolic Arc-Sine

Reference Type:	Supported	
Category:	Hyperbolic Trigonometric Functions	
ICR #:	10144	
Description:	The $$ASINH^XLFHYPER extrinsic function returns the hyperbolic arc sine, with radians output.	
Format:	`$$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.

31.6.5.1 Example

Figure 402: $$ASINH^XLFHYPER API—Example

```
>S X=$$SINH^XLFHYPER(3,12)

>W X
10.0178749273
```

31.6.6 $$ATANH^XLFHYPER(): Hyperbolic Arc-Tangent

Reference Type:	Supported	
Category:	Hyperbolic Trigonometric Functions	
ICR #:	10144	
Description:	The $$ATANH^XLFHYPER extrinsic function returns the hyperbolic arc tangent, with radians output.	
Format:	`$$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.

31.6.6.1 Example

Figure 403: $$ATANH^XLFHYPER API—Example

```
>S X=$$ATANH^XLFHYPER(.3,12)

>W X
.3095196042
```

31.6.7 $$COSH^XLFHYPER(): Hyperbolic Cosine

Reference Type:	Supported	
Category:	Hyperbolic Trigonometric Functions	
ICR #:	10144	
Description:	The $$COSH^XLFHYPER extrinsic function returns the hyperbolic arc cosine, with radians output.	
Format:	`$$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.

31.6.7.1 Example

Figure 404: $$COSH ^XLFHYPER API—Example

```
>S X=$$COSH^XLFHYPER(3,12)

>W X
10.0676619957
```

31.6.8 $$COTH^XLFHYPER(): Hyperbolic Cotangent

Reference Type:	Supported	
Category:	Hyperbolic Trigonometric Functions	
ICR #:	10144	
Description:	The $$COTH^XLFHYPER extrinsic function returns the hyperbolic cotangent, with radians output.	
Format:	`$$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.

31.6.8.1 Example

Figure 405: $$COTH^XLFHYPER API—Example

```
>S X=$$COTH^XLFHYPER(3,12)

>W X
1.00496982332
```

31.6.9 $$CSCH^XLFHYPER(): Hyperbolic Cosecant

Reference Type:	Supported	
Category:	Hyperbolic Trigonometric Functions	
ICR #:	10144	
Description:	The $$CSCH^XLFHYPER extrinsic function returns the hyperbolic cosecant, with radians output.	
Format:	`$$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.

31.6.9.1 Example

Figure 406: $$CSCH^XLFHYPER API—Example

```
>S X=$$CSCH^XLFHYPER(3,12)

>W X
.09982156967
```

31.6.10 $$SECH^XLFHYPER(): Hyperbolic Secant

Reference Type: Supported

Category: Hyperbolic Trigonometric Functions

ICR #: 10144

Description: The $$SECH^XLFHYPER 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.

31.6.10.1 Example

Figure 407: $$SECH^XLFHYPER API—Example

```
>S X=$$SECH^XLFHYPER(3,12)

>W X
.09932792742
```

31.6.11 $$SINH^XLFHYPER(): Hyperbolic Sine

Reference Type: Supported

Category: Hyperbolic Trigonometric Functions

ICR #: 10144

Description: The $$SINH^XLFHYPER 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.

31.6.11.1 Examples

31.6.11.1.1 Example 1

Figure 408: $$SINH^XLFHYPER API—Example 1

```
>S X=$$SINH^XLFHYPER(.707)

>W X
.767388542
```

31.6.11.1.2 Example 2

Figure 409: $$SINH^XLFHYPER API—Example 2

```
>S X=$$SINH^XLFHYPER(.3,12)

>W X
.30452029345
```

31.6.12 $$TANH^XLFHYPER(): Hyperbolic Tangent

Reference Type:	Supported	
Category:	Hyperbolic Trigonometric Functions	
ICR #:	10144	
Description:	The $$TANH^XLFHYPER extrinsic function returns the hyperbolic tangent of **x** (TAN **x** = SIN **x**/COS **x**), with radians output.	
Format:	`$$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.

31.6.12.1 Example

Figure 410: $$TANH^XLFHYPER API—Example

```
>S X=$$TANH^XLFHYPER(3,12)

>W X
.99505475368
```

31.7 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.

31.7.1 $$ABS^XLFMTH(): Absolute Value

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$ABS^XLFMTH extrinsic function returns the absolute value of the number in **x**.	
Format:	`$$ABS^XLFMTH(x)`	
Input Parameters:	**x**:	(required) Number for which you want the absolute value.
Output:	returns:	Returns the absolute value of a number.

31.7.1.1 Example

Figure 411: $$ABS^XLFMTH API—Example

```
>S X=$$ABS^XLFMTH(-42.45)

>W X
42.45
```

31.7.2 $$ACOS^XLFMTH(): Arc-Cosine (Radians)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$ACOS^XLFMTH extrinsic function returns the arc cosine, with radians output.	
Format:	`$$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.

31.7.2.1 Example

Figure 412: $$ACOS^XLFMTH API—Example

```
>S X=$$ACOS^XLFMTH(.5)

>W X
1.047197551
```

31.7.3 $$ACOSDEG^XLFMTH(): Arc-Cosine (Degrees)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$ACOSDEG^XLFMTH extrinsic function returns the arc cosine, with degrees output.	
Format:	`$$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.

31.7.3.1 Example

Figure 413: $$ACOSDEG^XLFMTH API—Example

```
>S X=$$ACOSDEG^XLFMTH(.5)

>W X
60
```

31.7.4 $$ACOT^XLFMTH(): Arc-Cotangent (Radians)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$ACOT^XLFMTH extrinsic function returns the arc cotangent, with radians output.	
Format:	`$$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.

31.7.4.1 Example

Figure 414: $$ACOT^XLFMTH API—Example

```
>S X=$$ACOT^XLFMTH(.5)

>W X
1.107148718
```

31.7.5 $$ACOTDEG^XLFMTH(): Arc-Cotangent (Degrees)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$ACOTDEG^XLFMTH extrinsic function returns the arc cotangent, with degrees output.	
Format:	`$$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.

31.7.5.1 Example

Figure 415: $$ACOTDEG^XLFMTH API—Example

```
>S X=$$ACOTDEG^XLFMTH(.5)

>W X
63.43494882
```

31.7.6 $$ACSC^XLFMTH(): Arc-Cosecant (Radians)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$ACSC^XLFMTH extrinsic function returns the arc cosecant, with radians output.	
Format:	`$$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.

31.7.6.1 Example

Figure 416: $$ACSC^XLFMTH API—Example

```
>S X=$$ACSC^XLFMTH(1.5)

>W X
.729727656
```

31.7.7 $$ACSCDEG^XLFMTH(): Arc-Cosecant (Degrees)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$ACSCDEG^XLFMTH extrinsic function returns the arc cosecant, with degrees output.	
Format:	`$$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.

31.7.7.1 Example

Figure 417: $$ACSCDEG^XLFMTH API—Example

```
>S X=$$ACSCDEG^XLFMTH(1.5)

>W X
41.8103149
```

31.7.8 $$ASEC^XLFMTH(): Arc-Secant (Radians)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$ASEC^XLFMTH extrinsic function returns the arc secant, with radians output.	
Format:	`$$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.

31.7.8.1 Example

Figure 418: $$ASEC^XLFMTH API—Example

```
>S X=$$ASEC^XLFMTH(1.5)

>W X
.841068671
```

31.7.9 $$ASECDEG^XLFMTH(): Arc-Secant (Degrees)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$ASECDEG^XLFMTH extrinsic function returns the arc secant, with degrees output.	
Format:	`$$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.

31.7.9.1 Example

Figure 419: $$ASECDEG^XLFMTH API—Example

```
>S X=$$ASECDEG^XLFMTH(1.5)

>W X
48.1896851
```

31.7.10 $$ASIN^XLFMTH(): Arc-Sine (Radians)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$ASIN^XLFMTH extrinsic function returns the arc sine, with radians output.	
Format:	`$$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.

31.7.10.1 Example

Figure 420: $$ASIN^XLFMTH API—Example

```
>S X=$$ASIN^XLFMTH(.5)

>W X
.523598776
```

31.7.11 $$ASINDEG^XLFMTH(): Arc-Sine (Degrees)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$ASINDEG^XLFMTH extrinsic function returns the arc sine, with degrees output.	
Format:	`$$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.

31.7.11.1 Example

Figure 421: $$ASINDEG^XLFMTH API—Example

```
>S X=$$ASINDEG^XLFMTH(.5)

>W X
30
```

31.7.12 $$ATAN^XLFMTH(): Arc-Tangent (Radians)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$ATAN^XLFMTH extrinsic function returns the arc tangent, with radians output.	
Format:	`$$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.

31.7.12.1 Example

Figure 422: $$ATAN^XLFMTH API—Example

```
>S X=$$ATAN^XLFMTH(.5)

>W X
.463647609
```

31.7.13 $$ATANDEG^XLFMTH(): Arc-Tangent (Degrees)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$ATANDEG^XLFMTH extrinsic function returns the arc tangent, with degrees output.	
Format:	`$$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.

31.7.13.1 Example

Figure 423: $$ATANDEG^XLFMTH API—Example

```
>S X=$$ATANDEG^XLFMTH(.5)

>W X
26.56505118
```

31.7.14 $$COS^XLFMTH(): Cosine (Radians)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$COS^XLFMTH extrinsic function returns the cosine, with radians input.	
Format:	`$$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.

31.7.14.1 Example

Figure 424: $$COS^XLFMTH API—Example

```
>S X=$$COS^XLFMTH(1.5)

>W X
.070737202
```

31.7.15 $$COSDEG^XLFMTH(): Cosine (Degrees)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$COSDEG^XLFMTH extrinsic function returns the cosine, with degrees input.	
Format:	$$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.

31.7.15.1 Example

Figure 425: $$COSDEG^XLFMTH API—Example

```
>S X=$$COSDEG^XLFMTH(45)

>W X
.707106781
```

31.7.16 $$COT^XLFMTH(): Cotangent (Radians)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$COT^XLFMTH extrinsic function returns the cotangent, with radians input.	
Format:	`$$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.

31.7.16.1 Example

Figure 426: $$COT^XLFMTH API—Example

```
>S X=$$COT^XLFMTH(1.5)

>W X
.070914844
```

31.7.17 $$COTDEG^XLFMTH(): Cotangent (Degrees)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$COTDEG^XLFMTH extrinsic function returns the cotangent, with degrees input.	
Format:	`$$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.

31.7.17.1 Example

Figure 427: $$COTDEG^XLFMTH API—Example

```
>S X=$$COTDEG^XLFMTH(45)

>W X
1
```

31.7.18 $$CSC^XLFMTH(): Cosecant (Radians)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$CSC^XLFMTH extrinsic function returns the cosecant, with radians input.	
Format:	`$$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.

31.7.18.1 Example

Figure 428: $$CSC^XLFMTH API—Example

```
>S X=$$CSC^XLFMTH(1.5)

>W X
1.002511304
```

31.7.19 $$CSCDEG^XLFMTH(): Cosecant (Degrees)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$CSCDEG^XLFMTH extrinsic function returns the cosecant, with degrees input.	
Format:	`$$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.

31.7.19.1 Example

Figure 429: $$CSCDEG^XLFMTH API—Example

```
>S X=$$CSCDEG^XLFMTH(45)

>W X
1.414213562
```

31.7.20 $$DECDMS^XLFMTH(): Convert Decimals to Degrees:Minutes:Seconds

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$DECDMS^XLFMTH extrinsic function converts a number from decimal to degrees:minutes:seconds.	
Format:	`$$DECDMS^XLFMTH(x[,n])`	
Input Parameters:	**x**:	(required) Decimal number to be converted to degree:minutes:seconds.
	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.

31.7.20.1 Example

Figure 430: $$DECDMS^XLFMTH API—Example

```
>S X=$$DECDMS^XLFMTH(30.7)

>W X
30:42:0
```

31.7.21 $$DMSDEC^XLFMTH(): Convert Degrees:Minutes:Seconds to Decimal

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$DMSDEC^XLFMTH extrinsic function converts a number from degrees:minutes:seconds to a decimal.	
Format:	`$$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.

31.7.21.1 Example

Figure 431: $$DMSDEC^XLFMTH API—Example

```
>S X=$$DMSDEC^XLFMTH("30:42:0")

>W X
30.7
```

31.7.22 $$DTR^XLFMTH(): Convert Degrees to Radians

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$DTR^XLFMTH extrinsic function converts degrees to radians.	
Format:	`$$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.

31.7.22.1 Example

Figure 432: $$DTR^XLFMTH API—Example

```
>S X=$$DTR^XLFMTH(45)

>W X
.7853981634
```

31.7.23 $$E^XLFMTH(): e—Natural Logarithm

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$E^XLFMTH extrinsic function returns **e** (natural logarithm).	
Format:	`$$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.

31.7.23.1 Example

Figure 433: $$E^XLFMTH API—Example

```
>S X=$$E^XLFMTH(12)

>W X
2.71828182846
```

31.7.24 $$EXP^XLFMTH(): e—Natural Logarithm to the Nth Power

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$EXP^XLFMTH extrinsic function returns **e** (natural logarithm) to the **x** power (exponent).	
Format:	`$$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.

31.7.24.1 Example

Figure 434: $$EXP^XLFMTH API—Example

```
>S X=$$EXP^XLFMTH(1.532)

>W X
4.6274224185
```

31.7.25 $$LN^XLFMTH(): Natural Log (Base e)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$LN^XLFMTH extrinsic function returns the natural log of **x** (Base **e**).	
Format:	`$$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.

31.7.25.1 Example

Figure 435: $$LN^XLFMTH API—Example

```
>S X=$$LN^XLFMTH(4.627426)

>W X
1.532000774
```

31.7.26 $$LOG^XLFMTH(): Logarithm (Base 10)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$LOG^XLFMTH extrinsic function returns the logarithm (Base **10**) of **x**.	
Format:	`$$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.

31.7.26.1 Example

Figure 436: $$LOG^XLFMTH API—Example

```
>S X=$$LOG^XLFMTH(3.1415)

>W X
.4971370641
```

31.7.27 $$MAX^XLFMTH(): Maximum of Two Numbers

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$MAX^XLFMTH extrinsic function returns the maximum value by comparing the number in **x** with the number in **y**.	
Format:	`$$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.

31.7.27.1 Example

Figure 437: $$MAX^XLFMTH API—Example

```
>S X=$$MAX^XLFMTH(53,24)

>W X
53
```

31.7.28 $$MIN^XLFMTH(): Minimum of Two Numbers

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$MIN^XLFMTH extrinsic function returns the minimum value by comparing the number in **x** with the number in **y**.	
Format:	`$$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.

31.7.28.1 Example

Figure 438: $$MIN^XLFMTH API—Example

```
>S X=$$MIN^XLFMTH(53,24)

>W X
24
```

31.7.29 $$PI^XLFMTH(): PI

Reference Type: Supported

Category: Math Functions

ICR #: 10105

Description: The $$PI^XLFMTH 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.

31.7.29.1 Example

Figure 439: $$PI^XLFMTH API—Example

```
>S X=$$PI^XLFMTH(12)

>W X
3.14159265359
```

31.7.30 $$PWR^XLFMTH(): X to the Y Power

Reference Type: Supported

Category: Math Functions

ICR #: 10105

Description: The $$PWR^XLFMTH 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.

31.7.30.1 Example

Figure 440: $$PWR^XLFMTH API—Example

```
>S X=$$PWR^XLFMTH(3.2,1.5)

>W X
5.7243340224
```

31.7.31 $$RTD^XLFMTH(): Convert Radians to Degrees

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$RTD^XLFMTH extrinsic function converts radians to degrees.	
Format:	`$$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.

31.7.31.1 Example

Figure 441: $$RTD^XLFMTH API—Example

```
>S X=$$RTD^XLFMTH(1.5,12)

>W X
85.9436692696
```

31.7.32 $$SD^XLFMTH(): Standard Deviation

Reference Type: Supported

Category: Math Functions

ICR #: 10105

Description: The $$SD^XLFMTH 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."[2]

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.

31.7.32.1 Example

Figure 442: $$SD^XLFMTH API—Example

```
>S X=$$SD^XLFMTH(5,25,2)

>W X
3.53553390593
```

[2] 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.

31.7.33 $$SEC^XLFMTH(): Secant (Radians)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$SEC^XLFMTH extrinsic function returns the secant of a number, with radians input.	
Format:	`$$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.

31.7.33.1 Example

Figure 443: $$SEC^XLFMTH API—Example

```
>S X=$$SEC^XLFMTH(1.5)

>W X
14.1368329
```

31.7.34 $$SECDEG^XLFMTH(): Secant (Degrees)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$SECDEG^XLFMTH extrinsic function returns the secant of a number, with degrees input.	
Format:	`$$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.

31.7.34.1 Example

Figure 444: $$SECDEG^XLFMTH API—Example

```
>S X=$$SECDEG^XLFMTH(45)

>W X
1.414213562
```

31.7.35 $$SIN^XLFMTH(): Sine (Radians)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$SIN^XLFMTH extrinsic function returns the sine of a number, with radians input.	
Format:	`$$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.

31.7.35.1 Example

Figure 445: $$SIN^XLFMTH API—Example

```
>S X=$$SIN^XLFMTH(.7853982)

>W X
.707106807
```

31.7.36 $$SINDEG^XLFMTH(): Sine (Degrees)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$SINDEG^XLFMTH extrinsic function returns the sine of a number, with degrees input.	
Format:	`$$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.

31.7.36.1 Example

Figure 446: $$SINDEG^XLFMTH API—Example

```
>S X=$$SINDEG^XLFMTH(45)

>W X
.707106781
```

31.7.37 $$SQRT^XLFMTH(): Square Root

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$SQRT^XLFMTH extrinsic function returns the square root of a number.	
Format:	$$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.

31.7.37.1 Example

Figure 447: $$SQRT^XLFMTH API—Example

```
>S X=$$SQRT^XLFMTH(153)

>W X
12.3693168769
```

31.7.38 $$TAN^XLFMTH(): Tangent (Radians)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$TAN^XLFMTH extrinsic function returns the tangent of a number (TAN **x** = SIN **x**/COS **x**), with radians input.	
Format:	`$$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.

31.7.38.1 Example

Figure 448: $$TAN^XLFMTH API—Example

```
>S X=$$TAN^XLFMTH(.7853982)

>W X
1.000000073
```

31.7.39 $$TANDEG^XLFMTH(): Tangent (Degrees)

Reference Type:	Supported	
Category:	Math Functions	
ICR #:	10105	
Description:	The $$TANDEG^XLFMTH extrinsic function returns the tangent of a number, with degrees input.	
Format:	`$$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.

31.7.39.1 Example

Figure 449: $$TANDEG^XLFMTH API—Example

```
>S X=$$TANDEG^XLFMTH(45)

>W X
1
```

31.8 Measurement Functions—XLFMSMT

This routine contains APIs to allow conversion between U.S. (English) and Metric units.

31.8.1 $$BSA^XLFMSMT(): Body Surface Area Measurement

Reference Type:	Supported	
Category:	Measurement Functions	
ICR #:	3175 & 10143	
Description:	The $$BSA^XLFMSMT extrinsic function returns the body surface area.	
Format:	`$$BSA^XLFMSMT(ht,wt)`	
Input Parameters:	**ht**:	(required) Height in centimeters.
	wt:	(required) Weight in kilograms.
Output:	returns:	Returns the body surface area measurement.

31.8.1.1 Examples

31.8.1.1.1 Example 1

Figure 450: $$BSA^XLFMSMT API—Example 1

```
>S X=$$BSA^XLFMSMT(175,86)

>W X
2.02
```

31.8.1.1.2 Example 2

Figure 451: $$BSA^XLFMSMT API—Example 2

```
>S
X=$$BSA^XLFMSMT($$LENGTH^XLFMSMT(69,"IN","CM"),$$WEIGHT^XLFMSMT(180,"LB","KG
"))

>W X
1.98
```

31.8.2 $$LENGTH^XLFMSMT(): Convert Length Measurement

Reference Type:	Supported
Category:	Measurement Functions
ICR #:	3175 & 10143
Description:	The $$LENGTH^XLFMSMT extrinsic function converts U.S. length to Metric length and vice versa. It returns the equivalent value with units.
Format:	`$$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 42).
	to:	(required) Unit of measure to which the **value** input parameter is converted (see Table 42).

Valid units in either uppercase or lowercase are:

Table 42: $$LENGTH^XLFMSMT API—Valid Units

Metric	US
km—kilometers	**mi**—miles
m—meters	**yd**—yards
cm—centimeters	**ft**—feet
mm—millimeters	**in**—inches

Output:	returns:	Returns the length measurement.

31.8.2.1 Examples

31.8.2.1.1 Example 1

Converting U.S. length to Metric length:

Figure 452: $$LENGTH^XLFMSMT API—Example 1

```
>S X=$$LENGTH^XLFMSMT(12,"IN","CM")

>W X
30.48 CM
```

31.8.2.1.2 Example 2

Converting Metric length to U.S. length:

Figure 453: $$LENGTH^XLFMSMT API—Example 2

```
>S X=$$LENGTH^XLFMSMT(30.48,"cm","in")

>W X
12 IN
```

31.8.3 $$TEMP^XLFMSMT(): Convert Temperature Measurement

Reference Type: Supported

Category: Measurement Functions

ICR #: 3175 & 10143

Description: The $$TEMP^XLFMSMT 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 43).

to: (required) Unit of measure to which the **value** input parameter is converted (see Table 43).

Valid units in either uppercase or lowercase are:

Table 43: $$TEMP^XLFMSMT API—Valid Units

Metric	US
C—Celsius	F—Fahrenheit

Output: returns: Returns the temperature measurement.

31.8.3.1 Examples

31.8.3.1.1 Example 1

Converting Fahrenheit to Celsius:

Figure 454: $$TEMP^XLFMSMT API—Example 1: Converting Fahrenheit to Celsius

```
>S X=$$TEMP^XLFMSMT(72,"F","C")

>W X
22.222 C
```

31.8.3.1.2 Example 2

Converting Celsius to Fahrenheit:

Figure 455: $$TEMP^XLFMSMT API—Example 2: Converting Celsius to Fahrenheit

```
>S X=$$TEMP^XLFMSMT(0,"c","f")

>W X
32 F
```

31.8.4 $$VOLUME^XLFMSMT(): Convert Volume Measurement

Reference Type:	Supported
Category:	Measurement Functions
ICR #:	3175 & 10143
Description:	The $$VOLUME^XLFMSMT extrinsic function converts U.S. volume to Metric volume and vice versa. Converts milliliters to cubic inches or quarts or ounces. It returns the equivalent value with units.
Format:	`$$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 44).
	to:	(required) Unit of measure to which the **value** input parameter is converted (see Table 44).

Valid units in either uppercase or lowercase are:

Table 44: $$VOLUME^XLFMSMT API—Valid Units

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

Output: returns: Returns the volume measurement.

31.8.4.1 Examples

31.8.4.1.1 Example 1

Converting U.S. volume to Metric volume:

Figure 456: $$VOLUME^XLFMSMT API—Example 1

```
>S X=$$VOLUME^XLFMSMT(12,"CF","ML")

>W X
339800.832 ML
```

31.8.4.1.2 Example 2

Converting Metric volume to U.S. volume:

Figure 457: $$VOLUME^XLFMSMT API—Example 2

```
>S X=$$VOLUME^XLFMSMT(339800.832,"ml","cf")

>W X
11.998 CF
```

31.8.5 $$WEIGHT^XLFMSMT(): Convert Weight Measurement

Reference Type: Supported

Category: Measurement Functions

ICR #: 3175 & 10143

Description: The $$WEIGHT^XLFMSMT extrinsic function converts U.S. weights to proximate 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 45).

to: (required) Unit of measure to which the **value** input parameter is converted (see Table 45).

Valid units in either uppercase or lowercase are:

Table 45: $$WEIGHT^XLFMSMT API—Valid Units

Metric	US
t—metric tons	tn— tons
kg—kilograms	lb—pounds
g—grams	oz—ounces
mg—milligram	gr—grain

Output: returns: Returns the weight measurement.

31.8.5.1 Examples

31.8.5.1.1 Example 1

Converting U.S. weight to Metric weight:

Figure 458: $$WEIGHT^XLFMSMT API—Example 1

```
>S X=$$WEIGHT^XLFMSMT(12,"LB","G")

>W X
5448 G
```

31.8.5.1.2 Example 2

Converting Metric weight to U.S. weight:

Figure 459: $$WEIGHT^XLFMSMT API—Example 2

```
>S X=$$WEIGHT^XLFMSMT(5448,"g","lb")

>W X
12.011 LB
```

31.9 String Functions—XLFSTR

These functions are provided to help process strings.

31.9.1 $$CJ^XLFSTR(): Center Justify String

Reference Type:	Supported	
Category:	String Functions	
ICR #:	10104	
Description:	The $$CJ^XLFSTR extrinsic function returns a center justified character string.	
Format:	`$$CJ^XLFSTR(s,i[,p])`	
Input Parameters:	**s**:	(required) Character string.
	i:	(required) Field size. If this second parameter contains a trailing **T**, this extrinsic function returns the output truncated to the field size specified.
	p:	(optional) Pad character.
Output:	returns:	Returns the Center justified string.

31.9.1.1 Examples

31.9.1.1.1 Example 1

Figure 460: $$CJ^XLFSTR API—Example 1

```
>W "[",$$CJ^XLFSTR("SUE",10),"]"
[   SUE    ]
```

31.9.1.1.2 Example 2

Figure 461: $$CJ^XLFSTR API—Example 2

```
>W "[",$$CJ^XLFSTR("SUE",10,"-"),"]"
[---SUE----]
```

31.9.1.1.3 Example 3

Figure 462: $$CJ^XLFSTR API—Example 3

```
>W $$CJ^XLFSTR("123456789",5)
123456789
```

31.9.1.1.4 Example 4

Figure 463: $$CJ^XLFSTR API—Example 4

```
>W $$CJ^XLFSTR(123456789,"5T")
12345
```

31.9.2 $$INVERT^XLFSTR(): Invert String

Reference Type:	Supported	
Category:	String Functions	
ICR #:	10104	
Description:	The v 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.

31.9.2.1 Example

Figure 464: $$INVERT^XLFSTR API—Example

```
>S X=$$INVERT^XLFSTR("ABC")

>W X
CBA
```

31.9.3 $$LJ^XLFSTR(): Left Justify String

Reference Type:	Supported	
Category:	String Functions	
ICR #:	10104	
Description:	The $$LJ^XLFSTR extrinsic function returns a left justified character string.	
Format:	`$$LJ^XLFSTR(s,i[,p])`	
Input Parameters:	**s**:	(required) Character string.
	i:	(required) Field size. If this second parameter contains a trailing **T**, this extrinsic function returns the output truncated to the field size specified.
	p:	(optional) Pad character.
Output:	returns:	Returns the left justified string.

31.9.3.1 Examples

31.9.3.1.1 Example 1

Figure 465: $$LJ^XLFSTR API—Example 1

```
>W "[",$$LJ^XLFSTR("TOM",10),"]"
[TOM       ]
```

31.9.3.1.2 Example 2

Figure 466: $$LJ^XLFSTR API—Example 2

```
>W "[",$$LJ^XLFSTR("TOM",10,"-"),"]"
[TOM-------]
```

31.9.3.1.3 Example 3

Figure 467: $$LJ^XLFSTR API—Example 3

```
>W $$LJ^XLFSTR("123456789",5)
123456789
```

31.9.3.1.4 Example 4

Figure 468: $$LJ^XLFSTR API—Example 4

```
>W $$LJ^XLFSTR(123456789,"5T")
12345
```

31.9.4 $$LOW^XLFSTR(): Convert String to Lowercase

Reference Type:	Supported
Category:	String Functions
ICR #:	10104
Description:	The $$LOW^XLFSTR extrinsic function returns an input string converted to all lowercase.
Format:	`$$LOW^XLFSTR(x)`

Input Parameters:	**x**:	(required) Character string.
Output:	returns:	Returns the input string converted to all lowercase.

31.9.4.1 Example

Figure 469: $$LOW^XLFSTR API—Example

```
>S X=$$LOW^XLFSTR("JUSTICE")

>W X
justice
```

31.9.5 $$REPEAT^XLFSTR(): Repeat String

Reference Type:	Supported	
Category:	String Functions	
ICR #:	10104	
Description:	The $$REPEAT^XLFSTR extrinsic function returns a string that repeats the value of **x** for **y** number of times.	
Format:	`$$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.

31.9.5.1 Examples

31.9.5.1.1 Example 1

Figure 470: $$REPEAT^XLFSTR API—Example 1

```
>S X=$$REPEAT^XLFSTR("-",10)

>W X
----------
```

31.9.5.1.2 Example 2

Figure 471: $$REPEAT^XLFSTR API—Example 2

```
>S X=$$REPEAT^XLFSTR("blue water ",5)

>W X
blue water blue water blue water blue water blue water
```

31.9.6 $$REPLACE^XLFSTR(): Replace Strings

Reference Type:	Supported	
Category:	String Functions	
ICR #:	10104	
Description:	The $$REPLACE^XLFSTR extrinsic function uses a multi-character **$Translate** to return a string with the specified string replaced.	
Format:	`$$REPLACE^XLFSTR(in,.spec)`	
Input Parameters:	**in**:	(required) Input string.
	.spec:	(required) An array passed by reference.
Output:	returns:	Returns the replaced string.

31.9.6.1 Examples

31.9.6.1.1 Example 1

Figure 472: $$REPLACE^XLFSTR API—Example 1

```
>SET spec("aa")="a",spec("pqr")="alabama"
>S X=$$REPLACE^XLFSTR("aaaaaaqraaaaaaa",.spec)

>W X
aaaaalabamaaaaa
```

31.9.6.1.2 Example 2

Figure 473: $$REPLACE^XLFSTR API—Example 2

```
>SET spec("F")="VA File",spec("M")="Man"
>S X=$$REPLACE^XLFSTR("FM",.spec)

>W X
VA FileMan
```

31.9.7 $$RJ^XLFSTR(): Right Justify String

Reference Type:	Supported	
Category:	String Functions	
ICR #:	10104	
Description:	The $$RJ^XLFSTR extrinsic function returns a right justified character string.	
Format:	`$$RJ^XLFSTR(s,i[,p])`	
Input Parameters:	**s**:	(required) Character string.
	i:	(required) Field size. If this second parameter contains a trailing **T**, this extrinsic function returns the output truncated to the field size specified.
	p:	(optional) Pad character.
Output:	returns:	Returns the right justified string.

31.9.7.1 Examples

31.9.7.1.1 Example 1

Figure 474: $$RJ^XLFSTR API—Example 1

```
>W "[",$$RJ^XLFSTR("TOM",10),"]"
[       TOM]
```

31.9.7.1.2 Example 2

Figure 475: $$RJ^XLFSTR API—Example 2

```
>W "[",$$RJ^XLFSTR("TOM",10,"-"),"]"
[-------TOM]
```

31.9.7.1.3 Example 3

Figure 476: $$RJ^XLFSTR API—Example 3

```
>W $$RJ^XLFSTR("123456789",5)
123456789
```

31.9.7.1.4 Example 4

Figure 477: $$RJ^XLFSTR API—Example 4

```
>W $$RJ^XLFSTR(123456789,"5T")
12345
```

31.9.8 $$SENTENCE^XLFSTR(): Convert String to Sentence Case

Reference Type:	Supported
Category:	String Functions
ICR #:	10104
Description:	The $$SENTENCE^XLFSTR extrinsic function returns an input string converted to Sentence case. The initial character of each sentence in the input string is 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 any of the following:

- Period (**.**)
- Exclamation point (**!**)
- Question mark (**?**)

NOTE: This API was released with Kernel Patch XU*8.0*400.

Format:	`$$SENTENCE^XLFSTR(x)`	
Input Parameters:	**x**:	(required) Character string.
Output:	returns:	Returns the string converted to Sentence case format.

31.9.8.1 Example

Figure 478: $$SENTENCE^XLFSTR API—Example

```
>S X=$$SENTENCE^XLFSTR("HELLO WORLD!!! THIS IS A CAPITALIZED SENTENCE. this
is not.")

>W X
Hello world!!! This is a capitalized sentence. This is not.
```

31.9.9 $$STRIP^XLFSTR(): Strip a String

Reference Type:	Supported	
Category:	String Functions	
ICR #:	10104	
Description:	The $$STRIP^XLFSTR extrinsic function returns a string stripped of all instances of a specified character.	
Format:	`$$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.

31.9.9.1 Examples

31.9.9.1.1 Example 1

Figure 479: $$STRIP^XLFSTR API—Example 1

```
>S X=$$STRIP^XLFSTR("hello","e")

>W X
hllo
```

31.9.9.1.2 Example 2

Figure 480: $$STRIP^XLFSTR API—Example 2

```
>S X=$$STRIP^XLFSTR("Mississippi","i")

>W X
Msssspp
```

31.9.10 $$TITLE^XLFSTR(): Convert String to Title Case

Reference Type: Supported

Category: String Functions

ICR #: 10104

Description: The $$TITLE^XLFSTR 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.

NOTE: This API was released with Kernel Patch XU*8.0*400.

Format: `$$TITLE^XLFSTR(x)`

Input Parameters:	**x**:	(required) Character string.
Output:	returns:	Returns the string converted to Title case format.

31.9.10.1 Example

Figure 481: $$TITLE^XLFSTR API—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.
```

31.9.11 $$TRIM^XLFSTR(): Trim String

Reference Type:	Supported	
Category:	String Functions	
ICR #:	10104	
Description:	The $$TRIM^XLFSTR extrinsic function trims spaces or other specified characters from the left, right, or both ends of an input string.	
Format:	`$$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.

31.9.11.1 Examples

31.9.11.1.1 Example 1

In Figure 482, 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):

Figure 482: $$TRIM^XLFSTR API—Example 1

```
>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.

31.9.11.1.2 Example 2

In Figure 483, 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):

Figure 483: $$TRIM^XLFSTR API—Example 2

```
>S X="["_$$TRIM^XLFSTR("//A B C//",,"/")_"]"

>W X
[A B C]
```

The second input parameter defaults to **LR**.

31.9.11.1.3 Example 3

In Figure 484, we are trimming the slashes from the left end of the string (the brackets are added to more clearly display the trimmed string):

Figure 484: $$TRIM^XLFSTR API—Example 3

```
>S X="["_$$TRIM^XLFSTR("//A B C//","L","/")_"]"

>W X
[A B C//]
```

31.9.11.1.4 Example 4

In Figure 485, we are trimming the slashes from the right end of the string (the brackets are added to more clearly display the trimmed string):

Figure 485: $$TRIM^XLFSTR API—Example 4

```
>S X="["_$$TRIM^XLFSTR("//A B C//","r","/")_"]"

>W X
[//A B C]
```

31.9.12 $$UP^XLFSTR(): Convert String to Uppercase

Reference Type:	Supported	
Category:	String Functions	
ICR #:	10104	
Description:	The $$UP^XLFSTR extrinsic function returns an input string converted to all uppercase.	
Format:	`$$UP^XLFSTR(x)`	
Input Parameters:	**x**:	(required) Character string.
Output:	returns:	Returns the string converted to all uppercase.

31.9.12.1 Example

Figure 486: $$UP^XLFSTR API—Example

```
>S X=$$UP^XLFSTR("freedom")

>W X
FREEDOM
```

31.10 Utility Functions—XLFUTL

These functions are provided to help with a variety of tasks.

31.10.1 $$BASE^XLFUTL(): Convert Between Two Bases

Reference Type:	Supported	
Category:	Utility Functions	
ICR #:	2622	
Description:	The $$BASE^XLFUTL extrinsic function converts a number from one base to another. The base *must* be between **2** and **16**, both **from** and **to** parameters.	
Format:	`$$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.

31.10.1.1 Examples

31.10.1.1.1 Example 1

Figure 487: $$BASE^XLFUTL API—Example 1

```
>S X=$$BASE^XLFUTL(1111,2,16)

>W X
F
```

31.10.1.1.2 Example 2

Figure 488: $$BASE^XLFUTL API—Example 2

```
>S X=$$BASE^XLFUTL(15,10,16)

>W X
F
```

31.10.1.1.3 Example 3

Figure 489: $$BASE^XLFUTL API—Example 3

```
>S X=$$BASE^XLFUTL("FF",16,10)

>W X
255
```

31.10.2 $$CCD^XLFUTL(): Append Check Digit

Reference Type: Supported

Category: Utility Functions

ICR #: 2622

Description: The $$CCD^XLFUTL extrinsic function returns a number appended with a computed check digit. To check if the original number corresponds with the appended check digit, use the $$VCD^XLFUTL(): Verify Integrity API.

Format: `$$CCD^XLFUTL(x)`

Input Parameters: **x**: (required) Integer for which the check digit is computed.

REF: See "The Taylor Report" in Computerworld magazine, 1975, for the algorithm.

NOTE: This Check Digit algorithm is considered obsolete. Developers are advised to consider other alternatives to validate data integrity. Alternatives include using:

- AES Encryption/Decryption: $$AESENCR^XUSHSH and $$AESDECR^XUSHSH.
- Secure Hash Algorithm (SHA) hashing: $$SHAHASH^XUSHSH or $$SHAN^XLFSHAN for strings.
- Other SHA hash APIs can be used to validate data integrity for: files: $$FILE^XLFSHAN or $$HOSTFILE^XLFSHAN; routines: $$ROUTINE^XLFSHAN;

globals: $$GLOBAL^XLFSHAN;
and messages: $$LSHAN^XLFSHAN.

Output:	returns:	Returns the number with appended check digit.

31.10.2.1 Examples

31.10.2.1.1 Example 1

Figure 490: $$CCD^XLFUTL API—Example 1

```
>S X=$$CCD^XLFUTL(99889)

>W X
998898
```

31.10.2.1.2 Example 2

Figure 491: $$CCD^XLFUTL API—Example 2

```
>S X=$$CCD^XLFUTL(7654321)

>W X
76543214
```

31.10.3 $$CNV^XLFUTL(): Convert Base 10 to Another Base

Reference Type: Supported

Category: Utility Functions

ICR #: 2622

Description: The $$CNV^XLFUTL 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.

31.10.3.1 Examples

31.10.3.1.1 Example 1

Figure 492: $$CNV^XLFUTL API—Example 1

```
>S X=$$CNV^XLFUTL(15,2)

>W X
1111
```

31.10.3.1.2 Example 2

Figure 493: $$CNV^XLFUTL API—Example 2

```
>S X=$$CNV^XLFUTL(255,2)

>W X
11111111
```

31.10.3.1.3 Example 3

Figure 494: $$CNV^XLFUTL API—Example 3

```
>S X=$$CNV^XLFUTL(255,8)

>W X
377
```

31.10.4 $$DEC^XLFUTL(): Convert Another Base to Base 10

Reference Type:	Supported	
Category:	Utility Functions	
ICR #:	2622	
Description:	The $$DEC^XLFUTL extrinsic function converts a number from a specified base, which *must* be between **2** and **16**, to Base **10**.	
Format:	`$$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**.

31.10.4.1 Example

Figure 495: $$DEC^XLFUTL API—Example

```
>S X=$$DEC^XLFUTL("FF",16)

>W X
255
```

31.10.5 $$VCD^XLFUTL(): Verify Integrity

Reference Type:	Supported	
Category:	Utility Functions	
ICR #:	2622	
Description:	The $$VCD^XLFUTL extrinsic function verifies the integrity of a number with an appended check digit. The check digit *must* be appended by the $$CCD^XLFUTL(): Append Check Digit API.	
Format:	`$$VCD^XLFUTL(number)`	
Input Parameters:	**number**:	(required) Number to verify, including appended check digit.
Output:	returns:	Returns: • **1**—Number corresponds to check digit. • **0**—Number does *not* correspond to check digit.

31.10.5.1 Examples

31.10.5.1.1 Example 1

Figure 496: $$VCD^XLFUTL API—Example 1

```
>S X=$$VCD^XLFUTL(76543214)

>W X
1
```

31.10.5.1.2 Example 2

Transposing “**32**” to “23”:

Figure 497: $$VCD^XLFUTL API—Example 2

```
>S X=$$VCD^XLFUTL(76542314)

>W X
0
```

31.11 IP Address Functions—XLFIPV

These calls are provided to standardize the storage and processing of Internet Protocol (IP) addresses. Storing addresses in a standardized format simplifies VA FileMan search and sort functions. It also simplifies the processing of addresses in M routines. When VistA is used in an IPv4/IPv6 dual-stack environment, some performance degradation can occur due to the need to try multiple IP address combinations when making network connections. Therefore, it is important to simplify and standardize this process whenever possible.

31.11.1 $$CONVERT^XLFIPV(): Convert any IP Address to Standardized IP Address Format

Reference Type: Supported

Category: IP Address Functions

ICR #: 5844

Description: The $$CONVERT^XLFIPV extrinsic function converts an Internet Protocol (IP) address (either **IPv4** or **IPv6**) into an IP address in a standardized format, depending upon the system settings:

- **IPv4**—$$FORCEIP4^XLFIPV(): Convert any IP Address to IPv4 API.
- **IPv6**—$$FORCEIP6^XLFIPV(): Convert any IP Address to IPv6 API.

Format: `$$CONVERT^XLFIPV(ip)`

Input Parameters: **ip**: (required) **IPv4** or **IPv6** address (string; in quotes) to be converted.

Output: returns: Returns:

- An **IPv4** address if **IPv6** is disabled on the system.
- An **IPv6** address if **IPv6** is enabled on the system.
- An **IPv4** or **IPv6** **NULL** address if the input *cannot* be converted.

31.11.1.1 Examples

31.11.1.1.1 Example 1 (IPv6 Enabled)

Figure 498: $$CONVERT^XLFIPV API—Example 1

```
>S X=$$CONVERT^XLFIPV("10.126.3.1")

>W X
0000:0000:0000:0000:0000:FFFF:0A7E:0301
```

31.11.1.1.2 Example 2 (IPv6 Disabled)

Figure 499: $$CONVERT^XLFIPV API—Example 2

```
>S X=$$CONVERT^XLFIPV("10.126.3.1")

>W X
10.126.3.1
```

31.11.1.1.3 Example 3 (IPv6 Enabled)

Figure 500: $$CONVERT^XLFIPV API—Example 3

```
>S X=$$CONVERT^XLFIPV("2001:db8::8a2e:370:7334")

>W X
2001:0DB8:0000:0000:0000:8A2E:0370:7334
```

31.11.1.1.4 Example 4 (IPv6 Disabled)

Figure 501: $$CONVERT^XLFIPV API—Example 4

```
>S X=$$CONVERT^XLFIPV("2001:db8::8a2e:370:7334")

>W X
0.0.0.0
```

31.11.2 $$FORCEIP4^XLFIPV(): Convert any IP Address to IPv4

Reference Type:	Supported	
Category:	IP Address Functions	
ICR #:	5844	
Description:	The $$FORCEIP4^XLFIPV extrinsic function converts an IP address (either **IPv4** or **IPv6**) into an **IPv4** address in a standardized format consisting of four decimal numbers, each in the range **0** to **255**. For example: `REDACTED`	
Format:	`$$FORCEIP4^XLFIPV(ip)`	
Input Parameters:	**ip**:	(required) **IPv4** or **IPv6** address (string; in quotes) to be converted.
Output:	returns:	Returns: • An **IPv4** address in "***nnn.nnn.nnn.nnn***" notation if the input address is valid and has an **IPv4** equivalent. • The **NULL** address "**0.0.0.0**" if the input address is invalid. • The **NULL** address "**0.0.0.0**" if an **IPv6** address is input that does *not* have an **IPv4** equivalent.

31.11.2.1 Examples

31.11.2.1.1 Example 1

Figure 502: $$FORCEIP4^XLFIPV API—Example 1

```
>S X=$$FORCEIP4^XLFIPV("REDACTED")

>W X
REDACTED
```

31.11.2.1.2 Example 2

Figure 503: $$FORCEIP4^XLFIPV API—Example 2

```
>S X=$$FORCEIP4^XLFIPV("REDACTED")

>W X
0.0.0.0
```

31.11.2.1.3 Example 3

Figure 504: $$FORCEIP4^XLFIPV API—Example 3

```
>S X=$$FORCEIP4^XLFIPV("2001:db8::8a2e:370:7334")

>W X
0.0.0.0
```

31.11.2.1.4 Example 4

Figure 505: $$FORCEIP4^XLFIPV API—Example 4

```
>S X=$$FORCEIP4^XLFIPV("::ffff:REDACTED")

>W X
10.126.3.1
```

31.11.2.1.5 Example 5

Figure 506: $$FORCEIP4^XLFIPV API—Example 5

```
>S X=$$FORCEIP4^XLFIPV("::ffff:c000:2eb")

>W X
192.0.2.235
```

31.11.3 $$FORCEIP6^XLFIPV(): Convert any IP Address to IPv6

Reference Type:	Supported	
Category:	IP Address Functions	
ICR #:	5844	
Description:	The $$FORCEIP6^XLFIPV extrinsic function converts an IP address (either **IPv4** or **IPv6**) into an **IPv6** address in a standardized format consisting of eight groups of hexadecimal numbers separated by colons. For example: `2001:0DB8:85A3:0042:0000:8A2E:0370:7334`	
Format:	`$$FORCEIP6^XLFIPV(ip)`	
Input Parameters:	**ip**:	(required) **IPv4** or **IPv6** address (string; in quotes) to be converted.
Output:	returns:	Returns: • An **IPv6** address in "***hhhh:hhhh:hhhh:hhhh:hhhh:hhhh:hhhh:hhhh***" notation if the input address is valid and has an **IPv6** equivalent. • The **NULL** address "**0000:0000:0000:0000:0000:0000:0000:0000**" if the input address is invalid.

31.11.3.1 Examples

31.11.3.1.1 Example 1

Figure 507: $$FORCEIP6^XLFIPV API—Example 1

```
>S X=$$FORCEIP6^XLFIPV("10.126.3.1")

>W X
0000:0000:0000:0000:0000:FFFF:0A7E:0301
```

31.11.3.1.2 Example 2

Figure 508: $$FORCEIP6^XLFIPV API—Example 2

```
>S X=$$FORCEIP6^XLFIPV("10.999.3.1")

>W X
0000:0000:0000:0000:0000:0000:0000:0000
```

31.11.3.1.3 Example 3

Figure 509: $$FORCEIP6^XLFIPV API—Example 3

```
>S X=$$FORCEIP6^XLFIPV("2001:db8::8a2e:370:7334")

>W X
2001:0DB8:0000:0000:0000:8A2E:0370:7334
```

31.11.3.1.4 Example 4

Figure 510: $$FORCEIP6^XLFIPV API—Example 4

```
>S X=$$FORCEIP6^XLFIPV("::ffff:10.126.3.1")

>W X
0000:0000:0000:0000:0000:FFFF:0A7E:0301
```

31.11.3.1.5 Example 5

Figure 511: $$FORCEIP6^XLFIPV API—Example 5

```
>S X=$$FORCEIP6^XLFIPV("127.0.0.1")

>W X
0000:0000:0000:0000:0000:0000:0000:0001
```

31.11.4 $$VALIDATE^XLFIPV(): Validate IP Address Format

Reference Type:	Supported	
Category:	IP Address Functions	
ICR #:	5844	
Description:	The $$VALIDATE^XLFIPV extrinsic function validates the format of an IP address (either **IPv4** or **IPv6**).	
Format:	`$$VALIDATE^XLFIPV(ip)`	
Input Parameters:	**ip**:	(required) **IPv4** or **IPv6** address (string) to be validated.
Output:	returns:	Returns: • **1**—If the IP address is in a valid format. • **0**—If the format is invalid or **NULL** input.

31.11.4.1 Examples

31.11.4.1.1 Example 1

Figure 512: $$VALIDATE^XLFIPV API—Example 1

```
>S X=$$VALIDATE^XLFIPV(10.126.3.1)

>W X
1
```

31.11.4.1.2 Example 2

Figure 513: $$VALIDATE^XLFIPV API—Example 2

```
>S X=$$VALIDATE^XLFIPV(10.999.3.1)

>W X
0
```

31.11.4.1.3 Example 3

Figure 514: $$VALIDATE^XLFIPV API—Example 3

```
>S X=$$VALIDATE^XLFIPV(2001:db8::8a2e:370:7334)

>W X
1
```

31.11.4.1.4 Example 4

Figure 515: $$VALIDATE^XLFIPV API—Example 4

```
>S X=$$VALIDATE^XLFIPV(2001:db8::8g2h:370:7334)

>W X
0
```

31.11.5 $$VERSION^XLFIPV: Show System Settings for IPv6

Reference Type: Supported

Category: IP Address Functions

ICR #: 5844

Description: The $$VERSION^XLFIPV extrinsic function determines the system settings for **IPv6**.

Format: `$$VERSION^XLFIPV`

Input Parameters: none.

Output: returns: Returns:

- **1**—If **IPv6** is enabled.
- **0**—If **IPv6** is disabled.

31.11.5.1 Examples

31.11.5.1.1 Example 1: IPv6 Enabled

Figure 516: $$VERSION^XLFIPV API—Example 1: IPv6 Enabled

```
>S X=$$VERSION^XLFIPV

>W X
1
```

31.11.5.1.2 Example 2: IPv6 Disabled

Figure 517: $$VERSION^XLFIPV API—Example 2: IPv6 Disabled

```
>S X=$$VERSION^XLFIPV

>W X
0
```

31.12 JSON Conversion Functions—XLFJSON

These calls are provided to standardize the conversion of a global or array to the JavaScript Object Notation (JSON) format, and JSON to a global or array format. They also include extrinsic functions to prepare strings for the JSON conversion process, by escaping (making JSON compliant) or unescaping (making code compliant) strings.

31.12.1 DECODE^XLFJSON(): Convert a JSON Object into a Closed Array Reference

Reference Type: Supported

Category: JSON Conversion Functions

ICR #: 6682

Description: The DECODE^XLFJSON API converts a **JSON** object into a closed array reference.

NOTE: This API was released with Kernel Patch XU*8.0*680.

Format: `DECODE^XLFJSON(xujson,xuroot[,xuerr])`

Input Parameters: **xujson**: (required) A string or array containing a serialized JSON object.

Output Parameters: **xuroot**: (required) A closed array reference for M representation of the object.

xuerr: (optional) This contains error messages. If *not* defined, defaults to **^TMP("XLFJERR",$J)**.

31.12.1.1 Example

Figure 518: DECODE^XLFJSON API—Example

```
>S INJSON(1)="{""menu"":{""id"":""file"",""popup"":{""menuitem"":[{""value
"": ""New"",""onclick"":""CreateNewDoc()""},"

>S INJSON(2)="{""value"": ""Open"",""onclick"": ""OpenDoc()""},{""value"":
""Close"",""onclick"": ""CloseDoc()""}]} ,"

>S INJSON(3)="""value"":""File""}}"

>D DECODE^XLFJSON("INJSON","OUTJSON","ERRORS")

>ZW OUTJSON
OUTJSON("menu","id")="file"
OUTJSON("menu","popup","menuitem",1,"onclick")="CreateNewDoc()"
OUTJSON("menu","popup","menuitem",1,"value")="New"
OUTJSON("menu","popup","menuitem",2,"onclick")="OpenDoc()"
OUTJSON("menu","popup","menuitem",2,"value")="Open"
OUTJSON("menu","popup","menuitem",3,"onclick")="CloseDoc()"
OUTJSON("menu","popup","menuitem",3,"value")="Close"
OUTJSON("menu","value")="File"
```

31.12.2 ENCODE^XLFJSON(): Convert Closed Array or Global Reference to a JSON Object

Reference Type: Supported

Category: JSON Conversion Functions

ICR #: 6682

Description: The ENCODE^XLFJSON API converts a closed array or global reference to a JSON object.

NOTE: This API was released with Kernel Patch XU*8.0*680.

Format: `ENCODE^XLFJSON(xuroot,xujson[,xuerr])`

Input Parameters: **xuroot**: (required) A closed array reference for M representation of the object.

Output Parameters: xujson: (required) A string or array containing a serialized JSON object.

xuerr: (optional) This contains error messages. If *not* defined, defaults to **^TMP("XLFJERR",$J)**.

31.12.2.1 Example

Figure 519: ENCODE^XLFJSON API—Example

```
>S Y("menu","id")="file"

>S Y("menu","popup","menuitem",1,"onclick")="CreateNewDoc()"

>S Y("menu","popup","menuitem",1,"value")="New"

>S Y("menu","popup","menuitem",2,"onclick")="OpenDoc()"

>S Y("menu","popup","menuitem",2,"value")="Open"

>S Y("menu","popup","menuitem",3,"onclick")="CloseDoc()"

>S Y("menu","popup","menuitem",3,"value")="Close"

>S Y("menu","value")="File"

>D ENCODE^XLFJSON("Y","OUTJSON","ERRORS")

>W OUTJSON(1)
{"menu":{"id":"file","popup":{"menuitem":[{"onclick":"CreateNewDoc()","value
":"N
ew"},{"onclick":"OpenDoc()","value":"Open"},{"onclick":"CloseDoc()","value":
"Clo
se"}]},"value":"File"}}
```

31.12.3 $$ESC^XLFJSON(): Escape String to JSON

Reference Type: Supported

Category: JSON Conversion Functions

ICR #: 6682

Description: The $$ESC^XLFJSON extrinsic function returns an escaped string in a JSON format.

NOTE: This API was released with Kernel Patch XU*8.0*680.

Format: `$$ESC^XLFJSON(x)`

Input Parameters:	x:	(required) A string to be escaped to a JSON format.
Output:	returns:	Returns a JSON escaped string.

31.12.3.1 Example

Figure 520: $$ESC^XLFJSON API—Example

```
>W $$ESC^XLFJSON("\one\two\three\")
\\one\\two\\three\\
```

31.12.4 $$UES^XLFJSON(): Unescape JSON to a String

Reference Type: Supported

Category: JSON Conversion Functions

ICR #: 6682

Description: The $$UES^XLFJSON extrinsic function returns a unescaped string from a JSON format.

NOTE: This API was released with Kernel Patch XU*8.0*680.

Format: `$$UES^XLFJSON(x)`

Input Parameters:	x:	(required) A JSON escaped string to be unescaped.
Output:	returns:	Returns a unescaped string representation of the escaped JSON input string.

31.12.4.1 Example

Figure 521: $$UES^XLFJSON API—Example

```
>W $$UES^XLFJSON("\\one\\two\\three\\")
\one\two\three\
```

32 XML Parser (VistA): Developer Tools

32.1 Overview

The VistA Extensible Markup Language (XML) Parser is a full-featured, validating XML parser written in the M programming language and designed to interface with the VistA suite of M-based applications. It is *not* a standalone product. Rather, it acts as a server application that can provide XML parsing capabilities to any client application that subscribes to the application programmer interface (API) specification detailed in this document.

The VistA XML Parser employs two very different API implementations:

- Event-Driven Interface
- World Wide Web Consortium Document Object Model Specification

The choice of which API to employ is in part dependent on the needs of the application developer. The event-driven interface requires the client application to process the document in a strictly top-down manner. In contrast, the in-memory model provides the ability to move freely throughout the document and has the added advantage of ensuring that the document is well formed and valid before any information is returned to the client application.

The VistA XML Parser employs an Entity Catalog to allow storage of external entities such as document type definitions. The Entity Catalog is a VA FileMan-compatible database and can be manipulated using the usual VA FileMan tools.

32.1.1 Event-Driven Interface

The event-driven interface is modeled after the widely used Simple API for **XML (SAX)** interface specification. In this implementation, a client application provides a special handler for each parsing event of interest. When the client invokes the parser, it conveys *not* only the document to be parsed, but also the entry points for each of its event handlers. As the parser progresses through the document, it invokes the client's handlers for each parsing event for which a handler has been registered.

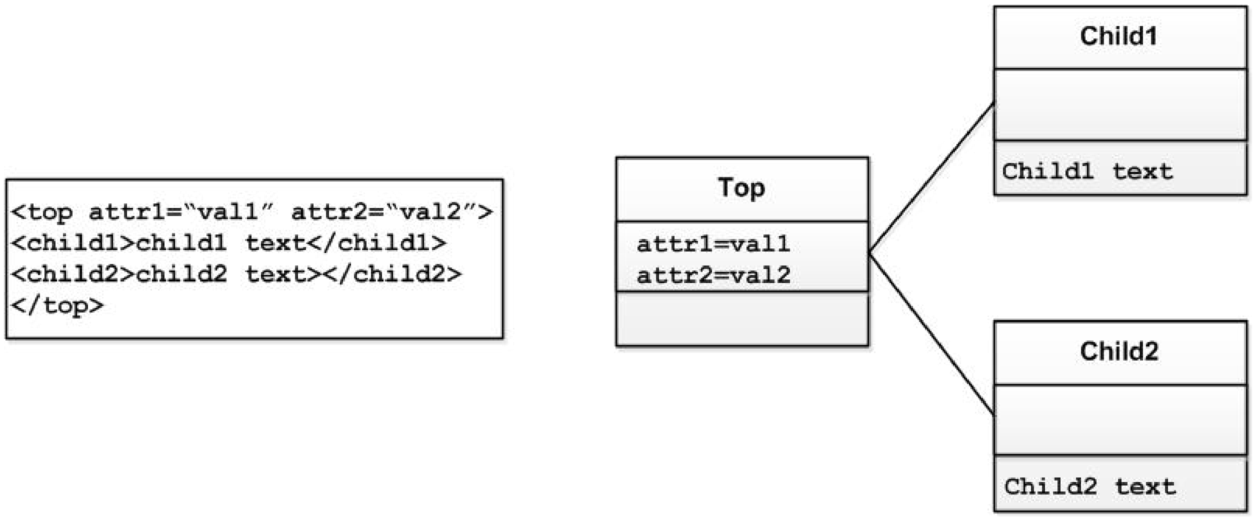
32.1.2 World Wide Web Consortium Document Object Model Specification

This API implementation is based on the World Wide Web Consortium (W3Cs) Document Object Model (DOM) specification. This API, which is actually built on top of the event-driven interface, first constructs an in-memory model of the fully parsed and validated document. It then provides methods to navigate through and extract information from the parsed document.

This API is actually layered on top of the event-driven API. In other words, it is actually a client of the event-driven API that in turn acts as a server to another client application.

The document image is represented internally as a tree with each node in the tree representing an element instance. Attributes (names and values), *non*-markup text, and comment text may be associated with any given node. For example, in Figure 522 the XML document on the left is represented by the tree structure on the right.

Figure 522: XML Document (left)—Tree Structure Diagram (right)



```
<top attr1="val1" attr2="val2">
<child1>child1 text</child1>
<child2>child2 text></child2>
</top>
```

32.1.3 Entity Catalog

The XML ENTITY CATALOG (#950) file is used to store external entities and their associated public identifiers. When the XML parser encounters an external entity reference with a public identifier, it first looks for that public identifier in the entity catalog. If it finds the entity, it retrieves its value. Otherwise, it attempts to retrieve the entity value using the system identifier. The problem with using system identifiers is that they often identify resources that may have been relocated since the document was authored. (This is analogous to the problem with broken links in HTML documents.) Using public identifiers and an entity catalog allows one to build a collection of commonly used and readily accessible external entities (e.g., external document type definitions).

The XML ENTITY CATALOG (#950) file is a VA FileMan-compatible file that is very simple in structure as shown in Table 46.

Table 46: XML ENTITY CATALOG (#950) File—Stores External Entities and Assoc Public Identifiers

Field #	Field Name	Datatype	Description
.01	ID	Free text (1-250)	The public identifier associated with this entity.
1	VALUE	Word Processing	The text associated with the entity.

32.1.4 Term Definitions and XML Parser Concept

To understand the terms used in this section and the concept of the operation of an XML Parser, please review the W3C Architecture Domain website, Extensible Markup Language (XML) page at: http://www.w3.org/XML/

The Toolkit VistA XML Parser Application Programming Interfaces (APIs) have been developed to assist you in creating an XML document.

Integration Control Registration #3561 defines the various callable entry points in the **MXMLDOM** routine. These APIs are based on the W3C's Document Object Model (DOM) specification. It first builds an "in-memory" image of the fully parsed and validated document and then provides a set of methods to permit structured traversal of the document and extraction of its contents. This API is actually layered on top of the event-driven API. In other words, it is actually a client of the event-driven API that in turn acts as a server to another client application.

REF: The VistA Extensible Markup Language (XML) Parser technical and user documentation can be found on the VA Software Document Library (VDL) located at: http://www.va.gov/vdl/application.asp?appid=137

32.1.5 Known Issues

The following are known issues in this version of the XML parser:

- Unsupported Character Encodings
- Retrieval of External Entities Using Non-Standard File Access Protocols
- File Access
- Entity Substitutions Text
- Enforcing Whitespace

Some of these are due to certain limitations of the M programming language.

32.1.5.1 Unsupported Character Encodings

Unlike languages like Java that have multiple character encoding support built-in, M does *not* recognize character encodings that do *not* incorporate the printable ASCII character subset. Thus, **16**-bit character encodings (e.g., Unicode) are *not* supported. Fortunately, a large number of **8**-bit character encodings do incorporate the printable ASCII character subset and can be parsed. Because of this limitation, the VistA XML Parser rejects any documents with unsupported character encodings.

32.1.5.2 Retrieval of External Entities Using Non-Standard File Access Protocols

The current version of the VistA XML Parser does *not* support retrieval of external entities using the HTTP or FTP protocols (or for that matter, any protocols other than the standard file access protocols of the underlying operating system). Client applications using the event-driven interface can intercept external entity retrieval by the parser and implement support for these protocols if desired.

32.1.5.3 File Access

The parser uses the Kernel function FTG^%ZISH for file access. This function reads the entire contents of a file into an M global. There are several nuances to this function that manifest themselves in parser operation:

- Files are opened with a **time-out** parameter. If an attempt is made to access a *non-*existent file, there is a delay of a few seconds before the error is signaled.
- Files are accessed in text mode. The result is that certain imbedded control characters are stripped from the input stream and never detected by the parser. Because these control characters are disallowed by XML, the parser does *not* report such documents as *non-*conforming.
- A line feed/carriage return sequence at the end of a document is stripped and *not* presented to the parser. Only in rare circumstances would this be considered significant data, but in the strictest sense should be preserved.

32.1.5.4 Entity Substitutions Text

The parser allows external entities to contain substitution text that in some cases would violate XML rules that state that a document *must* be conforming in the absence of resolving such references. In other words, XML states that a *non*-validating parser should be able to verify that a document is conforming without processing external entities. This restriction constrains how token streams can be continued across entities. The parser recognizes most, but *not* all, of these restrictions. The effect is that the parser is more lax in allowing certain kinds of entity substitutions.

32.1.5.5 Enforcing Whitespace

Parsers vary in how they enforce whitespace that is designated as required by the XML specification. This parser flags the absence of any required whitespace as a conformance error, even in situations where the absence of such whitespace would *not* introduce syntactic ambiguity. The result is that this parser rejects some documents that may be accepted by other parsers.

32.2 Application Programming Interface (API)

The Toolkit VistA XML Parser Application Programming Interfaces (APIs) have been developed to assist you in creating an XML document.

Integration Control Registration #3561 defines the various callable entry points in the **MXMLDOM** routine. These APIs are based on the W3C's Document Object Model (DOM) specification. It first builds an "in-memory" image of the fully parsed and validated document and then provides a set of methods to permit structured traversal of the document and extraction of its contents. This API is actually layered on top of the event-driven API. In other words, it is actually a client of the event-driven API that in turn acts as a server to another client application.

Several APIs are available for developers to work with the EXtensible Markup Language (XML). These APIs are described below.

32.2.1 $$ATTRIB^MXMLDOM(): XML—Get First or Next Node Attribute Name

Reference Type:	Supported	
Category:	XML Parser (VistA)	
ICR #:	3561	
Description:	The $$ATTRIB^MXMLDOM extrinsic function returns the first or next attribute associated with the specified node.	
Format:	`$$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: • Name (string) of the first or next attribute associated with the specified node. • **NULL** if there are none remaining.

32.2.2 $$CHILD^MXMLDOM(): XML—Get Parent Node's First or Next Child

Reference Type: Supported

Category: XML Parser (VistA)

ICR #: 3561

Description: The $$CHILD^MXMLDOM extrinsic function returns 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 returns 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.

32.2.3 $$CMNT^MXMLDOM(): XML—Extract Comment Text (True/False)

Reference Type: Supported

Category: XML Parser (VistA)

ICR #: 3561

Description: The $$CMNT^MXMLDOM 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.

32.2.4 CMNT^MXMLDOM(): XML—Extract Comment Text (True/False)

Reference Type: Supported

Category: XML Parser (VistA)

ICR #: 3561

Description: The CMNT^MXMLDOM 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.

32.2.5 DELETE^MXMLDOM(): XML—Delete Document Instance

Reference Type:	Supported	
Category:	XML Parser (VistA)	
ICR #:	3561	
Description:	The DELETE^MXMLDOM 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.	

32.2.6 $$EN^MXMLDOM(): XML—Initial Processing of XML Document, Build In-memory Image

Reference Type: Supported

Category: XML Parser (VistA)

ICR #: 3561

Description: The $$EN^MXMLDOM 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 of the following:

- Closed reference to a global root containing the document.
- 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.

- 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**—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:

- **Successful**—A *non*-**zero** handle of the document instance if parsing completed successfully.
- **Unsuccessful—Zero** handle 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.

32.2.7 $$NAME^MXMLDOM(): XML—Get Element Name

Reference Type:	Supported	
Category:	XML Parser (VistA)	
ICR #:	3561	
Description:	The $$NAME^MXMLDOM extrinsic function retrieves the name of the element at the specified node within the document parse tree.	
Format:	`$$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.

32.2.8 $$PARENT^MXMLDOM(): XML—Get Parent Node

Reference Type:	Supported	
Category:	XML Parser (VistA)	
ICR #:	3561	
Description:	The $$PARENT^MXMLDOM extrinsic function returns the parent node of the specified node, or **zero** (**0**) if there is none.	
Format:	`$$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.

32.2.9 $$SIBLING^MXMLDOM(): XML—Get Sibling Node

Reference Type:	Supported	
Category:	XML Parser (VistA)	
ICR #:	3561	
Description:	The $$SIBLING^MXMLDOM extrinsic function returns the node of the specified node's immediate sibling, or **zero** (**0**) if there is none.	
Format:	`$$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.

32.2.10 $$TEXT^MXMLDOM(): XML—Extract Non-markup Text (True/False)

Reference Type:	Supported	
Category:	XML Parser (VistA)	
ICR #:	3561	
Description:	The $$TEXT^MXMLDOM extrinsic function extracts *non*-markup text associated with the specified node.	
Format:	`$$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: • **True (*non*-zero)**—Text was retrieved. • **False (zero)**—Text was *not* retrieved.

32.2.11 TEXT^MXMLDOM(): XML—Extract Non-markup Text (True/False)

Reference Type: Supported

Category: XML Parser (VistA)

ICR #: 3561

Description: The TEXT^MXMLDOM 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.

32.2.12 $$VALUE^MXMLDOM(): XML—Get Attribute Value

Reference Type: Supported

Category: XML Parser (VistA)

ICR #: 3561

Description: The $$VALUE^MXMLDOM extrinsic function returns 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.

32.2.13 EN^MXMLPRSE(): XML—Event Driven API

Reference Type: Supported

Category: XML Parser (VistA)

ICR #: 4149

Description: The EN^MXMLPRSE API is an event-driven interface that is based on the well-established Simple API for XML (SAX) interface employed by many XML parsers. This API has a single method.

In this implementation, a client application provides a special handler for each parsing event of interest. When the client invokes the parser, it conveys not only the document to be parsed, but also the entry points for each of its event handlers. As the parser progresses through the document, it invokes the client's handlers for each parsing event for which a handler has been registered.

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: (required) 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>
```

REF: For the types of supported events and their required parameters, see the “Details” section.

opt: (optional) This is a list of option flags 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.

32.2.13.1 Details

The VistA XML Parser recognizes the event types listed in Table 47:

Table 47: XML Parser—Event Types

Event Type	Parameters	Description
STARTDOCUMENT	None	Notifies the client that document parsing has commenced.
ENDDOCUMENT	None	Notifies the client that document parsing has completed.
DOCTYPE	ROOT PUBID 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 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: **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 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 PUBID GLOBAL	The parser has encountered an external entity reference whose system and public identifiers are given by **SYSID** and **PUBID**, 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

Event Type	Parameters	Description
		entity altogether, it should set both **SYSID** and **PUBID** to **NULL**.
NOTATION	NAME SYSID 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: • **ERR("SEV") —**Severity of the error; Where: o **Zero (0) —**Warning. o **1—**Validation error. o **2—**Conformance error. • **ERR("MSG")—**Brief text description of the error. • **ERR("ARG")—**Token value the triggered the error (optional). • **ERR("LIN")—**Number of the line being processed when the error occurred. • **ERR("POS")—**Character position within the line where the error occurred. • **ERR("XML")—**Original document text of the line where the error occurred.

32.2.13.2 Example

This is a simple example of how to use the VistA XML Parser with an XML document (file). The XML file contains a parent node named **BOOKS**. Nested within that parent node are child nodes named **TITLE** and **AUTHOR**.

Remember the following:

- The parent node is the node whose child nodes are being retrieved.
- The child node, if specified, is the last child node retrieved. The function returns the next child in the list. If the parameter is **zero** or missing, the first child is returned.

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 for the parser entry point. This sample application simply prints a summary of the parsing events as they occur.

1. Create an XML file:

Figure 523: VistA XML Parser Use—Example: Create XML File

```
^TMP($J,1) = <?xml version=‘1.0’?>
^TMP($J,2) = <!DOCTYPE BOOK>
^TMP($J,3) = <BOOK>
^TMP($J,4) = <TITLE>Design Patterns</TITLE>
^TMP($J,5) = <AUTHOR>Author1</AUTHOR>
^TMP($J,6) = <AUTHOR>Author2</AUTHOR>
^TMP($J,7) = <AUTHOR>Author3</AUTHOR>
^TMP($J,8) = <AUTHOR>Author4</AUTHOR>
^TMP($J,9) = </BOOK>
```

2. Invoke simple API for XML (SAX) interface:

Figure 524: VistA XML Parser Use Example—Invoke SAX Interface

```
D EN^MXMLTEST($NA(^TMP($J)),"V")
```

3. Check Document Object Model (DOM) interface:

Figure 525: VistA XML Parser Use Example—Check DOM Interface

```
>S HDL=$$EN^MXMLDOM($NA(^TMP($J)))
```

Write the name of the first node.

```
>W $$NAME^MXMLDOM(HDL,1)
BOOK
```

Get the child of the node.

```
>S CHD=$$CHILD^MXMLDOM(HDL,1)
```

Write the child name.

```
>W $$NAME^MXMLDOM(HDL,CHD)
TITLE
```

Get the text of the child.

```
>W $$TEXT^MXMLDOM(HDL,CHD,$NA(VV))
1

>ZW VV
VV(1)=Design Patterns
```

4. List all sibling nodes:

Figure 526: VistA XML Parser Use Example—List All Sibling Nodes

```
>S CHD=$$CHILD^MXMLDOM(HDL,1)
>S SIB=CHD
>F  S SIB=$$SIBLING^MXMLDOM(HDL,SIB) Q:SIB'>0  W
!,SIB,?4,$$NAME^MXMLDOM(HDL,SIB)
3   AUTHOR
4   AUTHOR
5   AUTHOR
6   AUTHOR
>
```

32.2.14 $$SYMENC^MXMLUTL(): XML—Replace XML Symbols with XML Encoding

Reference Type:	Supported	
Category:	XML Parser (VistA)	
ICR #:	4153	
Description:	The $$SYMENC^MXMLUTL extrinsic function replaces reserved Extensible Markup Language (XML) symbols in a string with their XML encoding for strings used in an XML message.	
Format:	`$$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.

32.2.14.1 Example

Figure 527: $$SYMENC^MXMLUTL API—Example

```
>S X=$$SYMENC^MXMLUTL(“This line isn’t &”“<XML>”“ safe as is.”)

>W X
This line isn&os;t &amp;&quot;&lt;XML&gt;&quot; safe as is.
```

32.2.15 $$XMLHDR^MXMLUTL: XML—Get XML Message Header

Reference Type:	Supported	
Category:	XML Parser (VistA)	
ICR #:	4153	
Description:	The $$XMLHDR^MXMLUTL extrinsic function returns a standard Extensible Markup Language (XML) header for encoding XML messages.	
Format:	`$$XMLHDR^MXMLUTL`	
Input Parameters:	none.	
Output:	returns:	Returns a standard XML header.

32.2.15.1 Example

Figure 528: $$XMLHDR^MXMLUTL API—Example

```
>S X=$$XMLHDR^MXMLUTL

>W X
<?xml version="1.0" encoding="utf-8" ?>
```

33 ^XTMP Global: Developer Tools

33.1 Overview

There is a recurring need by VistA software to store data in a translated global for relatively short periods of time. However, this data needs to be accumulated for a period longer than an individual user's logon session and longer than the time a specific process/job might run. The **^UTILITY**, **^TMP** and **^XUTL** globals do *not* meet the basic requirements for storing this type of data due to the following:

- These globals are *not* translated, and thus, *cannot* be relied upon for transferring data from one job to another.
- The data is *not* stored for excessively long periods of time and is constantly being processed and purged.
- The data is stored in an intermediate form, temporarily, so that it can be further processed in an efficient manner.
- The original data is stored in a VA FileMan file from which the temporary data can be recreated, or on another system (usually *non*-VistA) from which it can be resent, if necessary. Hence, the creation of a VA FileMan file, while feasible, would add unnecessary overhead to the VistA systems.

Therefore, the Standards and Conventions Committee (SACC) asked Kernel to establish the **^XTMP** global, which can be used by *any* VistA software application. This global is dynamic in size and activity, with one copy accessible to *all* members of a UCI, and should be placed accordingly.

CAUTION: The ^XTMP global should *not* be used for long-term storage of data; data requiring long-term storage should be placed within a file. The ^XTMP global should only be used for near-term storage needs and should respect size constraints.

33.2 Rules for Use of the ^XTMP Global

The structure of each top node of the **^XTMP** global has the following format:

```
^XTMP(namespaced- subscript,0)=purge date^createdate^optional descriptive
information
```

(Both dates *must* be in VA FileMan internal date format.)

As per the Standards and Conventions (SAC, Section 2.11.8), developers are encouraged to include other descriptive information on the third piece of the **0** node of the **^XTMP** global (e.g., task description and creator **DUZ**).

1. **First Subscript Must be Namespaced**—The first subscript of the **^XTMP** global *must* be namespaced; however, other characters can follow the namespace. For example, if the namespace for the software is "**RA**," the first subscript could be **"RA"_DUZ**, **"RA"_literal**, **"RA"_$J**, etc. This allows the developer to use the global in different parts of the software.
2. **0 Node Must Exist**—There *must* be a **0** node for the global in which the first piece contains the PURGE DATE in VA FileMan internal date format, and the second piece contains the CREATE DATE in VA FileMan internal date format. For example:

   ```
   ^XTMP("RA1",0)=2920416^2920401
   ```
3. **KILL ^XTMP After Use**—The developer is responsible for **KILL**ing **^XTMP(*x*)** when its use is complete (where "***x***" is their namespaced subscript).
4. **Code Cleanup**—Kernel has included the necessary code in the **XQ82** routine to clean up the **^XTMP** global (e.g., **^XTMP("RA1"**). It **KILL**s this global under any of the following conditions:
   - There is no **0** node (e.g., **^XTMP("RA1",0**).
   - The **0** node does *not* contain a purge date as the first piece.
   - The date in the first piece of the **0** node is the same as or before the system date.

33.3 SAC Exemptions

As of May 17, 2002, the Standards and Conventions (SAC) document has the following exemptions regarding the **^XTMP** global:

- Section 2.3.2.1—Subscripts used in the **^TMP** and **^XTMP** globals can be lowercase.
- Section 2.3.2.5—The **^TMP**, **^UTILITY**, and **^XTMP** globals do *not* have to be VA FileMan compatible.
- Section 2.3.2.5.2—The **^XTMP** global will be translated, with one copy for the entire VistA production system at each site.
- Section 2.7.3.3—All documented temporary scratch global nodes (e.g., **^TMP** and **^UTILITY**) are created by a called supported reference, with the exception of **^XTMP** global data.
- Section 2.7.3.4—All local variables, locks, and scratch global nodes (except **^XTMP**, or other scratch globals designed to be passed between parts of a package) are created by the application.

A new extension *must* be added to the SAC stating that this global should be used as a scratch area when a translated scratch global is required by software applications.

REF: To view the entire SAC document, see the SACC VA Intranet website: REDACTED

Glossary

Term	Definition
ALERTS	An alert notifies one or more users of a matter requiring immediate attention. Alerts function as brief notices that are distinct from mail messages or triggered bulletins. Alerts are designed to provide interactive notification of pending computing activities (e.g., the need to reorder supplies or review a patient's clinical test results). Along with the alert message is an indication that the **View Alerts** common option should be chosen to take further action. An alert includes any specifications made by the developer 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 (#8992) file.
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 developer 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).
ALPHA TESTING	In VA terminology, Alpha testing is when a VistA test software application is running in a site's account.
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.

Term	Definition
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.
BETA TESTING	In VA terminology, Beta testing is when a VistA test software application is running in a Production account.
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 referred to as the "up-arrow" symbol.
CHECKSUM	A numeric value that is the result of a mathematical computation involving the characters of a routine or file.
CIPHER	A system that arbitrarily represents each character as one or more other characters. (See also: ENCRYPTION.)
COMMON MENU	Options that are available to all users. Entering two question marks (**??**) at the menu's select prompt displays 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 do *not*, however, see the results of the mathematical function on the screen. Only when you are printing or displaying information on the screen do 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.

Term	Definition
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 developer 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 rejects it and prompts 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.
INIT	Initialization of a software application. **INIT*** routines are built by VA FileMan’s **DIFROM** and, when run, recreate a set of files and other software components.
JSON	JavaScript Object Notation.

Term	Definition
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 to which you wish to jump. (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.

Term	Definition
PACKAGE IDENTIFIER	An optional identifier that the developer can use to identify the alert for such purposes as subsequent lookup and deletion (**XQAID** input variable).
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 system administrators. This menu should include most of the computing activities the user needs.
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 developer's at-sign security code, which sets the variable **DUZ(0)=@**.
PROTOCOL	An entry in the PROTOCOL (#101) file. 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.
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.
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 developer and implemented by system administrators. The process is controlled via the RESOURCE (#3.54) file.

Term	Definition
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. (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 system administrators (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 (#19) file. 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 (#19) file 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 (#3.5) file. It uses the associated operating system's spool facility, whether it is 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 subsequently transfers the text to a global for subsequent despooling (printing).
SYNONYM	In VistA, a field in the OPTION (#19) file. Options can be selected by their menu text or synonym.
TASKMAN	The Kernel module that schedules and processes background tasks (also called Task Manager).

Term	Definition
TIMED READ	The amount of time Kernel waits for a user response to an interactive **READ** command before starting to halt the process.
UP-ARROW JUMP	In the menu system, entering a caret (**^**) followed by an option name accomplishes a jump to the target option without needing to take the usual steps through the menu pathway.
XINDEX	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 ^XINDEX**).
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 VistA application developers so that separate versions of the software need *not* be written for each operating system.

REF: For a list of commonly used terms and definitions, see the OIT Master Glossary VA Intranet Website.

For a list of commonly used acronyms, see the VA Acronym Lookup Intranet Website.

Index