DEPARTMENT OF HEALTH & HUMAN SERVICES

Public Health Service

Indian Health Service
Albuquerque NM 87110

JUN 27 1996

TO: Director, Office of Information Resource Management

FROM: Acting Director, Division of Systems Management

SUBJECT: RPMS Software Handbook and RPMS Programming Standards & Conventions

Issue

The attached RPMS Software Handbook which includes the RPMS Programming Standards and Conventions (SAC), Volume 2.3.2, is approved for distribution and implementation.

Background

The Resource and Patient Management System (RPMS) is a suite of software applications utilized throughout IHS facilities in support of administrative and clinical functions. The environment that evolved from a myriad of development effort is highly complex and integrated. The Software Review and Certification Branch has developed a RPMS Software Handbook to guide the maintenance and development effort.

The RPMS Software Handbook contains a myriad of documents including the completely revamped RPMS Programming Standards and Conventions, RPMS Documentation Standards, and a multitude of various policies, procedures and crib notes aimed at guiding the software development process. In the past most of these documents existed in one format or another but were never readily available in one place. By placing all these documents in one place any developer or person involved with software development and maintenance would have a convenient handbook of all procedures at their fingertips. In addition, the handbook has divider pages for each of the various sections. The electronic format has a bookmark defined at each of these dividers for easy access to a specific chapter. The entire document has been indexed and has an extensive Table of Contents. Future additions can easily be made by updating a specific section or adding a new section. The document is intended to be used in a loose leaf notebook to accommodate future changes or additions.

Approval

The attached June 27, 1996 edition of the SAC document is approved as the official RPMS ADP Systems Manual, Volume 2.3.2, Programming Standards and Conventions. There will be a phase-in period of six months (December 27, 1996) where applications submitted for certification may adhere to either the new or the old SAC. Applications submitted under the old SAC must be resubmitted for verification within 3 years of the effective date to retain their certified standing. The only exception is the file naming standards which will take effect immediately.

[illegible]
Robert F. Dolan

Attachments: RPMS Software Handbook

cc: Area ISCs
IHS Developers
DSM

DEPARTMENT OF HEALTH & HUMAN SERVICES Public Health Service

Indian Health Service
Albuquerque NM 87110

TO: All Developers, All ISCs

FROM: Software Review & Certification Branch

SUBJECT: 1996 RPMS Software Handbook - Addendum 1 Procedure for Adding Entries to New Person File (#200)

Issue

The attached addendum to the June 27, 1996 RPMS Software Handbook and RPMS Programming Standards & Conventions is approved for distribution.

Background

The RPMS Software Handbook was released for national use on June 27, 1996. The Handbook is in a format that allows updates or changes. The attached is the first update to this manual. This procedure will define the appropriate process for development personnel to add entries to the all important NEW PERSON file, File 200.

Please add the attached pages, 210 and 211 to the end of your handbook following the section titled Procedures for Submitting Patches to Certified RPMS Software. Also, add page xxi to the Table of Contents at the beginning of the Handbook.

Approval

The attached August 7, 1996 addendum to the RPMS Software Handbook is approved for national distribution.

[illegible]
Robert F. Dolan

Attachment: RPMS Software Handbook

cc: Director, Office of Information Resource Management

Indian Health Service
Office of Information Resource Management

RPMS Software Handbook

June 27, 1996

Table of Contents

Indian Health Service
Office of Information Resource Management

Purpose, Policy & Standards and Conventions Committee

RPMS ADP Systems Manual
Volume 2.3.2
June 27, 1996

Purpose, Policy, & Standards and Conventions Committee

1. Purpose

The purpose of this document is to provide a cornerstone for all national software developed and distributed throughout the Indian Health Service (IHS). Programming Standards and Conventions (SAC) mandate functional soundness and technical correctness of Resource & Patient Management System (RPMS) programs and all other programs which interface with RPMS. RPMS software includes programs written in M and other programming environments.

2. Policy

a. **Conformance** All RPMS software developed within IHS will conform to the Programming SAC. In areas where specific IHS standards have yet to be adopted, software development will conform to accepted industry standards.

b. **Quality Control** The IHS Programming SAC have been established as a key aspect of quality control under the Software Review & Certification Branch (SRCB), Division of Systems Manangement (DSM).

c. **Definition/Appeal** The IHS Standards and Conventions Committee (SACC) defines Programming SAC and serves as the source of appeal when issues of conformity with Programming SAC arise.

d. **Development** SACC members and other technical experts within IHS will participate in the development of Programming SAC in collaboration with public and private subject experts and with national and international standards organizations.

3. General Responsibilities

a. **Promulgation** The Director, DSM will promulgate all Programming SAC.

b. **Review/Recommendation** The Chief, SRCB is responsible for the review and recommendation of SACC proposals.

c. **Modifications** To assist in this responsibility, the SACC has been established with representation from various elements. This committee will meet as necessary (normally once a year) to formally review and modify RPMS programming standards and conventions and will utilize MailMan and conference calls on an interim basis to discuss any changes proposed or to further interpret standards as necessary. The

SACC is responsible to the Director, DSM, for defining RPMS application programming and design standards and conventions to be followed in developing RPMS application packages.

1. **Membership** Membership will include one representative from each of the following:

   . DSM Hardware Branch
   . DSM Software Branch
   . ANMC Development Center
   . DSD Albuquerque Development Center
   . DTM Albuquerque Telecommunications Center
   . Area ISC Representative

2. **Selection** Office of Information Resource Management (OIRM) members will be selected by the Directors of each of the above Divisions. The Area Information Systems Coordinators (ISC) will provide a member to the committee.

3. **Period** Membership will be for a period of 4 years, with half of the members rotating on the SACC each 2 years.

4. **Terms** Members cannot serve two consecutive terms; a one-year interval will be required off the SACC each 2 years.

5. **Chairperson** The SACC will elect a chairperson each year to serve a one-year term.

6. **Selection of New Members** The Director, Division of Systems Development (DSD), will coordinate with the Director, DSM, on the selection of the SACC members replacing those outgoing. The outgoing chairperson will turn over all relevant SACC records to the new Chairperson.

4. Procedures

a. **Membership** Membership rules for the SACC are outlined above.

b. **New Programming SAC**

1. The SACC will forward the proposed Programming SAC changes to the PROGRAMMER mail group on IHS Mail for comments at least 30 days prior to voting on the proposed changes.

2. After the SACC votes to accept the proposed Programming SAC, the SACC will forward it to the Chief, SRCB for recommendation, and then to the Director, DSM for approval.

3. The new SAC becomes effective on the date of approval by the Director, DSM. After approval, the new SAC will be announced and made available on IHS Mail.

c. Conformance to new Programming SAC

1. For a period of 6 months following the effective date of the new Programming SAC, newly released RPMS packages may adhere to either the new or old SAC.

d. Programming SAC Extensions

1. At least quarterly but no later than 30 days prior to voting on a proposed extension, the SACC will forward the proposed SAC extensions to the Programmer mail group on IHS MailMan for comments.

2. After the SACC votes to accept the proposed extensions, the SACC will forward the extensions to the Chief, SRCB for approval.

3. New extensions become official the day of approval by the Chief, SRCB.

e. Conformance to Programming SAC Extensions

1. Each SAC Extension will contain an explicit statement regarding the required date of compliance. Until the date of required compliance is reached, newly released packages are not required to adhere to the new SAC Extension.

f. Requests for Exemptions from Programming SAC

1. If a developer identifies the need for an exemption from the SAC for a given package, a request must be made to the SACC either via written notification using the form in Appendix B or electronic mail to the SACC mail group on IHS Mail. Exemption requests should be submitted as early as possible in the development cycle.

2. Exemption requests should include the following information:

   a. Package name and version number.
   b. Type (Permanent or temporary).

   c. Violated Standard.
   d. Justification and comments.

3. After review of the need and impact of the proposed exemption, the Chairperson of the SACC will call for a vote on the request. A two-thirds decision by the SACC shall be required to grant an exemption from the SAC.

4. Notification of the decision of the SACC will be made to the developer requesting the exemption, and will be entered on IHS Mail.

g. Publication of Current Programming SAC

1. In order to promote rapid dissemination, the current version of the IHS SAC will be announced on IHS Mail for all users to access. All developers should recognize the need for periodic review of the SAC on IHS Mail.

Indian Health Service
Office of Information Resource Management

Programming Standards and Conventions

RPMS ADP Systems Manual
Volume 2.3.2
June 27, 1996

RPMS Programming Standards and Conventions (SAC

This version of the Programming SAC was adopted by IHS on June 27, 1996.

1. General Programming Standards and Conventions

The definitions, standards and conventions within this section apply to all programming and user environments for use in the RPMS. This includes all applications that use commercial software products as an integral part of the RPMS (but not to the commercial application itself) and to all development environments including, but not limited to programming languages such as M or Pascal and graphical user interface (GUI) environments such as Visual Basic, Delphi, etc.

1.1 Definitions Applicable to this Document:

Conventions - Programming guidelines which are designed to promote consistency and safety across RPMS applications. Exemptions from conventions are not required, but developers are strongly encouraged to follow them.

DBA - Database Administrator

DBIC - Database Integration Committee

DHCP - Decentralized Hospital Computer Program. Veterans Administration (VA) software equivalent to the RPMS.

Entry Point - Entry point within a routine that is referenced by a "DO" or "GOTO" command from a routine internal to a package.

Exemption - Authority granted by the SACC to a specific RPMS/DHCP application which allows that application to not comply with a particular section of the SAC for a specified timeframe.

Extensions - An addition, deletion, or modification to the current SAC Conventions document. Each extension will contain an appropriate effective date, and may optionally contain an expiration date or event. (The SACC will update the SAC quarterly via extensions, which have the full weight of the original SAC. Every effort will be made to keep all RPMS/DHCP development SAC in one location.)

IA - Integration Agreement. IA’s can be accessed via the DBA menu on IHS MailMan system.

Kernel - The Kernel is a set of VA software utilities that form the foundation of DHCP/RPMS and include elements that start with the namespaces XG*, XLF*, XPD*, XQ*, XT*, XU*, XV*, ZI*, ZO*, ZT*, ZU*.

MailMan - MailMan is a set of VA software utilities that form the foundation of DHCP/RPMS electronic mail and communications and include elements that start with the namespace XM*.

Namespace - A unique set of between 2 and 4 alpha characters assigned by the DBA.

New - A way to create a new version of a variable either by explicit declaration or implicitly through parameter passing. Some protected variables may not be modified through the use of either the implicit or explicit NEW command.

Package - A set of routines, files, options, templates, security keys, screens, bulletins, functions, help frames, forms, blocks, objects, protocols, dialogues, list templates, windows, etc., namespaced according to DBA requirements that function as a unit.

Programmer Mail Group - The Programmer mailgroup (G. Programmer) is established on the IHS MailMan system for discussion of technical issues. This mail group is used to disseminate information to the IHS RPMS development community.

Published Entry Point - Entry point within a routine that is referenced by a"DO" or "GOTO" command from a routine external to the package.

Standard - A rule which all RPMS/DHCP software must follow.

Supported Reference - Routine labels, extrinsic functions, files or global nodes that are accepted and documented by the DBIC and listed on the DBA menu on IHS Mail.

User - A person interacting with computer applications.

Utility - A callable routine line tag or function. A universal routine like XB*, ZIB*, AUPN*, usable by anyone.

Up-hat - "^" which is denoted on the keyboard by pressing Shift+6, a circumflex, also known as "hat" or "caret". It is used as a piece delimiter in a global.

VA FileMan - The Database Management System for RPMS/DHCP, with namespaces DD*, DI*, and DM*.

1.2 Files

1.2.1 Naming Requirements for Files Used by RPMS/DHCP Packages

1.2.1.1 VA FileMan All VA FileMan files in the M Language environment must be numberspaced or namespaced in the numberspace or namespace assigned to the package by the DBA. See Appendix A.

1.2.1.2 DOS, VMS, UNIX or Other Host Files All DOS, VMS, UNIX or other host files created or exported as part of a RPMS/DHCP application must be namespaced in the namespace assigned by the DBA. See Appendix C.

1.2.2 Exporting Script Files Packages exporting script files should provide script files for a variety of the terminal emulation packages commonly in use in the VA/IHS.

1.2.3 Exporting Spreadsheets Packages exporting spreadsheet templates should apply protection to embedded formulas to prevent accidental deletion by a user. Spreadsheet templates should contain documentation describing the purpose of the template, complex functions, and user help.

2. M Language Programming Standards and Conventions

All M-based RPMS/DHCP software will meet the following standards, and comply with the spirit of the conventions.

2.1 ANSI Standards The 1995 ANSI/MDC X11.1 Sections 1 and 2 will be adhered to unless explicitly modified by this document.

2.1.1 Standard Dictionary Releases All development within an Area or at Development Centers will be produced using the most recent standard dictionary releases as distributed by the OIRM.

2.1.2 Fields Within A String Fields within a string will be separated by the "up-hat" symbol except when the data within the string must contain the "up-hat" itself.

2.1.3 Reindexing of File Manager MUMPS Cross-References Reindexing of individual File Manager MUMPS cross-references must not result in an error. This affects both "set" and "kill" logic.

e.g., Use I $D(XYZ),XYZ'="" S ^GBL("AC",XYZ)=""

2.1.4 FileMan Entry Points Only standard documented FileMan entry points will be called by IHS routines.

2.1.5 Files/Data Restrictions When Creating a Package For Distribution All packages delivered to the SRCB for IHS distribution will contain only those files which belong to the package and shall not contain global data in the package inits. If data is required it will be delivered in a separate file. Files which are normally part of one package will not be distributed with another package.

2.1.6 File Manager File Access Code Security All packages delivered to the SRCB for verification and distribution will come with complete File Manager file access code security in the data dictionary. All files will have programmer only data dictionary access. This assures that security will be present initially if dictionaries are deleted prior to installation. The only exception to having file access security codes is in the READ field in a DD in which case it can be without access or blank. If developers need file access codes for their packages, they will coordinate this with the DBA.

2.1.7 Version Information Complete version information will be set into the package file (DIC(9.4)) and all files for a package will have the version number in DD(file#,"VR").

2.1.8 LAYGO Restriction Application programs will not allow LAYGO entry to the New Person file. Packages which need to be able to add to this file will have separate locked options which come with the package, but which are not assigned to any of the package menus as distributed. Application programs will not allow "LAYGO" to standard tables.

2.1.9 Entry in File 9000010 Packages external to PCC that are passing data to the Visit and V-files must use the APCDALV routine to select or create a visit entry in file 9000010. The APCDALVR will be used to append a V-file entry to a visit file entry.

2.2 Routines (Routine Structure and Format)

2.2.1 First Line The first line of a routine will have the following:

(routine name) ; (Agency)/(site)/(programmer)-(brief description) ; (date of edit)

e.g., AGPAT ;IHS/OKC/LD - Patient registration driver; JUL 20, 1986

2.2.1.1 First Line The first line of a routine cannot contain the formal list for parameter passing.

2.2.1.2 Generated Routines Routines generated by VA FileMan or Kernel (e.g., INITs, ONITs, NTEG, and compiled routines) and other compiled routines used in exporting a package, need not comply with this standard.

2.2.1.3 Agency/Site/Programmer This shall identify the developing Agency and site, and the programmer who is currently responsible for maintenance and development of the package.

2.2.1.4 Date of Edit The date of edit is optional. When it is included, the use of time is optional.

2.2.2 Second Line The second line of a routine must be in the following format:

LABEL-optional<ls>;;version number;package name;**pm,...pn**;version date

Where:

2.2.2.1 Version Number The version number must be the same on all of the package-namespaced routines including the Inits, Onits, etc.

2.2.2.2 Patch Numbers "pm,...pn" are the applied patch numbers separated by commas, this ";" piece is null if there are no patches.

2.2.2.3 Version Date The version date must be the same on all of the package namespaced routines including the Inits, Onits, etc.

2.2.2.4 **Compiled routines** Routines compiled from templates, cross-referenced, etc., by VA FileMan during or after package installation are exempt from the second line requirement.

2.2.3 **Local Modifications** If local modifications to a routine are restricted or prohibited by policy or directive, the third line should contain an appropriate notice. (e.g., "Per VHA Directive 10-92-142, this routine should not be modified")

2.2.4 **Labels** Labels are limited to eight (8) characters and may not contain lower case characters.

2.2.4.1 **Label+Offset** Label+offset references will not be used except for $TEXT references.

2.2.4.2 **Lines Referenced by $TEXT** Lines referenced by $TEXT for use other than to check for the existence of the routine must be in the following format: (LABEL-optional)<ls>;;text or M code.

2.2.5 **Linebody** The linebody must contain at least 1 character, must not exceed 245 characters in length, and must contain only the ASCII characters values 32-126. Commands, functions, local and global variable names, system variables, SSVNs, etc., must be uppercase.

2.2.6 **Routine Names** Package routine names of the following forms will not be used:

2.2.6.1 **NAMESPACE_I*** (with the exceptions of Kernel, VA FileMan, and routines created to support the INIT process).

2.2.6.2 **NAMESPACE_NTE*** (with the exception of the package integrity routines).

2.2.7 **Routine Size** The maximum routine size, as determined by executing ^%ZOSF("SIZE"), is 10,000 characters. The combination of routine and symbol table must run in the partition size specified in the appropriate VA/RPMS operating system manual/cookbook.

2.2.8 **Vendor Specific Subroutines** Vendor specific subroutines may not be called directly except by Kernel, MailMan and VA FileMan.

2.2.9 Taskman Globals All applications will use documented TaskMan utilities to interface with TaskMan globals.

2.2.10 Naked References Naked references must either be appropriately preceded by the full reference defining it or be documented.

2.2.10.1 Full Reference An appropriate preceding full reference is one that is on the same physical routine line as the naked reference and has no code between it and the naked reference that branches in any manner to other lines of code or executables.

2.2.10.2 Documenting Those naked references requiring documentation must be documented within the routine in the immediate vicinity of the naked reference. Those naked references that are preceded by a full reference which is outside of the routine where the naked reference is used must have documentation in both the routine containing the full reference and the routine containing the naked reference. This documentation must be in the immediate vicinity of the appropriate reference.

2.2.10.3 In Called Utilities Uses of naked references in called utilities are exempt, e.g., S DIC=200,DIC(0)="AEQ",DIC("S")="I $L($P($G(^(1)),"^",9))" D ^DIC is a legitimate use of the naked reference.

2.2.11 % Routines

2.2.11.1 %Routines No application will distribute % routines. (Exemptions: Kernel and VA FileMan)

2.2.11.2 %Routines No % routines shall execute variables which could be set by a programmer prior to executing the code.

2.2.11.3 View Commands No routine which will be resident in the Library (MGR) account will use VIEW commands using variables as arguments which could be set by a programmer prior to executing the code.

2.2.12 Z Routines

2.2.12.1 **Z Routines** No application will export routines whose names start with the letter "Z". (Exemption: Kernel)

2.2.12.2 **Creation of Local Routines** When creating local routines to be added to an existing national package, the routine name will begin with the namespace followed by the letter "Z".

2.2.13 **Extended Reference Syntax** Routines may not be invoked using the extended reference syntax, i.e., D ^|"VAH"|TAG^ROUTINE is illegal.

2.2.14 **Entry Points** Lines that are entry points (referenced by a DO or GOTO in other routines) will consist only of a label and a semi-colon followed by "entry point" or the abbreviation of "EP". An optional comment which describes the entry point may follow the entry point comment.

```
e.g.,    CLEAR ; EP - CLEARS SCREEN
e.g.,    CLEAR ; ENTRY POINT - CLEARS SCREEN
```

2.2.15 **Changed Lines** Lines changed from the standard release will be marked with a semicolon and comment. This comment will contain the Agency and site identification, the programmer's initials, the date of the change, and reason for the change. Duplicate and comment out the original line of code. Additional comment lines may be used.

```
e.g.,    CHGLINE;
 ;S ZXXX=3
S AXXX=3 ; IHS/ABQD/FBD 04/01/96
;This was changed to conform to IHS SAC.
```

2.2.16 **IHS Changes to VA Packages** IHS changes to VA packages will be incorporated into separate routines or templates using a proper IHS namespace, rather than embedding the changes in the middle of VA programs. (A "DO" statement will be used to exit to the IHS routines, if needed). When it is not possible to use an external IHS routine when modifying VA packages, add changes in the format outlined in Standard 2.2.15.

2.2.17 **Comments** Comments will be provided at the start of each independently callable routine specifying the items listed in paragraphs 2.2.17.1 through 2.2.17.3.

2.2.17.1 **Purpose or Function** Overall functions or purpose of routine.

2.2.17.2 **Input Variables** Input variables (variables set up by other routines which are used by this routine).

2.2.17.3 **Output Variables** Output variables (variables set by this routine).

2.2.18 **XB/ZIB Prefixed Routines** AU/AZ prefixed utility routines have been re-namespaced XB/ZIB respectively. All XB prefixed utility routines will refer to utility routines which are totally universal from machine to machine regardless of operating system or hardware. All ZIB prefixed utility routines will refer to utility programs which are operating system or hardware dependent. All program calls to previous AU/AZ prefixed utility routines must be changed to make the program calls to the XB/ZIB equivalent utility routines.

2.2.19 **Facility or Area Specific Information** Facility or Area specific information needed by a routine will be obtained from tables or input parameters, rather than being hard-coded in the routine.

2.2.20 **Approved Exemptions** Approved exemptions to standards must be documented in the routine with a separate comment line adjacent to the line containing the exemption in the form:

;IHS exemption approved on DATE

2.2.21 **"Namespace" Var Routine** Packages that need to set local variables prior to entering a package will use a routine which is named in the form 'namespace' VAR.

2.2.22 **Data Conversion Processes** All data conversion processes should be restartable at the point of interruption in the execution.

2.2.23 **Syntactically Correct Lines** All lines must be syntactically correct. Blanks will not appear at the end of lines.

2.3 Variables

2.3.1 Local Variables

2.3.1.1 **Local Variables** Lowercase characters in local variable names are prohibited.

2.3.1.2 Length of Local Variables

The full evaluated length of a local variable name including subscripts must not exceed 200 characters. The evaluated length is calculated as follows

Example subscripted variable:
NAME(sub1,sub2,...,subn)

(1.) +$L(NAME)+3
(2.) +$L(sub1) + $L(sub2) + ... +$L(subn)
(3.) + 2 * number of subscripts n
(4.) +15

VAR("XXX",123,1,2,0) would evaluate to a string length of 42 (6+11+10+15)=42.

2.3.1.3 System Wide Variables

2.3.1.3.1 Variables

The following are system wide variables. Any application setting system wide variables must conform to the following definitions.

AGE - Patient age in years from date of birth to DT expressed as an integer, or, if deceased, the date of death.

DFN - Internal number of an entry in the Patient File (#2).

DOB - Patient date of birth expressed in internal VA FileMan format.

SEX - Patient sex; either "F" or "M".

SSN - Social security number with 9 contiguous digits, or 9 digits and a "P".

VA("BID") - Brief patient identifier; up to 7 characters.

VA("PID") - Patient identifier; up to 15 characters.

2.3.1.3.2 Assumed Variables

The following variables, referenced elsewhere in this document, are set by Kernel during sign-on, or by VA FileMan, and can be assumed to exist by all DHCP/ RPMS applications.

DT - Current date, without time, in internal VA FileMan format.

DTIME - Time-out parameter for a read command in seconds.

DUZ array - Contains user specific information.
- DUZ(0) User's FileMan Access Code.
- DUZ(2) Division (pointer to the Institution File).
- DUZ("AG") Agency Code (from the Kernel Site Parameter File).

U - Circumflex (i.e., "^").

IO - The hardware name of the last selected input/output device.

IO(0) - The assigned principal device (primary device).

ION - The logical name of the IO device.

IOST - The last selected input/output device's subtype from the Terminal Type file.

IOST(0) - The internal entry number in the Terminal Type file of the last selected IO device's terminal type.

IOM - The width of the IO device.

IOSL - The length of the IO device.

IOF - The code to start output at the top of a page (e.g., W @IOF).

IOBS - The backspace of the IO device.

2.3.1.4 **DUZ or DUZ-Array** RPMS/DHCP packages are not allowed to KILL, NEW, SET, MERGE, READ (into) or otherwise modify the variable DUZ or any DUZ-array element with the exception of DUZ(2) (Exemptions: Kernel and VA FileMan). In order to allow programs to run stand alone or for transport to non-Kernel environments, DUZ and its descendants can be set after confirming that they do not already exist.

2.3.1.4.1 **DUZ(2)** The VA local variable DUZ(2) will be set for all users upon logon by Kernel, using the FACILITY field in the New Person. A user will be able to log on to only those sites listed under the FACILITY field in his or her New Person file entry. Any application allowing a user to switch facilities in a package will use this multiple field in the New Person file when selecting a valid site for the user to switch to. If a package modifies DUZ(2) after the original set, it will be reset to its original value before exiting the package. A value of 0 in DUZ(2) or the existence of the local variable AUPNLK("ALL") will allow complete lookup ability for all facilities in the Patient File.

2.3.1.5 **Special Variables** The variables DT, DTIME, and U have no array elements and shall be initially defined by Kernel or VA FileMan.

2.3.1.5.1 **Variable U** The variable U will not be KILLed or NEWed or changed from the value defined by Kernel or VA FileMan. (It is legal to SET U="^".)

2.3.1.5.2 **Variable DT** The variable DT will not be KILLed or NEWed. If changed it must be set using the supported reference S DT=$$DT^XLFDT.

2.3.1.5.3 **Variable DTIME** The variable DTIME may be changed, but must be restored to its original value before exiting the option.

2.3.1.5.3.1 **Documentation** All locations where DTIME is changed but not restored, must be documented in the Technical Manual.

2.3.1.5.3.2 **Resetting DTIME** The Kernel-supported reference $$DTIME^XUP will reset DTIME to its original value, e.g.,
S DTIME=$$DTIME^XUP(DUZ).

2.3.1.6 **IO Variables** RPMS/DHCP packages are not allowed to KILL, NEW, SET, MERGE, READ (into) or otherwise modify IO namespaced variables and any of their array elements except those documented as modifiable in the Kernel System Manual. (Exemption: Kernel, MailMan, and VA FileMan) The routine XBKVAR can be invoked to set these variables.

2.3.1.7 **% Variables** RPMS/DHCP packages are not allowed to KILL, NEW, SET, MERGE, READ (into) or otherwise modify % variables. Exceptions to this are the single character variable "%" and the variables set for and/or returned by Kernel and VA FileMan supported references. (Exemption: Kernel, VA FileMan and MailMan)

2.3.1.8 **Scope of Namespaced Variables** A RPMS/DHCP package may declare namespaced, local variables as package-wide. The variables and all of their array elements must be described in the package Technical Manual. A RPMS/DHCP package may not kill or change another RPMS/DHCP package's package-wide variables.

2.3.1.8.1 **Documentation** Documentation on how to create and kill package-wide variables created by an option that is removed from its exported menu path must be included in the Technical Manual.

2.3.1.9 **Lock Tables/Local Symbol Tables** All supported references must leave the lock tables and local symbol tables unchanged upon exit with the exception of the following:

- Documented input and output variables (including globals).
- Supported reference namespaced variables may be changed or killed (for example, the VA FileMan ^DIC call kills the variable DIE, which may exist in the symbol table prior to the call).
- Documented side effects, such as lock table changes, and changes to files.
- Variables composed of a single alpha character followed optionally by one numeric.
- The variable %.

These supported references must be documented in the package Technical Manual and on IHS MailMan with a descriptive list of ALL input and resulting output variables.

2.3.1.10 Variables Passed Between Packages Naming requirements for variables passed between packages.

2.3.1.10.1 Actual List Input variables in an Actual List passed by reference between packages must be package namespaced.

```
Legal:
 D BLD^DIALOG(3500010,"",.IBDATA,"IBX")
Illegal:  D BLD^DIALOG(3500010, "",.Y,"IBX")
```

2.3.1.10.2 Input Variables Input variables that represent local variables into which data will be exchanged must represent a data location that is package namespaced.

```
Legal  :  S DA=10,DR=".01;.104",
          DIC="^DPT(",DIQ="IBX" D EN^DIQ1
Illegal:  S DA=10,DR=".01;.104",
          DIC="^DPT(",  DIQ="Y"  D EN^DIQ1
```

2.3.2 Global Variables

2.3.2.1 Global Name Lowercase characters in global names and global subscripts are prohibited. (Exemption: Cross-references created using field values containing lowercase characters and subscripts used in the ^TMP and ^XTMP globals.)

2.3.2.2 Length of Global Reference The full evaluated length of a global reference must not exceed 200 characters. The evaluated length is calculated as follows.

Example subscripted variable:
^NAME(sub1,sub2,...,subn)

(1.) +$L(NAME)+3
(2.) +$L(sub1) + $L(sub2) + ... +$L(subn)
(3.) + 2 * number of subscripts n
(4.) +15

^TMP("XXX",123,1,2,0) would evaluate to a string length of 42
(6+11+10+15)=42.

2.3.2.3 **Unsubscripted Globals** The KILLing of unsubscripted globals is prohibited. (VA FileMan's EN^DIU2 utility allows the deletion of files stored in unsubscripted globals, and therefore, allows the killing of unsubscripted globals. Application developers must document when calls to EN^DIU2 are made to delete files stored in unsubscripted globals.)

2.3.2.4 **%Globals** READing, KILLing, SETting or MERGing ^% globals is prohibited. (Exemption: Kernel) %Globals will not be created without approval from DSM.

2.3.2.5 **Globals** All globals must be VA FileMan compatible. ^TMP, ^XTMP and ^UTILITY have a standing exemption from this requirement.

2.3.2.5.1 **^TMP Global** The global ^TMP will be used as a scratch global within a session. The first subscript shall be $J, or the first two subscripts shall be a package namespaced subscript followed by $J. The ^UTILITY global will not be used unless necessary to retrieve output from a Kernel or FileMan Utility.

2.3.2.5.2 **^XTMP Global** The global ^XTMP will be translated, with one copy for the entire RPMS production system at each site. The structure of each top node shall follow the format ^XTMP(namespaced-subscript,0)=purge date^create date^optional descriptive information, and both dates will be in VA FileMan internal date format.

2.3.2.6 **Executable Code** Fields in VA FileMan files which contain executable code must be write protected in the DD with "@" (e.g., ^DD(file,field,9)="@"), or be defined as VA FileMan data type of "MUMPS".

2.3.2.7 **DD Global** References to the DD Global requires a formal

Data Base Integration Agreement (DBIA) with the VA FileMan Development team and must be registered with the DBA.

2.3.2.8 **Global Variables** All global variables executed by % routines must be in write protected globals.

2.3.2.9 **Global Names** Extended reference syntax may not be used to reference global variables, i.e.,
S X=^|"VAH"|GLOBAL(1,1) is illegal.

2.3.3 Intrinsic (System) Variables

2.3.3.1 **Intrinsic Variables** Lowercase intrinsic variables are prohibited.

2.3.3.2 **Intrinsic Variables** No DHCP/RPMS package may use the following intrinsic (system) variables unless they are accessed using Kernel or VA FileMan supported references: $D[EVICE], $EC[ODE], $ES[TACK], $ET[RAP], $I[O], $K[EY], $P[RINCIPAL], $Q[UIT], $ST[ACK], $SY[STEM], $Z*. (Exemption: Kernel and VA FileMan)

2.3.4 Structured System Variables (SSVNS)

2.3.4.1 **SSVNS** Lowercase SSVNs are prohibited.

2.3.4.2 **Restricted SSVNS** The following structured system variables may be used only by Kernel or VA FileMan or through their supported references: ^$CHARACTER, ^$DEVICE, ^$DISPLAY, ^$EVENT, ^$GLOBAL, ^$JOB, ^$LOCK, ^$ROUTINE, ^$SYSTEM, ^$Z*, and ^$WINDOW.

2.4 Commands

2.4.1 Commands Lowercase commands are prohibited.

2.4.2 BREAK Command

2.4.2.1 **BREAK** Direct use of the BREAK command is prohibited.

Use ^%ZOSF("BRK") and ^%ZOSF("NBRK"). (Exemptions: Kernel and VA FileMan)

2.4.3 CLOSE Command

2.4.3.1 **CLOSE** Direct use of the CLOSE command is prohibited. Use the routine ^%ZISC. (Exemptions: Kernel, MailMan and VA FileMan)

2.4.4 HALT Command

2.4.4.1 **HALT** Direct use of the HALT command is prohibited. Use the supported reference H^XUS. (Exemption: Kernel and VA FileMan)

2.4.5 JOB Command

2.4.5.1 **JOB** Direct use of the JOB command is prohibited. Use the Kernel Task Manager's supported calls to create jobs. (Exemption: Kernel and MailMan)

2.4.6 KILL Command

2.4.6.1 **Argumentless** The argumentless form of the KILL command is prohibited. (Exemption: Kernel)

2.4.6.2 **Exclusive** The exclusive form of the KILL command is prohibited. (Exemptions: Kernel and VA FileMan)

2.4.7 LOCK Command

2.4.7.1 **LOCK** All LOCKs shall be of the incremental or decremental form. All routines will lock nodes to be created or updated. Appropriate error recovery action will be taken if the lock is not obtained. (Exemption: Kernel)

2.4.7.2 **Incremental LOCK** All incremental LOCKS must have a timeout.

2.4.8 NEW Command

2.4.8.1 **Argumentless NEW** The argumentless form of the NEW command is prohibited.

2.4.8.2 **Exclusive NEW** The exclusive form of the NEW command is prohibited. (VA Only)

2.4.9 OPEN Command

2.4.9.1 **OPEN** The use of the OPEN command is prohibited. (Exemptions: Kernel, MailMan and VA FileMan)

2.4.10 READ Command

2.4.10.1 **READ** All READ commands shall read into local variables, or one of the non-VA FileMan compatible globals, ^TMP and ^XTMP.

2.4.10.2 **READ** All user input READs must have a timeout. If the duration of the timeout is not specified by the variable DTIME and the duration exceeds 300 seconds, documentation in the package Technical Manual is required.

2.4.10.3 **READ** All user input READ commands shall be terminated by a carriage return. (Exemption: Kernel and VA FileMan) (Developers desiring to implement escape processing [function keys, arrow keys, etc.] must use Kernel supplied supported references [XGF]).

2.4.11 USE Command

2.4.11.1 **USE** The use of the USE command with parameters is prohibited. (Exemptions: Kernel and VA FileMan)

2.4.12 VIEW Command

2.4.12.1 **VIEW** The use of the VIEW command is prohibited. (Exemptions: Kernel and VA FileMan)

2.4.13 MWAPI Commands

2.4.13.1 **MWAPI** No RPMS/DHCP package may use the MWAPI Commands, ESTART, ESTOP, and ETRIGGER. (Exemption: Kernel)

2.4.14 Z* Commands

2.4.14.1 **Z* Commands** The use of Z* commands is prohibited. (Exemptions: Kernel and VA FileMan)

2.5 Functions

2.5.1 **Functions** Lowercase functions are prohibited

2.5.2 Intrinsic Functions

2.5.2.1 **$NEXT** Use of the $NEXT function is prohibited. All RPMS packages should remove all occurrences of $NEXT. This includes occurrences generated by VA FileMan.

2.5.2.2 **$VIEW** The use of the $VIEW function is prohibited. (Exemptions: Kernel and VA FileMan)

2.5.2.3 **$Z*** The use of $Z* functions are prohibited. (Exemptions: Kernel and VA FileMan)

2.5.3 Extrinsic Functions

2.5.3.1 **Parameters** Supported References that use parameters will document the elements of the formal list internally within the routine and in the package Technical Manual. Documentation will specify which elements of the formal list are required and which are optional, if any, and those elements which must be passed by reference.

2.5.3.2 **Supported Extrinsic Special Variables** Extrinsic functions with an empty formal list will be documented within the routine and in the Technical Manual.

2.6. Name Requirements

2.6.1 **Package Namespace** Unless approved by the DBA, routine, global, security key, option, template, bulletin, function, screen, help frame, protocol, form, block, list templates, objects, dialogues, remote procedures, etc., names must be consistent with the assigned DBA namespace for the package. The namespace of all IHS-developed packages assigned after January 1, 1994 must begin with *B*.

2.6.2 "NAMESPACE"MENU All IHS packages will have a top menu option in the form of "namespace"MENU with a lock on that menu option which is of the form 'namespace'ZMENU. When entering this menu the current version of the package will be displayed above the options along with the location determined by the current DUZ(2) setting.

2.7 Options (Option file entries)

2.7.1 Options Option selection must be made through Menu Manager or using VA FileMan's DIR utility. DIR utility may only be used at the lowest menu tree level. Hardcoded menu management systems are not allowed.

2.7.2 Options All options in a package must be path independent once the steps described in the Technical Manual for creating and killing package wide variables have been taken.

2.7.3 Options The following must not exist after exiting an option:

2.7.3.1 Output Variables All documented output variables created by a called supported reference.

2.7.3.2 Locks All documented locks created by a called supported reference.

2.7.3.3 Global Nodes All documented temporary scratch global nodes (e.g., ^TMP and ^UTILITY) created by a called supported reference, with the exception of ^XTMP global data.

2.7.3.4 Variables/Locks/Globals All local variables, locks, and scratch global nodes (except ^XTMP, or other scratch globals designed to be passed between parts of a package) created by the application.

2.7.4 Menu Options All menu options in a package will utilize alpha synonyms, not numeric. This is to allow use of the Kernel menu jumping capability.

2.8 Device Handling

2.8.1 Device Handling All device selection and closing will be made through the use of the Kernel supported references. See Sections 6.3 and 6.9 for specific information about the OPEN and CLOSE Commands

2.8.2 Output Queuing All output to a hard copy device (e.g., printer) must allow for queuing or use of an auxport printer. Internal device selection must be by device name rather than by $I. Developers should bear in mind that either IO or $I variables may be non-numeric. Forced queuing, which is sometimes desirable, will be local site parameter driven.

2.8.3 Outputs Any output directed to a hard copy device (e.g., printer) will not start with a form feed or line feeds with the purpose of creating a form feed, and will leave the device at top-of-form when the output is finished.

2.9 Miscellaneous

2.9.1 Implementation Specific Functions Application software must use documented Kernel-supported references to perform all M operating system specific functions. (Exemptions: Kernel and VA FileMan) In addition, %ZISH is to be used for Host File functions.

2.9.1.1 VA FileMan Application software must use documented VA FileMan standards such as defaults, editing text, date/time format, spacebar recall, help prompts, deleting stored values, control of logic flow, escapes, etc. (See VA FileMan Users Guide) Developers are encouraged to use documented FileMan calls to the fullest extent possible. (See VA FileMan Programmers Guide)

2.9.2 Data Element Interpreted as Number Any data element which may be interpreted as a number must contain no more than 15 significant digits.

2.9.3 Supported References Packages may phase out supported references (as callable from outside the application and documented by DBA) by providing a minimum 18 month notice to the PROGRAMMER, and ISC mail groups on IHS Mail.

2.9.4 Character Set All globals and routines will use only the M character set profile.

2.10 Supported Type A Extensions

2.10.1 Type A Extensions No Type A extensions to the M Language standard are currently approved for use. Type A extensions may be made available through a SAC extension, after all operating systems in wide use in the VA/IHS have implemented the Type A extension.

2.11 Conventions

2.11.1 ^UTILITY Only Kernel and VA FileMan and existing Supported References should use ^UTILITY.

2.11.2 Deleting Tasks Tasks should be deleted from Task Manager's list by setting the variable ZTREQ equal to "@" just prior to the application QUITing.

2.11.3 Routine Documentation

2.11.3.1 Entry Points Routine line tags referenced from outside the routine should be documented before, on or after the line tag. Documentation should include a description of function.

2.11.3.2 Supported References/Routines All supported references or routines invoked initially from an option or protocol should contain documentation explaining the functionality and any required, passed input and output variables.

2.11.4 READ READ commands should not be used in the Data Dictionary.

2.11.5 WRITE Write commands should not be used in data dictionaries (except for VA FileMan generated ID nodes). The call to EN^DDIOL should be used.

2.11.6 Device a CRT The proper method of determining if a device is a CRT is to check that the variable IOST starts with the string "C-". (e.g., I $E(IOST,1,2)="C-")

2.11.7 ^XTMP Developers are encouraged to include other descriptive information on the third piece of the 0 node of the XTMP global, such as task description and creator DUZ.

2.11.8 Linebody The linebody of a routine must contain at least 1 character. Generally a single semicolon is used to demark a blank line.

2.11.9 Location Name All packages will display the location name determined by the current DUZ(2) setting above all menus of the package.

3. Interface Programming Standards and Conventions

It is the intention of this section of the SAC to provide a consistent path for users as applications migrate from scrolling mode to a screen mode (either ScreenMan, List Manager, or screen-oriented editors) to a GUI environment.

3.1 User Interface Standards for Scroll Mode and Screen Mode

3.1.1 Deletion Deletion of a data value, if permissible, must be initiated by the user entering the at-sign "@".

3.1.2 READ All user-input READs which are in any way evaluated by the application must be escapable by entering a circumflex "^", which takes an action other than a reread.

3.1.3 Help All prompts requesting user input must provide additional help when the user enters a question mark ("?"). Any unrecognized or inappropriate response must be handled properly; i.e., at a minimum in a manner similar to the way VA FileMan handles responses (see VA FileMan User's Manual). Responses to READs that are in no way evaluated by the application are excluded from this requirement.

3.1.4 Defaults In scrolling mode, defaults must be so indicated with a double slash ("//") or "replace" indicating that "replace/with" editing is allowed. The null response (i.e., typing only the RETURN key) shall select the default.

3.1.5 READ Timeouts When a user input READ command times out, if the argument of the read is in any way evaluated by the application, the program must return to the Menu Manager with no more than one intervening read. A timeout at the menu level must halt through H^XUS

3.2 User Interface Conventions For Scroll Mode And Screen Mode

3.2.1 Terminology Developers are encouraged to use the following terminology.

3.2.1.1 **EXIT** Exit ends a function or application and removes from the screen all windows and pop-ups associated with that function or application. If information has been changed, the application may automatically save the information, or prompt the user to save or discard the information

3.2.1.2 **QUIT** Like Exit, Quit ends a function or application and removes from the screen all windows and pop-ups associated with that function or application. If information has been changed, the application may automatically discard the information, or prompt the user to save or discard the information.

3.2.1.3 **CANCEL** Cancel allows users to back out of a function or application, one pop-up at a time, until they reach the highest level window. At that point, another Cancel request has the same effect as an Exit action.

When users Cancel a pop-up, the application can decide whether to discard or retain the information in that pop-up, depending on how the application wants to establish the default values the next time the pop-up is displayed. If the information is discarded and the pop-up is later re-displayed, the pop-up contains the default values set by the application. If the information is retained and the pop-up is later re-displayed, the pop-up contains the same values as it did when the user canceled the pop-up.

3.2.1.4 **CLOSE** Synonymous with Cancel.

3.2.2 **Key Assignments** Developers are encouraged to use the following key assignments:

3.2.2.1 **PF1 Key Gold** May result in different actions based on the next key selected.

3.2.2.2 **PF2 Key Context-sensitive Help** Provides context-sensitive help about a specific item, field, or window.

3.2.2.3 **PF3 Key Exit** Exit is defined in 3.2.1.1 above.

3.2.2.4 **PF4 Key Backtab** Moves the cursor to the previous entry field. The cursor moves from right to left, bottom to top.

3.2.2.5 **F10 Key Menu Bar** Moves the cursor to the menu bar, if one is available, at the top of the window or pop-up currently in focus.

3.2.2.6 **F12 Key Cancel** Cancel is defined in 3.2.1.3 above.

3.2.2.7 **TAB Key Tab** Moves the cursor to the next entry field. The cursor moves from left-to-right, top-to bottom.

3.2.2.8 **PF1,H Key Sequence** Application Help. Provides information about the particular segment of the application being used.

3.2.3 Lock If a user is waiting for a lock which times out, then appropriate notification should be given to the user.

3.3 User Interface Conventions for GUI Mode

3.3.1 GUI Standards Document DHCP/RPMS packages should follow the guidelines for GUI applications set forward in the DHCP/RPMS Standards document. See Appendix G.

4. Unix Shell Programs and Files Standards

4.1 Purpose To provide guidance in certifying and distribution UNIX Shell Programs and Files that support the RPMS MUMPS applications.

4.2 Standards

4.2.1 Namespacing All UNIX Shell Programs and Files shall be namespaced with the assigned namespace (i.e., aibxxxx).

4.2.2 Directory All applications-related Shell Programs and Files shall reside in a unique subdirectory under the /usr directory. The subdirectory will be named using the assigned namespace (e.g., /usr/aib).

4.2.3 Version The first line of all UNIX Shell Programs and Files shall contain the name of the Shell Program or File and the version number of the application.

The version number shall be the same as the associated MUMPS application version number.

4.2.4 Install Shell Program An installation UNIX Shell Program shall be included in all distributions. This Shell Program shall perform the following:

4.2.4.1 Ownership Ensure that the ownership of the UNIX Shell Programs and Files are properly set.

4.2.4.2 Group Membership Ensure that the group membership of the UNIX Shell Programs and Files are properly set.

4.2.4.3 Modes Ensure that the modes of the UNIX Shell Programs and Files are properly set.

4.2.4.4 Subdirectories Ensure that all UNIX Shell Programs and Files reside in the proper subdirectories.

4.2.5 UNIX Commands/Utility Files The RPMS application packages submitted for certification should not include standard UNIX commands and utilities files. Utilities include the 3780 and uucp packages.

4.2.6 Patches All corrections of errors identified in the production release should be treated similar to corrections in the associated MUMPS application by using the IHS Patch System. Each patch should be documented in the UNIX Shell Program or File beginning with the second line.

4.2.7 Enhancements/Modifications All future enhancements and modifications of the application should be coordinated with the person who is responsible for developing or maintaining those applications affected.

Appendix A - RPMS Namespace Assignments

1. Purpose

Systems developed for the RPMS are identified by a set of program and file names, and are assigned a range of file numbers with the computer. Standards must be followed in assigning these names and numbers to ensure that they are unique and do not overlap between systems. The purpose of this documentation is to define these standards.

1.1 Responsibility

1.1.1 Program/File Name Assignments

The DBA will maintain a master list of all RPMS name assignments, and will assign and/or approve of new names for new systems.

1.1.2 File Numbers

Blocks of file numbers have been allocated to RPMS core systems, the Division of Data Processing Services (DDPS), and to each Area, as indicated in Figure 2.

The DBA will be responsible for maintaining and assigning file numbers for RPMS applications; the Area ISCs will maintain and assign file numbers for local systems.

2. General Standards

The general standards to be followed in assigning names are as follows:

2.1 Naming Assignments

2.1.1 Namespace Position 1 The namespace of all IHS-developed packages assigned after May 5, 1993 must begin with B.

2.1.2 Position 2-3 The next two to three characters will be the system prefix, and will be used to identify the application system itself (i.e., CHS for Contract Health Services, or CA for Cost Accounting).

2.1.3 Remaining Positions The remaining characters will be used to identify names used within the application.

2.2 Other Standards

2.2.1 All Names The use of "Z" as the fourth or fifth character, immediately after the system prefix, to designate locks, to distinguish locks from menus and options.

2.2.2 XB/ZIB - Utilities XB/ZIB will be assigned to utilities that have applicability for multiple systems.

Initially, there will be no central assignment of XB/ZIB names. A developer will only need to select a name not already being used. If this should become a problem later, the DBA will coordinate assignments of XB/ZIB names. Routines that only apply to a particular application should carry the name prefix of that application.

2.2.3 BZ - Local Area Routines

BZ will be reserved for local program development, when there is no immediate plan or intention to distribute the system outside of the Area. ISC's should notify everyone within their Area who may be writing programs to use BZ as the first two characters of their application name. The ISC's should monitor and/or assign names with the BZ format to assure there is no duplication within the Area.

It is recommended that Areas use the third digit to identify the Area in which the system was developed. In this way, you can ultimately share the program with another Area with the assurance that it will not interfere with the local programs developed by the other Area. A list of Area codes to be used for this purpose is shown in Figure 1. BZ should be used when there is no intention to share the system outside of the Area. In all other cases, the ISC should contact the DBA for a global name assignment.

2.2.4 Universal Globals - the DBA will coordinate the name assignments of globals that have applicability for multiple systems. Globals that only apply to a particular application should carry the namespace of that application.

2.2.5 Standard Table Globals - The first two characters of the IHS Standard Tables will begin with AU. The third character will indicate the type of global as follows:

T = Table
P = Patient File
A = Administrative File

The fourth character will indicate whether the global supports UCI translation (i.e., can be accessible from multiple UCI's). These codes are:

T = Translation required
N = Not required

Thus a table, supporting UCI translation, would have as its first four characters:

AUTT _ _ _ _

Care should be taken not to mis-use the AU name. Routines and globals that only apply to a particular system should carry the name prefix of that system.

2.2.6 **Use of Prefix AVA** - This prefix is reserved for IHS changes to VA files such as the New Person file. Its use should be coordinated through the DBA.

Codes for Area Use
in Local Program Development

A	Alaska
B	Billings
C	Aberdeen
D	Bemidji
H	OIRM
L	California
N	Navajo
O	Oklahoma
P	Portland
Q	Albuquerque
S	Tucson
U	Nashville
X	Phoenix

Figure **1**

Reserved File Numbers

File Number Range	**Reserved For**
0-9999	VA DHCP Supported Systems
90000-99999	IHS RPMS Systems (After 5/5/93)
8000000-8999999	DSM, DSD, DTM, & DDPS
9000000-9999999	IHS RPMS Systems (Pre-1993)
1000000-1099999	Area 10, Site 00-99, File 000-999 - Aberdeen
1100000-1199999	Area 11, Site 00-99, File 000-999 - Alaska
1200000-1299999	Area 12, Site 00-99, File 000-999 - Albuquerque
1300000-1399999	Area 13, Site 00-99, File 000-999 - Bemidji
1400000-1499999	Area 14, Site 00-99, File 000-999 - Billings
1500000-1599999	Area 15, Site 00-99, File 000-999 - California
1600000-1699999	Area 16, Site 00-99, File 000-999 - Nashville
1700000-1799999	Area 17, Site 00-99, File 000-999 - Navajo
1800000-1899999	Area 18, Site 00-99, File 000-999 - Oklahoma
1900000-1999999	Area 19, Site 00-99, File 000-999 - Phoenix
2000000-2099999	Area 20, Site 00-99, File 000-999 - Portland
2100000-2199999	Area 21, Site 00-99, File 000-999 - Tucson

Figure **2**

3. File Numbering Conventions

3.1 Reserved File Numbers Blocks of file numbers have been reserved for each Area, the DDPS, and for RPMS core systems as shown in figure 2 above.

3.2 Multiple Files When applications are developed that involve multiple files, the same integer may be used for all files directly related to the package, and decimal numbers used for members of the group. For example, a Personnel Package developed in Area 10, Site 05, might have the file number 1005007 and the group of files might be numbered as follows:

Personnel	1005007.1
Title	1005007.2
Wage Scale	1005007.3

3.3 Common Pointer Files Common pointer files that are pointed to by various applications (e.g., LOCATION file) should be numbered in a different manner to indicate these files are not unique to a single application. For example, the common pointer files in the IHS RPMS Systems will be numbered 9999999.n where n is the next available sequential canonic number. If you have more than nine files in any single group you should append two digit numbers if you want them to sort out properly (e.g., 01, 02)--(.11 sorts lower than .2).

3.4 Area ISCs Files The Area ISCs will assign file number ranges to their various sites, keeping in mind that there may be more than one system at a site. A range of numbers should also be retained for the Area office. The allocation of numbers will vary from Area to Area, depending on circumstances. Not that one may typically want to assign different site numbers to FileMan globals in PRD and PVT UCI's.

3.5 Locally Developed Application When a locally developed application is to be incorporated into the IHS Core System, the files will be renumbered with the range 9000000 - 9999999. If one site wants to incorporate another site's application, the receiving site may want to renumber the files, or it may choose to run the application with the original site's file numbers. When each site conforms to these conventions, the transfer of applications from site to site should not be difficult.

3.6 File Manager File Numbers File numbers in File Manager must be canonic; i.e., must not have leading zeros or trailing zeros following a decimal. This includes sub-files generated by multi-valued fields.

Appendix B - Request for Exemption to RPMS Programming Standards

Request for Exemption to RPMS Programming Standards

Package: Date:

Program:

Line Number:

Applicable Standard:

Reason for Exemption:

Developer

SRCB Review Date:

Recommend APPROVAL___ DISAPPROVAL___

Comments:

Verifier(s)

DSD Action Date:

Request APPROVED___ DISAPPROVED___

Comments:

Director, DSD

Appendix C - UNIX/DOS File Naming Standards

1. **Purpose** Applications developed for the RPMS are distributed in sets of files, the names of which must conform to the specifications of the host operating system. A defined operating system-independent file name format will simplify the activities of staff responsible for manipulation of these files (Area support staff, facility site managers, etc.) and provide a structured paradigm that can be used for automated manipulation of these files by future applications. This standard is being established for this purpose.

   1.1 **Distributed Applications** This standard applies only to files of distributed applications. It is not necessarily binding upon filenames for test and verification copies of an application, due to their inordinately long version numbers (See **Appendix D - Version Numbering for RPMS Systems).**

2. **General Standards** The general standards to be followed in assigning names for all host operating system files are as follows. Patches are addressed separately.

   2.1 **Positions 1-4** Positions 1-4 are reserved for the namespace of the RPMS application. If the application namespace is less than four characters, the remainder of the character positions may be padded with underline (_) character(s).

   2.2 **Position 9** Position 9 is a period or decimal point (.).

   2.3 **Namespace Case** Namespace and alphabetic codes will be in lower case for Distribution filenames.

3. **Standards for Distribution Files** The standards to be followed in assigning names for RPMS application distribution files are as follows.

   3.1 **Positions 5-6** Positions 5-6 are reserved for the major portion of the RPMS application's version number (the portion preceding the decimal point). If less than two digits, this value should be right-justified and padded with one or more number 0 digits.

   3.2 **Positions 7-8** Positions 7-8 are reserved for the minor portion of the RPMS application's version number (the portion following the decimal point). Position 8 will always be a 0 except when annotating a .pdf file as outlined elsewhere.

   3.3 **Position 10** Position 10 is reserved for an alphabetic code indicating the type of distribution file. The codes to be used are as follows:

Code	**Description**
r	Distribution Routines
g	Distribution Globals
u	Unix Files. Archived Unix Files

Example: apch0190.r would denote production UCI routines for Version 1.9 of the PCC Health Summary package

3.4 Position 11

Position 11 is a flexible field. Possible values for this field are as follows.

Code	**Description**
m	Routines or globals to be installed in MGR UCI
s	Required if using "u" in position 10 to denote Unix scripts/files

Example: xu_0710.rm would denote VA Kernel Version 7.1 routines to be installed in the manager UCI

3.4.1 Multiple Files

A single-digit numeric value used to denote one of multiple files to be installed in a production UCI.

Example: ade_0520.gl would denote the first set of globals to be considered in an installation of Version 5.20 of the IHS Dental package

3.5 Position 12

Position 12 is an optional numeric field. Possible values are as follows:

3.5.1 Multiple MGR/Script Files

A single digit value used to denote one of multiple files to be installed in the manager UCI or multiple UNIX script files.

Example: xu_0710.gm2 would denote the second set of manager UCI globals to be considered in an installation of Version 7.1 of the VA Kernel

4. Standards for Patch Files

The standards to be followed in assigning names for RPMS application patch files are as follows.

4.1 Position 10

Position 10 is reserved for a numeric. The numeric is to be the "tens" position of a two digit patch number. If there is no ten digit, a zero is to pad this space.

Example: xu__0710.10p would denote routines file for patch number 10 of Kernel Version 7.1

Example: xu__0710.02p would denote routines file for patch number 2 of Kernel Version 7

4.2 Position 11 Position 11 is reserved for a numeric. The numeric is the "ones" position of a two digit patch number.

Example: xu__0710.01p would denote a routines file for patch number 1 of Kernel Version 7.1

4.3 Position 12 Position 12 is reserved for an alphacharacter as follows.

Code	**Description**
p	Patch Routines
b	Patch Globals
n	Patch Notes

Example: bw__0200.01p Routines for patch 2 of Women's Health Version 2.0

5. Standards for Host Operating Shell Programs

Shell programs shall be in the form as outlined above through position nine. Position 10-16 will contain "install".

Example: aib_0300.install

6. Standards for Documentation Files

The standards to be following in naming documentation files is as follows.

6.1 Position 8 Character position 8 in a documentation file will contain one of the following codes.

Code	**Description**
u	User Manual
t	Technical Manual
s	Security Manual
r	Readme File

i	Installation Guide
n	Release Notes
o	Other

6.2 Position 10-12 Character positions 10-12 will contain "pdf" to designate that the particular manual was prepared in "pdf" format.

Example: bw__020u.pdf - User Manual in PDF format for the Women's Health Version 2.0

bw__020t.pdf - Technical Manual in PDF format for the Women's Health Version 2.0

6.3 Distributed Documentation File All documentation files will be "zipped" prior to final distribution. The "zipped" file will contain those files as outlined above.

Example: bw__0200.zip will contain the User, Technical and Readme File for Women's Health Version 2.0

7. Standard for Final Distribution File

The final distribution file will be a compressed tar file named using the above general standards for Positions 1-4 and position 9. The file will contain all the files necessary for the application, i.e., routines, globals, notes, Readme, all documentation files and any other files specified.

Example: bw__0200.tar.gz contains:

bw__0200.r	Routines
bw__0200.g	Globals
bw__0200.zip	Archived Documentation files
bw__0200.us	Archived Host Operating System Files
bw__0200.install	Host Operating System Installation Program Files

Appendix D - Version Numbering for RPMS Systems

1. Purpose

Applications developed for the RPMS go through three primary phases of evolution; testing (both alpha and beta), verification, and distribution. A single version of an application can go through multiple iterations of the first two steps before being distributed, with each iteration differing from the previous one. This standard establishes a standard format whereby multiple iterations of an application can be easily managed by developers, testers, and verifiers.

2. Definitions

2.1 **Distribution Version**- the version number assigned to the application when released for IHS-wide installation.

2.2 **Target Version** - the intended distribution version number assigned to the application while going through the testing and verification phases.

2.3 **Iteration** - a single set of files containing all routines, globals and notes necessary to install the application.

2.4 **Sequence Number** - the number identifying the current iteration of the application in either testing or verification phases. Testing sequence numbers and verification sequence numbers are not consecutive.

3. Format

Format of UNIX file name:

3.1 **Testing** - Versions of an application submitted for alpha or beta testing will be signified by an lowercase letter t immediately following the target version. Immediately following the lowercase letter t will be the sequence number of the test version. There is no distinction between alpha and beta test iterations.

Example: bw__0350.t3r would identify the third test iteration of version 3.5 routines file of Women’s Health.

3.2 **Distribution** - Applications that have passed verification will assume the target version of the last verification iteration.

Example: If bw__0350.t3r passes verification, the application will be distributed as version 3.5 and will be annotated in the unix file as bw__0350.r.

Format of M Routines

3.3 Second Routine Line The 2nd line of all package M routines will reflect the test iterations.

Example: ABCTest; IHS/ABDEV/MMM-Example M routine;
11/4/95;;1.0t3;Test;*0*;11/4/95

3.4 Package File The Package File will reflect the test iterations in the current version field. When the package passes verification the Package File will be cleaned to only include the final distributed version number.

Appendix E - Standards for Submission of Data to DDPS

1. Overview Many applications being developed to run on facility computers will have a requirement to generate and transmit data into an IHS-wide reporting system maintained at the DDPS, in Albuquerque, New Mexico. Examples are the PCC Data Entry System, Patient Registration, Dental Data System and the Admission, Discharge and Transfer (ADT) System.

For these systems, the data will flow from the facility to the Area, where data will be consolidated for all facilities for each system, and then forwarded periodically via IHS wide-area network for each system to the DDPS. The DDPS will not accept data directly from individual facilities.

Standards and procedures have been developed to facilitate this transfer, and are described in this document.

2. Facility Procedure

2.1 Creation of Transaction Global

A programmer or analyst developing a system with IHS-wide reporting requirements will need to determine the criteria by which data will be selected from the facility data base for transmission to the Area/DDPS.

There are a number of ways this can be done, including the following:

- Selection based on the transaction encounter date,
- Selection based on the facility posting date,
- A special flag to indicate whether data has been transmitted,
- Creation of a special global identifying records that have been created or modified since the last transmittal.

However, the data is identified, a program will be required to extract the data from the data base concerned, and generate a global that can be written to the host operating system (or transmitted directly) to the Area.

The standards to be used in creating this global are as follows:

a. The name of the global will be the four digit namespace assigned to the system, such as AAPC, ACHS, BCHR, etc., concatenated with the word “DATA”. If the namespace has less than 4 digits assigned, characters should be added to fill out the four spaces. Examples of these names are:
^AGTXDATA - Patient Registration

^AAPCDATA - PPC Data Entry
^ADENDATA - Dental Data System

b. The global will have a “0" node, followed by a series of data records subscripted from “1" to “n” for that transmission. For example:

^AGTXDATA(0)=
^AGTXDATA(1)=
^AGTXDATA(2)=

c. The format of the “0" node is illustrated in Figure 1. Pieces 1-5 and 7 are required. The pieces are as follows:

1. Facility code. This is the standard IHS 6 digit code identifying the facility.
2. Facility Name. Narrative name of the facility.
3. Date of Run. The date that the global was created in the format YYYMMDD, where YYY is the number of years since 1700; i.e. 1988 would be 288, MM is the month (01-12), DD is the day of the month.
4. Beginning Date. This field is required, but the definition of the field is program specific. For many programs, data will be selected by a range of dates (i.e., posting dates), and this field will be the earliest date in the date range. If this date is not important or used by the program concerned for data extraction, the date can be the date of the run.
5. Ending date. See above. If not important to the data extraction, the date can be the date of the run.
6. Last Record Transmitted. This field is optional, and was intended for use in cases where data was extracted based on some type of sequential number.
7. Number of Records. This is a count of the total number of records contained in the global (not including the “0" node), and is required.
8. Cartridge Number. The number of the cartridge on which the global will be written. This is for facility audit purposes. Not required.

9. Date Transmitted. The date the data file was transmitted. This field is normally not used, since this date and the date of the run are usually the same.

d. The format of data in each data node is as follows:

Globalname(n)=xxx^data string (fields separated by a “^”)

where: n is the next sequential subscript for the transmission,

xxx is the transaction type, i.e., RG1, RG2, IR1, etc.,

data string is the actual data being transmitted. Each field is separated by the “^”. A piece must exist for each field in the transaction, regardless of whether there is any data for that piece, and all pieces must be in the same sequence as the fixed length transaction required at DDPS.

Care must be taken to assure that the total length of the data in the node will never exceed 255 characters in length. If this should ever happen, the data needs to be broken down into two record types (i.e., RG1 and RG2).

An example of a global created for a particular application might be as follows:

```
^AGTXDATA(0)=508201^CARL ALBERT
HOSPITAL^2860804^2860701^2860731^^^3^^^
^ATGXDATA(1)=RG1^508201^336^SMITH^JAMIE^H^01^0608908^..
^AGTXDATA(2)=RG2^TOAHTY^MARY^^^Y^223098761A^...
^AGTXDATA(3)=RG1^508201^947^BEGAY^JOHNATHON^L^01^021
4893^...
^AGTXDATA(4)=RG2^ADAMS^EVA^^^N^123456789A^...
```

The AIB Record Consolidation System at the Area will eventually receive the above data and convert each variable length record into the required fixed length record for processing on the central computer. If there are two record types, as in the above example, they will be combined into a single record at DDPS, with the data from the second record (RG2) added onto the end of the data from the first (RG1).

2.2 Transmission of Global to Area

A general purpose utility routine XBGSAVE has been developed to read a global as described above, and copy it to a tape cartridge. This utility requires that the name of the global be placed in the variable XBGZL prior to execution, i.e.,
S XBGL=”AGTXDATA:

D ^XBGSAVE
Other variables can be set to select various IO options. See XBGSAVE for details.

Execution of this routine can be initiated automatically by incorporating the above routine at the end of the program that creates the global, or can be designed as a separate option to be selected from the program's main menu.

For MSM Systems:

The XBGSAVE routine will create global save format to a UNIX text file.

When the copy operation has been completed, the original global needs to be deleted before additional data is extracted and posted from the facility data base. If an operator fails to do this, new data will be merged with the data already sent to the Area, and this will all be re-forwarded to DDPS on the next transmission. This could cause problems, depending on the system concerned. A programmer might want to consider automatically killing the global after successfully copying to the host operating system.

The above process can be done at whatever frequency is agreed upon between the Area and the facility.

3. Area Procedure

When facility data files are received at the Area for a particular application, they are merged into a file containing similar data from other facilities with the utility routines "AIB".

The AIB consolidate routines will determine the name of the file to be updated from the name on the incoming file (i.e., AGTXDATA). If the file already exists on disk (i.e., AGTXBLOB), the AIB consolidation routines will merge the data from the incoming data file into the global file. If the file does not exist, it will create the file.

At periodic intervals, normally once or twice a month, the Area will create a transmission file for all information in the global files using the AIB routines. This will delete the global files on the Area disk, and the cycle will start over when the next data file is received from a facility.

These convert programs refer to a table which identifies the fixed length transaction to be created from the incoming data records. The pieces in the data string must be in the same sequence as the transaction to be created. The table identifies the column in which each field starts, and the length of the field. For instance:

;1;3;	RECORD TYPE	(Starts in col.1;3 characters long)
;4;6;	IHS FACILITY CODE	(Starts in col.4;6 characters long)
;10;2;	TYPE OF ACTIVITY	
;12;5;	REFERRAL CODE	

If an incoming field is longer than the field in the fixed length transaction, it will be truncated.

If more than one record type is contained in the incoming global, e.g., RG1 and RG2, the convert program will combine the two records into a single transaction by adding the data from record two (RG2) onto the end of record one (RG1).

The fixed length transactions will be written out to a transaction tape at DDPS, or stored in a transaction file, and processed on the next update of the application concerned.

Any programmer/analyst designing a system with reporting requirements to DDPS will need to notify DSD with as much lead time as possible to allow the convert program to be developed by DSD staff.

Each Area will be required to transmit monthly data sets to DDPS computer. Users can submit their data using the Area Data Consolidation System (AIB).

4. DDPS Procedure

The process for updating the master database will revolve around a program called *incoming* which begins by identifying file types and processing order of files received, updating each subsystem in the master database, activating a report generator, and electronically informing the submitting area of the status of its data.

4.1. IHS Subsystems

The master database is comprised of several systems which include Patient Registration (GTX), Patient Eligibility (ELG), Health Record Add (GHA), Inpatient Care (APC), Outpatient Care (INP), Inpatient and Outpatient Contract Health Services (CHS), and Patient Merge (GDM). The systems are listed in the order in which they are processed.

4.1.1 Patient Registration (GTX)

This subsystem is the first to be processed since it adds patients' records which are updated by the rest of the systems listed. The master database is searched for an existing facility code and health record number. If an entry is found, the health record is scanned for the associated patient record link. If no link is found, this indicates that this record was inserted by the health record add (GHA) subsystem and needs to be updated with the rest of the patient's personal information. If an entry is not found, the patient and associated personal information is inserted.

4.1.2 Patient Eligibility (ELG)

This subsystem is the second to be processed due to the remaining subsystems requiring updated eligibility information. This system updates an existing patient's eligibility record or creates a new eligibility record, depending on whether a starting or ending data is included in the incoming record.

4.1.3 Health Record Add (ELG)

This subsystem is the third to be processed because the remaining subsystems update records according to facility code and health record number. This system inserts a new record into the chart table with only a health record number and an associated facility code. A flag is set in the new chart record to signal the GTX subsystem to connect this health record with a person. If the associted patient record exists, the GTX system updates the patient record with GTX information and the flag is removed.

4.1.4 Ambulatory Patient Care (APC)

This subsystem, which inserts an outpatient record, is the fourth to be processed.

4.1.5 Inpatient Care (INP)

This subsystem, which inserts an inpatient record, is the fifth to be processed.

4.1.6 Contract Health Services (CHS)

This subsystem, which inserts a contract health services inpatient or outpatient record, is the sixth to be processed. The source of this information not only comes from the Areas but from Blue Cross/Blue Shield (BCBS).

4.1.7 Patient Merge (GDM)

Formerly Delete/Merge, this subsystem is the last to be processed. This subsystem closes out old health record numbers by setting an ending date in the record, and creating a new health record using the updated information provided and populating the rest of the new record with the most recent information available.

4.2 Post Updating

To close out a month's processing, a table containing each Area's submission information is updated with the status of each system's results. If an Area has not submitted its data, the table will contain NULL values which will flag the operator and send a message to the Area contact and the appropriate DDPS individual(s). A log fle is also created for every file processed and resides in the same directory as the proessed file which contains a full account of the processing and a summary of the essential totals. These totals are inserted into the Area's submit table which informs DDPS that an Area has submitted its data. This process also allows for a previous month's count to be compared to the current month's count. If the current counts are within a preset percentage, an error is generated and sent to the Area contact and DDPS individuals.

FORMATS OF GLOBALS CREATED AT FACILITY

```
AGTXDATA(0)=508201^CARL ALBERT^2860804^2860701^2860731^ ^45^ ^2860805
- - - - -   - - -  - - - - - -  - - - -  - - - - - - -  - - - - - - -
|           |      |           |       |       |       | |  | |Date
|           |      |           |       |       |       | |  | |Mailed
|           |      |           |       |       |       | |     - - - - - -
|           |      |           |       |       |       | |  |Cartridge
|           |      |           |       |       |       | |  |Number
|           |      |           |       |       |       | |  - - - - - - -
|           |      |           |       |       |       | |Number of
|           |      |           |       |       |       | |  Records
|           |      |           |       |       |       | - - - - - - - - -
|           |      |           |       |       |       |Last Record
|           |      |           |       |       |       |   Posted
|           |      |           |       |       |       - - - - - - - - -
|           |      |           |       |       | Ending Date
|           |      |           |       |       - - - - - - - - - -
|           |      |           |       | Beginning Date
|           |      |           |       - - - - - - - - - - - -
|           |      |           | Date of Run
|           |      |           - - - - - - - - - -
|           |      | Facility Name
|           |      - - - - - - - - - - -
|           | Facility Code (AR-SU-Fc)
|            - - - - - - - - - - - - - - - - - -
| Global Name
- - - - - - - - - - -
```

```
AGTXDATA(1)=XXX^DATA STRING
- - - - -   - -  - - - - - -
|           |   |
|           |   | Transaction data, with fields separated by the “^”
|           |   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
|           | Transaction type
|           - - - - - - - - - - - - -
| Global Name
- - - - - - - - - - -
```

Appendix F - RPMS Documentation Standards

1. Purpose The purpose of these documentation standards is to provide a basic documentation structure that can be applied to every software package, to provide consistency in all documentation, and to provide criteria by which documentation of a national package can be verified.

2. General Standards The general standards to be follow in preparation of all RPMS documentation is as follows.

2.1 Definitions

RPMS Package - An RPMS package is RPMS software intended for IHS and tribal distribution and implementation that is fully supported by the DSD. Each package is considered a component of RPMS and is assigned by the Director, DSD, to a development center for development and maintenance.

Non-RPMS Automated Information Systems Package A software package not developed by an approved development center, and not supported by DSD, but may be for use within IHS.

Package Documentation Package documentation is the information that describes the functions, implementation, use, maintenance, and distribution of RPMS packages.

Sensitive Information - Sensitive information is information that requires protection due to the risk and magnitude of loss or harm that could result from inadvertent or deliberate disclosure, alteration, or destruction.

Adobe Acrobat Reader/Exchange - Commercial software designed to bring electronic documents to a wide range of users. Cross-platform documents, which are created in Adobe Acrobat Portable Document Format (PDF) are called PDF documents. Acrobat Reader enables Windows and Windows NT, Macintosh, DOS, and UNIX users to review, navigate through and print any PDF document.

2.2 Mandatory Components The four mandatory components of RPMS software documentation consists of:

2.2.1 Installation Guide and/or Release Notes

2.2.2 Technical Manual

2.2.3 User Manual

2.2.4 Security Manual, if applicable

2.3 Preparation All RPMS software documentation will be prepared in electronic format using the Adobe Acrobat software (pdf format). The pdf file will be distributed as part of the national release and the file will reside as part of the archived distribution file.

2.4 Documentation Descriptions and Standards

2.4.1 Installation Guide (Mandatory)

2.4.1.1 Identify the package by name, version number, and release date.

2.4.1.2 Provide an instructional guide for installing the software. Describe issues that should be considered prior to initialization and how to prepare for initialization.

2.4.1.3 Describe the installation process in logical steps and include a statement recommending where the software should be loaded (e.g., "initially load software into a test account and then finally into the production account"). Note any routines, globals, etc., that may be removed from the system after completion of the installation.

2.4.1.4 Provide guidance and suggestions for system configuration and global placement. List any requirements necessary for successful installation of the package. List any reference material that may be required during the installation process. List items that the installer should produce from the system after installation of the national package.

2.4.1.5 Describe the resources required for the national package. Include Central Processing Unit (CPU) capacity, disk space, unique devices, and other pertinent resources. Provide a formula for sizing, if applicable.

2.4.2 Release Notes (Mandatory for subsequent releases - may be included as part of the Installation Guide but must be distinguished as such)

2.4.2.1 Identify the RPMS package by name, version number, and release date.

2.4.2.2 Describe any modifications and enhancements to the national package software since prior release. This information is needed in advance of loading the software.

2.4.3 Technical Manual

2.4.3.1 **Purpose** Technical documentation should provide sufficient information about the software for programmers, ISCs and OIRM technical personnel to operate and maintain the program applications without additional assistance from the package developer(s). The Technical Manual must contain, at a minimum, the mandatory sections listed below.

2.4.3.2 **Title Page (Mandatory)** Include the name of the national software package, the version number, the preparation date, the name of the development center and the logo "IHS RPMS".

2.4.3.3 **Preface (Mandatory)** Supply a brief statement identifying the document in terms of its purpose, scope and targeted audience.

2.4.3.4 **Table of Contents (Mandatory)** Provide a table of contents with page references to major chapters and/or sections of the manual.

2.4.3.5 **Introduction (Mandatory)** Include an overview that describes the package. The introduction should convey to the reader the major function(s) and purpose(s) of the package, and how the software accomplishes the objective(s).

2.4.3.6 **Orientation (Optional)** Address package-specific notations or directions (e.g., symbols used to indicate terminal dialogues or user responses).

2.4.3.7 **Implementation and Maintenance (Mandatory)** Provide information to assist the ISCs, OIRM personnel, and PSGs in the implementation and maintenance of the national package.

This section may include information regarding the entry of required site-specific data; a description of parameters configured to meet the needs of individual sites; sample configurations; and work sheets to assist in determining the parameters to be entered for the site.

2.4.3.8 **Routine Descriptions (Mandatory)** Provide a list of routines with comprehensive descriptions of the function.

2.4.3.9 **File List (Mandatory)** Include a list and brief description of files that come with the package. The description should indicate what data comes with the file and whether or not that data will overwrite existing data, if applicable (this will normally apply only to VA packages since including data with a file is against IHS Standards).

2.4.3.10 **Exported Options (Mandatory)** Provide a list of the options in the package. Indicate distribution of menus to users and note any restrictions on menu distribution.

2.4.3.11 **Cross-references (Mandatory)** Provide a brief description of all cross-references exported with the package.

2.4.3.12 **File Diagram/Flowchart (Optional)** For packages that include numerous files, provision of a chart representing the relationship among files is highly desirable.

2.4.3.13 **Archiving and Purging (Mandatory)** Describe any archiving or purging capabilities of the package and any necessary instructions or guidelines.

2.4.3.14 **Callable Routines (Mandatory)** List all entry points in the package that can be called by other applications. This list must include the actual entry points, a brief description of the function of these entries, a description of all required variables, and any restrictions on the use of the entry points.

2.4.3.15 **External Relations (Mandatory)** Explain any special relations and agreements between the routines and/or files/fields in this package and the routines and/or files/fields in other packages. List any routines essential to the functions of this package, for example: Could an outpatient facility function without programming related to inpatient activity and avoid system failure? Specify the version of VA FileMan, VA Kernel, and other packages required to run the package.

2.4.3.16 **Internal Relations (Mandatory)** Identify, if applicable, any routines, files or options within this package which cannot function independently of other programs. For example, Which menus can stand alone? Does the functioning of a particular option assume that entry/exit logic of another option has already occurred? List such options with their programming SACC approval dates.

2.4.3.17 **How to Generate On-line Documentation (Mandatory)** Provide the file numbers and/or file number ranges, and namespaces along with any special templates. Inform the users where to find the Kernel documentation and how to print the data dictionaries and menu diagrams.

2.4.3.18 **Glossary (Mandatory)** Provide a glossary of terms that relate to the specific national package.

2.4.3.19 **Index (Optional)** Provide a package-specific index.

2.4.3.20 **SAC Requirements/Exemptions** Provide a listing of any items required by the SAC to appear in the Technical Manual. All exemptions granted by the SACC must noted specifying the date of the exemption and the actual exemption.

2.4.4 User Manual

2.4.4.1 **Purpose** The User Manual is a document designed to be helpful to the user. This documentation will provide sufficient information for users to competently operate the national software package. Variability in the content is accepted. The User Manual must contain, at a minimum, the mandatory sections listed below.

2.4.4.2 **Title Page (Mandatory)** Include the name of the software package, the version number, the preparation date, the name of the Development Center, and the logo "IHS RPMS".

2.4.4.3 **Preface (Mandatory)** Supply a brief statement identifying the document in terms of its purpose, scope, and targeted audience.

2.4.4.4 **Table of Contents (Mandatory)** Provide a table of contents with page references to chapters and/or sections of the manual.

2.4.4.5 **Introduction (Mandatory)** Provide an overview that sets forth a description of the software package. Note related RPMS and/or VA manuals and other reference materials for a modular manual and the purpose for the individual module. Distinguish the major topics and issues within the package. The introduction should convey to the reader the major function(s) and purpose(s) of the package, and how the software accomplishes the objective(s).

2.4.4.6 **Orientation (Optional)** Address any package-specific notations or directions (e.g., symbols used to indicate terminal dialogues or user responses).

2.4.4.7 **Package Management (Mandatory)** Address unique legal requirements pertaining to the package and necessary security measures to protect the integrity of the package and its data (e.g., a package may use an electronic signature code or data that may not be changed because it is supplied by another agency).

2.4.4.8 **Package Operation (Mandatory)** Describe what the user needs to know in order to competently operate the package. The information should include how the user can access on-line documentation.

2.4.4.9 **Glossary (Mandatory)** Provide a glossary of terms that relate to the specific package.

2.4.4.10 **Index (Optional)** Provide a package-specific index.

2.4.5 Package Security Manual or Guide

2.4.5.1 **Purpose** A package security manual or guide will be created for controlling the release of sensitive information related to the national software package. This document will not be included in any Freedom of Information Act (FOIA) request releases. Distribution of this document is limited to the ISC and OIRM personnel. Since certain keys and authorizations must be delegated for proper management of the system, information about these items may be found elsewhere in the technical and user manuals.

2.4.5.2 **Title Page (Mandatory)** Include the title "Package Security Guide", the name of the software package, the version number, the name of the Development Center, the preparation date, the logo "IHS RPMS" and the words "SENSITIVE INFORMATION".

2.4.5.3 **Package Security (Mandatory)** Provide a section on security requirements for the national package. Document all locks and keys, special FileMan access codes, and other security measures included in the package. Describe the purpose of security keys. Include official policy unique to the package regarding the modifications of software and distribution of the package.

2.4.6 ***Users Guide to Computing*** The *Users Guide to Computing* has been adopted as a stand alone instructional guide for general computer usage. This guide describes programming conventions common to all national packages and is to be used as a reference source.

2.4.7 Patch Documentation Requirements

2.4.7.1 **Notes** All patches to certified RPMS software require a notes file outlining system requirements, contents of the distribution, modifications to the software, reason for the patch, etc. Developers should follow the format outlined in Notes File procedure elsewhere in this handbook.

2.4.7.2 **Patch Module Entry** Each patch to a certified RPMS package will be entered into the IHS Patch Module by the responsible developer. In addition, another developer is responsible for completion of the patch prior to review and verification by SRCB.

3. Documentation Style Standards

The following style standards are provided to promote consistency in IHS RPMS package documentation.

3.1 Abbreviations and Acronyms

Spell out the abbreviation or acronym when it is first used and put the abbreviation or acronym in parentheses after it is initially spelled out (e.g., Indian Health Service (IHS)). After initially spelled out, use the abbreviation or acronym without parentheses.

3.2 Appendices

Appendices are considered supplemental information and should be used for appropriate materials such as listings of descriptions or definitions of data elements, sample reports generated by the package, summaries of commands and biographies of supportive reference materials.

3.3 Change Pages

Change Pages will be identified with the new version number and release date. Additional distinguishing marks (e.g., vertical lines in margins) can be used as needed. Distribution of change pages should be coordinated with the SRCB.

3.4 Computer Dialogue

Recreate the program's computer dialogue in COURIER font. COURIER closely resembles the screen display. Distinguishing the computer dialogue from the manual text is very important.

3.5 Date

Use the release date of the manual on the title page and footer.

3.6 Double Sided

Design documentation to be reproduced in a double-sided format. Any blank pages that are added for order of sections and organization of the manual should be numbered. No statement is necessary on blank pages.

3.7 Field Names

Use all uppercase letters (e.g., NAME).

3.8 File Names

Use initial capital letters (e.g., Patient file).

3.9 Fonts

3.9.1 TIMES 10 POINT font is to be used throughout for the text, with TIMES 10 POINT BOLD used for headings and items to be emphasized, such as Notes, Cautions, or Warnings.

3.9.2 COURIER 10 POINT CAPS is to be used to indicate the computer prompts. Computer printouts and sample reports are to be so annotated.

3.9.3 HELVETICA 10 POINT BOLD will be used in the computer-interactive sections to indicate the information typed by the user in response to the computer prompts.

3.10 Headers and Footers Headers and footers are required for all manuals. The information contained in the headers and footers is:

3.10.1 Package name and version number

3.10.2 Release date (month/year)

3.10.3 Page number

3.10.4 Manual type (User, Technical, etc.)

3.10.5 Chapter or Section name

3.10.6 Header and footer information can be arranged at the discretion of the documenter.

3.10.7 Lines separating the header and footer from text is mandatory.

3.11 Headings Place headings at the left margin (block style), distinguish them in bold or uppercase, and do not underline.

3.12 Margins 1.0 inch top and bottom with headers and footers placed in the margin area; left and right margins shall be 0.75 inch.

3.13 Option Names Use initial capitalization to set off option names. Option and file names (since they both use initial caps) should be distinguished with the body of text by the word option or file. For example, Enter or Edit File Entries option and Patient file. Do not use single or double quotes with options.

3.14 Page Dimensions 8.5 by 11 inches.

3.15 Package Names Use initial capitalization when referring to package names (e.g., Laboratory). Initially completely spell out the package name when used in text, after that time, the abbreviated package name may be used.

3.16 Page Numbers Place page numbers in the footer. All pages should be numbered except for the title pages and their blank back sheet. Use lower case Roman numerals to number pages that contain the table of contents, preface, acknowledgements or lists of tables.

3.17 **Paragraph Numbers** Paragraph numbers, if used, shall be in legal format throughout the document. The lowest level allowable in outline style shall be level four (e.g., paragraphs 1.1.1.1, 1.1.12, etc.)

3.18 **Prompts** Use double quotes around prompts used within text (e.g., "Select PATIENT NAME:"). Use single quotes within double quotes only. Don't use quotes around prompts in a recreated computer dialogue.

3.19 **Return Key/Enter Key** Use this symbol: RETURN when referring to users entering information. The RETURN symbol does not need to be underlined. It is recommended that this key be defined in the orientation section of the manual.

3.20 **Sub-Headings** Place sub-headings at left margin (block style), distinguish them in bold or uppercase, smaller size type than the heading, and do not underline.

3.21 **Text** In most cases, the text is to be presented flush left with no paragraph indents and no hyphenation.

3.22 **Up-Arrow** Use this symbol: ^ when referring to the up-arrow or hat. Do not use single quotes around the up-arrow or hat. It is recommended that this key be defined in the Orientation section of the manual.

3.23 **Version Number** Spell out and capitalize (or initial cap) the word version when used in the cover (e.g., Laboratory Version 5.0). Abbreviate version to V space when used in the header or footer (e.g., Lab V 5.1).

3.24 **Computer Menus/Options** In the manuals, computer menus or options are shown inside a box resembling a computer screen. Use a different type of box for report examples. Be consistent in whatever format is chosen and use it throughout the entire document.

3.25 **Wording Convention** "Press" rather than "strike" or "hit" is used in instructions - i.e., "Press RETURN". Likewise, "Type" is preferred to "enter" in instructions to the user - i.e., "Type TRANSACTION".

Appendix G - DHCP/RPMS GUI Standards

(Note these standards are still pending VA approval and may need some modification for IHS)

1. Purpose The purpose of these GUI Standards is to provide a basic structure that can be applied to every software package, to provide consistency in all user interface software, and to provide criteria to follow for national certification.

2. DHCP/RPMS GUI Standards All VHA Decentralized Hospital Computer Program (DHCP)/RPMS user interface software will, at a minimum, meet the following standards and comply with the spirit of the guidelines (in the respective reference works).

2.1 Reference works For all user interfaces for applications that run in the Microsoft Windows graphical environment, the book, The Windows Interface, An Application Design Guide (for Macintosh, ISBN: 0-201-62216-5) and The Windows Interface Guidelines for Software Design (for Windows 95 and Windows NT, ISBN: 1-55615-679-0) shall be adhered to unless modified by this document. In addition, to make applications more accessible to individuals who have disabilities or who are aging, the document Designing Accessible Applications (draft version 15 or later) from Microsoft Corporation, should be followed.

For all user interfaces for applications that run in other graphical environments, corresponding guidelines should be followed. For example, for the Macintosh, the Macintosh Human Interface Guidelines shall be adhered to unless modified by this document.

Standard metaphors shall be used in the major application areas of clinical, management, and support applications. The standard metaphors shall be approved by the Application Requirements Group responsible for the application area.

3. DHCP/RPMS GUI Guidelines

3.1 General Principles

The intent of using this standard in DHCP/RPMS applications is to provide the end-user with enough consistency in the use of DHCP/RPMS applications that he or she can approach applications with confidence, and readily transfer knowledge gained from previous experience to new applications and functions. For presentation options not yet covered by standards, developers must apply the following three criteria which have been and will continue to be the underpinnings of an effective interface.

3.1.1 Simplicity

Simplicity will be achieved through limiting each single operation to one that can achieve predictable closure, provide informative feedback to every action, allow easy reversal of actions, and generally ensure that each dialog is the result of action initiated by the user.

3.1.2 Consistency

By being consistent throughout all applications the DHCP will maximize the opportunity for users to transfer and apply skills to new applications. Consistency will be achieved, to a great extent, through the widespread reuse of applications software. The use of standard controls and adherence to conventions widely used in commercial products are important components of consistency.

3.1.3 Intuitiveness

Intuitiveness (or self-evidency) will be achieved through the uniform use of the “Object/Action” paradigm, whereby the user selects an object (e.g., an icon, a file, or a block of text) and selects an action that is to be applied to that object (e.g., an icon, a file, or a block of text) and selects an action that is to be applied to the object (e.g., run, delete, or copy). Intuitiveness is also achieved through use of real world metaphors, where objects are identified in a way that is meaningful to the user. Intuitiveness will be supplemented with a rich help facility throughout all DHCP/RPMS applications.

4. GUI Controls

At the user level, the human interface consists of controls or widgets that permit the user to manipulate the information presented there. Graphical User Interface (GUI) metaphors, for the most part, still control procedural applications. The GUI may be built from reusable objects and may portray the information presented as data objects, but the underlying applications and their data are often procedural. Only when both the interface and the underlying applications are data-centric can the user interface itself be an object-oriented user interface (OOUI).

Since the manipulation of data in a GUI is still application-centered rather than document-data-centered, the GUI creates only an illusion of data objects. When that illusion is convincing, the difference between GUI and OOUI is insignificant. When it is not possible to maintain that illusion, the difference is painfully obvious.

For instance, using Cut-and-Paste editing or Drag-and-Drop manipulation, makes it easy to move a prescription to a progress note. The prescription information can then appear there as a text string. However, a progress note does not accept a prescription as a container accepts a piece of paper. Instead, it changes the representation of the prescription's structured information to create a text string. The user cannot readily invert the process by using Drag-and-Drop to move text string prescription from a progress note back to the medication profile.

In their current form, these guidelines define some components of DHCP/RPMS GUIs. The intent is to provide a descriptive framework to help developers produce the same GUI functionality using different tools on different platforms. The framework is user-centered; the framework specifies what the user does and sees at the GUI rather than how the GUI or its underlying applications accomplish it.

These guides address general issues as well as details concerning the controls from which the interface is built. Menus, check boxes, buttons of various kinds are all controls - visual components of the GUI. They are characterized here from the users' viewpoint; they are seen to consist of Revealed Features, and to exhibit certain Standard Behaviors. The user perceives a control's Revealed Features visually. Its associated Standard Behaviors are perceived as responses of the control to various stimuli (mouse motions, clicks and keystrokes). This characterization, while similar to the object-oriented language paradigm of object properties and methods, does not correspond exactly to that model.

Most controls have several Revealed Features. A check box, for instance, consists of an area that indicates whether it is checked (ON or OFF) and an area that displays its caption. Some controls may exhibit different behaviors when the same stimulus is applied to different features of the same control.

In order to create an interface that users will be able to learn, every control must demonstrate predictable behavior. All controls that present the same Revealed Features must exhibit the same Standard Behaviors.

5. General Guidelines

5.1 **General Guideline #1**

Controls having the same Revealed Features must have the same Standard Behaviors. If a novel behavior is required of an existing control, then a new widget is required for that control. The new widget must have a unique visual appearance to allow the user to distinguish it from existing widgets.

Windows and dialogs are controls too, and they should obey General Guideline #1. Application windows, modal dialogs, and semi-modal dialogs should each have a distinctive visual appearance. Each distinctive appearance should be associated with its own standard behaviors. For example all interactive warnings of a given severity should appear in dialogs having the same visual appearance, reveal a predictable choice of options for exiting from the dialog, and use the same icons.

The "natural" reading sequence in English is left to right from top to bottom. The organization of text and data entry controls in each window should respect this convention.

5.2 **General Guideline #2a**

The preferred organization of controls within a window is one that is correctly interpreted by the user when read according to the natural style of the user's native written language. For English this is left-to-right across the page from top-to-bottom.

5.3 **Guideline #2b**

When space limitations dictate an alternative presentation, a newspaper column type organization is permissible. The sequence in which controls are Focused when the user completes an action or the user strikes a special key indicating that the previous or next control should be focused (often accomplished with cursor keys or TAB key) should follow the user's natural reading style. The first portion of the screen presents static, or read-only information (an exception would be a time & date that is presented by the system but could be modified by the user). The central portion of the active area contains controls that can be selected and varied within the window, but are generally not affected until the window is closed or the dialog is exited. Command buttons which effect action are placed to be encountered after other information has been presented and the user has a basis for selecting some particular action or mix of actions.

For example, with columns of radio buttons or check boxes, the tab order would be from the top of the column to the bottom, then over to the next column. Likewise, such order would hold for panels in a window. There may also be exceptions where the center of attention would for example be in the center of the page, so the sequence might appropriately begin there.

5.4 **General Guideline #3** The preferred sequencing of controls within a window is one that moves the focus according to General Guideline #2. Not every control within a window need to be enabled. Controls (menus and other widgets) should share a common appearance that indicates that they are inactive. In many GUI environments this appearance is a gray shade (the control or its caption appears dimmed) that informs the user that the control cannot be used. Controls that cannot be used should not accept the Focus; they should be skipped.

5.5 **General Guideline #4a** All disabled controls must have a similar, distinctive visual appearance (ex: dimmed) and must not be allowed to accept the Focus.

Appropriate use of inactivation limits the user's possible actions at every point in the user session to those choices that are valid. This prevents many needless error messages and dialogs.

5.6 **General Guideline #4b** Any control (or a possible choice within a control) that is not valid at a given moment must appear disabled.

5.7 **General Guideline #4c** Although GUI controls are usually activated using a pointing device, it is important that users be able to perform all tasks using only their keyboards. This not only suits the preferences of some users but also provides for an alternative interface control method should the pointer be lost or broken. Because uniformity and simplicity are crucial to making the GUI easy to learn and memorable, there must be a uniform keyboard method for accessing GUI controls.

5.8 **General Guideline #5** A control affording a keyboard shortcut or accelerator must identify the shortcut by emphasizing the appropriate character in its caption. The corresponding shortcut is then a Special key-Character keystroke combination. The choice of the Special key and the type of emphasis is determined by a convention consistent with the operating system and other applications.

In MS Windows, the preferred Special key is Alt and the type of emphasis is an underline. Thus, an underlined E means, "use the Alt-E keystroke combination." If a second Special key (ex: the Ctrl key) is used, it must have a corresponding, unique character emphasis. The preferred shortcut character is the first character of a control's caption (the first letter in a menu caption, for instance).

The choice of interface control captions (for menus, buttons, and other widgets) must involve user testing to select those captions that users most consistently identify with the action performed or option selected.

5.9 General Guideline #6 Control captions must be meaningful to the intended users.

Although the first character of a caption is the ideal shortcut character, user testing may reveal captions that are meaningful to users but when taken together create a conflict due to multiple occurrences of the same initial character. When this occurs, Guideline 7 must yield to Guideline 6.

5.10 General Guideline #7 The preferred shortcut key for a control is the first character of its caption.

Standard PC keyboards are equipped with a number of special function keys. When these are used (singly or combined with either Alt or Ctrl) to activate a control, the shortcut must appear as part of the control's caption so that it is visible. In MS Windows the convention is that the accelerator appears at the end of the caption (ex: Spelling F7).

5.11 General Guideline #8 All shortcuts, whether implied (General Guideline #5) or explicit must be visible to the user.

Shortcuts for application-independent services that the operating system provides for application developers should follow operating system guidelines. Shortcuts for Cut, Paste, Copy, Print, and others occur in word processing, spreadsheet, and other non-DHCP/RPMS applications with which the intended users may already be familiar. When these same actions are implemented in DHCP/RPMS packages, the developers should conform to existing shortcut key conventions to provide inter-application consistency.

5.12 General Guideline #9 Application-independent shortcuts should conform to the target operating system's existing conventions.

Similarly, any task that is commonly accomplished in native applications by using operating system level common dialogs (ex: Font selection, palette color changes, etc.) should be handled in a new application with those same dialogs. The reason for this is that users change applications more frequently than they change platforms.

5.13 General Guideline #10 When a system level Common Dialog exists for an application function, it is preferable to use the system dialog rather than to create an application-specific dialog. At the user interface level, consistency among applications on a given platform trumps consistency of one application across platforms.

Similarly, the details for implementing direct manipulation techniques (drag-and-drop) should comply with the guidelines that apply to the host operating system and other applications on that platform. Again, consistency with other applications on the host platform trumps application consistency across platforms.

5.14 Windows-specific Guideline #11 With the mouse pointer on the source control, the user initiates Drag-and-drop by pressing the Left Mouse Button. With the mouse pointer over the destination control, the user completes the process by releasing the button. A change in the mouse pointer to an icon that represents the information contained in the source control signals the drag-initiation event. A change in the visual appearance of a control as the user drags a source over it signals that the underlying control may accept the source data (ex: a new border appears around the destination, the destination background color changes, etc.). Dragging over a control that cannot accept the source may be indicated by a change in the mouse pointer icon to the international NO symbol (a circle around a backslash).

6. Discussion

In general, anything that is standard for the GUI platform (Windows, Mac, etc.), should not be controversial. Any control (or metaphor) used by any widely used commercial application (i.e., the top 3 or 4 word processors, spreadsheets, databases, etc.) is also not controversial. For example, the current Software Services client application development environment, Delphi/Visual Basic, provides a large number of components whose behaviors should not be superseded.

When deciding whether to create new object classes or to utilize existing components, if the added functionality of a newly designed class is not expected to be utilized by more than one object instance, that class should not be built. Instead, code should be within the appropriate event handler of an instance of an existing class.

Color - Color should never be the only means of setting apart information. In clinical settings, red may be reserved for “emergency” or “panic” use. In accounting, red may be a convention for a negative balance (which may be an “emergency” or “panic” circumstance for an accountant!).

Fonts - Fonts not exported with Windows should be avoided or be exported and installed with the application.

Content areas - Content areas that are “writeable” areas shall have a white background and content areas that are “read-only” shall have a pale yellow, slightly grayed, or other non-bright background.

Patient name selection - Delphi/Visual Basic components from the Software Services Standard Component Repository should be used for patient name selection.

Sound - Use of sound should be curtailed to rare and special circumstances.

These guidelines are minimal development constraints meant to help users develop a reliable "intuition" about GUI behavior. They are not a formula for producing a successful GUI. Following these guidelines at best ensures that DHCP/RPMS GUIs will show some consistency. It does not guarantee that those GUIs will be usable. It is useability testing that will ultimately reveal both GUI design errors and missing DHCP/RPMS functionality.

Indian Health Service
Office of Information Resource Management

Procedures, Policies and Guidelines

June 27, 1996

Indian Health Service
Office of Information Resource Management

XB Programmer Tools

June 27, 1996

XB Programmer Tools

1. Summary / Overview

This document is designed primarily for RPMS application programmers. Area and Site IRM personnel can find this document helpful in understanding how the XB/ZIB utility routines. These utilities are provided as a result of the pursuit to use cpu cycles to save programmer cycles, enhance productivity, and enhance support ability.

The IHS/VA Utilities are in the XB namespace for routines that are not MUMPS (M) implementation specific. Routines that are implementation specific will be in the ZIB namespace.

Programmer tools are available from programmer mode thru the menu-driver routine XB. There are no files associated with the XB/ZIB package.

2. Routine Descriptions

2.1 XB

This routine lists available utilities in the form of a menu with a brief description of what the utility does. New utilities may be added to this routine by adding the appropriate “;;” entries to the bottom of this routine. See routine XB1 for further documentation and the menu options for the XB/ZIB Utility package.

2.2 XB1

In this routine each label represents a menu.

2.3 XBARRAY*

This utility provides a word processing format of free text and local variable references to build an array.

2.4 XBBPI

This routine builds a pre-init routine for a specified package. The pre-init routine will delete FileMan dictionaries being created by the package. Data globals and templates will be saved.

2.5 XBCLM

This routine displays a column number header followed by the passed string.

2.6 XBCLS

This routine clears the screen.

2.8 XBCNODE

This routine counts unique values in a selected global node.

2.9 XBCSPC

This routine checks selected fields to see what percent of the time it exists in the entries in a file, and if it should be unique, makes sure it is unique.

2.10 XBAD0

This routine sets the DA array from D0, D1, etc., or D0, D1, etc., from the DA array. If the variable XBDAD0=2, it sets the DA array, otherwise, it sets D0, D1, etc.

2.11 XBDATE

This routine limits routines selected by %RSEL to routines edited after some date.

2.12 XBDBQDOC

This routine contains double queuing shell handler documentation.

2.13 XBDBQUE

This utility gives the programmer a very friendly way to provide single, double, or no queuing at all to the applications. Report programming is structured into a compute routine, a print routine, and an exit routine. The call to XBDBQUE handles all the devices and host files as necessary.

- o %ZIS with "PQM" is called by XBDBQUE
- o The user will be asked to queue if queuing has not been selected.
- o IO variables as necessary are automatically stored.
- o XBDBQUE can be nested. The compute and print phases can call XBDBQUE individually (XBIOP is then required)
- o The appropriate %ZTSK node is killed.

<u>Input Variables</u>

(Mandatory)

Either	XBRC = Compute Routine
Or	XBRP = Print Routine

(Optional)

XBRC	=	Compute Routine
XBRP	=	Print Routine
XBRX	=	Exit Routine that cleans variables (<u>Highly Suggested</u>)
XBNS	=	Namespace of variables to auto load in ZTSAVE("NS*")=""
	=	"DG;AUPN;PS;..." ; (will add '*' if missing)

Or

XBNS("xxx")="" Where xxx is structured as in ZTSAVE variable arrays where xxx is as described for ZTSAVE("xxxx")=""

XBFQ	=	1 Force Queuing
XBDTH	=	FM date time of computing/printing
XBIOP	=	pre-selected printer device constructed with ION ; IOST ; IOSL ; IOM (mandatory if the calling routine is a queued routine itself)
XBPAR	=	%ZIS("IOPAR") values for host file with XBIOP, if needed

```
EX:     S XBRC="C^AGTEST",XBRP="P^AGTEST"
        S XBRX="END^AGTEST",XBNS="AG"
        D ^XBDBQUE ;handles foreground and tasking
        Q
```

2.14 XBDIQ0

Documentation routine for XBDIQ1.

2.15 XBDIQ1

This utility provides an easy pulling of data from the FM data base. The data is returned in an array and format designated by the programmer. (An enhanced EN^DIQ1)

Input Variables

The following variables are the same as for EN^DIQ1 but with friendlier results.

1. Data arrays are returned into the target array, @DIQ, in a variety of formats controlled by the setting of DIQ(0). The default return array is @DIQ(Field Number)= external value of field of the field.

2. Data retrieval is antiseptic! It does not disturb any local variables.

3. The input variable DA can either be an array or a literal of explicit values or a literal of variables.

Entry Points

ENP^XBDIQ1(DIC,DA,DR,DIQ,DIQ(0))
Returns @DIQ(FLD)= data for One Entry for fields indicated in DR

ENPM^XBDIQ1(DIC,DA,DR,DIQ,DIQ(0))
Returns @DIQ(DA,FLD)= data for Multiple Entries
DIC("S") can be set and used for screening entries

For ENPM the lowest level DA must be set to 0 (zero)

$$VAL^XBDIQ1(DIC,DA,DR)
Returns External value of one field.

$$VALI^XBDIQ1(DIC,DA,DR)
Returns Internal value of one field.

$$DIC^XBDIQ1(DIC)
    Returns constructed DIC from file/subfile number

Input Variables

DIC, DR, DIQ are defined as in the call to EN^DIQ1

DIQ(0) Format Options

If DIQ(0) is not present the default is set to NULL

0 OR NULL	@DIQ(FLD)	= external data base value
1	@DIQ(DA,FLD)	= ""
2	@DIQ(DA(x),..,DA,FLD)	= ""
(1 or 2)_I	@DIQ(...,FLD,"I")	= internal data base value
(1 or 2)_N	NULL fields are not returned	

DA Can be the array DA or a literal string in descending order. The literal string may be made up of explicit values or variables.

EX: DA = "1,23,45"

Or DA = "1,PATDFN,BLDFN"

Or DA = BARVDA("EOBSUB") :: ="BAFCLDA,BARITDA,BAREDA"

For ENPM the lowest level DA must be set to 0 (zero)

2.16 XBFORM*

This utility provides two entry points. The first is the editing of a word processing form where free text and markers for variables are placed. The second is for the generation of an array as defined by the form being referenced. Several options in the form definition enhance its flexibility. XBFORM0 contains the XBFORM documentation.

The programmer must supply a file for the forms with the .01 field being the name of the form and another field that is a WP field to hold the form itself.

(Requires XBLM and VALM utilities to be present.)

Entry Points

EDIT^XBFORM(NAME,DIC,FLD)

Edits and displays the form. Place the call to EDIT in the code where the data or variables have been gathered. Typically this is one line previous to the call to $$GEN^XBFORM. Once the form is designed the EDIT call is commented out. It uses XBLM to display the run time data as it will be structured into the array.

Exiting the editor portion the XBLM display can also provide markers in the document for those that are working with forms.

See FORM DEFINITION OPTIONS for instructions on format options within the form.

Input Variables

NAME	Name of form.
DIC	File number of the file with the forms
FLD	Field number of the WP field holding the form definition.

Y = $$GEN^XBFORM(NAME,DIC,FLD,%Y,FORMAT,OFFSET)

Generates the form into the root array indicated by %Y. The call to $$GEN must have all variables referenced already present in the partition. The return value of $$GEN is equal to the last line set in the array.

Input Variables

NAME	Name of form.
DIC	File number of the file with the forms.
FLD	Field number of the WP field holding the form definition.
NAME	Name of the form
%Y	The root of the target array to be built. Either a global or a variable root as in the format used for a %XY^%RCR call. (%RCR is actually used)

FORMAT	null or zero	The array is built %Y(line)="...
	1	The array is built %Y(line,0)=".... as used by VALM.

OFFSET	The offset is line numbers in building the array. The array will start construction at OFFSET +1. The value of the last line created is returned $$GEN.

WP Format Definition Options

Free Text:	Free text is key stricken where desired. Do not use "~" as it is used to mark variables and their placement.
Variables:	The reference to a variable is marked with a beginning "~" and a trailing "~". The trailing ~ is always required even if the variable is last item on the line.

Functions

Comment Line	Programmers comments can be put into the form and are ignored by the generator.

Mnemonic Variables

This is a short hand for variables.

Output Transform

Mumps code can be input that will transform a selected variable's output.

Functions

Comment Line	Begin the line with a ';'

Mnemonic Variable

Namespaced variables can be long. A mnemonic reference is available to make life simple. Mnemonic definitions are place at the top of the form. Mnemonic variables then can be placed anywhere in the form.

Begin each line of definition with a '#'. The mnemonic is separated from it reference by a '|' (vertical bar). Multiple mnemonic references on the same line are separated by '*'

Mnemonic definitions are placed at the top of the form

(M|R) MNEMONIC|REFERENCE

```
Example:    #D|DUZ*V|BARVPT
            #I|BARIPT

        ~D    will be interpreted as meaning ~DUZ
        ~V    will be interpreted as meaning ~BARVPT
        ~I    will be interpreted as meaning ~BARIPT

        (BARIPT is an array storing IHS Patient Information)
        (BARVPT is an array storing  VA Patient Information)
```

MNEMONIC MARKER

The mnemonic markers can be used anywhere in the WP form. It is marked by a beginning and ending pair of '~'s. A vertical bar separates the mnemonic and the value of the subscript.

(M|S) MNEMONIC|SUBSCRIPT

```
Format      ~mnemonic|variable subscript~
Example     ; following the mnemonic reference definition
Define      #D|DUZ*I|BARIT
                ~D|~     for DUZ
                ~D|0~    for DUZ(0)
                 ~I|.01~  for BARIPT(.01)
```

Output Transform

A MUMPS expression of X. 'X' must be used literally in the function defined.

A simple mumps output transform capability is also provided to aid in form design. A variable or mnemonic indicated will have its output transformed prior to being put into the form. The definition line is placed at the top of the form. It is started with a '*', followed by the variable reference, followed by a ':', and then the function of x 'f(x)' to be performed. Multiple lines can be used and multiple outputs defined on a line separated by a '*'.

Setup

*var1:mumps code1*var2:mumps code2
*mnemonic3:mumps code3*mnemonic4:mumps code4

Ex: *DUZ(2):$J(X,10,2) will transform ~DUZ(2)~ to $J(DUZ(2),10,2)
*D|2:$J(X,10,2) mnemonic notation of same

Special Output Transforms provided by XBFORM

$$MDY(X) Returns a date format of MM/DD/YY

Many times only a mm/dd/yy is desired. This function automatically converts any external date to mm/dd/yy. (An external form of date is returned by XBDIQ1.

*M|S:$$MDY(X) a literal ~"NOW"~ or variable ~IT|9~
ex: *"NOW":$$MDY(X) or *IT|9:$$MDY(X)
returns mm/dd/yy

$$WP("X") Word Processing field printing

Word Processing fields are handled by this output transform.

*M|S:$$WP("X")	for a word processing field array ~M|S~
NOTE: "X"	THE QUOTES ARE ABSOLUTELY NECESSARY! The variable array must have the form VAR(subscript,n) where n = 1:1
$$FL(X)	Fill Lines

Sometimes it is necessary to jump to a specific line

Blank lines are used to fill from the present line through the line indicated.
*19:$$FL(X) ~19~ fill lines through 19 with a " "

TIPS for VALM Users

The compilation of the form resides in the ^TMP($J,"XBFORM","Form Name", nodes. Those subscripts are organized ..."Form Name",Line,Column)=. Lines and columns are straight forward for text and variables start at their column +.5 to indicate the expression has to be evaluated. If the $$FL output transform has been used, the programmer will have to calculate the new line numbering offsets manually.

Adding video attributes with VALM calls needs to be done within the INIT lines of the VALM program being called.

Examples of XBFORM Calls

```
BARFORM0       ; IHS/ADC/PDW - FORMS FOR XBFORM ;  [ 07/06/95  11:03 AM ]
        ;;1.0c4;IHS ACCOUNTS RECEIVABLE;;JUN 21, 1995
TEST    ;;
        ;** set up variables
        D ENP^XBDIQ1(200,DUZ,".01:.116","BARU(")
        ;** setup a word processing field
        F I=1:1:5 S BARWP(101,I)="   LINE "_I_" has the value of "_I
        ;** setup form name
        S BARFORM="PW TEST"
        ;** call form editor
        D EDIT^XBFORM(BARFORM,90053.01,1000)
        ;** call array generator
        S LASTLINE=$$GEN^XBFORM(BARFORM,90053.01,1000,"BARFM(",0)
        Q
```

Editor Form Example

```
;------ Mnemonic References Definitions
#G|BARU*D|DUZ
#W|BARWP|
;------  Output Transforms - Mumps Expressions - $$MDY - $$WP("X") - $$FL(X)
*"TODAY":$$MDY(X)
*DUZ(2):$J(X,10,2)
*W|101:$$WP("X")
*39:$$FL(X)
;--------------------------------START OF FORM---------------------
          DUZ       DUZ(2)       DUZ(0)    DT

  Variable Reference   ~DUZ~    *~DUZ(2)~     ~DUZ(0)~  ~DT~
  Mnemonic|Subscript   ~D|~      ~D|2~        ~D|0~
                  * DUZ(2) has a $J(X,10,2) output transform
```

```
   /----------------------------------\
   |  NAME   ~G|.01~   |
   |  ADD2   ~G|.112~  |
   |  ADD3    ~G|.113~ |
   |  CITY     ~G|.114~ |
   |  STATE  ~G|.115~ |
   |  ZIP       ~G|.116~ |
   \----------------------------------/


    TERMINAL CHARACTERISTICS
    TYPE                ~IOST~
    R MAR               ~IOM~
    FORM LENGTH ~IOSL~
```

Word processing example W|101 with an output transform W|101:$$WP("X")

```
=============================================================
~W|101~|
=============================================================
SKIP TO LINE 40
~39~
LINE 40
/////////
END OF FRAME
```

ARRAY GENERATION EXAMPLE

```
[DEV,DSD]>ZW BARFM
BARFM(1)="                     DUZ       DUZ(2)       DUZ(0)    DT"
BARFM(2)=" "
BARFM(3)="  Variable Reference   60       *   1546.00   @         2950706"
BARFM(4)="  Mnemonic|Subscript   60       1546         @"
BARFM(5)="                             * DUZ(2) has a $J(X,10,2) output transform"
BARFM(6)=" "
BARFM(7)="              /----------------------------------\"
BARFM(8)="              |  NAME   WESLEY,PAUL |"
BARFM(9)="              |  ADD1   PO BOX 958 |"
BARFM(10)="             |  ADD2   'CRAVEN ELMS MOBILE HOME |"
BARFM(11)="             |  ADD3   ' #5 |"
BARFM(12)="             |  CITY   EDGEWOOD |"
BARFM(13)="             |  STATE  NEW MEXICO |"
BARFM(14)="          |  ZIP    87015  |"
BARFM(15)="          \-----------------------------------/"
BARFM(16)=" "
BARFM(17)=" "
BARFM(18)="             TERMINAL CHARACTERISTICS"
BARFM(19)="             TYPE       C-VT100"
BARFM(20)="             R Mar      80"
BARFM(21)="             FORM LENGTH 24"
BARFM(22)=" "
BARFM(23)="Word processing example   W|101 with an output transform
W|101!$$WP("X")"
BARFM(24)="=================================================="
BARFM(25)="   LINE 1 has the value of 1"
BARFM(26)="   LINE 2 has the value of 2"
BARFM(27)="   LINE 3 has the value of 3"
BARFM(28)="   LINE 4 has the value of 4"
BARFM(29)="   LINE 5 has the value of 5"
BARFM(30)="=================================================
BARFM(31)="SKIP TO LINE 40"
BARFM(32)=" "
BARFM(33)=" "
BARFM(34)=" "
BARFM(35)=" "
BARFM(36)=" "
BARFM(37)=" "
BARFM(38)=" "
BARFM(39)=" "
BARFM(40)="LINE 40"
```

2.17 XBLM

XBLM provides simplified utility entry points for the programmer to utilize the VALM utility without having to design their own browser. It has an interface already built in to access the host file systems so that hard coded or FM generated displays can be loaded into the browser. It also has an array entry point.

This utilizes and requires the presence of the VALM software as distributed by the VA and/or IHS.

Requirement

The VALM software must be installed and inited.

The DEFAULT HOST FILE as identified in FILE(1) by $$PWD^%ZISH(.FILE) must have its permission for WR and group set for RPMS users.

D ^XBONIT	Installs the XBLM Protocol
D ^XBL	Installs the XBLM List Manager Template.

Entry Points

FILE^XBLM("Directory","File Name")
Displays file indicated

Directory	Host file directory
File Name	File Name to be displayed

SFILE^XBLM — Manual selection of host file for display. Allows the user real time access to host files. Wildcarding of the file during selection is allowed.

VIEWR^XBLM("TAG^ROUTINE","Header")
Displays printout of the routine. (non - FM, using IO)

VIEWD^XBLM("TAG^ROUTINE","Header")
Displays printout of the routine. (FM - using EN1^DIP)

DIQ^XBLM("DIC","DA")
Displays EN1^DIQ for the DIC,DA

ARRAY^XBLM("ARRAY(","Header")
Displays the array(...,n,0) as necessary in VALM format.

2.18 XBNEW

This routine provides the programmer with a SACC-approved manner of performing an exclusive NEW preserving the required Kernel variables at the same time. It also includes wild carding.

Entry Point

EN^XBNEW("TAG^ROUTINE","Variable List")

Input Variables

"TAG^ROUTINE"	The routine to be executed within the exclusive newed environment.
"Variable List"	The list of variables to be carried into the exclusive newed environment. EX: "AGDFN;AGINS;AGP*" Wild card allowed. Required Kernel variables are automatically added to the list.

2.19 XBDH*

XBDH is the Header Editor main routine. XBDHD sets basic info about file and fields. XBDHD1 compiles header line. XBDHD2 works with special choices. XBDHDF gets field information for header line editor. XBDHDF1 checks jump syntax. XBDHDIP is an overlay of DIP2 for Auto FileMan. XBDHDSV compiles header info for auto entry into DIP. XBDHDSP puts spaces between headers.

2.20 XBDHNTEG

XB integrity checker.

2.21 XBDICV

This routine sets FileMan dictionary version numbers.

2.22 XBDIE

Use this routine to nest DIE calls.

2.23 XBDIFF

The difference between two dates/times is returned with this routine.

2.24 XBDINUM

Use to convert a non-dinum file to a dinum file.

2.25 XBDIR

The purpose of this routine is to provide an interface methodology for a call to ^DIR, to ensure correct handling of variables, and to provide for the expressiveness of an extrinsic function.

2.26 XBDR*

This routine builds a string which sets variables DIR and its descendants for use in a routine. The string is stored in the variable "%", and in the "Temp" storage area for the screen editor for the current device.

2.27 XBDSET

This routine selects FileMan dictionaries individually, by a range, or for a specific package. This routine can be called from another routine by setting the variables XBDSLO, XBDSHI, and then D EN1^XBDSET.

2.28 XBENHANC

This routine prints enhancements to a package from the entry in the package file. Entry point EN^XBENHANC(ns) is used with the caller providing the namespace of the package.

2.29 XBFCMP

This routine compares FileMan files in two UCIs.

2.30 XBFDINFO

Given a file/subfile number, a field number, and an array root, this routine will return information about the specified field. The information will be returned as a subscripted variable from the root passed by the caller.

2.31 XBFIXL1

This routine asks the user to select a set of routines, for the programmer information, and standardizes the format of the first line of each routine.

2.32 XBFIXPT

This routine fixes all “PT” nodes for files 1 through highest file number in the current UCI.

2.33 XBFLD*

This routine lists dictionaries which may be selected individually or by a range of dictionary numbers. XBFLD0 prints field triggers.

2.34 XBFMK

This routine kills variables left by FileMan.

2.35 XBCDIC*

This routine cleans up ^DIC and ^DD. XBCDIC2 checks dictionary names and data globals. XBCDIC3 checks ^DD. XBCDICD deletes bad files.

2.36 XBFIX

This routine counts entries in FileMan files and fixes.

2.37 XBCFXREF

Use this routine to check and fix cross references.

2.38 XBCOUNT

This routine counts entries in a FileMan file.

2.39 XBFRESET

Routine is used to reset file globals.

2.40 XBFUNC*

These routines make up the Function Library.

2.41 XBGC

Use to copy a global at any level.

2.42 XBGCMP

Routine compares two different globals. XBCMP2 contains help for XBCMP.

2.43 XBGLDFN

Routine to get last DFN.

2.44 XBGSAVE

See separate section in this manual for further guidance. Generic global save for transmission globals.

2.45 XBGTI

Use to restore globals saved in DSM %GTO format.

2.46 XBGTOT

Use for fast save to tape.

2.47 XBGXFR

Routine is used to transfer global trees.

2.48 XBGXREFS

Use to get cross references for one field in one file.

2.49 XBHEDD*

Contains the components for the Electronic Data Dictionary. The EDD provides a more user friendly method of reviewing data dictionary structures and global structures. It contains on-line documentation.

2.50 XBHELP

Use to display help text from a routine.

2.51 XBHFMAN*

These routines are for the help frame manual.

2.52 XBKD*

Use to kill DICs and globals.

2.53 XBKERCLN

Use this routine to clean out Kernel namespace items prior to install.

2.54 XBKSET

This routine sets minimal Kernel variables.

2.55 XBKTMP

Use to clean ^TMP nodes for the current job.

2.56 XBKVAR

This routine sets minimal Kernel variables.

2.57 XBL

This routine is used for a list template exporter.

2.58 XBLCALL

Use to provide a list of callable subroutines.

2.59 XBLML

Use to enter or reset XB display in List Template File for List Manager.

2.60 XBLUTL

This routine lists all entries in the ^UTILITY global for the current $J where $J is the first or second subscript. This is most useful from programmer mode. If used thru the XB menu, ^UTILITY($J) is killed in ^XBKSET before this routine is run.

2.61 XBLZRO

This routines lists the 0th nodes of FileMan files.

2.62 XBMAIL

This utility generates a mail message to everyone on the local machine that holds a security key according to the namespace, range, or single key provided in the parameter. The text of the mail messages must be provided by the developer, and passed to the utility as a line reference. The utility uses the first line after the line reference as the mail message subject, and subsequent lines as the body of the message, until a null string is encountered. This places an implicit limit on your mail messages to the maximum size of a routine. Suggested text may be used to inform the users that a patch has been installed, and to describe any changes in displays or functionality, or problems addressed, and provide a contact number for questions, e.g:

```
------------------------------------------------------------------
Please direct your questions or comments about RPMS software to:
OIRM / DSD
Albuquerque NM  87110
505-837-4189
```

2.63 XBNODEL

This routine sets FileMan dictionaries so users cannot delete entries. Protection is provided by SET'ing the "DEL" node of the .01 fields in the selected dd's to "I 1".

2.64 XBOFF

Use to set reverse video off.

2.65 XBON

Use to set reverse video on.

2.66 XBPATSE

Use to search routines for patched versions.

2.67 XBPFTV

This routine is used to return pointer field terminal value.
NOTE TO PROGRAMMERS; Use entry point PFTV. Do not use the first line of this routine, as pending initiatives in MDC might make a formal list on the first line of a routine invalid. Given a file number, file entry number, and variable name into which the results will be placed, return the terminal value after following the pointer chain. U must exist and have a value of "^".

Formal list
- 1) F = file number (call by value)
- 2) E = file entry number (call by value)
- 3) V = variable for results (call by reference)

Scratch vars:
- D = Flag, 1 = Done, 0 = continue
- G = Global for file F

2.68 XBPKDEL

Programmers can use this routine to remove options, input, sort, print templates, help frames, bulletins, functions, and if indicated, security keys for a package.

> XBPKNSP must be set to the namespace, e.g., "AICD", if this routine is called from a preinit. If you want security keys deleted, set XBPKEY=1 if this routine is called from a preinit. Call LIST^XBPKDEL to list all namespaced options, templates, etc. Call RUN^XBPKDEL to delete all namespaced options, templates, etc. The RUN and LIST entry points are for programmer use and are not to be called from a preinit. Preinit calls XBPKDEL directly with variables set as indicated above.

2.69 XBPOST

XB/ZIB installation postinit.

2.70 XBPRE

Preinit that checks requirements, etc.

2.71 XBRESID

This routine deletes residual entries in ^DD by a range of dictionary numbers. A residual entry is one that has no parent. The process is reiterative, so an entry that has a parent in ^DD, and the parent is deleted because it has no parent, will also be deleted. The parent of an entry in ^DD is defined as another entry in ^DD for sub-files, and an entry in ^DIC for primary files.

The range of dictionary numbers is inclusive but residual entries for the high file number will not be deleted at the sub-file level. This is because sub-files are numbered with the primary file number with decimal numbers appended. The terminating check is ^DD entry greater than high file number specified, so by definition all sub-files for the high number are greater than the high number.

This routine can be called by another routine by setting XBRLO and XBRHI and then D EN1^XBRESID.

2.72 XBRESTL1

Routine to restore first line of routines from a save file.

2.73 XBRLL

This routine lists a single routine line by line noting the length of the line plus the cumulative character count.

2.74 XBRPRTBD

This functionality has been moved to ZIBRPTRD because of the use of non-standard $Z special variables. The GO is provided for backwards compatibility.

2.75 XBRPTL

This routine prints the selected routine down to the first line label.

2.76 XBRSBD

This routine saves selected routines edited after a given date.

2.77 XBRSELM

Routine selector.

2.78 XBRSIZ

List routine names and sizes with overall total.

2.79 XBRSRCH*

Search data dictionary (DD) for called routines, common check logic, search input transform for routines, search output transform for routines, search cross references for routines, and search miscellaneous for routines.

2.80 XBRXREF*

This routine re-cross references selected cross references(xrefs) for a file. The xrefs are killed at the highest level and then reset. This is very different from what FileMan does when you RE-INDEX a field. FileMan does a logical kill and then sets the new xrefs. The reason for this is multiple fields may set the same xref so one would want to kill only the ones set by the field being RE-INDEXed. One must re-xref all fields that set any one of the xrefs being killed and reset, unless the xref is set the same from multiple fields. This is very hard to explain. Therefore, if you do not understand the problem, you probably should not be running this routine.

This routine executes an entry point in ^DIK to build the xref logic for all xrefs on the file. It then deletes the logic for all xrefs not selected, and executes another entry point in ^DIK to actually xref the file.

TRIGGERS are very complex animals which do not have a xref to kill and may be conditional and may have no affect.

2.81 XBSAUD

This routine sets 'audit' on at the file level for selected files.

2.82 XBSAUTH

This routine sets FileMan dictionary authorities:
"AUDIT" "DD" "DEL" "LAYGO" "RD" "WR".

2.83 XBSFGBL

This routine returns a subfile global reference.

NOTE TO PROGRAMMERS: Use entry point EN. Do not use the first line of this routine, as pending initiatives in MDC might make a formal list on the first line of a routine invalid. Given a file or subfile number and global reference form, this routine will return the global reference in the form specified.

F (form) is optional but if passed should equal 1 or 2.
If F is not passed the default form will be 1.

F = 1 will be in the form ^GLOBAL(DA(2),11,DA(1),11,DA,
F = 2 will be in the form ^GLOBAL(D0,11,D1,11,D2,

Formal list:

1) S = subfile number (call by value)
2) G = global reference (call by reference)

2.84 XBSITE

Routine to set DUZ(2).

2.85 XBSUMBLD

This routine requests the user to select a set of routines and generates an integrity checking routine for the selected routines. The user is asked to enter the name of the generated routine.

The VA's equivalent routine is XTSUMBLD, which also creates integrity checking routine(s).

2.86 XBTM*

This routine, and subsequent routines in the XBTM* namespace, produce a technical manual from information contained in the package. The manual is approximately 80 pages. All, or individual chapters can be printed.

2.87 XBUPCASE

Call to convert to uppercase.

2.88 XBVCH*

Routine(s) to intelligently change variable names.

2.89 XBVCHV

Use to pull in variables and routines from a %INDEX.

2.90 XBVIDEO

Set various video attributes. $X is saved and the cursor is returned to it's original position thru X IOXY (except certain attributes). In addition to the attributes supported by ENDR^%ZISS, some color attributes are supported, and other mnemonics are provided for backward compatibility.

2.91 XBVK

This is the front end for killing local variables in the namespaced parameter. Implementation specific routines are called from this routine which is in the ZIBVK* namespace.

This routine is intended to be called by applications that are done executing in order to KILL any remaining namespaced local variables. E.g., D EN^XBVK("AG") will KILL any local variables that exist in the AG namespace.

Notice that if called in background, and the OS is not supported, the routine will quit, unpleasantly. If your implementation is other than what is supported, and your vendor has implemented all Type A extensions to the 1990 ANSI M standard, you can safely remove the two lines that check for OS, and use the existing call to the MSM-specific routine.

2.92 XBVL

This is the front end for listing local variables. Implementation specific routines are called from this routine. Therefore, it has been moved to the ZIBVL* namespace.

2.93 XBVLINE

This routine asks the user to select a set of routines, asks for the version number, package, date, and sets the second line of each routine.

The form of the version line will be as follows:

```
;;n;package name;patch level;date    E.G.
;;1.1;PCC DATA ENTRY;**1,2**;Sep 9, 1989
```

2.94 XBXTSS

Use for extract and table subscripts.

2.95 XBPATC

This routine will $Order through the patient and 3rd party globals looking for missing entries. It will compare IHS/VA Patient files to define unequal DFN's. It will also look for a null pointer value in Medicaid global.

2.96 ZIBCKPKG

This routine checks UCI for package content.

2.97 ZIBCLU*

This is a general purpose clean up utility global and driver to get UCI. This routine will initiate a job running ^%ZIBCLU0 in each UCI and then wait 5 seconds to elapse before getting the next UCI, skip the UCI this task is in, and then run ^%ZIBCLU0 here. %ZIBCLU0 will remove all dangling ^UTILITY, ^XUTL, ^ZUT entries. This routine is usually started via TaskMan by scheduling the -ZIBCLU- option which runs this routine.

DSM ONLY - $ZU(ZIBI) returns <NOUCi> error at end of UCI list
MSM ONLY - $ZU(ZIBI) returns -NULL- value at end of UCI list

2.98 ZIBER*

MSM error report routines.

2.99 ZIBFIND

MSM-specific utility for finding blocks which contain a specific GBL.

ZIBCC=common count, ZIBUC=unique count
ZIBCHAR=string of characters

2.100 ZIBFMD

Routine to display FileMan installation data.

2.101 ZIBFR

Given a routine name, this routine searches all UCIs and reports the first line of the selected routine to the user.

2.102 ZIBGCHAR

This routine contains non-interactive modifications of global characteristics.
Not all capabilities of the implementation-specific global characteristics routines are reflected in this routine. The argument for each entry point is the unsubscripted name of the global whose characteristics you want to change with the circumflex present. If the call is successful, 0 is returned. If the call is not successful, a positive integer is returned, and the cause can be retrieved at the ERR() entry point.

```
E.C.'s:
S %=$$NOJOURN^ZIBGCHAR("^AUTTSITE")
I % W !,$$ERR^ZIBGCHAR(%)
```

2.103 ZIBGCHR

Routine to search for control characters in globals.

2.104 ZIBGD

This routine displays a selected range or subset of the global directory.

2.105 ZIBGSVED

Routine to save global(s) to tape - DSM specific.

2.106 ZIBGSVEM

Routine to save global to MSM Unix.

2.107 ZIBGSVEP

Routine to save global to DOS media.

2.108 ZIBGTOT

This routine creates a global save in the same format as ^%GTO. The difference is it uses $ZO which is much faster. Because tests showed no significant difference in speed between CDT and CAVL4 this routine accepts CDT only. It would be difficult, although not impossible, to allow partial globals. Therefore, this routine will save complete globals only. This routine always rewinds the tape before saving the globals. That means only one file per tape but that one file may use multiple volumes.

The primary purpose of this routine is to move globals from DSM to MSM. Therefore, the limitation is use of tape only.

2.109 ZIBJRNI

This utility is used to initialize all except the active journal areas of a system. The STU entry point is used by the automatic partition reference for a particular configuration. This was developed for the PC Network Configuration and has only been tested using MSM-PC/386.

2.110 ZIBNSSV

Use to return non-standard ($Z) special variables, e.g.,
W $$Z^ZIBNSSV("ERROR") will write the contents of the error message most recently produced by the OS. These are the variables supported:

ERROR : Text of error message most recently produced.
LEVEL : Number of the current nesting level.
NAME : Name of routine currently loaded in memory.
ORDER : Data value of the next global node that follows

the current global reference.
TRAP : Line label and routine name of the program that is to receive control when an error occurs.
VERSION : Name and release of M implementation.

2.111 ZIBPKGF

Use this routine to obtain an installation status report.

2.112 ZIBPKGP

Use this routine to process implementation status files.

2.113 ZIBRD

Use this routine to display a MSM directory of selected routines .

2.114 ZIBRER*

This routine provides remote error reporting. It $ORDERs through the ERROR LOG, extracting errors executed since the last run. It then WRITEs the errors to a file, and SENDs the file to the identified destination(s) according to the parameter (ENTRY ACTION of option). It removes errors in the ERROR LOG that are more than 180 days old. This routine is non-interactive. It is designed to run in the background from TaskMan only.

Entry point OPT is used to set an option into the OPTION file which is scheduled for every 6 days at 9 PM. The process begins at START.

2.115 ZIBRERP

ZIB remote error file processing routine. An option will be placed in the OPTION file for daily processing of files sent to this machine by the Remote Error Reporting utility beginning the next night at 10:30 PM. The user can change the frequency/time of scheduling by using the TaskMan option thru the Kernel.

2.116 ZIBRNSPC

Use this routine to namespace previously written routines.

2.117 ZIBRPI*

This utility creates an entry in the OPTION file which is scheduled to run daily in TaskMan. It searches for files matching the naming conventions for patch files (specified in the SAC) in the directory indicated by the user. If the package version currently installed on the system and the patch version match, the routines are restored from the file, an entry is made in the VERSION multiple of the PACKAGE file entry, and a report file is sent to the systems indicated by the user. If an action routine (A9 or B9) is detected during the ZLOAD, and the user has the indicated permission to run action routines, the action routine is called after all routines have been restored.

NOTE: Use the same entry point, OPT^ZIBRPI, to edit any changes to the parameters. If the user unschedules the option, the TaskMan options must be used to reschedule it.

2.118 ZIBRPRTD

This routine lists routines edited after given date.

2.119 ZIBRSEL

This is a non-interactive routine select which returns the number of selected routines set into the indicated variable.

E.g.:
```
I '$$RSEL^ZIBRSEL("B-BZZZZZZZ","ARRAY(") W "NONE SELECTED" Q
```

If routines exists in the list or range, their names will be returned as the last subscript of indicated variable in the 2nd parameter. The default is ^TMP("ZIBRSEL",$J,.

If routine B exists, then node ^TMP("ZIBRSEL",$J,"B") will be null. It is the programmer's responsibility to ensure the name of the array is correctly formed.

Variables used:

X = String indicating list or range of routines.
Y = String indicating variable into which to set the selected routines. Default = ^TMP("ZIBRSEL",$J,
F = First routine, if range.
L = Last routine, if range.
N = Number of routines returned.

Q = Quote character.

2.120 ZIBRUN

Use to check for active routine in a specific UCI.

2.121 ZIBSSD

This utility is used as the nightly shutdown routine for MSM PC and is initiated by the scheduled option AZSJ SHUTDOWN. It was developed for the PC Network configuration.

2.122 ZIBTCP

TCP Print Test - This routine must be DONE from the CLOSE EXECUTE when printing to a TCP printer. See below for further documentation.

H = Host IP address
P = Port number
I = Counter

Technical Notes:

MSM TCP uses the "!" to clear the TCP buffer. FileMan (RPMS) uses "!" for a carriage return, line feed. Further, TCP does not recognize "?30" as 30 spaces from left margin. To circumvent these problems, write to a temporary host file, which formats the document, and then read it back into the TMP global. Once it's in the TMP global, $O through the global and write each line with a $C(10) and $C(13) concatenated to the string. This process handles the CR/LF problem at the remote end.
Port 2501 is the assigned port from the vendor for the Net Que.
As of 3Jan95, this has only been tested on the Unix platform using MSM. It should work in a DOS environment using FTP Software's TCP, but needs to be tested.

Below is an inquiry of the Device file and Terminal Type file.

```
OUTPUT FROM WHAT FILE: DEVICE//
NAME: P-TCP TEST PRINTER              $I: 51
ASK DEVICE: YES                  ASK PARAMETERS: NO
VOLUME SET(CPU): TUC                       SIGN-ON/SYSTEM
DEVICE: NO
```

```
FORCED QUEUING: N0
LOCATION OF TERMINAL: MAT PARKENSON PRINTER
ASK HOST FILE: NO                MARGIN WIDTH: 255
FORM FEED: #                   PAGE LENGTH: 256
BACK SPACE: $C(8)   OPEN PARAMETERS:
                     ("XM"_DUZ_$G(ZIBH)_".DAT":"M")
SUBTYPE: P-TCP PRINTER    TYPE: HOST FILE SERVER
```

```
Select TERMINAL TYPE NAME: P-TCP PRINTER
NAME: P-TCP PRINTER           SELECTABLE AT SIGN-ON: NO
RIGHT MARGIN: 255               FORM FEED: #
PAGE LENGTH: 256               BACK SPACE: $C(8)
OPEN EXECUTE: S XMREC="R X#255:1"      CLOSE EXECUTE: D
^ZIBTCP Q
DESCRIPTION: Special Terminal Type used only for P-TCP Printer
Device.
```

2.123 ZIBVCHV

Use to read variables and routines from a %Index.

2.124 ZIBVGE

This routine changes the Volume Group Name in ^SYS(and ^%ZOSF. It is used for the rapid change of the volume group names in the PC Network Configuration. It has only been tested using MSM-PC/386.

2.125 ZIBVKIL

Use to build a namespace variable killer routine in ^.ns.KVAR. Select a %INDEX host file summary on which to build the routine. Select a namespace for the variables and the routine to be built. Enter any package-wide variables. Add D ^.ns.VKL0 to all menu exit actions where package variables are to remain. Add D KILL^XUSCLEAN to the exit action of all other menus.

2.126 ZIBVKMSM

This routine kills variables in the namespace of the variable passed in the parameter and is accessed thru the front end routine XBVK.

2.127 ZIBVLMSM

This routine lists variables that begin with the string entered by the user. Selection of variables is case sensitive. This routine is specific to Micronetics. It will work with any M implementation that has all Type A extensions to the 1990 M ANSI standard implemented. The front end routine, XBVL, stops if any other than an MSM implementation is encountered.

2.128 ZIBZUCI

Swap UCI between volume sets for MSM-UNIX - Save this routine as %ZUCI in the MGR UCI.

This utility permits switching between UCIs and Volume Groups when run in programmer mode. D ^%ZUCI. If switching to a UCI in a Volume Group other than the System Volume Group (0), enter either the Volume Group Number or Volume Group Name along with the UCI Number or Name. A 'help' display identifies all UCIs and Volume Groups that are currently mounted. Use a '?' for 'help'.

A routine may be tied to the UCI,VOL switch. This will be called immediately after the UCI,VOL switch occurs.

XBGSAVE - Generic Global Save

XBGSAVE is a utility to save globals onto a peripheral medium. XBGSAVE determines the operating system (OS) in use, and will invoke OS-specific routines to perform the actual saves. Historically, MSM and DSM routines had been distributed with XBGSAVE. Other OS-specific routines must be provided by the user or submitted to the developer for incorporation into XBGSAVE. Current capabilities are to save MSM globals to cartridge tape, floppy disk, Unix file or 9-track tape, and to save DSM globals to cartridge tape or 9-track tape.

XBGSAVE Routine

XBGSAVE is a parameter-driven routine that: (1) verifies that the global to be saved exists, (2) verifies parameters, (3) determines the operating system under which it is running, then calls the appropriate OS-specific routine to process the global.

ZIBGSV* Routines

These OS-specific routines saves the global entries to the output device specified.

Technical Notes

MSM Write Protect The MSM operating system cannot test for write protection on tape, therefore, write enable the tape before processing.

Kernel Environment Assumed XBGSAVE assumes it's running under the KERNEL with appropriate variables set. If called from other than the Kernel environment, the user must ensure a defined environment prior to calling XBGSAVE.

File Name

A global saved to the Unix file will be named AAAAFFFFFF.JJJ.
"AAAA" is the global namespace.
"FFFFFF" is the six digit numeric facility code.
"JJJ" is the Julian date.

Device numbers

DSM - 47 for cartridge; 48 for 9-track
MSM - 51 - 54 for all devices

The tape format for MSM is in the %GS format.

Programmer Notes

Input:

In addition to the global to be saved, other parameters may be used to customize XBGSAVE. Only the global name parameter, XBGL, is required. For non-KERNEL applications, also specify DT for date, DUZ(2) for location and %ZOSF("OS") for operating system type.

Process:

1. Verify global to be saved exists.
2. Verify parameters.
3. Call OS-specific ZIBGS* routine.

Output:

Local variable XBFLG will contain the result of the call to XBGSAVE. If the save was successful, XBFLG will be 0 (zero). If the save was unsuccessful, local variable XBFLG(1) will have a narrative that can be displayed to the user indicating the cause of failure.

.

XBGSAVE Input Parameters

Name	Description/ Default/ Format/ Use
XBDT	Date of Global save. Default: NOW. Format: Enter date in FileMan format, leading zeros not required. Ex.: S XBDT=7991231 Use: Header dump comment.
XBE	Ending first-level numeric subscript Default: End of file. Format: Canonic number.
XBF	Beginning first-level numeric subscript, seed for $ORDER. Default: beginning of file. Format: Canonic number.
XBFLT	S= 1, saves as flat file. Default: Save as a subscripted global. Format: 1 or none. Ex.: S XBFLT=1 Use: If 1, just the data will be saved, in flat file format.
XBFN	Output file name. Default: "<ns><asufac>.<Julian date>"
XBGL	Global name to be saved. Default: **None. This is required.** Format: Global name with no “^”. Ex.: S XBGL=”ATAGLOB”.
XBIO	IO device parameter. Default: 51 - HFS for MSM and 47 - cartridge for DSM. Format: one to three digit number specifying device.
XBMED	Media used for output to global save. Default: If not defined, the user is asked to select the media. Format: “C” for cartridge tape “D” for diskette “F” for Unix file “T” for 9-track tape
XBNAR	Narrative for Operator display if help needed. Default: None.

	Format: Free text Ex.: S XBNAR=”APC Service Unit”.
	Use: Customizing HELP display. “This option saves the “_XBNAR_” “_XBGL_” transaction file to tape or diskette...”.
XBPAR	DSM IO parameter.
	Default: open parameter from DEVICE file for specified device.
	Format: “V” or undefined
	Use: for DSM operating systems only, ignored by a MSM routine.
XBQ	Y/N, to place file in uucp q.
	Default: "Y"
	Format: “Y” or “N”. Ex.: S XBQ=”N”
	Use: If “Y”, will place file in uucp q to send to Area Office.
XBQTO	'sendto' destination.
	Default: Area Office sysid in RPMS SITE file
XBTLE	Global dump comment.
	Default: DUZ(2) location name.
	Format: free text. Ex.: S XBTLE=”APC’s October trans”
	Use: Header dump comment for off-line media. XBTLE will be appended with DUZ(2) name for header dump comment.
XBUF	Full Path name for file (MSM only).
	Default: /user/spool/uucppublic for UNIX, C:\EXPORT for DOS.

Sample Set-up of Routine Call

To save a global to an operator-specified output device:

```
S XBGL=”gloname”,XBTLE=”T&A for FY87"  D ^XBGSAVE
```

To save a global to a MSM cartridge:

```
S XBGL=”gloname”,XBMED=”C”   D ^XBGSAVE
```

To save a global to a DSM cartridge:

```
S XBGL=”gloname”,XBMED=”C”   D ^XBGSAVE
```

Error Codes

Various types of errors are produced:

Parameter errors
Operator cancellation errors
I/O errors
Tape test errors

If errors are detected, XBFLG is set to “-1", the error narrative is stored in XBFLG(1) and XBGSAVE returns to the calling program WITHOUT SAVING THE FILE.

If no errors are detected, XBFLG is set to “0".

Below are some of the possible error messages returned:

"Abort at drive select"
"Device Not Available During Tape Testing"
"Facility Number 'DUZ(2)' is not defined"
"Job Aborted by Operator during Tape Test"
"Job Aborted by Operator During Floppy Mount"
"Job Terminated by Operator at Device Select"
"Job Terminated By Operator at Mount Message"
"Job Aborted by Operator During Tape Mount"
"Job Aborted by Operator During Tape Test"
"Job Aborted, Tape not Ready"
"Job Aborted by Operator During Floppy Mount"
"Media Type '"_XBMED_"' is incorrect"
"Operating system is not 'MSM' or 'DSM'"
"Queue of File to uucp Failed"
"Tape not rewound"
"Tape Test Failed During Testing"
"The variable 'XBGL' must contain the name of the global you wish to save."
"The ^%ZOSF(""OS"") node does not exist"
"Transaction File does not exist"
XBERRMSG_" Not Available"
XBFLG(1)," After 6 Minutes"
XBMSG_" Drive Not Available"

Indian Health Service
Office of Information Resource Management

Miscellaneous Tools for Programmers

June 27, 1996

ScreenMan Help - Crib Sheet

Cursor Movement

Move right one character	<Right>
Move left one character	<Left>
Move right one word	<Ctrl-L> or <PF1><Space>
Move left one word	<Ctrl-J>
Move to right of window	<PF1><Right>
Move to left of window	<PF1><Left>
Move to end of field	<PF1><PF1><Right>
Move to beginning of field	<PF1><PF1><Left>

Modes

Insert/Replace toggle	<PF3>
Zoom (invoke multiline editor)	<PF1>Z

Deletions

Character under cursor	<PF2> or <Delete>
Character left of cursor	<Backspace>
From cursor to end of word	<Ctrl-W>
From cursor to end of field	<PF1><PF2>
Toggle null/last edit/default	<PF1>D or <Ctrl-U>

Macro Movement

Field below	<Down>	Next page	<PF1><Down> or <PageDown>
Field above	<Up>	Previous page	<PF1><Up> or <PageUp>
Field to right	<Tab>	Next block	<PF1><PF4>
Field to left	<PF4>	Jump to a field	^caption
Pre-defined order	<Return>	Go to Command Line	^

Go into multiple or word processing field	<Return>

Command Line Options (Enter '^' at any field to jump to the command line.)

Command	**Shortcut**	**Description**
-------------	-----------	---------------
EXIT	see below	Exit form (asks whether changes should be saved)
CLOSE	<PF1>C	Close window and return to previous level
SAVE	<PF1>S	Save changes
NEXT PAGE	<PF1><Down>	Go to next page
REFRESH	<PF1>R	Repaint screen

<u>Other Shortcut Keys</u>

Exit form and save changes	<PF1>E
Quit form without saving changes	<PF1>Q
Invoke Record Selection Page	<PF1>L

Full Screen Editor - Crib Sheet

Summary of Key Sequences

Navigation

Incremental movement	Arrow keys
One word left and right	<Ctrl-J> and <Ctrl-L>
Next tab stop to the right	<Tab>
Jump left and right	<PF1><Left> and <PF1><Right>
Beginning and end of line	<PF1><PF1><Left> and <PF1><PF1><Right>
Screen up or down	<PF1><Up> and <PF1><Down>
	or: <PrevScr> and <NextScr>
	or: <PageUp> and <PageDown>
Top or bottom of document	<PF1>T and <PF1>B
Go to a specific location	<PF1>G

Exiting/Saving

Exit and save text	<PF1>E
Quit without saving	<PF1>Q
Exit, save, and switch editors	<PF1>A
Save without exiting	<PF1>S

Deleting

Character before cursor	<Backspace>
Character at cursor	<PF4> or <Remove> or <Delete>
From cursor to end of word	<Ctrl-W>
From cursor to end of line	<PF1><PF2>
Entire line	<PF1>D

Settings/Modes

Wrap/nowrap mode toggle	<PF2>
Insert/replace mode toggle	<PF3>
Set/clear tab stop	<PF1><Tab>
Set left margin	<PF1>,
Set right margin	<PF1>.
Status line toggle	<PF1>?

Formatting

Join current line to next line	<PF1>J
Reformat paragraph	<PF1>R

Finding

Find text	<PF1>F or <Find>
Find next occurence of text	<PF1>N
Find/Replace text	<PF1>P

Cutting/Copying/Pasting

Select (Mark) text	<PF1>M at beginning and end of text
Deselect (Unmark) text	<PF1><PF1>M
Delete selected text	<Delete> or <Backspace> on selected text
Cut and save to buffer	<PF1>X on selected text
Copy and save to buffer	<PF1>C on selected text
Paste from buffer	<PF1>V
Move text to another location	<PF1>X at new location
Copy text to another location	<PF1>C at new location

Fileman Sorting Crib Sheet

SORT OPTIONS

Formatting Codes:	Examples:	Explanations:
C-Column Assignment	SORT BY: Sex;C30	Print Sex sub-header in column 30
S-Skip Lines	SORT BY: Sex;S2	Skip 2 lines before printing the next sex sub-header
L-Left Justify	SORT BY: Nursing Home;L10	Print only the first 10 characters of the Nursing Home as the sub-header
" "-Print Your Header	SORT BY: Sex;"Gender"	Prints Gender: as a sub-header rather than Sex
@-Supress Sub-Header	SORT BY: @SEX	Sorts by the selected field but suppresses the Sub-Header

Sort Functions

!-Ranking Numbers	SORT BY: !Nursing Home	Items printed under Nursing Home sub-header will have ranking numbers
+-Sub Totals	SORT BY: +Nursing Home	All print fields with !,&,+,or# will be sub-totaled at each new sorted by value
#-Form Feed	SORT BY: #Nursing Home	A form feed will be generated at each new sorted by value
- Reverse Order	SORT BY: -Days Of Care	Will reverse order of print from lowest-highest to highest-lowest order
'-Select Entries	SORT BY: 'Placement Date	Selects items only, rather than selects and sorts

Special Features:

@ at START WITH prompt	SORT BY: Days Absent START WITH DAYS ABSENT: @	Prints all entries with a value in the days of care field first followed by null values for that field
@ at the START WITH and GO TO prompt	SORT BY: Days Absent START WITH DAYS ABSENT: @ GO TO DAYS ABSENT: @	Prints only entries with null values for in the days of care field

Templates

] forces FileMan to offer a template prompt	SORT BY:] FIRST PRINT FIELD:]	Forces FileMan to offer you a template
[used to call a template	SORT BY: [VINCE FIRST PRINT FIELD: [VINCE	Calls a previously created template sort or print template named Vince
[? will show all templates available to the user	SORT BY: [? FIRST PRINT FIELD: [?	Shows all sort or print templates

^ inserts	THEN PRINT FIELD:SEX//^SSN
	THEN PRINT FIELD: SEX//
		Inserts a field before another field

@ deletes	THEN PRINT FIELD: SEX//@	Deletes a field

Note: Any changes must be re-saved.

BOOLEAN LOGIC

= Equal	SORT BY: NAME// SEX="MALE"	Equal finds exact matches
> Greater Than	SORT BY: NAME// DAYS ABSENT>3	Finds all entries with DAYS ABSENT values greater than three
< Less Than	SORT BY: NAME// DAYS ABSENT<3	Finds all entries with DAYS ABSENT values less than three

[Contains	SORT BY: NAME// NAME["AR"	Finds all names having the letters 'AR' in them
] Follows	SORT BY: NAME// NAME]"ST"	Finds all names starting with 'ST' to the end of the alphabet (STEVENS to TURNER)
! OR	SORT BY: NAME// NAME["AR"!(SEX=”MALE”)	Finds entries with 'AR' in the name or the sex is male
& AND	SORT BY: NAME// NAME["AR'&(SEX="MALE")	Finds entries with 'AR' in the name and the sex is male
' Apostrophe	SORT BY: NAME// NAME'["AR"	Negates any condition; Finds all entries with-out AR in the name

Fileman Printing - Crib Sheet

PRINT OPTIONS

Formatting Codes:	**Examples:**	**Explanations:**
C-Column Assignment	FIRST PRINT FIELD: Name;C10	Print Name in column 10
	FIRST PRINT FIELD: Name;C-30	Print Name 30 columns from the right margin
S-Skip Lines	FIRST PRINT FIELD: Name;S1	Skip 1 line before printing the next name
L-Left Justify	FIRST PRINT FIELD: Name;L8	Print only the first 8 characters of the name
R-Right Justify	FIRST PRINT FIELD: Name;R30	Right justify the columns from the end of the last value plus 2 column spaces
W-Word Wrap	FIRST PRINT FIELD: Text;W20	Wrap after 20 columns of text but will not split words
D-Decimal Points	FIRST PRINT FIELD: Cost;D1	Use only one decimal place
N-No Repeat	FIRST PRINT FIELD: Nursing;N	Will not repeat consecutive occurrence of the same name
Y-Start at Row	FIRST PRINT FIELD: Name;Y10	Start printing 10 rows from the top
	FIRST PRINT FIELD: Name;Y-10	Start printing 10 rows from the bottom
@-Suppress Heading	HEADING: CONTRA..Replace @	Suppresses the entire Heading

X-Suppress Spacing	FIRST PRINT FIELD: .01 THEN PRINT FIELD : SSN;X	Suppress spacing between the name and the SSN
" "-Print Your Header	FIRST PRINT FIELD: Sex;"S"	Prints S as a column header rather than Sex
_ Concatenate (Join)	FIRST PRINT FIELD: City_", "_State	Joins field values with literals or other fields
other :-Forward Pointing	FIRST PRINT FIELD: Nursing Home: THEN PRINT CONTRACT NURSING HOME FIELD:	Follows the Nursing Home pointer field to the Nursing Home file

Arithmetic Operators:

!-Counts Any Field	FIRST PRINT FIELD: !Sex	Counts the entries that have values in the Sex field
&-Totals Numerics	FIRST PRINT FIELD: &Cost	Totals numeric fields
+-Totals,Count&Mean	FIRST PRINT FIELD: +Cost	Totals and Counts fields and provides a mean value
#-Totals,Count,Mean Minimum,Maximum, & Standard Deviation	FIRST PRINT FIELD: #COST	Totals and Counts fields and provides a minimum value and a maximum value found with the deviation

Binary Operators:

+ Addition	FIRST PRINT FIELD: SPER DIEM RATE+10	Add 10 to the Skilled Per Diem Rate
- Subtract	FIRST PRINT FIELD: SPER DIEM RATE-10	Subtract 10 from the Skilled Diem Rate

* Multiply	FIRST PRINT FIELD: SPER DIEM RATE*30	Multiply Skilled Per Diem Rate by 30
/ Divide	FIRST PRINT FIELD: NURSING HOME COST/DAYS OF CARE	Divide the Total Nursing Home Cost by the Total Days of Care
\ Integer Division	FIRST PRINT FIELD: TODAY-DATE OF BIRTH\365.25	Divide the Age by 365.25 leaving off all remainders

VI - UNIX Editor - Crib Sheet

Invoking vi

vi	invoke vi, load a new file
vi *file*	invoke vi, load a specified file
vi +n *file*	..... at line *n*
vi + *file*	..... at end of *file*
vi + */s file*	..... at *s*(string)
vi -r *file*	recover *file*
view *file*	read only mode

Quitting/Saving vi

:wq	write (save) and quit
:w	write (save) no quit
:q	quit without saving (only if no changes)
:x	write (save) (only if changes made)
:q!	quit without saving
ZZ	write (save) file and quit vi

Scrolling

<ctrl>f	forward one screen
<ctrl>b	back one screen
<ctrl>d	down half screen
<ctrl>u	up half screen

Line Positioning

H	first line on screen
L	last line on screen
M	middle lline on screen
G	go to end of file
*n*G	go to line *n*
j or ↓	down one line (same position)
k or ↑	up one line (same position)
<ret>	down one line

Character/Word Positioning

h or →	forward one character
l or ←	back one character
0	beginning of line
$	end of line
<spacebar>	forward one character
w	forward to next word
e	... end of next word
b	back to previous word

Insert & Replace

a	append after cursor
A	append at end of line
i	insert before cursor
I	insert at end of line
o	open line below
O	open line above

Deleting

dd	delete current line
*n*dd	delete *n* lines
D	... from cursor to line end
d}	... rest of paragraph
d0	... to left margin
dw	delete word
x	delete current character

Search/Replace

/*string*	search forward for *string*
?*string*	search backwards
n	repeat last search
:*#,#s*/*s1*/*s2*	replace first occurrence of *s1* on each line in range (#-#) w/ *s2*
:*#,#*s/*s1/s2/*gc	...but global and confirm

Changing/Undoing

r*x*	replace character w/ *x*
R	replace characters
cc	change line
*n*cc	change next *n* lines
cw	change word
C	Change to end of line
u	undo last change
U	undo current line changes
yy	yank line to buffer
*n*yy	... *n* lines to buffer
p	put lines at cursor
P	... before cursor
c)	change sentence starting at cursor to new text
J	join next line down to line with cursor

Other Helpful Commands

<ctrl> g	displays line status
<ctrl> l	redraw screen
:se nu	set line numbering
:se nonu	unset ...
:!*command*	execute shell command
:#	go to line #
:w *file*	write to *file*
:r!spell	spell check
:n,kw file2	write lines n-k into another file
:n,kw >> file2	append lines n-k to another file

NOTE: To obtain any command that is preceded by a ":", you press ESC key first then type in the colon.

Some Useful Unix Commands or Unix Scripts

gflist *filename*	provides a listing of globals contained in *filename*
rflist *filename*	provides a listing of routines contained in *filename*
sendto -l dev:dev cmbsyb *filename*	use this to send files to SRCB. This will place your files in the /usr/spool/uucppublic/VERIFY directory on cmbsyb.
sendto	send file to a certain location. Example: sendto tuclcl *filename*
getfrom	script to get files from some system. Ex.: getfrom -i -r DIST/96cert cmbsyb aum_9610.tar.gz *This example would immediately get the aum* file from cmbsyb which resides in the /usr/spool/uucppublic/DIST/96cert subdirectory*
gzip -v -9 *filename*	compresses *filename*, filename will have a .gz appended to the end (-9 is the best compression but is the slowest speed - the default is -6)
gunzip -v *filename*	decompresses *filename*
man *command*	to get manual help on *command*
uname -a	will give the version of UNIX install
mu	a UNIX function that will normally take the user to the MUMPS login
pub	a UNIX function to cd (change directories) to /usr/spool/uucppublic
u	... to /usr/lib/uucp
m	... to /usr/mumps
lpstat -t	will display status of the lp (printer) spooler

cancel *device name* -number	will free up hung printer jobs

FTP is another way of transferring files from system to system. Please see ftp help for further instructions on its use. Non-IHS facilities must use FTP instead of any of the unix scripts/commands described above.

Indian Health Service
Office of Information Resource Management

PCC Function Calls

Function Calls Used to Retrieve Data Items from PCC/Patient Files

June 27, 1996

Introduction to PCC Function Calls

This manual describes the many function calls available to developers for retrieving data from PCC or the Patient file. The PCC files include the Visit file and all “V” files. If you know the visit ien, you may use any of these calls to retrieve a particular PCC data item. To retrieve patient demographic information, you only need to know the patient DFN. This document describes published entry points from both the AUPN - IHS DICTIONARIES (PATIENT) package and from the APCL - PCC MANAGEMENT REPORTS package.

If you have any comments or would like to make additions to this manual, please contact one of the following people:

Lori Butcher
Information Systems Division
OHPRD
7900 S. J. Stock Rd.
Tucson, AZ 85746-9352
520-295-2535

Toni Jarland
Information Systems Division
OHPRD
7900 S. J. Stock Rd.
Tucson, AZ 85746-9352
520-295-2533

George Huggins
Acting Director, DSD
OIRM
5300 Homestead Road NE
Albuquerque, NM 87110
505-837-4189

Patient Demographic Functions

Routine - AUPNPAT

Data for the following function calls come from the PATIENT file, file 9000001, and from the Medicare Eligible, Medicaid Eligible, and Private Insurance Eligible files.

SEX(p) **Returns SEX of patient p**

arguments

```
    p - patient ien (DFN)
```

examples

```
    W $$SEX^AUPNPAT(234)              => F
```

DOB(p,f) **Returns DATE OF BIRTH of patient p in format f**

arguments

```
    p - patient ien (DFN)
    f - optional format; if null, returns internal fileman format of DOB
       E - external written-out format (MAR 05, 1995)
```

examples

```
    W $$DOB^AUPNPAT(1234)             => 2950305
    W $$DOB^AUPNPAT(1234,“E”)         => MAR 05,1995
```

SSN(p) **Returns SSN of patient**

arguments

```
    p - patient ien (DFN)
```

examples

```
    W $$SSN^AUPNPAT(234)              => 123456789
```

AGE(p,d,f) **Returns AGE of patient p on date d in format f**

arguments

```
    p - patient ien (DFN)
    d - optional date in internal fileman format; if null, will default to DT
    f - optional format; if null, returns age in years
        null - age in years
       R - age in readable format
```

examples

```
    W $$AGE^AUPNPAT(1234)             => 32
    W $$AGE^AUPNPAT(1234,“R”)         => 32 YRS
```

DOD(p,f) **Returns DATE OF DEATH of patient p in format f**

arguments

p - patient ien (DFN)
f - optional format; if null, returns internal fileman format of DOD
E - external written-out format (MAR 05, 1995)

examples

```
W $$DOD^AUPNPAT(1234)            => 2950305
W $$DOD^AUPNPAT(1234,“E”)        => MAR 05,1995
```

TRIBE(p,f) **Returns TRIBE OF MEMBERSHIP of patient p in format f**

arguments

p - patient ien (DFN)
f - optional format; if null, returns tribe code
I - internal format of tribe (tribe ien)
E - external written-out format of tribe
C - tribe code

examples

```
W $$TRIBE^AUPNPAT(1234,“I”)      => 31
W $$TRIBE^AUPNPAT(1234,“E”)      => CHOCTAW NATION OF OK
W $$TRIBE^AUPNPAT(1234,“C”)      => 031
```

COMMRES(p,f) **Returns COMMUNITY OF RESIDENCE of patient p in format f**

arguments

p - patient ien (DFN)
f - optional format; if null, returns STCTYCOM COMMUNITY code
I - internal format of COMMUNITY (community ien)
E - external written-out format of COMMUNITY
C - STCTYCOM code

examples

```
W $$COMMRES^AUPNPAT(1234,“I”)          => 31
W $$COMMRES^AUPNPAT(1234,“E”)          => PRINCESS BAY
W $$COMMRES^AUPNPAT(1234,“C”)          => 0210019
```

HRN(p,l,f) **Returns HEALTH RECORD NUMBER of patient p at location l in format f**

arguments

p - patient ien (DFN)
l - must be valid ien of location
f - optional, 2–HRN will have prefix of site abbreviation

examples

```
W $$HRN^AUPNPAT(1234,4585)             => 3456
W $$HRN^AUPNPAT(1234,4585,2)           => SE3456
```

ELIGSTAT(p,f) Returns ELIGIBILITY STATUS of patient p in format f

arguments

p - patient ien (DFN)
f - optional format; if null, returns internal format
I - internal format of eligibility status (set of codes)
E - external written-out format of eligibility status

examples

```
W $$ELIGSTAT^AUPNPAT(1234,“I”)          => D
W $$ELIGSTAT^AUPNPAT(1234,“E”)          => DIRECT ONLY
```

BEN(p,f) Returns CLASSIFICATION/BENEFICIARY of patient p in format f

arguments

p - patient ien (DFN)
f - optional format; if null, returns classification/beneficiary code
I - internal format of classification/beneficiary (pointer value)
E - external written-out format of classification/beneficiary
C - classification/beneficiary code

examples

```
W $$BEN^AUPNPAT(1234,“I”)          => 1
W $$BEN^AUPNPAT(1234,“E”)          => INDIAN/ALASKA NATIVE
W $$BEN^AUPNPAT(1234,“C”)          => 01
```

MCR(p,d) Returns 1 or 0: Is Patient p eligible for Medicare on date d?

arguments

p - patient ien (DFN)
d - required date in internal fileman format

examples

```
W $$MCR^AUPNPAT(1234,2950601)              => 1
Is patient 1234 eligible for Medicare on 6/1/95?  => yes
```

PI(p,d) Returns 1 or 0: Is Patient p eligible for private insurance on date d?

arguments

p - patient ien (DFN)
d - required date in internal fileman format

examples

```
W $$PI^AUPNPAT(1234,2950601)                          => 1
Is patient 1234 eligible for private insurance on 6/1/95?  => yes
```

MCD(p,d) Returns 1 or 0: Is Patient p eligible for Medicaid on date d?

arguments

p-patient ien (DFN)
d-required date in internal fileman format

examples

```
W $$MCD^AUPNPAT (1234,2950601)              =>1
Is patient 1234 eligible for Medicaid on 6/1/95?  =>yes
```

MCDPN(p,d,f) Returns Medicaid plan name for patient p on date d in format f

arguments

p - patient ien (DFN)
d - required date in internal fileman format
f - format, optional; if null, returns internal ien of insurer I

examples

```
W $$MCDPN^AUPNPAT(1234,2950601,“I”)  => 1
W $$MCDPN^AUPNPAT(1234,2950601,“E”)  => CARONDELET
```

PIN(p,d,f) Returns private insurance plan name for patient p on date d in format f

arguments

p - patient ien (DFN)
d - required date in internal fileman format
f - format, optional; if null, returns internal ien of insurer

examples

```
W $$PIN^AUPNPAT(1234,2950601,“I”)   => 1
W $$PIN^AUPNPAT(1234,2950601,“E”)   => BLUE CROSS/BLUE SHIELD
```

CDEATH(p,f) Returns CAUSE OF DEATH of patient p in format f

arguments

p - patient ien (DFN)
f - optional format; if null, returns Cause of Death ICD9 code
    I - internal format ICD9 ien
    E - external written-out format (ICD9 TEXT)
    C - ICD9 code

examples

```
W $$CDEATH^AUPNPAT(1234,“I”)   => 31
W $$CDEATH^AUPNPAT(1234,“E”)   => DIABETES MELLITUS
W $$CDEATH^AUPNPAT(1234,“C”)   => 250.00
```

ENC(p) Returns an encrypted patient identifier 12 bytes long. The entry-point DEC reverses the process and returns the decoded output in a 27-byte-long string.

arguments

p - patient ien (DFN)

examples

```
W $$ENC^AUPNPAT(1)           => V46332UMH763
```

DEC(p) **Reverses the process of ENC^AUPNPAT and returns the decoded output in a 27-byte-long string.**

arguments

p - patient ien (DFN)

examples

```
W $$DEC^AUPNPAT(V46332UMH763)  =>[THA,B__JAN 01,1933_0001]
```

Routine - AUPNPAT1

Data for the following functions comes from the PATIENT file, file 9000001.

BEN(p) **Returns Beneficiary/Non-Beneficiary Status**

arguments

p - patient ien (DFN)

examples

```
W $$BEN^AUPNPAT1(1)          =>1
                OUTPUT:
                              1 = yes
                              0 = no
                             -1 = no/old tribe or unable
```

Visit File Functions

Routine - APCLV

For each of the function calls in APCLV, the user is required to pass the visit ien as the first parameter. Data for the following functions come from the VISIT file, file 9000010.

VD(v,f) **Returns VISIT DATE for visit v in format f**

arguments

- v - visit ien
- f - optional format; if null, returns internal FileMan format of visit date
  - I - internal FileMan format
  - E - external written-out format (MAR 05, 1995)
  - S - slash format (03/05/95)

examples

```
W $$VD^APCLV(1234,“I”)        => 2950305
W $$VD^APCLV(1234,“E”)        => MAR 05,1995
W $$VD^APCLV(1234,“S”)        => 03/05/95
```

VDTM(v,f) **Returns VISIT DATE AND TIME for visit v in format f.**

arguments

- v - visit ien
- f - optional format; if null, returns internal FileMan format of visit date
  - I - internal FileMan format
  - S - slash format (03/05/95)
  - E - external written-out format (MAR 05, 1995)

examples

```
W $$VDTM^APCLV(1234,“I”)           => 2950305.1300
W $$VDTM^APCLV(1234,“E”)           => MAR 05,1995 1:00pm
W $$VDTM^APCLV(1234,“S”)           => 03/05/95 1:00pm
```

TIME(v,f) **Returns TIME of visit v in format f**

arguments

- v - visit ien
- f - optional format; if null, returns internal FileMan format of visit time
  - I - internal FileMan format
  - P - am/pm
  - E - external format

examples

```
W $$TIME^APCLV(1234, “I”)   => 1300
W $$TIME^APCLV(1234, “P”)  => 1:00pm
W $$TIME^APCLV(1234, “E”)  => 1:00
```

DOW(v,f) **Returns DAY OF WEEK for visit v in format f**

arguments

- v - visit ien
- f - optional format; if null, returns internal FileMan format of visit date
  - I - internal number of day 0-6 for Sunday to Saturday
  - E - external written-out format

examples

```
W $$DOW^APCLV(1234,“I”)    => 1
W $$DOW^APCLV(1234,“E”)    => Monday
```

TYPE(v,f) **Returns TYPE OF VISIT for visit v in format f**

arguments

- v - visit ien
- f - optional format; if null, returns internal FileMan format of type of visit
  - I - internal FileMan format
  - E - external format

examples

```
W $$TYPE^APCLV(1234,“I”)    => I
W $$TYPE^APCLV(1234,“E”)   => IHS
```

PATIENT(v,f) **Returns PATIENT for visit v in format f**

arguments

v - visit ien

f - optional format; if null, returns internal FileMan format of PATIENT entry (DFN)

I - internal FileMan format

E - external format

C - chart number (asufac_chart number)

examples

```
W $$PATIENT^APCLV(1234,“I”)   => 34
W $$PATIENT^APCLV(1234,“E”)  => JONES,LORI
W $$PATIENT^APCLV(1234,“C”)  => 000101000987
```

COMM(v,f) **Returns COMMUNITY OF RESIDENCE of the patient for visit v in format f**

arguments

v - visit ien

f - optional format; if null, returns internal FileMan format of community

I - internal FileMan format

E - external format

C - STCTYCOM code

examples

```
W $$COMM^APCLV(1234,“I”)       => 1234
W $$COMM^APCLV(1234,“E”)       => TUCSON
W $$COMM^APCLV(1234,“C”)       => 04023876
```

CHART(v) **Returns ASUFAC_HRN of patient for visit v**

arguments

v - visit ien

will return ASUFAC_HRN for the HRN at the location of the visit. If no chart exists at that location, the ASUFAC_HRN will be for DUZ(2). Otherwise, a blank will be returned. The HRN is left zero filled to 6 digits.

examples

```
W $$CHART^APCLV(1234)           =>000101003457
```

LOCENC(v,f) **Returns LOCATION OF ENCOUNTER for visit v in format f**

arguments

v - visit ien

f - optional format; if null, returns internal FileMan format of location entry (ien)

I - internal FileMan format

E - external format

C - IHS ASUFAC

examples

```
W $$LOCENC^APCLV(1234,“I”)   => 4585
W $$LOCENC^APCLV(1234,“E”)   => SELLS HOSPITAL
W $$LOCENC^APCLV(1234,“C”)   => 000101
```

SC(v,f) **Returns SERVICE CATEGORY for visit v in format f**

arguments

v - visit ien

f - optional format; if null, returns internal FileMan format of service category

I - internal FileMan format

E - external format

examples

```
W $$SC^APCLV(1234,“I”)  => A
W $$SC^APCLV(1234,“E”)  => AMBULATORY
```

CLINIC(v,f) **Returns CLINIC for visit v in format f**

arguments

v - visit ien

f - optional format; if null, returns internal FileMan format of clinic entry (ien)

I - internal FileMan format

E - external format

C - IHS Clinic Code

examples

```
W $$CLINIC^APCLV(1234,“I”)     => 1
W $$CLINIC^APCLV(1234,“E”)     => GENERAL
W $$CLINIC^APCLV(1234,“C”)     => 01
```

DLM(v,f) **Returns DATE LAST MODIFIED for visit v in format f**

arguments

v - visit ien

f - optional format; if null, returns internal FileMan format of visit date

I - internal FileMan format

E - external written-out format (MAR 05, 1995)

S - slash format (03/05/95)

examples

```
W $$DLM^APCLV(1234,“I”)          => 2950305
W $$DLM^APCLV(1234,“E”)          => MAR 05,1995
W $$DLM^APCLV(1234,“S”)          => 03/05/95
```

DVEX(v,f) **Returns DATE VISIT EXPORTED for visit v in format f**

arguments

v - visit ien

f - optional format; if null, returns internal FileMan format of visit date

I - internal FileMan format

E - external written-out format (MAR 05, 1995)

S - slash format (03/05/95)

examples

```
W $$DVEX^APCLV(1234,“I”)         => 2950305
W $$DVEX^APCLV(1234,“E”)         => MAR 05,1995
W $$DVEX^APCLV(1234,“S”)         => 03/05/95
```

APWI(v,f) **Returns APPT/WALK IN code from visit file for visit v in format f**

arguments

v - visit ien

f - optional format; if null, returns internal FileMan format of visit date

I - internal FileMan format

E - external written-out format (APPOINTMENT)

examples

```
W $$APWI^APCLV(1234,“I”)         => A
W $$APWI^APCLV(1234,“E”)         => APPOINTMENT
```

EM(v,f) **Returns EVALUATION AND MANAGEMENT CPT for visit v in format f**

arguments

v - visit ien

f - optional format; if null, returns internal FileMan format of CPT CODE (ien)

I - internal FileMan format

E - external format, description of code

C - CPT Code

examples

```
W $$EM^APCLV(1234,“I”)          => 99211
W $$EM^APCLV(1234,“E”)          => OFFICE VISIT, EXTENDED
W $$EM^APCLV(1234,“C”)          => 99211
```

CODT(v,f) **Returns CHECK OUT DATE AND TIME for visit v in format f**

arguments

v - visit ien

f - optional format; if null, returns internal FileMan format of visit date

I - internal FileMan format

E - external written-out format (MAR 05, 1995)

S - slash format (03/05/95)

examples

```
W $$CODT^APCLV(1234,“I”)        => 2950305
W $$CODT^APCLV(1234,“E”)        => MAR 05,1995
W $$CODT^APCLV(1234,“S”)        => 03/05/95
```

LS(v,f) **Returns LEVEL OF SERVICE for visit v in format f**

arguments

v - visit ien

f - optional format; if null, returns internal FileMan format of type of visit

I - internal FileMan format

E - external format

examples

```
W $$LS^APCLV(1234,“I”)   => B
W $$LS^APCLV(1234,“E”)  => BRIEF
```

APDT(v,f) **Returns APPT DATE AND TIME from visit file for visit v in format f**

arguments

v - visit ien

f - optional format; if null, returns internal FileMan format of visit date

I - internal FileMan format

E - external written-out format (MAR 05, 1995)

S - slash format (03/05/95)

examples

```
W $$APDT^APCLV(1234,“I”)        => 2950305
W $$APDT^APCLV(1234,“E”)        => MAR 05,1995
W $$APDT^APCLV(1234,“S”)        => 03/05/95
```

OUTSL(v) **Returns OUTSIDE LOCATION from visit file for visit v.**

arguments

v - visit ien

examples

```
W $$OUTSL^APCLV(1234)           => WALGREEN’S PHARMACY
```

Inpatient Information

The following data items are extracted from the V HOSPITALIZATION file and will be present only if the visit is a Hospitalization (service category = H).

Routine - APCLV

ADMSERV(v,f) **Returns ADMITTING SERVICE for visit v in format f**

arguments

v - visit ien, must be a hospitalization

f - optional format; if null, returns internal FileMan format of treating specialty entry (ien)

I - internal FileMan format

E - external format

C - IHS Code

examples

```
W $$ADMSERV^APCLV(1234,“I”)      => 3
W $$ADMSERV^APCLV(1234,“E”)      => PEDIATRICS
W $$ADMSERV^APCLV(1234,“C”)      => 15
```

DSCHSERV(v,f) **Returns DISCHARGE SERVICE for visit v in format f**

arguments

v - visit ien

f - optional format; if null, returns internal FileMan format of treating specialty entry (ien)

I - internal FileMan format

E - external format

C - IHS Code

examples

```
W $$DSCHSERV^APCLV(1234,“I”)     => 3
W $$DSCHSERV^APCLV(1234,“E”)     => PEDIATRICS
W $$DSCHSERV^APCLV(1234,“C”)     => 15
```

ADMTYPE(v,f) Returns ADMISSION TYPE for visit v in format f

arguments

v - visit ien, must be a hospitalization

f - optional format; if null, returns internal FileMan format of admission type entry (ien)

- I - internal FileMan format
- E - external format
- C - IHS Code

examples

```
W $$ADMTYPE^APCLV(1234,"I")        => 3
W $$ADMTYPE^APCLV(1234,"E")        => DIRECT
W $$ADMTYPE^APCLV(1234,"C")        => 02
```

DSCHTYPE(v,f) Returns DISCHARGE TYPE for visit v in format f

arguments

v - visit ien, must be hospitalization

f - optional format; if null, returns internal FileMan format of treating specialty entry (ien)

- I - internal FileMan format
- E - external format
- C - IHS Code

examples

```
W $$DSCHTYPE^APCLV(1234,"I")       => 2
W $$DSCHTYPE^APCLV(1234,"E")       => AWOL
W $$DSCHTYPE^APCLV(1234,"C")       => 03
```

DSCHDATE(v,f) Returns DISCHARGE DATE for visit v in format f

arguments

v - visit ien

f - optional format; if null, returns internal FileMan format of visit date

- I - internal FileMan format
- E - external written-out format (MAR 05, 1995)
- S - slash format (03/05/95)

examples

```
W $$DSCHDATE^APCLV(1234,"I")       => 2950305
W $$DSCHDATE^APCLV(1234,"E")       => MAR 05,1995
W $$DSCHDATE^APCLV(1234,"S")       => 03/05/95
```

CONSULTS(v) **Returns NUMBER OF CONSULTS for hospitalization for visit v**

arguments

v - visit ien

examples

```
W $$CONSULTS^APCLV(1234)     => 2
```

LOS(v) **Returns LOS for visit hospitalization visit v**

arguments

v - visit ien

examples

```
W $$LOS^APCLV(1234)      => 4
```

FACTX(v,f)Returns FACILITY TRANSFERRED TO for visit v in format f

arguments

v - visit ien, must be a hospitalization

f - optional format; if null, returns internal FileMan format of facility

- I - internal FileMan format
- E - external format
- C - IHS ASUFAC if an IHS facility

examples

```
W $$FACTX^APCLV(1234,“I”)       => DIC(4;123)
W $$FACTX^APCLV(1234,“E”)       => SELLS HOSPITAL
W $$FACTX^APCLV(1234,“C”)       => 000101
```

ATTPHY(v,f) **Returns ATTENDING PHYSICIAN for visit v in format f**

arguments

v - visit ien, must be a hospitalization

f - optional format; if null, returns internal FileMan format of provider entry (ien)

- I - internal FileMan format
- T - provider’s initials
- A - affiliation, internal format; e.g., 1
- B - affiliation, external format; e.g., IHS
- C - provider code; e.g., JDS
- D - provider’s discipline code; e.g., 01
- E - provider’s discipline, external format; e.g., PHYSICIAN
- F - provider’s discipline, internal format; e.g., 1 (ien of provider class)
- N - provider’s name; e.g., SMITH,JOHN DAVID
- O - provider’s affiliation_discipline; e.g., 101
- P - provider’s affiliation_discipline_code; e.g., 101JDS

examples

```
W $$ATTPHY^APCLV(1234,“P”)   => 101JDS
W $$ATTPHY^APCLV(1234,“E”)   => PHYSICIAN
```

ADMDX(v,f) **Returns ADMITTING DIAGNOSIS for visit v in format f**

arguments

v - visit ien, must be a hospitalization

f - optional format; if null, returns ICD Diagnosis code

I - internal FileMan format (ICD9 ien)

E - external of ICD9 code (HYPERTENSION)

C - ICD Code; e.g., 250.00

examples

```
W $$ADMDX^APCLV(1234,“I”)    => 2477
W $$ADMDX^APCLV(1234,“E”)    => HYPERTENSION
W $$ADMDX^APCLV(1234,“C”)    => 250.00
```

Visit Measurement Information

Routine - APCLV

The following data is extracted from the V Measurement file, file 9000010.01. One or more measurements may have been taken on any one visit; therefore, an array is passed back that contains all of the measurements taken on that visit.

Call to Pass Back an Array of Data from V Measurement

FORMAT: S E=$$PCCVF^APCLV(v,t,f)

An array (APCLV) will be passed back that contains a list of all measurements that were taken on visit v. The information passed back will be in format f. If an error was encountered, E will be passed back as one of the following error codes:

1 - no value for t, type of data wanted
2 - no value for f, format of data wanted
3 - no value for v, visit
4 - invalid visit ien passed
5 - invalid type of data passed in t

It is the caller's responsibility to kill array APCLV before and after the call to PCCVF^APCLV.

arguments

v - visit ien

t - is defined as the type of V File you want. For measurements, it must be the word MEASUREMENT.

f - is defined as the format of the data you want and it will depend on the V file being looked at. You may specify several numbers in the string: 7;8 or 1;3;7;8, for example. If you specify several items in f, they will be returned in the corresponding "^ piece of APCLV. In the case of 1;3;7;8, the value for 1 will be in the first piece, the value for 3 in the second piece, the value for 7 in the third piece, and the value for 8 in the fourth piece. Each value of f is described below.

1 - visit date and time in internal FileMan format; e.g., 2950402
2 - visit date and time in external FileMan format, e.g., 04/02/95
3 - internal IEN of patient; e.g., 234
4 - external name of patient; e.g., SMITH,JOHN
5 - internal value of measurement type; e.g., 4
6 - external value of measurement type; e.g., BLOOD PRESSURE

7 - coded value of measurement type; e.g., BP
8 - value of measurement; e.g., 120/90
9 - CPT Code for measurement type (if available)
09 - each of the above items in a "^" pieced string where each piece is equal to the number above; e.g.,
2950402^04/02/95^23^SMITH,JOHN^1^
WEIGHT^WT^175^^^^^^"

examples

```
S E= $$PCCVF^APCLV(1234,“MEASUREMENT”,“7”)
    Will return:
        APCLV(1)=BP
        APCLV(2)=WT
        APCLV(3)=HT
S E=PCCVF^APCLV(1234,“MEASUREMENT”,“7;8”)
    Will return:
        APCLV(1)=BP^120/80
        APCLV(2)=WT^125
        APCLV(3)=HT^65
S E=PCCVF^APCLV(1234,“MEASUREMENT”,“99”)
    Will return:
        APCLV(1)=2950402^04/02/95^123^SMITH,JOHN^4^BL
        OOD PRESSURE ^BP^120/90
        APCLV(2)=2950402^04/02/95^123^SMITH,JOHN^2^WE
        IGHT^WT^125
```

NMSR(v) **Returns NUMBER OF MEASUREMENTS taken on visit v**

arguments

v - visit ien

examples

```
W $$NMSR^APCLV(1234)  => 3
```

Visit Provider Information

Routine - APCLV

The following data is extracted from the V PROVIDER file, file 9000010.06. One or more providers may be listed for any one visit; therefore, an array is passed back that contains all of the providers for that visit.

Call to Pass Back an Array of Data from V Provider

FORMAT: S E=$$PCCVF^APCLV(v,t,f)

You will be passed back an array (APCLV) that contains a list of all providers that were seen on visit v. The information passed back will be in format f. If an error is encountered, E will be passed back as one of the following error codes:

1 - no value for t, type of data wanted
2 - no value for f, format of data wanted
3 - no value for v, visit
4 - invalid visit ien passed
5 - invalid type of data passed in t

It is the caller's responsibility to kill array APCLV before and after the call to PCCVF^APCLV.

arguments

v - visit ien

t - is defined as the type of V file you want. For providers, it must be the word PROVIDER.

f - is defined as the format of the data you want and it will depend on the V file being looked at. You may specify several numbers in the string: 7;8 or 1;3;7;8, for example. If you specify several items in f, they will be returned in the corresponding "^ piece of APCLV. In the case of 1;3;7;8, the value for 1 will be in the first piece, the value for 3 in the second piece, the value for 7 in the third piece, and the value for 8 in the fourth piece. Each value of f is described below.

1 - visit date and time in internal FileMan format; e.g., 2950402
2 - visit date and time in external FileMan format; e.g., 04/02/95
3 - internal ien of patient; e.g., 234
4 - external name of patient; e.g., SMITH,JOHN
5- internal value of provider entry; e.g., 4
6 - provider's initials; e.g., JDS
7 - affiliation, internal format; e.g., 1

8 - affiliation, external format; e.g., IHS
9 - provider code; e.g., JDS
10 - provider's discipline code; e.g., 01
11 - provider's discipline, external format; e.g., PHYSICIAN
12 - provider's discipline, internal format; e.g., 1 (ien of provider class)
13- provider's name; e.g., SMITH,JOHN DAVID
14 - provider's affiliation_discipline; e.g., 101
15 - provider's affiliation_discipline_code; e.g., 101JDS
16 - primary or secondary; e.g., P
17 - primary or secondary, external; e.g., PRIMARY
18 - operating/attending, internal; e.g., O
19 - operating/attending, external; e.g., Operating
99 - each of the above items in a "^" pieced string where each piece is equal to the number above; e.g., 2950402^04/02/95^23^JONES,MARY ^34^JS^1^IHS^JS^01^PHYSICIAN^1^SMITH,JOHN^101^101JS

examples

S E=PCCVF^APCLV(1234,"PROVIDER","7")
Will return:
APCLV(1)=1
APCLV(2)=1
APCLV(3)=1
S E=PCCVF^APCLV(1234,"PROVIDER","7;8")
Will return:
APCLV(1)=1^IHS
APCLV(2)=1^IHS
S E=PCCVF^APCLV(1234,"PROVIDER","99")
Will return:
APCLV(1)="2960125^1/25/96^50^SMITH,JIMALLEN^
618^LAB^3^TRIBAL^LAB^53^COMMUNITY HEALTH
REP.^47^BUTCHER,LORI
ANN^353^353LAB^P^PRIMARY^^"

PRIMPROV(v,f) Returns PRIMARY PROVIDER for visit v in format f

arguments

v - visit ien
f - optional format; if null, returns internal FileMan format of provider entry (ien)
I - internal FileMan format
T - provider's initials
A - affiliation, internal format; e.g., 1
B - affiliation, external format; e.g., IHS
C - provider code; e.g., JDS
D - provider's discipline code; e.g., 01

E - provider’s discipline, external format; e.g., PHYSICIAN
F - provider’s discipline, internal format; e.g., 1 (ien of provider class)
N - provider’s name; e.g., SMITH,JOHN DAVID
O - provider’s affiliation_discipline; e.g., 101
P - provider’s affiliation_discipline_code; e.g., 101JDS

examples

```
W $$PRIMPROV^APCLV(1234,“P”)        => 101JDS
W $$PRIMPROV^APCLV(1234,“E”)        => PHYSICIAN
```

SECPROV(v,f,n) **Returns the nth SECONDARY PROVIDER for visit v in format f**

arguments

v - visit ien
f - optional format; if null, returns internal FileMan format of provider entry (ien)
I - internal FileMan format
T - provider’s initials
A - affiliation, internal format; e.g., 1
B - affiliation, external format; e.g., IHS
C - provider code; e.g., JDS
D - provider’s discipline code; e.g., 01
E - provider’s discipline, external format; e.g., PHYSICIAN
F - provider’s discipline, internal format; e.g., 1 (ien of provider class)
N - provider’s name; e.g., SMITH,JOHN DAVID
O - provider’s affiliation_discipline; e.g., 101
P - provider’s affiliation_discipline_code; e.g., 101JDS

examples

```
W $$SECPROV^APCLV(1234,“P”)         => 101JDS
W $$SECPROV^APCLV(1234,“E”)         => PHYSICIAN
```

MIDWIFE(v) **Returns 1 if one of the providers is a midwife**

arguments

v - visit ien

examples

```
W $$MIDWIFE^APCLV(1234)       => 1
```

Purpose of Visit Information

Routine - APCLV

The following data is extracted from the V POV file, file 9000010.07. There may be one or more POVs for any one visit; therefore, an array is passed back that contains all of the POVs for that visit.

Call to Pass Back an Array of Data from V Pov

FORMAT: S E=$$PCCVF^APCLV(v,t,f)

The user will be passed back an array (APCLV) that contains a list of all POVs that were seen on visit v. The information passed backed will be in format f. If an error was encountered, E will be passed back as one of the following error codes:

1 - no value for t, type of data wanted
2 - no value for f, format of data wanted
3 - no value for v, visit
4 - invalid visit ien passed
5 - invalid type of data passed in t

It is the caller's responsibility to kill array APCLV before and after the call to PCCVF^APCLV.

arguments

v - visit ien

t - is defined as the type of V File you want; for POVs, it must be the word POV.

f - is defined as the format of the data you want and will depend on the V file being looked at. You may specify several numbers in the string: 7;8 or 1;3;7;8, for example. If you specify several items in f, they will be returned in the corresponding "^ piece of APCLV. In the case of 1;3;7;8, the value for 1 will be in the first piece, the value for 3 in the second piece, the value for 7 in the third piece, and the value for 8 in the fourth piece. Each value of f is described below.

1 - visit date and time in internal FileMan format; e.g., 2950402
2 - visit date and time in external FileMan format; e.g., 04/02/95
3 - internal IEN of patient; e.g., 234
4 - external name of patient; e.g., SMITH,JOHN
5 - internal value of POV entry; e.g., 4
6 - external ICD code text; e.g., HYPERTENSION

7 - ICD code; e.g., 250.00
8 - APC Recode; e.g., 020
9 - Cause of DX, internal; e.g., 1
10 - Cause of DX, external; e.g., Alcohol-related
11 - Cause of Injury; e.g., E900.2
12 - Place of Injury, internal; e.g., A
13 - Place of Injury, external; e.g., HOME-INSIDE
14 - Provider Narrative
15 - Primary/Secondary code, internal; e.g., P
16 - primary or secondary; e.g., PRIMARY
17 - Date of injury; e.g., 09/01/95
18 - Stage; e.g., 4
19 - Modifier, internal; e.g., 1
20 - Modifier, external; e.g., Rule Out
99 - each of the above items in a “^” pieced string where each piece is equal to the number above.

examples

S E=$$PCCVF^APCLV(72555,“POV”,99)
APCLV(1)=“2960125^1/25/96^50^SMITH,JAMES A^101531^DM UNCOMPL/T-II/NIDDM,NS UNCON^250.00^080^^^^^^DIABETES MELLITUS:PATIENT CARE - CHR^^^^^^”

PRIMPOV(v,f) Returns PRIMARY POV for visit v in format f

arguments

v - visit ien
f - optional format; if null, returns internal FileMan format of POV entry (ien)
I - internal FileMan format
E - ICD9 text
C - ICD9 code
A - APC RECODE
D - Cause of Dx (internal)
J - Cause of Injury code
P - Place of Injury (code)
N - Provider Narrative

examples

W $$PRIMPOV^APCLV(1234,“P”) => A
W $$PRIMPOV^APCLV(1234,“E”) => 250.00

SECPOV(v,f,n)Returns the nth SECONDARY POV for visit v in format f

arguments

v - visit ien
f - optional format; if null, returns internal FileMan format of POV entry (ien)

I - internal FileMan format
E - ICD9 text
C - ICD9 code
A - APC RECODE
D - Cause of Dx (internal)
J - Cause of Injury code
P - Place of Injury (code)
N - Provider Narrative

examples

```
W $$SECPOV^APCLV(1234,“P”)   => A
W $$SECPOV^APCLV(1234,“E”)   => 311
```

Miscellaneous PCC Calls

Information for these calls is taken from various V files.

NLAB(v) Returns NUMBER OF LABS done on visit v

arguments

v - visit ien

examples

W $$NLAB^APCLV(1234)   => 12

NRX(v)Returns NUMBER OF MEDICATIONS prescribed on visit v

arguments

v - visit ien

examples

W $$NRX^APCLV(1234)   => 3

ACTTIME(v) Returns ACTIVITY TIME for visit v

arguments

v - visit ien

examples

W $$ACTTIME^APCLV(1234)   => 10

TRAVTIME(v) Returns TRAVEL TIME for visit v

arguments

v - visit ien

examples

W $$TRAVTIME^APCLV(1234)   => 25

CHSCOST(v,f) Returns TOTAL COST from V CHS file

arguments

v - visit ien

examples

W $$CHSCOST^APCLV(1234)   => 12,000

PROC(v,f,n)Returns the nth PROCEDURE for visit v in format f

arguments

v - visit ien

f - optional format; if null, returns internal FileMan format of PROCEDURE entry (ien)

I - ien of ICD0 code

E - external of ICD0 code; e.g., appendectomy

C - ICD0 code; e.g., 44.10

P - CPT CODE

T - CPT internal ien
D - date of procedure/internal FileMan format
G - date of procedure/external format
F - infection Y/N
R - operating provider affl_disc_code
X- dx done for
N - provider’s narrative

examples

W $$PROC^APCLV(1234,“R”,1)=> 101JDS
W $$PROC^APCLV(1234,“E”,1)=> APPENDECTOMY

IMM(v,f,n) **Returns the nth IMMUNIZATION for visit v in format f**

arguments

v - visit ien
f - optional format; if null, returns internal FileMan format of IMMUNIZATION entry (ien)
I - ien of immunization entry
E - immunization external format - OPV
C - immunization code - 06
P - CPT CODE
S - series

examples

```
W $$IMM^APCLV(1234,“C”,1)     => 06
W $$IMM^APCLV(1234,“E”,1)     => OPV
```

DENT(v,f,n) **Returns the nth V DENTAL ENTRY for visit v in format f**

arguments

v - visit ien
f - optional format; if null, returns internal FileMan format of ADA CODE entry (ien)
I - ien of ADA entry
E - ADA CODE external format
C - ADA code - 0110

examples

```
W $$DENT^APCLV(1234,“C”,1)    =>   0110
```

Indian Health Service
Office of Information Resource Management

Modifications to Class I Package Components

June 27, 1996

Modifications to Data Dictionaries, Data Elements, and Routines in Class I Packages

Modifications to core data dictionaries, or dictionaries associated with Class I software must be restricted to adding new data elements and creating input and output templates to meet specific local needs. In order to assure the capability of installing new releases of the application packages, it is important that any local additions to the database be made in areas that will not conflict with elements contained in the nationally distributed database and should be coordinated with the DBA or be made a sub package so the changes can be reinstalled after a new dictionary release.

When adding new data elements to a data dictionary, the numbering conventions used for creating new files should be used for data elements. That is, a data element number should be entered that is in the numbering range of the Area making the change (or the sub-range of numbers assigned to a specific IHS site). The same numbering convention should be applied to global subscripts for local data add-ons to previously defined globals. Sub-files numbers should be assigned in an analogous fashion, putting the number after the decimal point.

When input or output templates are incorporated into the option file, the options names should be prefixed by the namespace followed by "Z" and the letter assigned to the Area making the addition. For example, a local option called LOG for the PS package made in the Tucson Area would have the option name "PSZSLOG". This will allow a site to readily differentiate between those options developed locally and those associated with the standard package, and will prevent collisions.

Any other types of local data modifications to the core and Class I packages are strongly discouraged. If local modifications are made to existing data elements in the data dictionaries, it will then be the responsibility of the site to maintain those modifications as new versions of the package are installed. Furthermore, should a data element be modified that is used in standard external reports, it will be the responsibility of the facility to ensure that the modification will not affect the accuracy or validity of the data contained in the report.

Class I software carries with it the guarantee that the RPMS Development Centers will support and maintain that software in conjunction with staff at the sites of installation. Local modifications of Class I routines invalidates that guarantee and transfers maintenance and support responsibility from the Development Centers to the modifying site. Debugging local variations of Class I software and assessing the "ripple effects" that such changes may have throughout the module are not the responsibility of the Development Centers.

Indian Health Service
Office of Information Resource Management

Principles of RPMS Data Base Design

June 27, 1996

Principles of RPMS Data Base Design

1. Objective

The primary objective of data base design for the RPMS is to integrate patient and cost data into a single ADP system, supporting the various information needs of specific disciplines and programs, while at the same time collecting and storing a core set of health and management data that cuts across disciplines and facilities.

This integration of data infers:

- That specific information is collected only once, by the program that generates the data, and made available to all users who need access to it.
- That like information is stored in a single, integrated file, rather than being scattered in a number of discipline-specific files.

An example of the above is the Patient Registration file. Patient data is collected only once, by Health Records staff, and made available to all users who need some form of patient data. The Contract Health Services (CHS) program, Third Party Billing program, Pharmacy system, Laboratory system, Dental system and others do not have to separately collect and maintain a basic patient file, which would result in redundant data collection efforts, redundant files, and errors and inconsistencies in patient data between programs.

Moreover, the data is maintained in a central registration file, not scattered in a number of dis-similar discipline specific files. If the latter were true, each program needing patient data would need to be aware of the existence of all such files and access all of the individual files to search for patient data. Furthermore, each time a new discipline-specific program was added to the system, all of the old programs needing patient data would need to be rewritten to access the new files.

The same principle applies to diagnoses, operations, laboratory procedures, etc. All of this data should be stored in a core set of centralized files, and not scattered in a number of discipline-specific or program-specific files. Cost accounting, third party billing and patient care programs should be able to obtain all diagnoses, operative procedures, basic visit data, etc., from the core set of files, and not have to access a myriad of different disciplinary files for relevant information. This integration requires each discipline or special program application to make sure that it's system stores core data into the appropriate integrated file. Additional, discipline-specific information, can be stored in discipline-defined files, as needed. The relationship of these sets of files is illustrated in the attached wheel diagram. (Figure 1). At the core is a standard set of IHS/VA tables shared by all systems. These include community tables, facility tables, discipline tables, tribal designations, etc.

Surrounding these tables is a core set of data files, shared by all disciplinary and programmatic groups as appropriate. Each group will feed information from its patient encounters into this core set of data.

The outer ring represents the discipline-specific program and management data, that is not in common with most other disciplines. For Contract Health Services, for instance, the outer ring would contain files on outstanding and completed CHS vouchers, commitment register balance, contract appropriation, etc.

In total, these various levels of files and tables represent the RPMS data dictionary. Figure 2 contains a listing of RPMS Patient Visit files which are part of the standard tables for IHS.

RPMS DICTIONARY RELATIONS

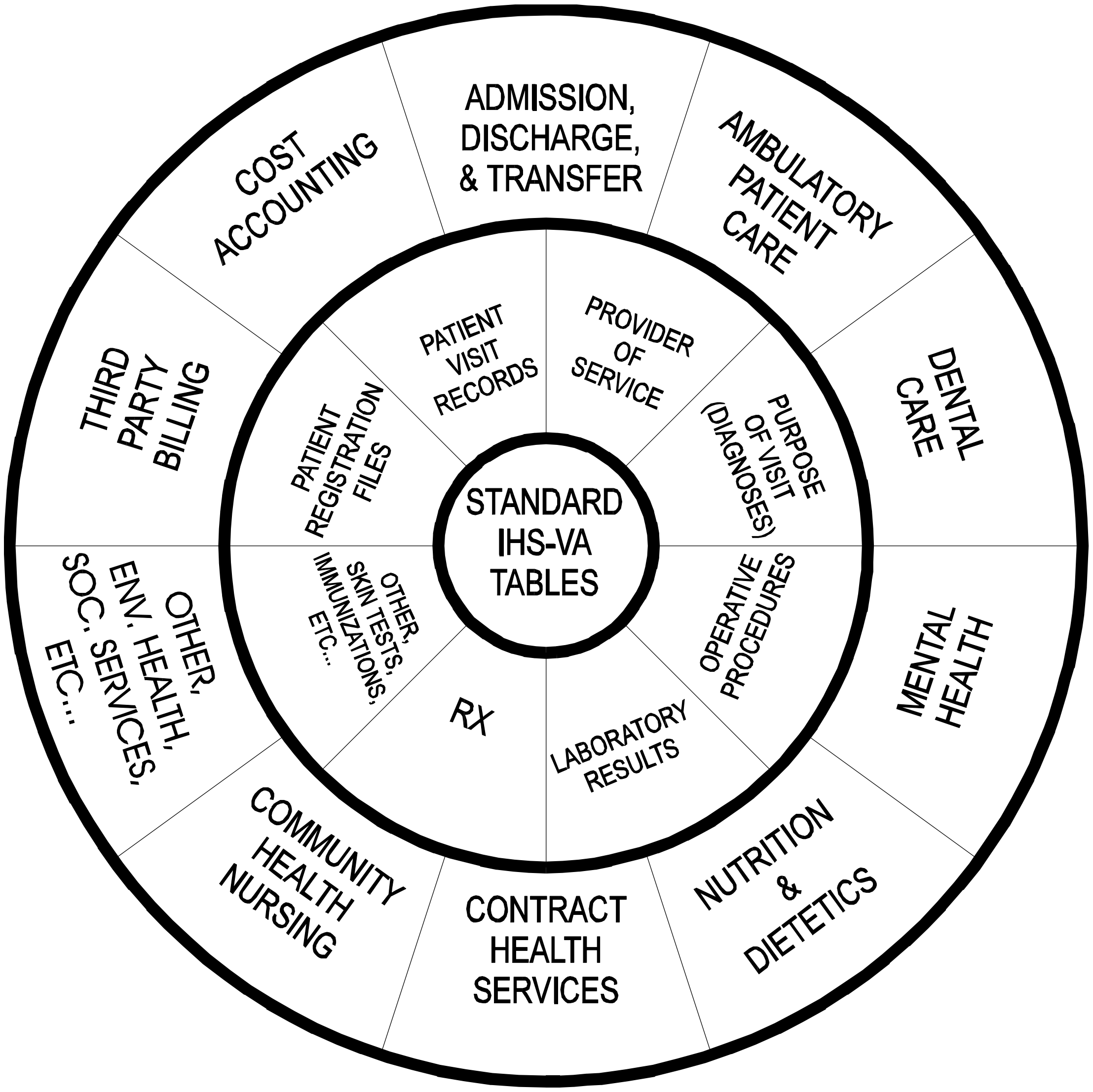

Figure 1

Inner Circle - Standard IHS/VA Tables
Middle Circle - RPMS Patient Care Files
Outer Circle - Discipline and Program Specific Files

List of RPMS Patient Visit Files

FILE	FILE NAME	GLOBAL NAME
9000010	Visit File	AUPNVSIT
9000010.01	V Measurement	AUPNVMSR
9000010.02	V Hospitalization (Direct)	AUPNVINP
9000010.03	V Contract Health Service	AUPNVCHS
9000010.04	V Eye Glass Prescription	AUPNVEYE
9000010.05	V Dental	AUPNVDEN
9000010.06	V Provider	AUPNVPRV
9000010.07	V POV (Purpose of Visit)	AUPNVPOV
9000010.08	V Procedure	AUPNVPRC
9000010.09	V Lab	AUPNVLAB
9000010.11	V Immunization	AUPNVIMM
9000010.12	V Skin Test	AUPNVSK
9000010.13	V Exam	AUPNVXAM
9000010.14	V Medication	AUPNVMED
9000010.15	V Treatment	AUPNVTRT
9000010.16	V Family Planning/Prenatal	AUPNVFMP
9000010.17	V Physical Therapy	AUPNVPT
9000010.18	V CPT	AUPNVCPT
9000010.19	V Activity Time	AUPNVTM
9000010.21	V Diagnostic Procedure Result	AUPNVDXP
9000010.22	V Radiology	AUPNVRAD
9000010.23	V Health Factors	AUPNVHF
9000010.701	V VA Mobile Visit Related	AUPNVMVR
9000010.702	V VA Mobile Visit Types	AUPNVMVT

FILE	FILE NAME	GLOBAL NAME
9000010.703	V VA Mobile Pres Actions	AUPNVMPA
9000010.704	V VA Mobile Refer for Outp	AUPNVMRO
9000010.705	V VA Mobile Specialty of Refer	AUPNVMSP
9000010.706	V VA Mobile Exams Ordered	AUPNVMEO

Figure 2

2. Minimum Required Patient Data

Each patient visit should generate data in three files at a minimum:

- the basic visit file
- a purpose of visit or diagnosis file
- the provider of service file.

Data will be generated into other RPMS core files as appropriate (i.e., immunizations, lab tests, physical exams, etc.)

The basic visit file (AUPNVSIT) contains the following information:

a. Visit date.
b. Posting date.
c. Type of visit (IHS, contract, tribal, other).
d. Patient name (pointer to Registration file).
e. Location of encounter (pointer to Location file).
f. Service category (ambulatory, hospitalization, in-hospital visit, dental, chart review, telephone, not found).
g. Clinic type (pointer to Clinic Type file).
h. Third party billed (Y or N, for Medicare/Medicaid eligible patients).

All other visit-related events stored in the data base (i.e., diagnosis, immunizations, lab tests) that were performed at this visit will be stored in the appropriate AUPNV... file, pointing to the visit in the AUPNVSIT file.

For each patient visit, a diagnosis or purpose of visit will be generated and stored in the AUPNVPOV file. The purpose of visit will be identified as an ICD9 diagnosis, with the ability to add appropriate user-defined qualifiers and narratives. There can be as many POV's identified for a visit as desired.

The information stored in the Purpose of Visit file is shown below:

a. Purpose of visit (pointer to ICD9 Table) - Required.
b. Date of visit (pointer to visit in AUPNVSIT) - Required.
c. Patient name (pointer to Registration file) - Required.
d. Narrative qualifiers (user-supplied narrative qualifiers, such as "severe, mild, not healing well, onset xx/xx/xx, etc.) (pointer to POV Narrative Qualifier file).
e. Disease stage (local use, and locally defined).
f. Modifier (doubtful, rule-out, probable, suspect, status post, resolved, follow-up).
g. Cause of diagnosis (hospital acquired, alcohol-related, battered child, employment-related).

h. First visit/revisit for this episode of problem - Required.
i. Cause of injury (pointer to ICD9 Cause of Injury table) - Required.
j. Place of accident - Required for cause of injury.
k. Primary/secondary diagnosis for this visit - Required.

At least one provider of service will be identified for each patient visit, and stored in the AUPNVPRV file. This file contains the following:

a. Provider (pointer to provider identified in the New Person file).
b. Date of visit (pointer to the visit in AUPNVSIT).
c. Patient name (pointer to Registration file).
d. Primary/secondary provider.
e. Type of provider (for hospital physician - attending or operating).
f. Vendor (pointer to the Vendor file, if appropriate).

3. Integration with VA Data Dictionary

The IHS is committed to integrating its data dictionary with the Veteran's Administration dictionary, to assure that systems obtained from the VA will run in a compatible mode with RPMS.

In general, where the VA has a dictionary for a specific application, the IHS will utilize that dictionary. Quite often, however, the IHS needs an additional sub-set of data not contained in the VA file. If there are only two or three additional fields needed, the IHS will generally add these fields to the VA file, utilizing a high-order set of field numbers that will never collide with VA fields.

If the number of differences between IHS data needs and VA data needs exceeds three or four fields, the IHS will generally define it's own file for the data set, which will point to the VA file for name and other basic information.

An example of this latter structure is shown in the patient file itself. The IHS has its own patient file (#9000001), which in turn points to the VA Patient file (#2), where the actual patient name, sex and date of birth are stored. These two files are DINUM so that the DFN is always the same for each patient.

The relationship of the RPMS patient visit files to other RPMS and VA files is shown in Figure 2.

4. Use of Standard Tables

Each developer should become familiar with the RPMS standard dictionary before initiating work on a new system, and should utilize tables and files already defined to the greatest extent possible.

The following are examples of some of the tables already developed:

File	File Name	Global
5	STATE	^DIC(5
7	PROVIDER CLASS	^DIC(7
10	RACE	^DIC(10
11	MARITAL STATUS	^DIC(11
12	OCCUPATION	^DIC(12
13	RELIGION	^DIC(13
25	TYPE OF DISCHARGE	^DIC(25
71	RADIOLOGY PROCEDURES	^RAMIS(71
80	ICD DIAGNOSIS	^ICD9(
80.1	ICD OPERATION/PROCEDURE	^ICD0(
80.2	DRG	^ICD(
80.3	MAJOR DIAGNOSTIC CATEGORY	^ICM(
9999999.01	USER CONVERSION	^AUTTUCV(
9999999.02	PATIENT RECORD DISPOSITION	^AUTTDIS(
9999999.03	TRIBE	^AUTTTRI(
9999999.04	IMM MANUFACTURER	^AUTTIMAN(
9999999.05	COMMUNITY	^AUTTCOM(
9999999.06	LOCATION	^AUTTLOC(
9999999.07	MEASUREMENT TYPE	^AUTTMSR(
9999999.08	RECODE ICD/APC	^AUTTRCD(
9999999.081	APC RECODE CATEGORY	^AUTTRCDC(
9999999.09	EDUCATION TOPICS	^AUTTEDT(
9999999.11	VENDOR	^AUTTVNDR(
9999999.12	RECODE INJURY	^AUTTRIJ(
9999999.13	LAB TEST	^AUTTLAB(
9999999.14	IMMUNIZATION	^AUTTIMM(
9999999.15	EXAM	^AUTTEXAM(
9999999.16	MEDICATION	^AUTTMED(
9999999.17	TREATMENT	^AUTTTRT(
9999999.18	INSURER	^AUTNINS(
9999999.19	SURVEILLANCE CODE	^AUTTSURC(
9999999.21	AREA	^AUTTAREA(
9999999.22	SERVICE UNIT	^AUTTSU(
9999999.23	COUNTY	^AUTTCTY(
9999999.24	CLINICAL REMINDER CODE	^AUTTCRC(
9999999.25	BENEFICIARY	^AUTTBEN(
9999999.26	SERVICE CATEGORY	^AUTTSC(
9999999.27	PROVIDER NARRATIVE	^AUTNPOV(
9999999.28	SKIN TEST	^AUTTSK(
9999999.29	DISTRICT	^AUTTDST(

9999999.31	ADA CODE	^AUTTADA(
9999999.32	MEDICARE SUFFIX	^AUTTMCS(
9999999.33	RAILROAD PREFIX	^AUTTRRP(
9999999.34	VENDOR TYPE	^AUTTVTYP(
9999999.35	QUANTUM CODE	^AUTTQUAN(
9999999.36	RELATIONSHIP	^AUTTRLSH(
9999999.37	U/R DENIAL REASONS	^AUTTREAS(
9999999.38	ERROR MESSAGES	^AUTTEMSG(
9999999.39	RPMS SITE	^AUTTSITE(
9999999.41	IMMUNIZATION LOT	^AUTTIML(
9999999.42	HOUSEHOLD STATUS	^AUTTHHS(
9999999.43	EMPLOYMENT STATUS	^AUTTEMP(
9999999.44	INCOME SOURCE	^AUTTINC(
9999999.45	RPMS APPLICATION PARAMETERS	^AUTNSYS(
9999999.46	PHYSICAL THERAPY	^AUTTPHTH(
9999999.47	RPMS RESERVATION	^AUTTRES(
9999999.48	BIC ELIGIBILITY	^AUTTBICE(
9999999.49	IHS TASK REPRINT	^AUTTZTSK(
9999999.51	APPROPRIATION NO.	^AUTTPRO(
9999999.52	ALLOWANCE NO.	^AUTTALLW(
9999999.53	NO LONGER USED	
9999999.54	BUDGET ACTIVITY	^AUTTBA(
9999999.55	SUB-ACTIVITY	^AUTTSA(
9999999.56	SUB-SUB-ACTIVITY	^AUTTSSA(
9999999.57	COMMON ACCOUNTING NUMBER	^AUTTCAN(
9999999.58	COST CENTER	^AUTTCCT(
9999999.59	OBJECT/SUB-OBJECT	^AUTTOBJC(
9999999.591	OBJECT CLASS GROUP	^AUTTOCG(
9999999.61	OBJECT CLASS CATEGORY	^AUTTOBCC(
9999999.62	FMS DEPARTMENT/PROGRAM	^AUTTPRG(
9999999.63	REFERENCE CODE	^AUTTDOCR(
9999999.64	HEALTH FACTORS	^AUTTHF(
9999999.65	COVERAGE TYPE	^AUTTPIC(
9999999.66	LOCATION CODE	^AUTTLCOD(
9999999.67	OBJECT CLASS REPORT GRP	^AUTTOCRG(
9999999.68	DIAGNOSTIC PROCEDURE RESULT	^AUTTDXPR(
9999999.69	ACCOUNTING POINT	^AUTTACPT(
9999999.71	IHS COMMUNICATIONS PARAMETERS	^AUTTTEL(
9999999.72	REVENUE CODES	^AUTTREVN(
9999999.73	NO LONGER USED	
9999999.74	CHA ICD RECODE TABLE	^AUTTCHA(
9999999.75	EMPLOYER	^AUTNEMPL(
9999999.76	TYPE OF BUSINESS	^AUTTTOB(
9999999.77	EMPLOYER GROUP INSURANCE	^AUTNEGRP(

9999999.78	SSN STATUS	^AUTTSSN(
9999999.79	GEOGRAPHICAL LOCATION	^AUTTGL(
9999999.81	SIZE OF SMALL BUSINESS	^AUTTSOB(
9999999.82	PERCENTILES	^AUTTPCT(

5. FileMan Compatible Files

All files utilized in RPMS applications will be FileMan compatible files, and FileMan protocols, (e.g., DIE or IX1^DIK), plus edits will be honored.

One of the main goals of the RPMS is to provide the tools to local facility staff to generate reports and make retrievals to satisfy their unique and constantly changing information needs. Historically, it was necessary to write a new program everytime new information needs arose. Under RPMS, the VA FileMan offers a flexible, sophisticated but easy-to-use capability for local report generation.

Using DIE to update the files ensures that all necessary housekeeping and cross-references updates are accomplished. This also allows users to add additional cross-indices to files to speed up retrieval time for specific applications in local use, without having to modify the update logic.

6. Review of Proposed Data Dictionary

Each developer will define data requirements early in the design layout.

For RPMS applications, it is recommended that this file description be submitted to the DBA early in the system design phase for review and possible suggestions on integration with other RPMS applications.

Relationship of RPMS Patient Visit Files to Other RPMS and VA Files

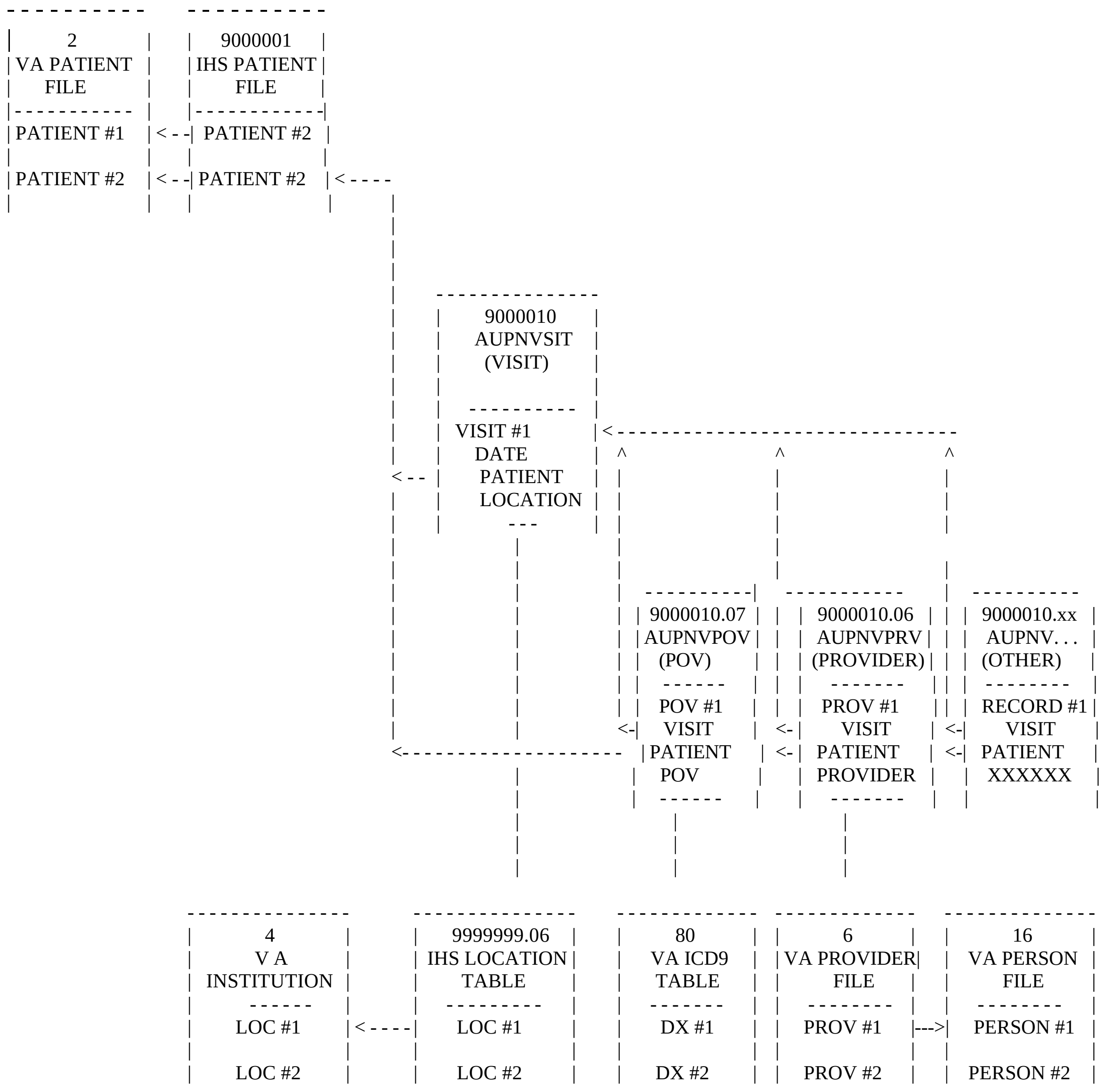

Figure 3

Indian Health Service
Office of Information Resource Management

Procedure for Building M Applications

June 27, 1996

Procedure for Building Application Packages

1. Procedure for Building M Packages

A designer of an RPMS application eventually will need to create an application package of all of the routines and globals to be sent to other computer sites. Package systems to be used by other members of the IHS must be completed in a standard way. The following rules must be followed, and if followed, will allow the recipients of the system to install the package without interfering with other applications running on their computer.

The Area ISC should assign a name and numberspace to any user that will be creating systems of any kind. Each Area has been assigned a range of numbers that will assure uniqueness and compatibility with other users of RPMS-developed software. This is done to protect systems within the Area even if there is no intent to send the system to other Areas.

For the RPMS sites with AVAINIT installed, there is a FileMan file #9.4, PACKAGE FILE, that is used for creating packages. The file is a typical FileMan dictionary with fields for entering data items that will become part of the package, i.e., routines, globals, options, menus and files. All parts supporting a package must begin with the assigned namespace. Globals should not be allowed to default to ^DIC or ^DIZ but instead should be assigned a name that begins with the namespace. Existing RPMS dictionaries should be used before creating new dictionaries that are redundant and which will force the users of the system to maintain data globals in duplicate. If the need exists to add fields to any RPMS data dictionary simply call the Area ISC. They will provide the procedures to follow.

One of the questions faced by the developer is whether to assume that the standard RPMS files and tables used in their package are available and already loaded on the recipient computer, or whether these should be included in the application package. Never include a file in the basic package that could already be installed, instead, create a subpackage that contains the support files. They can then be installed if needed, or ignored, if appropriate. In this way the recipient of the system has the option of not running all the INITS for the package and in doing so will not overlay or destroy any existing globals on the system.

2. Creating the Package

Update the PACKAGE file in the same UCI that contains the system. Never include data for any globals, unless absolutely sure that no data exists in these globals at any site that will be using the system. This will seldom be the case. Never include an RPMS dictionary in a package (one that falls in the 9NNNNNN.NN numberspace). These must be in a separate package along with yours, but with the option to load them or not. A suggestion is to name the packages with the namespace followed by a number, i.e., ADE1 and ADE2 for the IHS Dental package, where the ADE1 contains the Dental system and the ADE2 contains the RPMS support files. After updating the package file, create the system for shipment to other sites.

Note, the package file must exist in the same UCI as the system.

At the system prompt type:

```
D ^DIFROM
```

The system will display several questions. The first prompt requests a 2-4 character namespace; respond with the package namespace. The program responds with: Creating routine (namespace followed with INIT) and asks if it is OK?; respond with 'Y' and hit return.

The program prompts with: Selected Files:
Respond with the list of files contained in the package; remember that in almost all cases, include data dictionaries only, NO DATA. At the completion of the list, simply hit RETURN. This will start the INIT process, and the routine names will be displayed on the screen as they are created.

To create multiple packages, repeat the DIFROM process as many times as necessary.

Newly created routines can be saved to any device that is available on the computer, i.e., Floppy, Cartridge or Tape drive.

To initiate the Routine Save Utility, enter:

```
D ^%RS
```

(Save out the routines to be included in the package.
Remember to include the routines that were generated by the DIFROM process.
A wild card is allowed, i.e., namespace*.

The package should contain complete documentation of the system and a list of instructions that can be followed to install the package. These instructions must include the use of any menu processing and security key assignments necessary for the operation of the system. A complete list of menus, options, and security key assignments and how they are used must be included.

At the receiving site location, all that is required to install the system on the computer is to restore the routines into the UCI of choice, wherever the application should run. This UCI must have FileMan installed.

Each INIT has a sub-entry point that allows for the bypassing of any preinit execution.

This entry point is labeled EN, and entering D EN^XXXX will execute the INIT at this point. This is very helpful but the installer must understand what he is bypassing, and what the preinit was intended to accomplish. The preinits that are contained in the RPMS packages, for example, delete the old dictionary, and allow the installer to Kill off templates and or globals for the building of the new dictionary. This is the only mechanism for deleting fields in a dictionary after it has been installed.

To initiate the Routine Restore Utility, enter:

```
D ^%RR
```

After the routines are restored, the system is ready to install. Follow the instructions provided with the system. Remember after completing the installation process, delete the routines that were created from the INIT process. Be careful not to delete routines that were created from template generations. In general, it is safe to delete all routines that are constructed with the namespace followed by I*

Any globals that are necessary to support the package should be saved as a separate file with the package.

Indian Health Service
Office of Information Resource Management

RPMS Software Certification Policy and Guidelines

June 27, 1996

RPMS Software Certification Policy and Guidelines

1. Purpose

The purpose of this document is to provide policy and guidelines to be followed in the certification of RPMS Software. The purpose of certification is to ensure the functional soundness and technical correctness of RPMS software and documentation.

2. Overview

2.1 Requirement for Certification All IHS RPMS programs and packages which are to be distributed nationally throughout the IHS must first be certified by the SRCB, DSM, OIRM.

2.2 Certification Process Components The certification process consists of four components:

2.2.1 Technical Verification This will be performed by a member of the SRCB to ensure that the system conforms to RPMS Programming Standards and Conventions.

2.2.2 Functional Verification This will be accomplished by the appropriate Professional Specialty Group (PSG), or designated project leader, in conjunction with the alpha/beta test facilities to verify that the application conforms to system requirements, operates correctly and, in general, does what the PSG has specified. Limited functional verification will be done in SRCB.

2.2.3 Design Verification This will be performed by the DBA and will assure that the packages are integrated with other RPMS dictionaries in accordance with the Principles of Data Base Design as outlined elsewhere in this handbook.

2.2.4 Documentation Verification This will be accomplished by SRCB in conjunction with the alpha/beta test phase to ensure conformance to the RPMS Documentation Standards as outlined in Appendix F.

2.3 Non-RPMS Packages If an RPMS package conforms with the first two requirements but not the third, it will be certified for IHS distribution as a Class II package as defined in Appendix J.

2.4 **Certified RPMS Packages** If a package satisfies all four components, the system will be certified as a Class I system as defined in Appendix J.

3. RPMS Software Development Process

This section defines the procedures for RPMS software development and certification. The steps involved in producing RPMS software are:

3.1 **Functional Requirements/Design Phase** Members of the PSG conceptualize the functionality and specify the requirements for the system in close working relationship with the prototype developers. The PSG addresses the question, "What do we want this system to do?" In the absence of a PSG, the developer may work with interested parties to conceptualize the system with the approval of the Director of the Development Center that takes responsibility for the application. The Director, DSD, will assign responsibilities for the system development to a Development Center based on availability of resources and interest of the center. The development center may further suballocate portions of the development to other interested groups if appropriate.

3.2 **Prototype/Development Phase** The developers begin the prototype development process by addressing the question "How do we implement (technically) the requirements specified by the PSG?" Using rapid prototyping, the developers work closely with the PSG members to create the software. Design and development of prototype software will be in consonance with the methods and techniques contained in the SAC. The developer will prepare appropriate technical documentation of the system, and will work with the PSG in developing a user manual.

3.3 **Testing** The prototype package is tested comprehensively at multiple sites. The question addressed during testing is "Does the package function properly in all operating system environments and RPMS configurations (clinics, hospitals, multi-division, etc.)?" At least one test site will be co-located with a PSG member to ensure that requested functionality is present and meets the PSG’s preliminary requirements. Reevaluation or redesign of prototype requirements may be instituted at any point during the testing process.

When satisfied with the results of testing, the PSG will provide formal endorsement of the functional requirements of the package prior to beginning the final verification process. This will take the form of an endorsement letter that must accompany the package through final verification and release. The testing will be divided into two phases, alpha and beta.

3.4 **Alpha Testing** During the first phase, or the alpha test, the developer and PSG will actively participate in making necessary changes as defined by the PSG. The developer will execute a %Index and SAC Checker and will correct all items noted by these utilities. Once the package has stabilized and been approved by the PSG, the developer will notify the SRCB that the system is ready for the second test phase, or the beta test. Preliminary verification can be in parallel with the final stages of the alpha test, or at the initiation of the beta test phase.

3.5 Preliminary Technical Verification The Chief, SRCB, will assign a verifier to the package to complete the Preliminary Technical Verification (PTV). The PTV should be completed within 30 days from receipt of the system. The developer will forward the package files and all accompanying documentation to the appropriate SRCB directory. Upon completion of the PTV, the SRCB will coordinate any changes required with the developer prior to or during the initiation of the beta test.

3.6 Beta Testing

The PSG, with the assistance of the developer and/or Director, DSD, will select at least two (2), but no more than five (5), beta test sites. At least one (1) of the beta test sites must be in an area outside of the developer’s area. The developer will provide the SRCB with the test site information including site names and points of contact. The SRCB will coordinate the beta testing phase and prepare beta test agreements.

The PSG, the SRCB and the developer will actively participate in the beta test, making changes as problems are identified. When the package has remained stable for a minimum of 30 days and meets the requirements of the PSG it is ready for final certification. The PSG will notify the SRCB using the formal endorsement form that the system has met functional requirements and is ready for release. In addition, the PSG/beta test site will complete the beta test checklist and return it to the SRCB.

3.7 Final Verification

Functional and technical verification of the package is the last step in the development process. Verification addresses the question, "Does this software and documentation meet the functional and technical requirements necessary to be successfully implemented in all IHS facilities, regardless of size or complexity?". The end result of the verification process will be:

a) release of the package as submitted by the developer;

b) release of the package with exemptions annotated to be resolved in the next release; or

c) returning the package for rewrite or redesign.

3.8 Future Modifications

The PSG periodically will define changes desired in the certified system. These will be made by the developer and incorporated into a new release version of the system.

3.9 New Version Certification

The new release will be sent to the SRCB for re-certification, where it will follow the same procedures as a system in Beta testing.

3.10 Patches

Patches to distributed certified software will be made through the IHS Patch System. Please refer to the chapter in this volume on that subject.

4. Technical Verification Procedure

4.1 Objectives of Technical Verification

It is not the purpose of this verification to ensure that the program accomplishes its design objectives, since this will be done by the PSG in the functional verification of the system. The objectives of technical verification are basically to ensure the RPMS programs distributed throughout IHS meet the following requirements:

4.1.1 Correctness That there are no obvious errors in functionality, system design, or programming methodology.

4.1.2 Standards and Conventions That appropriate SAC are followed to facilitate maintenance and program readability, and to provide consistency across RPMS programs.

4.1.3 Documentation That adequate user and technical documentation has been developed for the system.

4.2 When Verification is Required

4.2.1 National Distribution Verification is required for any RPMS program developed by the IHS that is to be distributed nationally from the SRCB, or which is defined as a part of the RPMS "family" of systems.

4.2.2 Local Use Verification is not required for programs intended for local use only. However, it would be prudent to follow the RPMS SAC even for local development, since these standards are designed primarily to protect systems that run in the same environment, and to facilitate program maintenance.

Local systems may be distributed informally to other Areas without verification, but the receiver of the system should be forewarned that the unverified application may interfere with other approved application systems running on their computer(s) either now or at some time in the future.

A locally developed system may be submitted for verification at any time by following the guidelines described in this manual.

4.2.3 VA Software VA programs will not be verified by the SRCB since they have already undergone a certification by the VA. However, any modules or changes added by the IHS will be verified.

4.3 Preliminary Technical Verification

The Director, DSD, will coordinate with the Director, DSM and the Chief, SRCB, to ensure that the SRCB is not saturated with requests for PTV, and that the order of submission of systems for PTV is in accordance with the exigencies of the IHS.

4.3.1 Minimum Requirements for PTV Items submitted for verification will be exactly those items that are intended for national release. Minimum requirements follow.

4.3.1.1 **Package** Software including a notes, global (if applicable) and routines file.

4.3.1.2 **Documentation** User Manual (or updates), Technical Manual (or updates), Installation Guide and/or Release Notes, and Security Guide, if applicable.

4.3.1.3 **Exemptions** Requests for exemptions from the RPMS Programming Standards and Conventions.

4.3.1.4 **Design Criteria** Design criteria or system requirements as provided by the PSG or developer.

4.3.2 Reviewer(s) The SRCB will designate one or two persons for the technical verification review. Considerations involved in the selection are the availability of persons to perform verification, rotation among members to help share the load, and if possible, availability of the same person to subsequently perform the final review.

4.3.3 Technical Verification The SRCB will perform a technical verification which will involve auditing the software for adherence to RPMS SAC and evaluation software for installation and operational correctness. The PTV must be completed and results returned to the developer within 30 days of receipt of the request for PTV. Requests for extension of this timeframe must be arranged with the Chief, SRCB and Directors, DSM and DSD. If an extension is granted, the developer will be informed of the reason for the delay.

4.3.4 Functional Verification In addition, SRCB will perform functional verification to ensure that the software is operationally sound. The software will be tested for logic errors, inaccuracies and inconsistencies. This process will include:

1. Simple testing in a controlled environment, and

2. Complex testing involving interaction with other RPMS modules in an environment removed from control of the developers.

4.3.5 Review Findings Upon completion of the verification process, the reviewer will provide written findings (electronic or letter format) to the developer for action. Comments may be provided but they should be specific, understandable (<u>avoid technical jargon where possible</u>), constructive and tactful. Examples and alternatives may be presented where appropriate.

4.3.6 Timeliness Coordination between the SRCB and developers is paramount to successful and timely release of quality software. The SRCB will ensure that the verification is completed timely and developers will ensure that findings are acted upon in a timely fashion as well.

4.4 Final Technical Verification

4.4.1 Requirements The developer of the system will provide the same items as required for the preliminary verification but the products will be in final format. In addition, the developer will include the following:

a. A list of changes made since the last verification review. This does not need to be a specific line-by-line list of changes, but a general indication to the reviewer of the sections of the system that need re-verification because of changes.

b. The original reviewer’s comments with the developer’s response in electronic or letter form.

4.4.2 Reviewer(s) The reviewer(s) will complete the final verification as soon as possible, but within one month from date of receipt, following the same procedures outlined for the preliminary verification. Upon certification of the package and receipt of the endorsement form and beta checklist from the PSG, SRCB will prepare the software for national release.

5. Functional Verification Procedure

5.1 Objectives of Functional Verification

The objectives of the functional verification of a system are:

a. Ensure that the system performs according to the PSG design criteria,

b. Ensure that the system is easy to understand and use by an end user,

c. Ensure that the system is documented, and has a readable user manual

5.2 When Verification is Required

A functional verification of a system is performed by the PSG, or system designer, during an alpha test of the system, and again during the beta test of the system.

5.3 Definition of Alpha/Beta Test

5.3.1 Alpha Test

An alpha test of a system is the first comprehensive check-out of the system by someone other than the development staff. It is usually performed by end users in an operational setting in the developer’s Area, but could be a non-operational "laboratory" test by PSG members at the development site. In either case, the test should be coordinated and monitored by the PSG.

5.3.2 Beta Test

A beta test is the second formal review of a system, after all of the problems identified in the alpha test have been corrected. A beta test is definitely in an operational setting, and involves day-to-day use by end users. Preferably, it will be accomplished in an Area outside of the development Area, but this is not mandatory. The PSG is responsible for overseeing the pilot operation, and verifying that the system accomplishes all design objectives in an acceptable fashion.

5.4 Functional Verification of Program

Every option of the program must be thoroughly and accurately tested using a logical testing sequence to ensure completeness. The PSG will prepare a written test plan before alpha and/or beta test, to follow during the system check-out. Upon completion of the beta test phase, the PSG/test site will complete the formal endorsement form and beta checklist certifying that the package has been fully tested and is ready for release.

Beta testing involves not only a formal check-out as described in this document, but also routine day-to-day operation of the system by end users for some period of time, usually 30 days at a minimum.

5.4.1 Formal Functional Verification Plan

The formal functional verification plan should ensure the following.

a. Each option should function according to PSG design criteria, and must receive PSG endorsement.

b. The software must do what the documentation says it does. There should be no inconsistencies between performance and documentation.

c. If there are nuances in the way the software works, they should be documented, either in the software itself or in the User Manual.

d. If there are assumptions made that are not readily apparent, they should be documented.

e. The style of the package, in screen formats, data input, and option selection, should be consistent and unambiguous.

f. No unwarranted errors should be generated while using any of the options in any possible way.

g. The system should never "hang up" in an error condition, but should display the error message and return the user to an option selection.

h. Response times must be acceptable from the end-user standpoint.

i. "Help" prompts should be meaningful and provide examples when appropriate.

6. Documentation Verification

This process will involve reviewing all documentation for:

a. thoroughness,

b. accuracy,

c. readability,

d. functionality of option descriptions and examples, and

e. adherence to RPMS Documentation Standards for RPMS Software.

6.1 Minimum Documentation Required

a. User Manual

b. Technical Manual

c. Release Notes (Required for major revisions of existing systems) identifying all enhancements and changes made to the system since the last verified version

d. Installation Instructions (Notes File)

e. ReadMe File identifying the word processing package used to prepare the manual, type of printer used in the process, fonts used throughout the manual, and list of files included in the documentation files.

6.2 Preparation of Documentation

All documentation that is to accompany a RPMS package must be prepared in electronic form as outlined in the RPMS Documentation Standards.

6.3 Review of User Manual

The User Manual will be evaluated for the following:

a. all options are fully described;

b. all options should have appropriate examples;

c. the format should be readable, complete and easily understood and can be followed by a novice user;

d. the manual should be organized in a logical manner;

e. all prompts described in the manual should match what is actually on the screen; and

f. rules of grammar and correct spelling should be followed.

6.4 Review of Technical Manual

The Technical Manual will be evaluated for the following:

a. that content meets minimum requirements as set forth in the RPMS Documentation Standards

b. and that items required to be noted by a RPMS SAC are accurately documented.

6.5 Review of Documentation

The SRCB will complete the documentation review and provide a report of their findings in writing by electronic mail, or by letter to the responsible developer/ preparer of the manual. Unresolved issues will be handled as outlined in the RPMS Documentation Standards.

7. Design Verification Procedure

7.1 Responsibility The DBA will review the design of the system concurrently with Technical Verification to determine whether it is fully integrated with other RPMS data dictionaries. The section on Data Base Design in this handbook discusses data base design criteria for RPMS applications. Basically, these requirements are:

a. If the application generates patient-specific health data, this data will be integrated into the core set of patient care dictionaries;

b. Whether dealing with patient health data or not, the application should integrate with other pre-existing RPMS dictionaries as appropriate;

c. The application should use standard IHS tables where appropriate, rather than create new tables.

7.2 Results of Review Depending on the results of the design verification, the system will either be certified or returned to the developer for conformance to the RPMS design principles.

8. Certification and Release of RPMS Software

8.1 Items Required for Certification

a. Receipt of signed PSG Endorsement Form and Beta Test Checklist

b. Successful completion of final verification

c. Successful completion of beta test phase

d. Relevant data base design criteria are satisfied

e. Final versions of software and documentation

8.2 National Release If the software satisfies all outlined requirements, it will be submitted for final certification and national release. If the software does not meet all the outlined requirements it will be returned to the developer for rewrite or redesign. Any software released to users that is not classified as certified will not be supported by or maintained by OIRM.

8.2.1 Responsibility The SRCB is responsible for the coordination and release of RPMS packages and will maintain a central file of all systems that have been certified.

8.2.2 Release Letter A release letter signed by appropriate officials will be prepared and distributed through appropriate channels announcing the availability of the newly certified and released software. A copy of the PSG endorsement(s) will be attached to the release letter.

8.2.3 Certified Software When all concurrences are in place, the SRCB will make the newly certified software available for Area ISC personnel. An IHS MailMan message will be generated in addition to the written Release Letter announcing the package release.

9. Patch Certification

9.1 Patch Priority SRCB personnel will give highest operational priority to verification of completed patches classified as MANDATORY. Patches are to be completed by another IHS developer.

9.2 **Notification Procedure** Verifying personnel will be automatically notified of completed patches through the Patch Module on IHS MailMan.

9.3 **Nature of Verification** Verification of patches will consist of procedures listed under Final Technical Verification. Functional verification will be performed by the developer and completer of the patch.

9.4 **Unix File Containing the Patch** When the patch has been verified, the verifier will move the file to the uucppublic/PATCHES directory.

Indian Health Service
Office of Information Resource Management

Procedure for RPMS Documentation

June 27, 1996

Procedure for RPMS Documentation

1. Purpose

To establish policy for all IHS RPMS software package documentation. It may also apply to package documentation for non-RPMS automated information systems. This section defines package documentation and management of documentation. RPMS documentation standards (Appendix F) have been established to:

1.1 Provide a basic documentation structure that can be applied to every RPMS national package. For non-RPMS automated information systems software packages, documentation must be available to support the product.

1.2 Provide consistency in all RPMS documentation.

1.3 Provide criteria by which documentation of an RPMS package can be verified.

1.4 Ensure the highest standard of documentation in order to achieve the goal of providing optimal information to the targeted audiences.

2. Policy

2.1 RPMS Packages

2.1.1 Electronic copy documentation shall be provided for all RPMS national packages.

2.1.2 Package documentation shall be provided in sufficient detail for users and local site manager to install, operate, and manage the package. Documentation must provide clear, understandable materials that serve the software package users.

2.1.3 Package documentation shall comply with the RPMS documentation standards.

2.1.4 Documentation shall reference the availability of on-line tools (e.g., help frames).

2.1.5 The SRCB, DSM, OIRM shall review all nationally-developed RPMS software packages for compliance with the documentation standards.

2.2 Non-RPMS Automated Information Systems Packages

2.2.1 Documentation shall be provided in sufficient detail for users, programmers, and OIRM personnel to install, operate, and manage the systems. Documentation must provide clear, understandable materials that serve the software package users

3. General Responsibilities

3.1 Associate Director, OIRM The Associate Director, OIRM, has authority for documentation policy. The Director, DSM, establishes standards and procedures for RPMS documentation. The SRCB is responsible for reviewing all RPMS software package documentation for compliance with the documentation standards.

3.2 Development Centers Each Development Center is responsible for the preparation of documentation for its packages.

3.3 Division of Systems Management DSM will manage the reproduction, inspection, assembling, and distribution of all certified package documentation and all other related materials.

3.4 Information Systems Coordinators (ISC) Each ISC will distribute package documentation to its Area health care facilities.

4. Procedures

4.1 Documentation Preparation

4.1.1 Format All documentation will be prepared and distributed in electronic format as a PDF document using Adobe Acrobat.

4.1.2 Audiences Consider the wide variety of skill, education, and system familiarity in the targeted audience. Audiences include the health care facility user community, PSGs, and technical personnel at the facility and Area levels. The manual should be written or edited with these factors in mind. Give the users all the information they will need to satisfactorily perform their duties, but do not overwhelm them with nonessential details. Documentation offers, as appropriate, procedures, examples, exemptions, warnings, and helpful tips.

4.1.3 User Groups and Conventions In the beginning of each manual, clearly identify the user group and the notational conventions that will be used in the manual. When special terms, system names, unusual abbreviations or acronymns first appear, they should be explained. If there are a number of such terms, you may also want to present them in a glossary section or appendix for easy reference. The users should know by identified differences in format, fonts, and presentation of data, what type of information is being shown, i.e., background information, computer menus, prompts, displays, sample reports, etc. In computer interactive or tutorial-type manual sections, show the computer menus or prompts in the same form that the users will see on their computer screens, including capitalization and punctuation.

4.1.4 Guidelines Keep the following guidelines in mind.

4.1.4.1 Instructions that may be repeated many times in the same manual, such as instructions for queuing a report to a printer, should be consistent.

4.1.4.2 Avoid the use of underlines or all capital letters in text; they are difficult to read. Boldface letters are better for emphasis.

4.1.4.3 Avoid phrasing that, if taken literally, could be disastrous. For example, "Press any key to continue is a better choice of words than "Hit any key to continue" or "Strike any key to continue".

4.1.4.4 Documentation produced for the RPMS packages will be considered part of the official reference library.

4.1.4.5 The PSG and/or Program Officer associated with the package will assist in documentation preparation and is expected to review the documentation.

4.2 Documentation Review and Approval

4.2.1 Preliminary Set A preliminary set of all documentation will be prepared by each Development Center prior to beta test of an application.

4.2.2 Submittal A copy of the documentation will be submitted to SRCB at the time of beta test in PDF format. Hard copies of documentation will not normally be required.

4.2.3 Review The SRCB will review all documentation for adherence to the documentation standards, for spelling, formatting and grammar, and for clarity and accuracy. The responsible Development Center will be advised of the findings as well as any differences between the documentation and the application.

4.2.4 Development Center The responsible Development Center will finalize all documentation at the conclusion of the beta test. The final documentation will be submitted to SRCB in electronic PDF format along with the final set of software files.

4.3 Distribution and Reproduction

4.3.1 Responsibility The SRCB shall be responsible for any reproduction, inspection, assembling, and final distribution of the completed documentation.

4.3.2 Master Copy The DSM shall retain a master copy of all documentation. SRCB will maintain a copy of all certified software and its respective documentation in the public directory. Licensed or restricted software will reside in a separate protected directory. Areas may obtain any certified software from the public directory. Licensed or restricted software may be obtained by contacting DSM directly.

4.3.3 Updates, Changes, Deletions Updates, changes, or deletions to certified package documentation shall be submitted to SRCB for distribution and addition to the master libraries.

Indian Health Service
Office of Information Resource Management

Classification of RPMS Software

June 27, 1996

Classification of RPMS Software

1. Purpose

The purpose of this appendix is to provide policy and guidelines to be followed in the classification of RPMS software. The purpose of classification is to distinguish, for users, the different levels of software and the level of support they can expect from the OIRM.

2. Overview

2.1 Classes Within the RPMS, there are two classes of software based on whether the application has been certified by the RPMS/CMB, and whether the data base has been integrated into the RPMS Data Dictionary.

Software Classes

	Class I	Class II
Certified by RPMS/CMB	Yes	Yes
Integrated with RPMS Data Dictionary	Yes	No

3. Certified Attributes

A system certified by the SRCB (Class I and Class II) has the following attributes.

3.1 Technical Verification The system has received technical verification. This verification ensures that the system has been written in accordance with RPMS Programming SAC.

3.2 Functional Verification The system has received functional verification by the appropriate PSG, or in the absence of a PSG, by a user group outside of the developer's Area.

3.3 Support The system will be supported by OIRM.

3.4 Distribution The system is available for IHS national distribution as part of the RPMS.

4. RPMS Data Dictionary Integration

A system which has been integrated with the RPMS Data Dictionary is one which meets the following criteria.

4.1 Certification The system is certified by the SRCB both functionally and technically.

4.2 Standard RPMS Tables The system uses standard RPMS tables where appropriate, as opposed to local tables. Examples of non-complying applications might be those that use a local provider discipline table, rather than the RPMS Provider Class Table/File; or a local Tribe table rather than the RPMS Tribe Table/File.

4.3 Core Data The system stores RPMS core data in the appropriate RPMS file (i.e., visits are generated in the RPMS Visit File, diagnoses in the RPMS Purpose of Visit File, immunizations in the RPMS Immunization File, etc...)

5. Classification

Based upon the classification criteria, the two RPMS Classifications for software are further defined as follows.

5.1 Class I Software

5.1.1 Verified technically by the SRCB

5.1.2 Verified functionally by a PSG or user group outside of the developer's Area

5.1.3 Integrated into the RPMS Data Dictionary

5.1.4 Supported by a national development center

5.1.5 Available for IHS-wide distribution through DSM/SRCB

5.1.6 May or may not be mandated for implementation IHS wide

5.2 Class II Software

5.2.1 Verified technically by the SRCB

5.2.2 Verified functionally by a PSG or user group

5.2.3 Not integrated into the RPMS Data Dictionary

5.2.4 May or may not be supported by a national development center

5.2.5 Available for distribution through the SRCB

5.3 **Designation Documentation** The class designation of all distributed packages will be identified in the package documentation and will be recorded in the Package File which is distributed with the Kernel.

5.4 **Additional Information** For additional information on certification procedures, programming standards, and RPMS data base design criteria, refer to the appropriate appendices of this document.

Indian Health Service
Office of Information Resource Management

Developer's Checklist

June 27, 1996

Developer's Checklist

Developer's should go over this checklist prior to submitting their packages for verification.

	Has the VA's version of %INDEX been run? Are the problems noted corrected? Has the IHS SAC Checker been run and any problems noted corrected?
	Is the top menu option of the package in the form "NAMESPACE"MENU and the top key in the form "NAMESPACE"ZMENU? Is the institution file value of the current DUZ(2) and the version of the package displayed when entering the top menu of the package?
	Do the outputs from the package provide warnings to the user when displaying Privacy Act patient data?
	Are the proper version numbers set into all your routines and are the second lines all formatted properly? Is all pertinent information entered into the Package File (9.4)?
	Has the "DEL" node been set into the ^DD for each file in which deletion of data is not allowed?
	Has the "VR" nodes been set into ^DD for each file to describe which version of the file is being sent?
	Do the inits created with DIFROM carry along only the dictionaries? (No data unless exempted)
	If there is a pre-init, does it affect the same files as unfolded by the init?
	Has it been ensured that FM security codes have been included in all the files packaged in the inits?
	Is the general package requirements and disk storage requirements for the package included in the package documentation?
	Was the generic notes template used to prepare the notes file?
	Are the manuals ready to submit with the package, including electronic files for each?
	Are the unix files appropriately named?

Indian Health Service
Office of Information Resource Management

Procedure for Performing BETA Testing

June 27, 1996

Procedure for Performing Beta Testing

1. The following outlines the basic procedure for BETA Testing of an RPMS package. Development personnel should follow the steps outlined below. Changes to this process must be reviewed by the Directors, DSD and DSM and the Chief, SRCB before implementing.

1.1 The developer determines where the BETA testing is to be done at least two weeks prior to date the testing begins. Those Area/site names are provided to the SRCB who will then prepare the BETA Test Agreements. SRCB will obtain the appropriate signatures from the testing participants, as applicable, as outlined in Attachment A. Signed copies of the test agreement will be maintained in the SRCB Verification files for each package. BETA testing will not commence until all signed agreements are received.

1.2 Concurrent with Item 1.1.1, the responsible developer will submit a set of package files, i.e., routines, notes, globals and other files, to SRCB for a preliminary review. This must occur at least two weeks prior to the beginning of the BETA testing phase. This will assure the test sites that some form of review has occurred. The Preliminary Review will include review of the %Index and SAC Checker findings along with execution of each option in the package. This is not a full verification of the package. The full review will occur simultaneously with the BETA testing.

1.3 Upon receipt of the signed test agreements and completion of the Preliminary Review, the SRCB will confer with the developer to commence the BETA testing phase.

1.4 The SRCB will upon agreement from the developer, send out the necessary files to the test facilities, initiate a BETA test IHS MailMan message, provide the test facilities with a BETA Check List (Attachment B) and coordinate the entire BETA test process.

1.5 Problems or errors encountered during the BETA test phase will be reported to the developer via the IHS MailMan message or by phone. The developer will rectify the problem and send a new set of package files to the SRCB for dissemination to each test facility and for review. Communication between the developer and SRCB is crucial to coordinate the number of new files that are necessary due to changes.

1.6 The SRCB will coordinate and conduct at least one conference call between the developer and test facilities during the BETA phase. This call will be used to further convey any problems being encountered and to assure that the test facilities have installed and are testing the software.

1.7 The BETA testing phase can vary in length but all packages must complete a minimum of 30 days testing without any changes to the software. A change would restart the 30-day requirement.

1.8 BETA testing will end when the following is met:

a) Test sites have completed and returned the BETA Check List

b) Test sites have signed and returned the Endorsement Form, Attachment C. This form verifies the package functionality and recommends it for national release.

c) Package has been in testing 30 days with no changes.

d) Package has been certified using the RPMS SAC.

e) Developer approves the national release.

1.9 Upon completion of BETA testing, the SRCB will make the software available nationally.

2. BETA Test Forms Samples of BETA Test Forms and Agreements follow.

2.1 Sample BETA Test Agreement

TO: AREA/SITE, ETC.

FROM: Software Review & Certification Branch
Division of Systems Management
Headquarters West OIRM

SUBJECT: BETA Test Agreement - Package Name Version #

The FACILITY has been proposed as a BETA test site for Version # of the PACKAGE NAME (NAMESPACE) package. Preliminary discussions with the Area Information Systems Coordinator and a representative of the site have been supportive of this proposal. This memorandum is to formalize the conditions of the BETA test and to document agreement by the parties most affected by the testing process.

As a BETA test site, you can be assured that the developer has tested the package in both development and alpha test environments to uncover any major problems that may have existed. Also, package has received a preliminary review by the Software & Certification Branch. Further, you can be assured that this software system is operating successfully at other field facilities. As a BETA test site, your facility will assume a key role in proving that the software works properly. Your site will also serve to test the installation of the software in a "live" environment to assure its compatibility with other RPMS packages. There may, of course, be latent "bugs" in the system not detected at previous test sites, and these will be addressed by the developer immediately upon notification.

The formal terms of the beta test agreement are as follows:

1. The test period will officially begin when the package is put into production use and officially ends when the software installed at all BETA sites has been stable without change for a minimum of thirty (30) days. In addition, the software must meet RPMS Standards and Conventions, the test facilities have endorsed the release of the package and completed the attached BETA Check List and the developer agrees to the release.

2. The DEVELOPMENT CENTER NAME Development Center and PROGRAM Program agrees to the following responsibilities:

    A. We will provide the software and documentation to your site. These will be provided through SRCB. Changes to the software during the testing phase will also be routed through SRCB to your facility.

    B. We will provide training for your package coordinator, site manager, and area information systems support staff.

    C. We will work with your RPMS staff as the primary support for the package during the testing period. Support will be provided via telephone, including dialing into your system, if necessary. Your contact for this package is:

        DEVELOPER:
        Indian Health Service
        Address
        Phone:
        IHS Mail address

        VERIFIER:
        Indian Health Service
        DSM - SRCB
        5300 Homestead N.E.
        Albuquerque, NM 87110
        Phone: (505) 837-4378
        IHS Mail address

3. As a test site, we ask that you assume the following responsibilities:

   A. Identify the package coordinator who will be involved in operational use of the package and will be responsible for local implementation and oversight of the application. Assure this person has access to IHS Mail, if at all possible.

   B. Provide adequate hardware for testing the package. Convenient access to a terminal and printer is required for operation of the package. Beyond that, your RPMS system should not require any expansion to run the test version of the package.

   C. To assure prompt resolution of problems and to assure a complete beta test, install all software related to the package as quickly as possible. We ask that you notify the developer when production use of the software begins. This can be done on the IHS Beta Log message created for this package or by phone.

   D. Follow the guidelines provided in the attached Guidelines for BETA Test Sites and complete the BETA Check List when testing has finished. (Package specific items can be added here)

   E. Review and comment on the installation, user and technical documentation produced for this package.

   F. Participate in any conference calls scheduled during the testing phase.

   G. Report all problems to us promptly. When a problem arises, it is vital that the developer be contacted as soon as possible with as much detail as possible to enable us to identify and correct the source of the problem. Providing the developer with step-by-step description of what the user was doing when the problem occurred will help resolve the problem in a timely manner.

4. In addition, we ask that you meet the following conditions:

   A. Allow access to your system by the developer to support investigation and resolution of problems. The details of this access will be worked out with your RPMS staff, if needed.

   B. Use the software exactly as provided. To support you in this test, the developer will coordinate or personally make all necessary modifications to the software. Any emergency fixes made by the site must be reported to the developer immediately. During testing, local enhancements must not be implemented in order to preserve the reliability and integrity of the test.

5. Please indicate your concurrence or non-concurrence with these conditions as soon as possible. Return this memorandum to SRCB, DSM, OIRM, 5300 Homestead NE, Albuquerque, NM 87110. Please feel free to call us for clarification of any of these requirements (505) 837-4378.

____ ________________________

SIGNATURE DATE

Signatures of BETA Test Site Participants, Area ISC and other necessary signatures, i.e., Site Manager, Program Manager or PSG, Service Unit Director (If applicable), etc.

2.2 Sample BETA Checklist

RPMS BETA TEST CHECKLIST

Please fill in the blanks or circle appropriate response as necessary.

DATE: ___________________

BETA SITE NAME: ______________________________

COMPUTER: _______________________ OPERATING SYSTEM: UNIX DOS

PACKAGE NAME & VERSION: _________________________

SIZE OF DATABASE WHERE INSTALLED: _____________ INSTALL TIME: ____________

==

Check ONLY the items that were FUNCTIONALLY reviewed during the beta test.

PROBLEMS OR ERRORS ENCOUNTERED DURING THE BETA TESTING OF ANY PACKAGE MUST BE REPORTED IMMEDIATELY USING THE BETA TEST LOG IHS MAIL MESSAGE OR VIA TELEPHONE CONTACT THROUGH THE RESPONSIBLE AREA PERSONNEL TO THE DEVELOPER OF THE PACKAGE.

Review installation notes file before installation	
Ascertain that REQUIREMENTS in Notes file is met on target system	
Review all accompanying manuals	
Execute all menu options	
Review all help prompts for clarity and usefulness	
Execute all field prompts	
Test escapes at all prompts ("^")	
Test jumping capabilities at all menu options, prompts ("^Mnemonic")	
Manipulate all site parameters in various combinations	

Test various access levels of users (use the various security keys)	
Perform all available data entry functions	
Check that all fields are storing the appropriate data	
Test all reporting functionality of the package for accuracy of results and presentation	
Review data extractions for integrity of data	
Ascertain functionality of all bulletins in package, if applicable	
If an audit is included, test if functioning as intended	
Schedule all background tasks and determine if appropriately functioning and correct results	
Check performance of batch transmissions, if applicable	
Review hooks and integration of data with other packages, if applicable	
Test printer functionality where indicated, i.e., slave, queuing, direct printing, screen printing	
Report all problems encountered through proper channels to the developer	
Other items not listed, specify	

CHECKLIST COMPLETED BY:________________________________DATE:_____________________

Please provide any comments you may have on any area of testing the noted package.

____

____

Thanks for participating in the BETA testing process. Your assistance helps assure quality software for all of IHS.

At the end of the BETA testing phase, please return this form and your BETA test endorsement form to SRCB, Headquarters West, 5300 Homestead NE, Albuquerque, NM 87110. A FAX copy is acceptable

GUIDELINES FOR BETA TEST SITES

The following are items that the test site should evaluate while performing the BETA Testing. All concerns should be reported via proper channels to the responsible developer as noted.

Application package validation is complete when the PSG/Program Manager has determined and certifies that the system is correct based on the criteria contained in its requirements document.

- Installation steps - clear and concise, accurate. Pre- and post-init requirements are clear. Can they be simplified for the installer?
- All package requirements are noted.
- Resource requirements are noted, if applicable.
- User Manuals are clear, accurately present the screen displays, are useful and provide clear presentation.
- Technical Manuals are useful to the installer/support personnel.
- Screen displays are logical and well presented.
- Help frames, prompts or on-line documentation are accurate and useful.
- Menu options/prompts perform appropriately.
- Background tasks perform accurately.
- Data integrity is maintained.
- Data transmission performs as required.
- Security is sufficient.
- Devices function appropriately.
- All reports produce the desired results.
- Any system configuration/management concerns are appropriately addressed.
- Global management is clearly defined.
- Unnecessary routines or components are noted for deletion to assist with system management.
- Additional functionality to an existing package is clearly stated.
- Site configurable items are clearly defined.
- All information necessary to install and support the package was provided with package.

2.3 Sample PSG/BETA Test Site Endorsement Form

TO: Software Review & Certification Branch, Headquarters West

FROM: PSG/Program Manager/Area/Facility Involved in Testing

SUBJ: PSG/Program Manager/Beta Test Site Endorsement of PACKAGE NAME Version #

The PACKAGE (NAMESPACE) Version # package meets all functionality requirements developed by the PSG/Program Manager. We endorse this software and approve of its release to all IHS/Tribal facilities.

Signature & Title (PSG/Test Site)	Date
Signature & Title (Program Manager or Area)	Date

Indian Health Service
Office of Information Resource Management

Procedure for Submitting Packages for Verification

June 27, 1996

Procedure for Submitting Packages for Verification

1.1 The following outlines the basic procedure for submitting packages to the SRCB for verification. Packages not following these procedures will be returned to the developer for appropriate correction before the verification process commences.

1.1.1 Files submitted will conform to the naming conventions outlined in Appendix C of 1996 RPMS Standards & Conventions (SAC). Package documentation will be prepared as outlined in the RPMS Documentation Standards. Package documentation manuals will be submitted in draft format at the same time that the software is submitted to Verification for initial review. These manuals should not be submitted as a partial document. A final set of complete documentation must be submitted to the SRCB at least two weeks prior to the end of the BETA testing. This allows time for final review.

1.1.2 There will be a minimum of four or five individual files required. A tar file is not acceptable. They are:

Routines file (always required)
Globals file (if globals being included)
Installation Guide (always required)
Release Notes (if subsequent release)
Documentation file (always required)

If additional files are required, coordination is required between the responsible Developer and the SRCB.

1.1.3 Requests for Exemption to the standards must accompany the files being submitted to the SRCB for initial review. Use only the approved form in Appendix B of the SAC.

1.1.4 Files submitted to SRCB will be placed in the /usr/spool/uucppublic/VERIFY directory on cmbsyb (Risc 6000) located at Headquarters West. An IHS MailMan message will be generated to G.VERIFIERS notifying them of the arriving files with instructions to start the verification process.

1.1.5 The developer shall provide the names of the BETA test facilities chosen. The SRCB will coordinate and obtain signed BETA Test Agreements from the facilities chosen. Additionally, the SRCB will prepare and obtain final endorsement memorandum from each BETA Test Site and/or PSG/Program Manager. Routines, etc., being submitted for verification are to be the same as those installed at the BETA test sites at all times. The SRCB will transmit all files and any subsequent changes to these files to each BETA site as the developer submits them. Please refer to the policy addressing BETA testing for further guidance.

1.1.6 All packages submitted to the SRCB for final distribution will contain only those files which belong to the package as outlined in the SAC. To accommodate alpha and beta testing, files submitted to the SRCB may contain other packages files only if submitted as follows:

- One routine or global file is submitted which includes the namespace routines or globals and any additional routines or globals needed by the package in order to perform the testing. VA Packages are the exception and should be submitted in separate files following the naming conventions outlined in the SAC.

- The notes file must contain specific information as to why these additional routines or VA packages are being included. It must also state that these routines or packages will be officially released as certified patches or releases before or in conjunction with the release of said package.

- Written confirmation that coordination has been conducted with the other developer whose routines are being included must be presented to the SRCB. This confirmation should note that the developer understands that an official patch must be released prior to the release of said package.

- The final files submitted for distribution will not contain non-package routines.

Indian Health Service
Office of Information Resource Management

Procedure for Distribution of Certified RPMS Software

June 27, 1996

Procedure for Distribution of RPMS Certified Software

1. Certified RPMS packages will be made available to the Area ISC immediately upon certification.

2. The basic software release process is outlined as follows:

2.1.1 The developer submits package to the SRCB, DSM for review and certification. Alpha and beta testing will occur simultaneously with the review and certification.

2.1.2 Upon certification of the package and receipt of endorsement of functionality from PSG and test sites, the SRCB will notify Area ISCs of the certification of the package on the IHS MailMan System. Documentation will be done electronically and will reside with the rest of the unix files in a directory on the cmbsyb Risc 6000 located in the DSM, OIRM.

2.1.3 The Areas will retrieve the desired files from cmbsyb for installation at their facilities.

2.1.4. Coordination between developers is required to assure that all software required by the package is either already certified and available, or that the required software is being submitted in conjunction with the package for testing and certification.

2.1.5 A written endorsement of the functionality of the package will be required from those participating in the BETA testing, from the PSG and from the Program Manager (if any).

2.1.6 The distribution directory will reside on cmbsyb in /usr/spool/uucppublic/DIST. It will be called 96cert. The first two numbers will change with the fiscal year. This directory will be restricted to use by the Area ISC personnel and Development Centers. Licensed software will be maintained in a separate more secure directory. Access to licensed software will require a phone or E-mail contact to DSM to request the required software . Proof of licensure is required before the software will be released.

Procedure for Package Installation Guide and Patch Notes

1. **RPMS Application Installation Guide** All RPMS software must have an Installation Guide prepared as outlined in Appendix F of the 1996 RPMS SAC. This Guide is to be prepared using PDF format. In general, developers are encourage to try NOT to instruct the user to "see the manual" for information, as the manual is generally not available during installations.

2. **Patch Notes** All patches will have a notes file. When writing the notes file, developers should use the sample template located on IHS MailMan Bulletin or follow the sample noted below. This sample represents the minimum items required in all notes files. In general, developers are encouraged to NOT instruct the user to "see the manual" for information, as the manual is generally not available during installation.

2.1. Sample Patch Notes

```
INSTALLATION NOTES FOR (Application Package Name)
==============================================================
PREFIX:    {Prefix as found in the PACKAGE file}
CURRENT VERSION: {Version}    {Patch number if a patch}
==============================================================

*****NOTE*****NOTE*****NOTE*****NOTE*****NOTE*****NOTE*****NOTE*
**************************************************************************
*        Read the Entire Notes File Prior to Attempting Any Installation !!!        *
**************************************************************************
*****NOTE*****NOTE*****NOTE*****NOTE*****NOTE*****NOTE*****NOTE*
```

1. GENERAL INFORMATION

   a) Print all notes/readme files.

   b) It is recommended that the terminal output during the installation be captured using an auxport printer attached to the terminal at which you are performing the software installation. This will assure a printed audit trail if any problems should arise.

2. CONTENTS OF DISTRIBUTION

   a) <routine_file> - Routines

b) <notes_file> - This file

c) <global_file> - Globals

d) <List any other files>

3. REQUIREMENTS

a) Kernel V 7.1 or later

b) FileMan V 21 or later

c) {List any packages, including version, required to install and/or run this package}

4. INSTALLATION INSTRUCTIONS

In all UCI's running {application name}:

a) Disable logins or ensure all users are off, or disable [prefix]MENU.

b) Do routine save and global save of <old routines/globals>.
(Note: Remove these saves from your system 7-10 days after this installation if no problems noted with new install)

c) Delete the routines [prefix]*.

d) Do routine restore from the file <routine_file>, which should restore {number of routines} routines.

e) D ^[prefix]INIT and answer "YES" to all init questions. Time to complete the init will be approximately {number of minutes}.

f) Do global restore from the file <global_file>, which should restore {number of globals} globals.

g) Assign option [prefix]MENU and security key [prefix]ZMENU.

h) {Assign other menus and keys to designated users }

i) Enable logins and/or [prefix]MENU.

j) <pkginits and any other rtns> may be removed from the system after successful installation.

5. POINT OF CONTACT

[Name], [division/branch/dev ctr], [phone #]

Indian Health Service
Office of Information Resource Management

Procedure for RPMS Patches to Certified Software

June 27, 1996

Procedure for Submitting Patches to Certified RPMS Software

1. Introduction

The National Patch Module (NPM) is a software package designed to provide a database for the distribution of software patches and updates for certified IHS software. Options provide for systematic entry, review and completion of patches by developers, review and release of patches by verifiers, and display and distribution of the released verified patches to the users.

Once a problem is found in RPMS software and the solution identified, the developer enters a patch in the NPM identified by its namespace, version and patch number. The patch is assigned a status of "under development" and is accessible only by other developers. When the patch is completed and ready for review, a second developer changes the status to "completed/unverified" making the patch available for review by the verifiers. After the verifier(s) have checked the patch and determined that it is ready for release, the status is changed to "verified". The patch then is available for viewing by users and is distributed to all Areas. A master directory of current patches resides in the public directory.

2. Patch Process

2.1 Access Developers have access to a package by being defined for that package in the DHCP Patch/Problem Package file (#11007).

2.2 Adding Patch The Developer is responsible for adding a patch to the package. A patch designation (i.e., package namespace*version*patch number) is generated for the patch and status is set to "under development". The patch may now be edited by the responsible developer.

2.3 Dependencies on Other Patches To help in the development and release of patches that are dependent on other patches, there is an ASSOCIATED PATCHES field where other patches may be entered as references. If the associated patch has to be verified and installed first, it may be flagged to prevent verification of the patch being edited until the associated patch is verified. There is a HOLD DATE field that can be used to prevent verification/release until a certain date. IHS patches must be cumulative, i.e., contain all previous patches to that package version.

2.4 Completing the Patch Patches are to be completed by another developer who has reviewed the patch to determine if the fix is as it claims. Upon completion of the review by the second developer, the status of the patch is changed to "completed". A completed-unverified message is generated to the SRCB automatically.

2.5 Verifier/Support Access Verifiers have access to a package by being defined as support personnel and verifier for that package in File #11007. The verifier will review the patch for conformance with RPMS SAC and report findings, if any, back to the respective developer. Upon passing certification, the verifier will change the patch's status to "Verified". The patch will then be distributed to all Areas and placed in the master patch directory on the public directory.

2.6 Support Personnel Users may select packages to enable automatic notification of new verified patches. Users who have selected a package for notification can also utilize an option which will print all new verified patches for a selected package.

3. Patch Files

3.1 Standards Files for patches are to be submitted to the SRCB following the formats outlined in the RPMS SAC.

4. Patch Module Manuals

For more details on the NPM, see the VA DHCP Documentation for NPM.

Indian Health Service
Office of Information Resource Management

Procedure for Adding Entries to New Person (#200) File

August 7, 1996

Procedure for Adding New Entries to the New Person (#200) File

1. Introduction

Two public entry points have been developed for those instances when an application must add new entries to the New Person File, File 200.

2. Process

To add a person entry:

> S X=$$PERSON^AVA200(AVADR,AVADR1) where X is your local variable, AVADR is your list of additional identifiers and AVADR1 is your list of additional edit fields for the DIE call.

To add a provider entry:

> S X=$$PROVIDER^AVA200(AVADR,AVADR1) where X, AVADR and AVADR1 are used the same way as in the $$PERSON call.

AVADR and AVADR1 must be set up as the variable DIC("DR") or DR with each field separated by a semicolon.

The identifiers already used by the AVA200 call are:
> Initials; Sex; SSN (required if user does not have the XUSPF200 key)

For provider call, additional identifiers are:
> Provider Class; Affiliation; Code

The edit fields already used in the AVA200 call are:
> The identifiers listed above plus DOB, Title, Address, Phone Numbers, Service/Section

For the provider call, the additional fields are:
> IHS Local Code, Medicare Provider #, Medicaid Provider #, UPIN #, Authorized to Write Med Orders, DEA#, Provider Type, Remarks, Medical License numbers.

Index