

PA-ABW-417

1371

New Management Systems NMS

Developer Standards and Guidelines

(Rules Only, Quick Reference Version)

April 1995



**United States Agency for International Development
(M/IRM/SDM)**

New Management Systems NMS

Developer Standards and Guidelines

The "Rules Only," Quick Reference Version

This document is a subset of the New Management Systems' Developer Standards and Guidelines, "Rules, Recommendations and Discussion" version. Its purpose is to provide a quick reference to NMS development Rules. These rules are necessary to develop a consistent Graphical Users Interface. Adherence to rules also promotes consistency while developing new management system processes and access to databases while using the Visual Basic, Oracle and SQL tools.

This abbreviated document does not contain general dialogue, recommendations, hints, tips, examples and discussions associated within the context of specific rules.

The Table of Contents in the "Rules Only" version for Chapters 1, 2 and 3 is the same as that in the "Rules, Recommendations and Discussion" version, except for page number references. These are the only chapters containing rules. Section contents from the "Rules, Recommendations and Discussion" version were deleted from these chapters as were Chapters 4, 5, 6 and Appendices C through G. To expand context and understanding of a given rule the reader should refer to the same section, chapter, or appendix in the "Rules, Recommendations and Discussion" document.

Table of Contents

| | |
|--|-----|
| Preface | 1 |
| Introduction | 3 |
| Document Standards | 4 |
| Term Usage Peculiar to This Document | 4 |
| NMS Document Conventions | 5 |
| References | 7 |
| | |
| 1 COMMON USER INTERFACE FOR NMS APPLICATIONS | 1-1 |
| 1.1 Graphical User Interface (GUI) Design Objectives | 1-1 |
| 1.1.1 Placing Users in Control | 1-1 |
| 1.1.2 Reducing the Users' Memory Load | 1-1 |
| 1.1.3 Making Applications Easy to Learn and Use | 1-1 |
| 1.1.4 Providing a Consistent User Interface | 1-1 |
| 1.2 The Common User Interface | 1-1 |
| 1.2.1 The Visual (display screen) Elements of a CUI | 1-1 |
| 1.2.2 The Functional (user actions) Elements of the CUI | 1-1 |
| 1.3 Screen Architecture and Components | 1-1 |
| 1.3.1 Menus | 1-1 |
| 1.3.1.1 Top Menu Bar | 1-1 |
| 1.3.2 Colors | 1-2 |
| 1.3.3 Fonts | 1-3 |
| 1.3.3.1 Minimum Font Size | 1-3 |
| 1.3.3.2 Default Fonts | 1-3 |
| 1.3.4 3-D Effects | 1-4 |
| 1.3.5 Text Display Selection and Formats | 1-4 |
| 1.3.5.1 Selection from Text Lists | 1-4 |
| 1.3.5.2 Dates | 1-5 |
| 1.3.6 Tabular Data Display | 1-5 |
| 1.3.7 Objects | 1-5 |
| 1.3.8 Graphs | 1-5 |
| 1.3.8.1 Contents of Graph Screens | 1-5 |
| 1.3.8.2 Graph Screen Functionality | 1-5 |
| 1.3.8.3 Colors and Styles | 1-6 |
| 1.3.9 Help | 1-6 |

| | | |
|---------|---|------|
| 1.4 | User Navigation | 1-7 |
| 1.4.1 | Keyboard Methods as Alternatives to Mouse | 1-7 |
| 1.4.2 | Drill-Down Methods | 1-7 |
| 1.4.3 | Look-up Methods | 1-7 |
| 1.5 | Window Forms | 1-7 |
| 1.5.1 | General Rules | 1-7 |
| 1.5.1.1 | Resolution/Size | 1-7 |
| 1.5.1.2 | Labels | 1-7 |
| 1.5.1.3 | Exiting Forms | 1-7 |
| 1.5.1.4 | Window Titles | 1-7 |
| 1.5.1.5 | Graphics on Forms | 1-8 |
| 1.5.1.6 | Automatic Resizing | 1-8 |
| 1.5.1.7 | 3-D Appearances of Forms | 1-8 |
| 1.5.1.8 | Distraction/Program Entry Forms | 1-9 |
| 1.5.1.9 | Main Menu/System Entry Forms | 1-9 |
| 1.6 | MDI Parent Forms | 1-9 |
| 1.6.1 | Top Menu Bar | 1-9 |
| 1.6.2 | Toolbar | 1-10 |
| 1.6.3 | Standard Toolbar Icons | 1-11 |
| 1.6.4 | MDI Parent Status Bar | 1-14 |
| 1.7 | Controls | 1-14 |
| 1.7.1 | Command Buttons | 1-14 |
| 1.7.2 | Standard Control Placement on Forms | 1-16 |
| 1.7.2.1 | Command Buttons | 1-16 |
| 1.7.2.2 | Grouping of Controls | 1-16 |
| 1.7.2.3 | Labels and Label Controls | 1-17 |
| 1.8 | Data-Entry Controls | 1-17 |
| 1.8.1 | Speeding Data Entry Using Defaults | 1-17 |
| 1.8.2 | Speeding Data Entry Using Pop-up Windows | 1-17 |
| 1.8.3 | Speeding Data Entry Using Masks | 1-18 |
| 2 | CODING STANDARDS (VISUAL BASIC) | 2-1 |
| 2.1 | Standardizing Existing Code | 2-1 |
| 2.1.1 | Syntax Conventions | 2-1 |
| 2.2 | Declarations | 2-2 |
| 2.2.1 | Variable Declarations | 2-2 |
| 2.2.2 | Constant Declarations | 2-2 |
| 2.2.3 | DLL/API Subroutine Declarations | 2-2 |
| 2.3 | VB Environment | 2-4 |

| | | |
|--------|---|------|
| 2.3.1 | Option Explicit | 2-4 |
| 2.3.2 | DefInt A-Z | 2-4 |
| 2.3.3 | Save As Text | 2-4 |
| 2.3.4 | Option Base | 2-4 |
| 2.3.5 | Option Compare Binary vs. Option Compare Text | 2-4 |
| 2.3.6 | Save-Before-Run | 2-4 |
| 2.3.7 | Grid-Alignment Size = 60 | 2-4 |
| 2.3.8 | Config.Sys: Files | 2-5 |
| 2.4 | Source Code Documentation | 2-5 |
| 2.4.1 | Code Commenting | 2-5 |
| 2.4.2 | Standard Header Templates | 2-5 |
| 2.4.3 | Internal Comment Blocks | 2-9 |
| 2.5 | Indenting Source Code | 2-10 |
| 2.6 | Naming Conventions | 2-10 |
| 2.6.1 | Name Prefixes | 2-11 |
| 2.6.2 | Constant Names | 2-11 |
| 2.6.3 | Variable Names | 2-11 |
| 2.6.4 | Subroutine Name Prefixes: Indicating Data Types | 2-12 |
| 2.6.5 | Names for Database Objects | 2-12 |
| 2.6.6 | Names for GUI Objects -- Forms and Controls | 2-12 |
| 2.6.7 | Menu Naming Conventions | 2-13 |
| 2.7 | Subroutine Design, Coupling and Cohesiveness | 2-14 |
| 2.7.1 | General | 2-14 |
| 2.7.2 | Global Variables | 2-14 |
| 2.7.3 | Global Subroutines | 2-15 |
| 2.7.4 | Private/Local Subroutines | 2-15 |
| 2.7.5 | User-Defined Data Types | 2-15 |
| 2.7.6 | Constants and Variables -- Scope | 2-15 |
| 2.8 | Procedural Coding Standards | 2-15 |
| 2.8.1 | Concatenation Operators | 2-15 |
| 2.8.2 | Goto Statements | 2-15 |
| 2.8.3 | Error Trapping | 2-15 |
| 2.8.4 | If-Then-Else Structures | 2-16 |
| 2.8.5 | Loading Text into Combo Boxes, List Boxes and Grids in VB | 2-16 |
| 2.8.6 | AutoRedraw Property | 2-16 |
| 2.8.7 | Sending Messages from One Form to Another in VB | 2-16 |
| 2.8.8 | Writing Text to Labels in VB | 2-16 |
| 2.8.9 | Modularization -- File Organization | 2-16 |
| 2.8.10 | Data Management | 2-16 |

| | | |
|----------|--|------------|
| | 2.8.10.1 Validating Data Entry | 2-16 |
| | 2.8.10.2 Debugging SQL | 2-18 |
| | 2.8.10.3 Safely Interpreting / Converting Field Values | 2-19 |
| | 2.8.10.4 Programming Rules for Database Reliability | 2-22 |
| | 2.8.10.5 Increasing Database Performance in Visual Basic . | 2-23 |
| | 2.8.10.6 Database Error Trapping | 2-24 |
| 2.9 | Coding to Minimize Memory Use and Executable File Size | 2-24 |
| 2.10 | Coding to Maximize Performance (Execution Speed) | 2-26 |
| 2.11 | Shared, Common, Reusable Source Code | 2-26 |
| 3 | DATABASE ACCESS AND SQL CODING STANDARDS | 3-1 |
| 3.1 | SQL Calls to the Database | 3-1 |
| 3.2 | Creating and Using Indexes | 3-1 |
| 3.3 | ORACLE SQL Statement Processing Techniques | 3-2 |

APPENDICES

APPENDIX A - Standard VB GUI Object Name Prefixes

APPENDIX B - Standard Name Prefixes for Variables and VB Data-Access Objects

Preface

The U. S. Agency for International Development's *NMS Development Standards and Guidelines* is provided for NMS developers and for USAID officials responsible for planning, designing, developing, testing, and deploying the NMS. The standard describes the technical methodologies and organizational procedures for ensuring that the NMS is completed in an organized and businesslike manner. Adherence to the standard will help ensure that all USAID programs and databases behave consistently, will require less training, technical support, and documentation to implement the new management systems. Applications will be easier to use, fewer errors will be made, and developer productivity will be improved. Lastly, but of perhaps greatest importance, follow-on maintenance costs will be minimized.

The USAID "Common User Interface (CUI) Standard" document (Aug. 6, 1993), is incorporated in this document. The ISP (Information Systems Plan) "Report to Management" document (Feb. 1993) makes several references to the CUI Standard. Those standards have now been updated and included in this document.

Pursuant to its responsibilities defined in the Paperwork Reduction Act of 1980 (44 USC Chapter 35), USAID's Office of Information Resources Management (M/IRM) requires that all newly developed or acquired systems accessed by more than one user must adhere to this standard, effective October 1, 1993. Adherence to this standard is required when expending USAID funds to acquire programming services to develop computer programs or to modify existing programs. This standard is also applicable to the purchase of Commercial Off-The-Shelf (COTS) programs to be used by more than one user.

This standard is applicable to programs that will be used by USAID employees or by contractors performing work for USAID on the USAID network. Compliance with this standard is recommended, but not required for programs acquired by USAID for use by host countries. For instance, if it becomes necessary to procure a word processing program for a USAID employee or contractor to use in preparing an USAID project paper, these standards would not be mandatory for the staff of a host country ministry.

Although the use of the enclosed standards offers substantial benefits to USAID, there will be some instances in which its implementation may not be cost-effective. An example would be a special-purpose off-the-shelf program with a non-standard interface. It will not be cost-effective in most cases to modify an existing commercial program simply to change the user interface. M/IRM will grant waivers from the application of the standard in cases where its implementation would not be cost-effective. Requests for a waiver from the standard should be sent to M/IRM/OD with a justification for the waiver.

The following standards and guidelines recommend a development approach for all USAID NMS applications. The standard emphasizes the values of consistency, simplicity, and user empowerment to make the user's NMS computing experience more pleasant and productive. This document will evolve as technology improves, USAID business needs change, and as experience provides improved methods. Suggestions, questions, and comments are always welcomed.

For further information please call:

| | | |
|-------------------|----|-----------|
| Steve Polkinghorn | | Ron Burke |
| IRM/SDM | or | IRM/SDM |
| 875-1646 | | 875-1909 |

Send WordPerfect modifications (by E-mail) to:

Deborah Adams
IRM/SDM
875-1843

Introduction

Proper planning for NMS development and implementation is critical to USAID management.

The organization has made large investments in developing plans, choosing appropriate hardware, and training staff. A major challenge facing USAID management today is the necessity to incorporate the appropriate technology into the enterprise, while avoiding disruption and keeping productivity at acceptable levels.

This Standards and Guidelines manual addresses the requirement to set rules and guidelines for developers to follow while they design, document, and code software applications for the Agency's New Management Systems. These standards are tailored to Microsoft's Windows applications. Some are generic (not language-specific) and some are specific to Visual Basic for Windows and the Oracle data base management system.

Many sections contain rules, recommendations, and discussion. Thus, a person wishing to quickly learn or search only the rules can do so, but discussion giving details, examples, or justifications for the rules and recommendations is also provided. The rules are mandatory; the recommendations are not. A "rules only" version of the document will be supplied.

The need for creating, maintaining, and enforcing good standards is of the highest priority. Developers who balk at having to live by standards -- and it is natural to feel resistance to using standards that do not appear to make sense -- have to keep in mind that the system being built will be used by a large and diverse group of users, throughout the USAID enterprise. Productivity, deployment, usability and acceptance of the system by these users are critical to the organization's mission.

It is impossible to create standards that satisfy everyone. Individuals who choose to express their views can influence or add to the standards in meaningful ways. The rules and recommendations will change as conditions change or new understandings occur. The best source for new standards and rules to evolve from -- at least for coding standards -- is an intelligent developer base. Whatever makes code more readable, more maintainable, and of higher reliability is more desirable.

A goal within the NMS project is to get methodologies, procedures and standards clearly defined, followed, and enforced. To get these standards in place, compromises on personal preferences may sometimes become necessary, without compromising quality. Changes also have to take place one step at a time because the NMS effort requires a substantial technological change -- and new habits and ways of thinking. This cannot be done overnight. The process of implementing these changes will require much patience and willingness from all staff involved.

Document Standards

Term Usage Peculiar to This Document

Most terms can be found in the glossary. However, the following terms are defined here because they are used either in a non-standard way or with a more generic meaning than in a specific programming language.

The terms "application" and "program" are interchangeable and refer to a standalone, executable piece of software. "Program" is from the older, text-based programming tradition; and "application" is more commonly used in the newer Graphical User Interface (GUI)-based programming convention.

In Visual Basic (VB), the term "subroutine" is used to refer to a "procedure," as opposed to a "function." The term "subroutine" is used in this document in its more generic meaning to refer to both functions and procedures. A function returns a value through its name. A procedure does not return a value through its name. Both functions and procedures can return values through elements in their parameter lists.

A "control" is a GUI object that resides on a window-form (see below) and encapsulates certain functionality. Some examples of Visual Basic standard controls are scrollbars, listboxes, command buttons, check boxes, option ("radio") buttons, picture boxes, labels, or text boxes. A "custom control" is one that is built either by the local programming team or by a third-party vendor. A control, if it can be visible, is a lower-level type of window than forms. Some controls do not provide visual objects (except in design mode), but rather provide just functionality, such as the "common dialog" controls that can pop up a dialog box to handle common user operations such as "Open File" or "Select Font."

A "bound control" is one that is bound to a "data control," which is a special type of VB control that provides database access with little or no programming.

A "form" in Windows programming refers to a high-level window that can contain "controls" and float free from other windows -- either on the desktop or, for MDI (Multiple Document Interface) child forms, within the boundaries of an MDI parent form. A form (sometimes called a "window-form" for clarity) can generally be moved. Most forms can also be resized, minimized or maximized, if they are not modal. A modal form is one that will not let the user do anything else until that window has been closed. An example of this is an error message box that must be acknowledged by the user by clicking "OK" before it will close.

A global variable is one whose "scope" extends throughout the application; a "semi-global" or "form-global" variable is one whose scope lies within a module -- either a pure source code module (.BAS) file or a form-module (.FRM file).

NMS Document Conventions

The following conventions shall apply to all documents that are a part of USAID NMS library.

Rule 1: When a computer text string that must be strictly literal is being quoted, commas, periods, and other punctuation that are not actually part of the literal string shall be placed *outside* the quotation marks.

This rule applies to any literal text string for a computer command, SQL query, program source code sample, or other computer text string that must be strictly literal. The reader shall assume that any commas or periods inside such quotation marks for such literal strings are intended to be part of the command. This rule eliminates confusion concerning whether the punctuation mark is intended to be part of a text string intended to be strictly literal. This is to get around the limitations of the traditional, literature-oriented rule in English that commas and periods at the end of a quoted expression (that is, when quoting a person) are to be placed inside the quotation marks. The traditional rule is not practicable when documenting computer-oriented commands, which must be strictly literal. For example, if a document using the traditional syntax told a user to type the DOS "dir" command in the following manner, the user would be confused whether or not to type a period as part of the command:

Type "dir."

Rule 2: Programming source code samples or programming language keywords shall be printed in bold, 12-point, fixed-size font. "Courier" is the common standard for this, but if "Courier" font is used, "Courier New" (a TrueType font) should be used so that font size can be set. The normal font size for source code shall be 12-point, but 10 point can be used where it is needed to prevent long lines from being wrapped. Comments quoted as part of source code samples should be in non-bold, in the same fixed-size font and font-size as the source code.

Rule 3: Computer command strings and quoted blocks of source code should be separated from the surrounding text referring to them. That is, they should begin on a separate line and there should be a blank line preceding them and following them.

Rule 4: Computer command strings and quoted blocks of source code should be indented one indent space, if space permits and line-wrapping can be avoided.

Rule 5: Avoid automatic line-wrapping when quoting Visual Basic source code or user command strings that are supposed to be on one line. Where line-wrapping is necessary, the writer shall manually break the line using a graphic right-arrow symbol. Visual Basic does not support line-wrapping, since it is a single line-oriented language.

Rule 6: DOS commands or other user-keyboard input will be in ***bold Italic*** font and will be indented in from the left margin.

Rule 7: Text that will be displayed on a user-screen will be in bold, 14-pcint font. A fixed-size font shall be used if it is necessary to make columns of text or data line up or if multiple lines of text must be shown lined up exactly as it would be on a screen using a fixed-size font. If more than one or two words (such as quoting a menu-choice) is shown, then the whole block of text should be started on a separate line, separated from the surrounding text by empty lines, and, if space permits, indented from the left margin.

Rule 8: Backus-Naur syntax notation should be used for computer literal strings where there is some user-selectable variation in the content of the command, such as for parameters in the command string. In Backus-Naur format, the following symbols are common:

"< **variable parameter** >" A type of bracket-pair used to denote a variable or user-defined input parameter.

"[**statement elements**]" Square brackets used to denote an optional set of statement elements.

"|" A vertical "pipe" symbol, used to separate "either-or" elements of a statement.

"..." An ellipsis, used to denote the optional addition of multiple elements of the same type as the one preceding it.

For example, the use of Backus-Naur syntax notation would convert the following sample SQL string (used to execute a stored procedure from Visual Basic) from:

"Begin Name_of_Procedure; End;"

to:

"Begin <stored procedure name>; End;"

The literal parts of the command shall be in bold, and the variable elements and Backus-Naur syntax symbols in non-bold.

Rule 9: If quote marks are part of the command or other literal string, for example, if quoting a Visual Basic STRING where the intended quote marks are to be put into the source code of a module, then the quotation marks should also be in bold.

References

The following books should be used as a foundation reference on which these standards are built:

- The Handbook of Structured Design by Page-Meillor
- The Windows Interface -- An Application Design Guide, Microsoft Press
- Programmers Guide, Visual Basic 3.0 for Windows, Microsoft Press. Specifically refer to the section "Object Naming Conventions" on pages' 34-35 for naming conventions, and chapter 6 "Programming Fundamentals."
- Database Developers Guide with Visual Basic 3, by Roger Jennings, Sams Publishing 1994

(The rules in this document override any of those in the previous two references for which there is any conflict.)

- The Windows Interface -- Software distributed by Microsoft, Inc., including:
 - Visual Design Guide, (also distributed with Visual Basic)
 - Interactive Design Guide
 - Interface Design Guide

1 COMMON USER INTERFACE FOR NMS APPLICATIONS

1.1 Graphical User Interface (GUI) Design Objectives

1.1.1 Placing Users in Control

1.1.2 Reducing the Users' Memory Load

1.1.3 Making Applications Easy to Learn and Use

1.1.4 Providing a Consistent User Interface

1.2 The Common User Interface

1.2.1 The Visual (display screen) Elements of a CUI

1.2.2 The Functional (user actions) Elements of the CUI

1.3 Screen Architecture and Components

1.3.1 Menus

1.3.1.1 Top Menu Bar

Note: The same rules and discussion for the top menu bar under the "MDI Parent Forms" "Top Menu Bar" Section applies to forms in non-MDI applications as well.

The top menu bar with drop-down submenus is based on the Lotus 123 model. Some general rules concerning top-menu bars are:

- If more than three or four items, use a separator bar to group selections logically.
- Gray out non-available choices.
- Hide choices inaccessible to the user. They need not know about such choices if the user's access/security level totally prohibits them access. Leaving such choices visible but grayed out can confuse users if they think the menu items are grayed out due to some temporary state and can be activated based on some selection or action they make later.
- Indicate depth of choice with a black right-arrow triangle (>) for a cascade submenu or ellipses ("...") for entry to a dialog box.

- Define a mnemonic hot-key for each menu choice. Such hot keys should not conflict with standard Windows hot-keys. Indicate Alt-key or letter-key mnemonic hot-keys for menu items by underlining a letter or number in the Menu choice text. Preferably, use the first letter in the menu choice text or use the first character of the second word. If these are not possible, then consider using the next consonant of the first word, and so on ...
- Do not capitalize the hot-key unless this is normally done (like the first letter of the first word).
- List any control-key hot-keys to the right of the menu choice.

1.3.2 Colors

Rule 1-1: Use colors in forms and controls, where feasible, to make the applications more attractive and easier to use. Remember, using Windows, users can select foreground and/or background colors. When designating colors, always designate both foreground and background colors to maintain contrast and avoiding users being able to select a foreground or background that ends up with, for example, gray on gray.

Rule 1-2: Do not rely on colors alone for control identification. Remember that 8% to 10% of the user population may be color blind to a significant extent. Applications should be tested using color-blind users in addition to normal-vision users.

Rule 1-3: Avoid colors or color combinations which are irritating, garish, or which make objects hard to see. Avoid placing highly-saturated colors together.

Rule 1-4: Specifically, avoid color combinations which make it hard to see text. The standard colors for text boxes is black or dark blue text on white background. Avoid colors for text such as red, magenta, cyan, except where a suitable background color can make the text easy to see and non-irritating. There should be good reasons for using non-standard colors for text (such as to help identify different areas of a form where controls are grouped functionally or logically). Such uses should be approved by the standards review board.

Rule 1-5: Text boxes should never have a grey background, unless for some specific exception approved by the standards review board.

Rule 1-6: Highlight selected text using inverted colors (such as white text on black background -- or yellow or white on dark blue background). (Such inversion of selected text is usually automatically done in most textual controls.)

Rule 1-7: Avoid light or saturated blue for text, thin lines, and small shapes.

Rule 1-8: Avoid adjacent colors that differ only in the amount of blue.

Rule 1-9: Avoid edges created only by color difference; it is hard to focus on them. Particularly bad are adjacent red and green.

Rule 1-10: Background colors for forms and frames should be light, such as the pastel colors. Title bars and borders can be any pleasant color. Borders can sometimes be any color.

Rule 1-11: Use the standard Windows method for "graying out"; text in a text box, combo box, or other textual control which is "disabled" for editing.

Exception: It might be permissible in certain special situations to set non-standard colors for the background color and foreground (text) color of such a non-editable text box. For example, the message windows in a status bar might be given a pastel or gray background with black or a dark-color foreground color for the text. Another example would be to use a grey or -- say, green -- background for a text box which is always uneditable, such as a box showing current time and date.

Rule 1-12: Avoid inversion of colors in textual controls except to indicate selected text.

Rule 1-13: Let the user customize certain common colors in the application.

1.3.3 Fonts

1.3.3.1 Minimum Font Size

Rule 1-14: Minimum font size for all controls shall be **9.8 points** in order to accommodate small, low-resolution monitors. (Color VGA is the minimum resolution supported.) A few exceptions can be made where good reason exists, with approval of the local standards review board. The user might be allowed to select a different font size for certain classes of controls as part of their user-customizable configuration, but the range of font sizes should be limited to what is practical.

1.3.3.2 Default Fonts

Rule 1-15: Use the default system fonts available. Avoid exotic fonts. Avoid fonts from third-party font packages; these might not be installed on user workstations. (It is best if developer workstations not have such font packages installed. An exception would be TrueType fonts which come with the organization's standard Windows-based word processor.)

Rule 1-16: The default font for controls shall be **MS Sans Serif** (the default font in Visual Basic). (The user might be allowed to change the default font for all controls as part of their user configuration; however, it will involve extra programming.)

Rule 1-17: The default font for documents (word-processing controls or OLE objects) shall be Times New Roman. (The user might be allowed to change the default font as part of their user customization configuration.)

Rule 1-18: The default font for message boxes is controlled by the settings in the WIN.INI file, settable by the user through Control Panel.

1.3.4 3-D Effects

Rule 1-19: Use 3-D effects for all controls where feasible -- except on labels --, to make the applications more appealing to the eye and less boring.

Rule 1-20: Lighting should appear consistently, in 3-D effects, to come from over the user's left shoulder.

Rule 1-21: Do not use the 3-D effect DLL provided with Visual Basic to give 3-D effect to dialogue boxes. (It has bugs; there are several versions of it from different vendors with different bug-fixes and/or bugs; Microsoft has stated that it will not support it nor correct any bugs in it; and it slows the application.)

Rule 1-22: Labels should appear to be printed on the background of a form or frame panel. To this end, for label controls, avoid the use of 3-D effect and make the label background color the same as its parent form or frame. Use the Elastic frame to provide this functionality.

Rule 1-23: Label controls which are actually used as labels for visible GUI objects should appear to be written directly on the background of their parent form or panel. Otherwise, a label can be distracting -- calling the user's attention to them as if it were a data entry field. Thus, they should not have 3-D boxes around them, and their background colors should be the same as the background of their parent form or panel.

1.3.5 Text Display Selection and Formats

1.3.5.1 Selection from Text Lists

Rule 1-24: Enable look-ahead typing where feasible. When the user is typing in a string to look up a name or value in a list, except where this requires that each keystroke start a new query against the database. In the latter case, optimization techniques can be used to implement this, if deemed useful enough.

Caution: This may require intercepting a **Control_KeyPress()** event-handler and/or the use of extra state variables to control the search functions and to keep the search-ahead routine

from being called again before it has completed the current search. This may require optimization techniques.

1.3.5.2 Dates

Rule 1-25: Dates should be displayed in the "mm/dd/yyyy" format, like "12/21/1994"
-- especially in data entry/edit boxes.

1.3.6 Tabular Data Display

1.3.7 Objects

Rule 1-26: Document which graphic files go into a form. A copy of each icon or bitmap file used to build a toolbar or otherwise used in a Windows application are stored in the source code directory of each application in which it is used. Each form's descriptive header (described in the "Coding Standards (Visual Basic)" Section) should list which graphic files are used for which icons in which controls.

1.3.8 Graphs

1.3.8.1 Contents of Graph Screens

Rule 1-27: Graph: A graph shall be presented in a separate, typically rectangular, box on the screen with a medium blue border. The screen background should be black or white (See "Printing" Section below). Vertical and horizontal metrics or gradients shall also be inside the graph box using medium blue uppercase letters. The graph "box" should be as large as practical, and positioned in the upper, left portion of the screen.

Rule 1-28: Graph Title: A graph title shall be fully explanatory to any reader. The title will typically be segmented by major-to-minor subject nouns, uppercase, left adjusted, and positioned above the graph "box." An "Information Date:" with date, will be on the same line, right adjusted.

Rule 1-29: Legend: The legend, relating colors to graph sub-elements, will be outside the graph "box," to the right side of the screen and in a columnar list form, with color samples aligned and to the left of the legend description or indicator word(s).

1.3.8.2 Graph Screen Functionality

Rule 1-30: Each graph screen shall provide a user the capability to generate and print a report, with appropriate headings and formatting, of the values used to generate the graph.

Rule 1-31: Where applicable, the user shall have the capability to generate additional views for different time periods, such as fiscal year, month, etc.

Rule 1-32: The user shall have the capability to "toggle" between the graph and the report used to generate the graph.

Rule 1-33: Graphs shall be printable on a black and white printer or optionally to a color printer.

1.3.8.3 Colors and Styles

Rule 1-34: Colors depicting values or graphs shall be used in the following order:

- | | | |
|----------------|----------------------|---------------|
| 1. Medium Blue | 5. Purple (Magenta) | 8. Cream |
| 2. Yellow | 6. Light Blue (Cyan) | 9. Dark Green |
| 3. Light Red | 7. Orange | 10. Gold |
| 4. Light Green | | |

Though the above color selection list allows up to ten colors, the recommended maximum number of colors (values) used is six.

Rule 1-35: Any graph using a value which represents a plan, goal, standard, average, trend, etc., shall represent that value in light green.

Rule 1-36: Deviations from plan, goals, standards, average, trends, etc., (variance) shall be represented by incremental amounts; using yellow (caution) for a small variance, and light red (warning) for larger variances.

Rule 1-37: When printing a graph (black and white) which depicts plan, goal, standards, average, trends, etc., shall use the following "fill (or substitute) characters to represent the indicated colors.

| <u>Color</u> | <u>Fill Character</u> |
|--------------|-----------------------|
| Light Green | • |
| Yellow | * |
| Red | + |

1.3.9 Help

Rule 1-38: "Help" assistance shall be provided using the Microsoft Windows methodology. Text will be generated using Microsoft's "Word" word processor and implemented with key word hypertext using the RoboHelp tool. Context-sensitive help will be used where

applicable and to the greatest practical extent.

1.4 User Navigation

1.4.1 Keyboard Methods as Alternatives to Mouse

Rule 1-39: The user should use the keyboard for as many operations as possible -- especially in data entry applications.

1.4.2 Drill-Down Methods

Rule 1-40: A user should be able to "drill-down" to a more detailed view of a data set by double-clicking on a cell or through the **View** menu item. This will usually pop up another form displaying detail data -- often consisting of child records (from a one-to-many table relationship) or shown in a grid or spreadsheet control.

1.4.3 Look-up Methods

1.5 Window Forms

1.5.1 General Rules

1.5.1.1 Resolution/Size

Rule 1-41: Window forms should be designed so that they will fit in the lowest-resolution monitor supported (standard VGA Color @ 640x480 Pixels). Limit maximum window size at design time to 625x370 pixels. Limit MDI child window sizes accordingly to fit inside such an MDI parent form.

1.5.1.2 Labels

1.5.1.3 Exiting Forms

Rule 1-42: Each form should be provided a **Close/Cancel** button on the form. The button should say **Cancel** if data has been modified and may need saving. The **Cancel** button should not exit the form. Exiting a form with modified, unsaved data on it should pop up a dialog box asking the user whether they want to save changes first. Setting up this usually requires a Boolean state-control variable for each form called **blnDataModified**.

1.5.1.4 Window Titles

Note: Backus-Naur notation is used in the following title-bar specifications.

Rule 1-43: The title in the title bar of an MDI parent window or non-modal window in a non-MDI application, should use the following format recommended for the new MS GUI model in Windows 4.0 (code-named "Chicago"):

[<Document> | <Item Name>] and/or [<Current Window>] : <Application>

Obviously, the application name in the title bar should be short or abbreviated.

1.5.1.5 Graphics on Forms

Rule 1-44: Limit the use of bitmaps on the window forms, except the opening "distraction" and/or "main menu" window forms. Bitmaps use memory and slows screen repainting. Load such bitmaps at runtime to conserve memory and executable file size and unload the form as soon as possible. If possible, use .WMF files instead of Bitmap files for least memory use and maximum screen repainting speed.

1.5.1.6 Automatic Resizing

Rule 1-45: Unload window forms when possible, unless they are called up repeatedly and it is desired to make them appear faster when called -- in which case, make the decision to unload or to hide (or reduce) the form based on memory availability. This can be done at design time for expected minimum configurations or, more efficiently, at run-time by using Win API calls to check memory and resources.

1.5.1.7 3-D Appearances of Forms

Rule 1-46: Forms designed for applications should have a 3-D appearance, preferably with background colors used to help identify the forms and make the application more useful.

Rule 1-47: Avoid using special 3-D controls just for the sake of 3-D effects. Simpler controls will work with the Elastic frame or Tab control to give desired 3-D effects.

Rule 1-48: Avoid using the MicroHelp 3-D controls due to slowness in screen repainting.

Rule 1-49: Avoid using the Sheridan 3-D controls. Specifically, do not use the Sheridan 3-D frame panel.

Rule 1-50: Avoid using the set of controls in file THREED.VBX.

1.5.1.8 Distraction/Program Entry Forms

Rule 1-51: Each application -- or application subsystem consisting of several applications with a common main menu window form -- should show the user a "distraction screen" window form while program initialization is being done. This form should be full-screen and contain an attractive, graphical, full-screen image. The database logon form, if needed, should pop-up over this distraction form. The image should be loaded from an image file at runtime to reduce memory overhead and size of the executable file. After logon and initialization, this form should be unloaded. Use Windows Metafiles (.WMF) for graphic images, where possible, for minimum size and fastest resizing speed.

1.5.1.9 Main Menu/System Entry Forms

Rule 1-52: Each Business Area's software system shall provide a system-entry application containing a "main menu" window-form, shown after or as part of the initial distraction form (after the database logon form) to select a task (a Windows application) or task-group (a subsystem of applications) from large icons with titles. This acts to reduce confusion and provides a more user friendly alternative to normal Program Manager "groups" (of application icons). This also allows the developers to split the software into multiple, separate applications to reduce memory and .EXE files size. It can reduce user confusion from being offered too many choices at once. Since the icons can be picture boxes much larger than normal icons, detailed art images, scanned images or logos to reduce confusion and speed user access.

The main menu system entry application should be unloaded after a user selects a task-application, but each spawned application should provide a means of getting back to that main menu application through the "**Window**" menu item.

The main menu form can contain a large image in its background, but this image and the menu-icons should be automatically resized proportionally to fit different resolution monitors in full screen mode. A recommended technique is to use the Windows API bitmap resizing subroutines (BitBlockTransfer functions). Font sizes for menu item labels on such a form should also be automatically adjusted.

1.6 MDI Parent Forms

1.6.1 Top Menu Bar

Rule 1-53: The MDI parent form's top menu bar shall have a minimum of the following standard top-level menu items:

File -- This shall provide at least the minimum functions

- Database**
- Edit**
- Window** -- This shall show a list of currently open window forms within the application.
- Help**

In addition, where they are appropriate, the following extra top-level menu items may be used.

- View** -- This shall offer the user a means of changing the view, such as Zooming a view of the currently-selected control on the currently active form, or bringing up another form to provide further detail of the current document, work item or data.

Rule 1-54: The **Help** menu item shall be placed at the far right. This can be done at run-time by **HelpMenu.Caption = Chr\$(8)**. At design time, this can be done by pasting a backspace character created in Write by pressing **Alt** and numeric **keypad 8**.

1.6.2 Toolbar

Rule 1-55: The MDI parent form shall have a toolbar at the top, underneath the top menu bar, to enable the user to quickly select often-used actions. This toolbar shall use a set of icons in an order; these shall be standardized in common among the various Windows applications developed for this organization. Most of these icons will act as shortcuts to menu items, saving the user from navigating through multiple levels of the top menu bar. The icons on the toolbar should appear as 3-D raised push buttons. The toolbar might look something like this one from an A&A application:



Rule 1-56: Document which graphic files go into a form. A copy of each icon or bitmap used to build a toolbar or otherwise used in a Windows application should be stored in the source code directory of each appropriate application. Each form's descriptive header (described in the Coding Standards section) should list which graphic files are used for which icons in which controls. (This rule is repeated here for emphasis.)

Rule 1-57: For each icon on the toolbar, a help-bubble should appear after a brief time delay when the user moves the mouse cursor over the icon, describing the function of that icon.

1.6.3 Standard Toolbar Icons

Rule 1-58: The standard set of icons in a toolbar for common operations shall be as follows: (Some sample icons are shown, but not all examples are in the required 3-D appearance.)

Rule 1-59: Icons of 3-D objects should appear to face obliquely to the left, as in the printer icon at the right.



Toolbar Icons Corresponding to File Menu Items

Open a folder or a file: -- An open folder



Print -- Print the data in the current window.



Save to disk -- An arrow pointing to a diskette.



"Edit" Toolbar Icons

The edit functions of the toolbar (cut, copy, paste) should be accessible from the main, top menu bar as well as under *Edit*.

Edit-Cut -- A pair of scissors



Edit-Copy -- An arrow to a clipboard, or a camera.



| | | | | |
|----------------|----|--|---|---|
| Edit-Paste | -- | A paste-pot or an arrow from a clipboard to a page. |  |  |
| Undo last edit | -- | A U-turn arrow or eraser |  |  |
| Search | -- | A pair of eyeglasses or field glasses is common for this operation (including Search and Replace). |  |  |

Note: The magnifying glass symbol without a question mark is commonly used for "Zoom" function -- zooming a view -- however, with a question mark in it, it is commonly used for search. To eliminate possible user confusion, it is better to reserve the magnifying glass (without the question mark) for "Zoom" -- especially for the smaller, toolbar icons, in which it is harder to make the distinction.

Database Action Toolbar Icons

Use the appropriate action for a particular window form's task. The data record functions **Logon**, **Password**, **Next**, **Previous**, **First**, **Last**, **Insert New**, **Update**, **Delete**, and **Cancel** should also be accessible from the top menu bar. In addition, the forms which require the database functions **Insert New**, **Update**, **Delete**, and **Cancel**, should also be provided by clearly marked command buttons on each individual MDI child form.

| | | | |
|------------------------|----|---|---|
| Insert New Record | -- | An arrow showing insertion of one item between two others. |  |
| Delete Current Record | -- | An arrow showing removal of one item from between two others. |  |
| Update, commit changes | -- | a check mark, optionally with the letters "OK" |  |
| Cancel | -- | all changes to current record or work item |  |

| | | | |
|--------------------------------------|----|-------------------------------------|--|
| Previous | -- | Go to previous record or work item |  |
| Next | -- | Go to next record or work item |  |
| Go to beginning of current recordset | -- | A rewind Button |  |
| Go to end of current recordset | -- | A fast-forward button |  |
| Connect to Database | -- | |  |
| Password | -- | Change the user's database password |  |

Other Common Toolbar Icons

| | | | | |
|---|----|---|---|---|
| Attach a "Post-it" Note (to the current record or work item). | | |  |  |
| In-basket | -- | Open a task-specific workflow in-basket form. |  | |
| Show List | -- | Lookup from table |  | |
| Query | -- | Execute an ad-hoc query against the database |  | |
| Exit the application | -- | An open door with an arrow or a person walking out of it. |  | |

Help



1.6.4 MDI Parent Status Bar

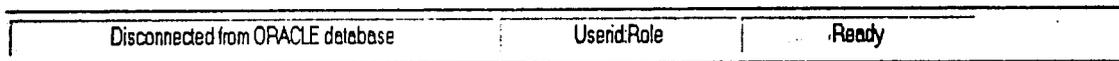
Rule 1-60: Each MDI parent form shall have a status bar at the bottom. This status bar shall contain as a minimum, the following:

Message window: At the left side of the status bar, a single-line text window shall display messages sent from the application to inform the user of the current state of things, to explain the use of the currently-selected edit window or other controls, to recommend the next action and provide guidance.

Although Visual Basic provides custom controls to show keyboard statuses, such as **Caps-Lock** or **Num-Lock**, it is recommended that they not be used. If they are used, the standard place for them is on the status bar at the right.

Below is an example of status bar from an A&A MDI parent form:

The second text window in this status bar shows the current user's database user-id.



Another useful item to show in this status bar is system (current) time and date.

Controls

1.7.1 Command Buttons

Rule 1-61: Captions for command buttons should be short, direct and clear. They should be verbs. A Hot-key using the *Alt*-key with a mnemonic letter key should be defined as a keyboard alternative for each command button. The use of these hot-keys should be consistent from form to form to avoid the user getting used to performing an action by hitting one key and being surprised when that key performs a different action in another form. Common hot-key usages in Windows applications are:

| | |
|--------------|----------------------|
| <i>Alt-O</i> | OK |
| <i>Alt-C</i> | Cancel |
| <i>Alt-X</i> | Exit the application |

Certain other, common hot-keys are defined in the Windows Interface Design Guide -- most commonly for top menu bar items, such as *Alt-F* for **File**, *Alt-E* for **Edit**, *Alt-W* for **Window**, and *Alt-H* for **Help**. Care must be taken to avoid conflict with key combinations.

In addition, the following keys have common meanings:

Esc Close the current lookup-window or menu window with nothing selected.

See the "Standard Control Placement on Forms" Section for rules on command button placement.

Common Command Buttons

| Caption | Hot Key | Function | Alternate Icon |
|-----------------|---|---|---|
| C lose | <i>Alt-L</i> or <i>Alt-C</i> (1) | Close the current window. |  |
| O k | <i>Alt-O</i> | Shall be used only for acknowledgement, not for "do-it" or "execute". |  |
| C ancel | <i>Alt-C</i> | Cancel the changes to the current record. |  |
| U ppdate | <i>Alt-U</i> | Write the current record to the database. |  |
| A dd New | <i>Alt-A</i> | Create a new document or record. |  |
| D elete | <i>Alt-D</i> | Delete the current record. |  |

| Caption | Hot Key | Function | Alternate Icon |
|-------------------------|--------------|--|---|
| Do It or Execute | <i>Alt-G</i> | Perform the selected operation, such as a user-defined ad-hoc query. |  |

(1) *Alt-C* is the standard hot-key for Cancel, but the **Cancel** button on a data-entry/edit form can change caption to "Close" when data on the form is in an unmodified state, and vice-versa.)

1.7.2 Standard Control Placement on Forms

1.7.2.1 Command Buttons

Rule 1-62: Command buttons shall be placed at the right or bottom of the form or panel in which they are used. If at the bottom, they should be centered or, preferably, at the right. It is important that the user get to expect that a *Close* button to close a window-form is always in the same place -- at the far right bottom.

Rule 1-63: Command buttons in a group should all be the same size, if possible.

Rule 1-64: Avoid one set of command buttons on one form looking the same but working differently or vice versa, as a similar or identical set of buttons on another form. For instance, **Add**, **Update**, **Delete** and **Cancel** buttons should always be in the same order from one form to another.

1.7.2.2 Grouping of Controls

Rule 1-65: Group controls on a form according to their logical or functional use or category. Use Elastic frames (preferable to normal Visual Basic frame panels), frame panels, or Tabs controls to section off different groups of controls on a form. Try to avoid drawing lines on a form or frame; because its `AutoRedraw` may have to be set on, slowing screen updates.

Rule 1-66: Make the focus of user action flow from left to right, top to bottom, both within and among groups of controls.

Rule 1-67: Arrange data-entry/edit controls within a group -- and arrange groups of controls on a form -- according any required sequencing necessary for data lookup or validation.

Rule 1-68: Avoid overcrowding of a form.

1.7.2.3 Labels and Label Controls

Rule 1-69: Labels associated with certain controls should generally be positioned according to the following guidelines:

- Group Box (such as a frame) -- on top of or in the upper left corner replacing the frame line.
- Single-field controls -- to the left of, or above, the control.
- Command button captions -- inside the command button.
- Check boxes or option boxes -- to the right of the button or box.
- All others -- above or left of control.

1.8 Data-Entry Controls

1.8.1 Speeding Data Entry Using Defaults

Rule 1-70: For certain date fields, it may make sense to automatically fill in the current day's date as a default, or to at least put in the current year. For date fields, the user should be able to type in just two digits of the year as a shortcut, and the application would automatically convert this to the correct four-digit number on the user leaving that field. This is a simple algorithm. A type-ahead correction can even be implemented for certain date fields. If the user starts the year with a "9" then expect a two-digit year like "199N" and change the "9" to "199"; otherwise, if the digit begins with a zero, one or two, expect the year to be like "20nn" and change the "1" or "2" to "201" or "202" before the user keys in the next number.

1.8.2 Speeding Data Entry Using Pop-up Windows

Rule 1-71: A user should be able to select certain data values from a pop-up dialog box form. On such a form, the user should be able to make a selection with a double click or by highlighting an item and clicking an "OK" button or pressing the *Alt-O* key. Alternatively, the user should be able to avoid selecting anything by clicking the "Cancel" button or using the *Alt-C* or *Escape* key.

Rule 1-72: A user should be able to pop up a calendar by double-clicking or clicking the

right mouse button on a date field. Selecting a date from the calendar should close the calendar and write the selected date into the date field box.

Rule 1-73: A data entry/edit box which can have only a fixed set of values should be implemented using a list box. Alternatively, if more detailed information must be displayed for each item in a list, or the list must be made up dynamically or "on-the-fly" by reading from the database based on other user-entered data values or selections, a modal dialog box can be popped up with a scrollable grid from which the user can select. Avoid codes in lists, use text names. The user should never have to memorize "codes". Even if the database or the software uses numeric or alphanumeric codes, always present a list of English-language descriptive item name to the user.

1.8.3 Speeding Data Entry Using Masks

Rule 1-74: Zip codes, telephone numbers, and social security numbers should be implemented as text fields (using masked text boxes), not numbers -- both for data entry boxes and in the database -- in order to maintain correct sort-orders. Such fields should have non-numeric characters filtered from the user input -- best done in the control's **KeyPress()** event handler subroutine.

2 CODING STANDARDS (VISUAL BASIC)

2.1 Standardizing Existing Code

These standards shall be applied to any new code developed. Existing code will be changed as time permits, to conform to this standard, then regression-tested to verify that it still functions correctly. Certain rules in these standards are so critical that they shall be applied to every module. These important coding standards are:

- The inclusion of the **Option Explicit** statement at the beginning of every module file
- The setting of each VB (Visual Basic) programmer's Environment Options to include **"Require Variable Declaration = Yes"**
- The setting of each VB (Visual Basic) programmer's Environment Options to include **"Default Save As Format = Text"**, and the conversion of existing source code stored as binary to text format
- The setting of each VB (Visual Basic) programmer's Environment Options to include **"Save Before Run = Yes"**
- The inclusion of descriptive headers at the beginning of all source code files and complex subroutines
- Explicit declaration of every variable, each on a separate line with the type explicitly declared
- Adherence to source code naming conventions -- particularly name prefixes. (See Appendices A & B)

All variables, GUI objects, data objects, and functions should be made to conform with the defined naming conventions. Search and replace, either globally or within a module or subroutine, is the best way to do this, but extreme care must be taken not to corrupt working code. A problem with global search and replace in VB is that, since VB search and replace operations are not case-sensitive, the programmer make unintended modifications to many defined constants at once, when only lower-case variable names were intended to be changed. Global search-and-replace is generally safe for words or substrings that are long and whose uniqueness is obvious -- especially for ones that include one or more underscores.

2.1.1 Syntax Conventions

This document specifies placing commas and periods outside quotation marks where confusion might arise about whether the punctuation mark is part of: a literal computer command, SQL query, a literal string in a programming language, or other string or text intended to be strictly literal.

Programming source code samples or programming language keywords shall be in **bold**. Where fixed-size font is necessary to maintain the original appearance and indentation, use "Courier New" 2-12-point font. Source code samples are indented from the left margin. DOS commands or other user-keyboard input shall be in **bold italic** and indented from the left margin. Examples of screen display text should be in **bold "Courier New"** 2-14-point font and indented from the left margin.

Where it is necessary to indicate optional parameters in examples of DOS commands or source code examples, Backus-Naur syntax notation should be used to indicate these options. (See the section on Backus-Naur notation in the "Document Conventions" Section under "Document Standards.")

2.2 Declarations

2.2.1 Variable Declarations

Rule 2-1: Variables shall always be declared before use.

Rule 2-2: Variable declarations shall contain "As <datatype>" to explicitly declare the data type.

Rule 2-3: Each variable shall be declared on a separate line with its own explicit data type declaration.

Rule 2-4: Variable declarations shall be followed by a comment defining its meaning when the name is not sufficient. The comment should also describe any peculiarities of the variable or the way it is used.

2.2.2 Constant Declarations

Constant (and variable) declarations should be grouped logically or functionally. Global constants must, of course, go into a global (.BAS) file. Groups of constants will have a header comment block to define their commonality and purpose. Each constant will have a comment describing its meaning or use if this is not totally obvious from its name and grouping. Constants that are not used (for instance, the VB constants found in the CONSTANT.BAS file) should be commented out.

2.2.3 DLL/API Subroutine Declarations

Warning: Be very careful in using subroutine prototype declarations for external DLLs. The ones which come with Visual Basic 3.0 in the files WIN30API.TXT and WIN31EXT.TXT have several incorrect data type declarations -- specifically several where the As **String** or **By**

Val As String is used where a pointer or reference to a user-defined type (data structure) should be used. This comes about from Microsoft's conversion of C-language function prototypes to Visual Basic function prototypes, where in old C, a pointer to a string (**char ***) was also used as a generic pointer to anything -- the same way that **"void *"** is now used in C and C++ as a generic pointer.

The rules for declaring parameters in Visual Basic are not clearly laid out in any one book. In Visual Basic, pointers are not used, but passing parameters "by reference" is the same thing.

Rule 2-5: Always explicitly declare the data type of all subroutine parameters, using the same rules as for variable declarations. (See "Variable Names" under the "Naming Conventions" Section .)

Rule 2-6: The correct way to declare a user-defined-type parameter passed by reference to an external API subroutine is to declare the parameter as data type **"As Any"** or **"As <Type name>"**, as in the following example from file WIN30API.TXT:

```
' Parameter Block description structure for use with subr. LoadModule():
Type PARAMETERBLOCK
    wEnvSeg As Integer
    lpCmdLine As Long
    lpCmdShow As Long
    dwReserved As Long
End Type

Declare Function LoadModule Lib "Kernel" (ByVal lpModuleName As String, \
    lpParameterBlock As PARAMETERBLOCK) As Integer
```

This could also have been done as:

```
Declare Function LoadModule Lib "Kernel" (ByVal lpModuleName As String, \
    lpParameterBlock As Any) As Integer
```

Rule 2-7: The correct way to declare a string buffer type parameter in an external API subroutine is to declare it as **"By Val"** and data type **"As String"**, as in the following example from file WIN30API.TXT:

```
Declare Function WinExec Lib "Kernel" (ByVal lpCmdLine As String,
    ByVal nCmdShow As Integer) As Integer
```

Rule 2-8: Never pass a VB **String** object to an external DLL subroutine without first allocating a memory block for the string. (There are certain exceptions where certain DLLs have subroutines specifically designed to take VB string parameters by reference and which

can dynamically allocate memory for these strings the same way that VB does.) Note that some DLL subroutines have an extra parameter for passing the size of the buffer.

Rule 2-9: Always terminate a VB **String** object passed to an external DLL subroutine with at least one C-type null character or **Chr\$(0)**. For fixed-size buffers fill out the end of the buffer with C-type null characters. (A C-type null character is a byte of numeric value zero, not the numeric character "0".)

Rule 2-10: Use constants to define the size of fixed size string buffers passed to an external DLL subroutine and in all references in the code to the buffer size.

2.3 VB Environment

2.3.1 Option Explicit

Rule 2-11: Set the "Require Variable Declaration" environment option in Visual Basic. This causes the **Option Explicit** statement to be inserted at the beginning of any new VB source code modules (.BAS or .FRM) created.

Rule 2-12: The **Option Explicit** statement must be at the head of each source code module (including form files). For ease in QA verification, this should be before the **Option Explicit** statement and the module's descriptive header.

2.3.2 DefInt A-Z

Rule 2-13: The statement **DefInt A-Z** shall be inserted at the beginning of each module file (including form files) -- preferably, for ease in verification, before the module descriptive header and after the **Option Implicit** statement.

2.3.3 Save As Text

Rule 2-14: Set the **Default Save As Format = Text** environment option in Visual Basic.

Rule 2-15: Convert any existing source code modules saved in binary format to text format.

2.3.4 Option Base

2.3.5 Option Compare Binary vs. Option Compare Text

2.3.6 Save-Before-Run

2.3.7 Grid-Alignment Size = 60

2.3.8 Config.Sys: Files

Rule 2-16: Set **Files=<N>**, where N = 75 to 2-125, in file "CONFIG.SYS" on developer workstations.

2.4 Source Code Documentation

2.4.1 Code Commenting

Rule 2-17: Language in comments should be short but clear. It can be informal. Correct spelling and grammar should be used because such errors interrupt the flow of reading.

Rule 2-18: Comments will be used to provide detail about what a subroutine call or section of code does and why, when this is not obvious from the subroutine name or from variable or object names in the code section. This is especially important where variable, object, or variable names need to be kept short and do not convey enough information about a process.

2.4.2 Standard Header Templates

Rule 2-19: Descriptive header sections -- using large comment blocks -- should be used at the beginning of each module file and at the beginning of all subroutines of any size to help make the code self-documenting. (A module file here means a source code file with a **.BAS** or **.TXT** suffix or a form file with a **.FRM** suffix.) Extremely short, simple subroutines, such as VB event-handlers, may not require a descriptive header. Subroutines with no parameters or with parameters whose meaning is obvious, may require only one or two lines of comment to describe what it does.

These headers shall list and describe global variables used or affected; database tables read or modified, and any file I/O.

Subroutine headers shall list and describe input parameters, output parameters, and return values. Parameters passed to a subroutine should be described in detail when their meanings are not obvious or when they need to be in a specific range. They shall include flexible-format, narrative sections that provide enough detail to enable a new programmer to understand the intended functionality. This description shall not describe the implementation details (how it does it) because these often change over time, resulting in unnecessary comment maintenance work, or worse yet -- erroneous comments; the code itself through local comments will describe the implementation.

There are four different standard header templates -- one each for:

- **Procedure** Procedure-type subroutines. These do not return a value; although they

can pass back values through their parameters. In VB they are called a **Sub**.

- **Function** Function-type subroutines. These return a value.
- **VB Module** A Visual Basic source code module, conventionally stored as a text file with a **".BAS"** suffix.
- **Form Module** A Visual Basic "Form" module defining a VB windows-form and containing source code conventionally stored as a text file with a **".FRM"** suffix.

Copies of these VB standard templates are found in files FUNCTION.TEM, SUBROUTN.TEM, MODULE.TEM, and FORM.TEM in the common, re-usable code repository on S:\COMMON\VBSOURCE\TEMPLATE. Examples of their use are given below.

Rule 2-20: Each descriptive header block shall identify who created the module or subroutine. It shall also list briefly and clearly describe the gist of the modifications in different revision levels of the module or subroutine, who modified it, when and why. Use the automatic change-description prompting (on module check-in) and keyword-expansion features of the VCS (version control system) to automate this process for modules.

Rule 2-21: The comment header block at the beginning of a module shall be of a standard header template format. It should give the file and module name, explain the purpose of the module; the common functionality of the classes, constants, global variables, and subroutines grouped within it.

Rule 2-22: The comment block at the beginning of the application's main module (the module containing the **Main()** subroutine for the application) will give it an overview description of the application, enumerating primary data objects, routines, algorithms, dialogues, database and file system dependencies, etc., and shall be included at the start of the **.BAS** module file that contains the project's Visual Basic generic constant declarations. An example follows, with sample source code in bold and comments for purposes of this document in non-bold:

The following standard header template for a form (**.FRM**) file contains sample variable, constant, subroutine, and table specifications.

```

'-----
' FILE NAME: <Module name>.BAS (for modules or <Form name>.FRM for forms)
' FORM NAME: frm_AcctSpecial (for form modules only)
'
' APP:    <application name> | Shared
'
' PURPOSE: The purpose of this module is ...
' It contains subroutines with the common purpose of ....
'
' GLOBAL SUBROUTINES EXPORTED: (only for .BAS modules, not for .FRM modules)
'   function  errUpdateAcctSpecial  (sample subroutine names)
'   function  lngGetNumSpecialAcctRecs
'   sub       DisplayAcctErrMsg
'
' GLOBALS IMPORTED:
'   g_arrAccountRecs      -- an array of account records
'   g_ds_Creditors        -- a dynaset of creditors associated with this
'                           gang.
' GLOBALS EXPORTED: (only for .BAS files)
'   g_arrAccountRecs      -- an array of account records
'
' FILE I/O:    (optional)
'
' TABLES READ:    (optional if none read)
'   (The following optional mapping to controls or grid columns is to be used only for VB form files:)
'
'   Map to
'   Table      Attribute      Control/Grid Column      Description
'-----
' <Table1 Name> Field_1      <Grid Column 2-1>      (sample specification text)
'                   Field_2      <Grid Column 2>
' <Table2 Name> Field_1      <Grid Column 3>
'
' TABLES MODIFIED:    (optional if none written to)
'-----
'HISTORY:
' Created by: Henry Hull, Nov. 04, 1994
' Modified by: Henry Hull, Dec. 25, 1994      Revision 1.1
'   Added safety check for null value in global dynaset fields:
'   g_ds_Creditors("Acct_No").
'-----

```

Rule 2-23: The comment block at the beginning of a subroutine should give its name and describe its functionality. It shall identify input parameters, output parameters, and return values. In addition, it will identify any global variables, database tables, files accessed or modified, and any other unusual I/O. An example is given below.

Recommendation: The comment block at the beginning of a subroutine preferably should be inside the subroutine just after its declaration line, indented 1 level, or outside the subroutine, just before the subroutine declaration, not indented but flush with the left margin.

Sub UpdateAccountSpecial (curPayment As Single)

```
'-----  
' This subroutine processes a payment for special case individuals.  
' It has this special functionality ..... and .....  
' It uses the input .... and ... to update a matching record in the  
' global array g_arrAccountRecs.  
,  
' ASSUMES:  
' This subr. assumes that the global dynaset and global array  
' listed below have been initialized.  
,  
' INPUTS: curPayment -- the payment being made for the  
' (List each non-obvious parameter on a separate line with in-line comments.)  
' OUTPUTS: A printed exception report is printed  
' (List each non-obvious output parameter or other output not listed below on a separate  
' line with in-line comments. Do not list function return values here.)  
,  
' RETURNS: data type -- description  
' (Use this only for functions, not procedures (called Sub in VB)).  
,  
' SIDE EFFECTS: (optional)  
' List each non-obvious external effect. This is usually obvious through  
' the list of globals affected, file output, and tables/fields affected.  
,  
' GLOBALS USED:  
' g_arrAccountRecs -- an array of account records  
' g_ds_Creditors -- a dynaset of creditors associated with this  
' gang.  
' GLOBALS MODIFIED:  
' g_arrAccountRecs -- an array of account records  
,  
' FILE I/O: (optional)  
,  
' TABLES READ: (optional if none read)  
,  
' TABLES MODIFIED: (optional if none written to)  
' (If any tables are affected, list specific fields affected.)  
'-----  
' HISTORY:  
' Created by: Henry Hull, Nov. 04, 1994  
' Modified by: Henry Hull, Dec. 25, 1994  
' Added safety check for null value in global dynaset.field:  
' g_ds_Creditors("Acct_No").  
'-----
```

2.4.3 Internal Comment Blocks

Rule 2-24: The code within long subroutines shall include normal comments explaining lines or data objects and section headers composed of comment blocks highlighted using lines formed of "-", "=" or "*" characters. Use comment blocks to make long subroutines more readable by breaking it up into sections, such that a programmer can quickly scan through the code, understanding the main logic flow without having to read detail, and to quickly get to a particular section where modifications may be required. A good technique is to insert a pseudocode description of the subroutine's algorithm into the subroutine body, split it up into comment blocks, adding explanatory text where appropriate, and then fill in the section of code underneath each comment block. Comment blocks should stand out visually. Examples of comment blocks follow.

For small modules, or comments embedded in large modules use:

```
'-----  
' ACTION SUMMARY STATEMENT  
' Comment detail ...  
'  
'-----
```

For large modules, where the need exists to use a higher level of comment blocks or to call attention to one or more particular sections, use the following style comment-block:

```
'*****  
' ACTION SUMMARY STATEMENT  
' Comment detail ...  
'  
'***** *****
```

or

```
'=====
```

```
' ACTION SUMMARY STATEMENT  
' Comment detail ...  
'  
'=====
```

Rule 2-25: The right end of a comment block should not be closed to form a box, since this takes too much time to manually reformat lines when making changes inside the comment blocks. (It also creates larger source code files because of the large number of trailing spaces required.)

Rule 2-26: Place the name of the subroutine in an in-line comment at the end of each subroutine that is longer than about five lines or which might cross page boundaries in printouts.

2.5 Indenting Source Code

Rule 2-27: Indent all code within a subroutine at least one level.

Rule 2-28: Indent all code inside a control structure, such as a loop or **If-Then-Else**, one level. VB **Select-Case** statements will be indented as in the following example:

```
Select Case Var1:
  Case 1:
    Statement 1...
    Statement 2...
  Case 1:
    Statement 3...
    Statement 4...
End Select
```

Rule 2-29: Four-space indenting shall be used for Visual Basic source code. Three is the ideal, but requires all programmers to set their VB environments the same. The same indenting should be used throughout a project.

Rule 2-30: When nesting control structures, put a comment at the end of each control structure -- and at the beginning if it helps -- to identify its beginning and end.

Rule 2-31: If using a programming editor, such as Brief, WinEdit, or Visual SlickEdit, the tab option shall be set to use all spaces and not to fill with tab-characters. Otherwise, printed source code will not look as it appeared on screen. There is virtually no discernable difference in compile times between using tabs and using spaces.

2.6 Naming Conventions

Rule 2-32: Names shall be shortened, within reason -- using standard abbreviations where necessary -- and should clearly indicate the meaning and use of the object named.

Rule 2-33: Except for constants, individual words within a name shall begin with a capital letter, and the rest will be lower-case letters or numbers.

Rule 2-34: Avoid acronyms in object names, where possible, except for a very small list, such as "FY" for fiscal year, "BY" for budget year, and "OY" for operating year. However, the statements declaring and using acronyms should always have explanatory comments. Always program for newcomers. (A list of these standard acronyms will be posted in an appendix later.)

Rule 2-35: Generally avoid underscores in names, except:

- after the prefix of a control name or a GUI object,
- following an acronym (such as "TEC" for total estimated cost) where the acronym must be clearly set off from the beginning capital letter of the next word or symbol in the name,
- where grouping words or symbols within a name makes the meaning clearer.

Exception: The use of underscores to separate words inside object names can be allowed where it helps readability, but the recommended standard is to avoid them within variable names. The use of underscores has the disadvantage of making names longer.

A variation of Hungarian notation (lowercase prefixes followed by an underscore) is recommended for GUI objects and database objects, since this provides quick recognition of those type objects versus variables. These are discussed in separate subsections below.

2.6.1 Name Prefixes

Rule 2-36: Use the standard name prefixes to indicate the type of any data object, database object, or GUI object.

Rule 2-37: Global variables and data objects shall be further prefixed with a "g" or "g_", as in example global database object `g_db_AIDMain`.

2.6.2 Constant Names

Rule 2-38: Constants shall be named using all uppercase letters. Numbers and underscores can be embedded in them.

Rule 2-39: Do not prefix global constant names with a "G" or "G_", since there are too many global constants already defined in VB and the Windows API that do not use this convention.

Rule 2-40: For a group of new constants that refer to a common class of objects or to a common API or custom control(s), prefix the constant names with an abbreviated, mnemonic code to indicate their common usage. A good example of this is the "DB_" prefix on all VB database constants, the "MAPI_" prefix to VB MAPI constants, or the "SS_" prefix to all FarPoint spreadsheet constants in the file "FPSREAD.BAS."

2.6.3 Variable Names

Rule 2-41: Use the standard Hungarian notation prefixes listed in Appendix B for variable names to indicate the data type or use. (These prefixes are based on Microsoft's recommended prefixes.)

Rule 2-42: Global variables shall be further prefixed with a lowercase "g". It is recommended to not use the underscore character to separate the "g" from the rest of the prefix.

Rule 2-43: Do not separate a variable name prefix from the rest of the variable name with an underscore.

Rule 2-44: Use the same rules for naming all subroutine parameters as for variable names (except that the "g" for global is not appropriate).

Rule 2-45: When referencing cells, rows and columns within a grid via a loop, always use loop-counter names like `iRow` and `iCol` instead of `i` and `j`. This makes the source code far more readable.

2.6.4 Subroutine Name Prefixes: Indicating Data Types

Rule 2-46: Procedure-type subroutine names do not need prefixes, since they do not return data.

Rule 2-47: Subroutine names shall sound like commands or verbs and shall have an object to reflect the data object acted on. This applies to both procedure-type (**Sub**) and function-type (**Function**) subroutines. For example, a function name can be like `intGetCurrentTotalEstimCost()`, which fulfills both requirements.

2.6.5 Names for Database Objects

Rule 2-48: Use the "Hungarian"-style notation naming prefixes from the table in Appendix B for database-access type objects.

Rule 2-49: The prefix for a database object shall be separated from the rest of the name with an underscore "_" character.

Rule 2-50: Names of database objects should clearly reflect the name of the object in the database (table-name, view-name, field-name, etc.) such as ("Account," "Country," etc.) which it accesses or displays. Names of database objects should generally sound like nouns.

2.6.6 Names for GUI Objects -- Forms and Controls

Rule 2-51: Names of GUI objects (forms and controls) shall be prefixed using standard prefixes listed in Appendix A.

IMPORTANT NOTE: A form name shall always be prefixed with "frm_". Appendix A defines variations to use for MDI parent and child forms. When first saving a newly named form, since the "frm_" prefix will be in the name, VB will try to name the form's file starting with "FRM_", thus using some of the eight characters available for a filename prefix. The programmer must manually override this in the VB File/Save As operation, the first time the form file is saved.

Rule 2-52: The prefix for a control or form shall be separated from the rest of the name with an underscore "_" character to distinguish it from prefixed-names of variables.

Rule 2-53: Third party controls shall be identified by including an additional two-letter pre-prefix in their name identifying their vendor. Standard prefixes for third party controls are listed in a table in Appendix A. This list is subject to revision as third-party controls are added to USAID's standard set of tools.

Rule 2-54: The main part of the control name should indicate clearly in near-English the use of the control. An underscore shall separate the prefix from the rest of the name, but underscores shall not be used within the prefix nor within the suffix. For instance, a text box might be **txt_CalculatedDepth**, or a label connected to a data control field might be **lbl_StartDate**. A Crescent Software CSDate custom control text box would be **csDate_Completed**, and a CSLabel might be called **csLbl_TimeLapsed**. A Sheridan Software spreadsheet control might be **ss_TimeItems**; although if there is only one tabular control (grid or spreadsheet) on a form and its use is clear, sometimes it might be acceptable to simply call the grid or spreadsheet **Grid_1**, **fpGrid_1**, or **ss_1**.

Rule 2-55: A VB data control's name shall indicate the table or view to which it is connected. If multiple databases are accessed in one application, the data control's name should indicate the database as well. (A data control is a GUI object, and is different from the Database Access Objects.)

Rule 2-56: Use **dat_** or **data_** as a prefix for data controls as defined in the second table in Appendix B.

Exception: One allowable exception to the custom control naming convention is the collection of simple labels on a form which are not referred to in the code so their names do not need to indicate any specific functionality. In this case, the default names **Label1**, **Label2**, etc., are quite adequate, although **lbl_1**, **lbl_2**, may be slightly preferable. (The programmer will want to make this collection of general labels a control array to reduce memory overhead.)

2.6.7 Menu Naming Conventions

Rule 2-57: All menu item names shall be prefixed with "mnu," as in the examples below.

Rule 2-58: Menu control prefixes shall be extended beyond the initial **mnu** label by adding an additional prefix for each level of nesting, with the final menu caption at the end of the name string.

2.7 Subroutine Design, Coupling and Cohesiveness

2.7.1 General

2.7.2 Global Variables

Rule 2-59: The scope of variables and constants shall be limited as much as possible to only those modules using them. Variables and constants declared within a VB form-module but outside any subroutines automatically have form-scope. In **.BAS** modules, the keyword **Global** shall be left off any module-scope variables not intended to be used outside the module.

Rule 2-60: Since global variables increase the possibilities for error to software development and maintenance phases, the use of them shall be limited as much as possible and values and variables passed as subroutine parameters where possible.

Rule 2-61: List all global variables -- used in, modified in, or exported from a module -- in the module's descriptive header block. (See "Code Commenting" under the "Source Code Documentation" Section.)

Rule 2-62: Never use short, cryptic names or single-letter names for global variables. An exception is that just "gdb" can be used as a global database object if there is only one database accessed in an application and only one database object is needed.

Rule 2-63: Never use global variables for loop counters or variables intended to be local in scope. Never define a local variable with the same name as any higher-scope variable.

Rule 2-64: Where form-scope variables are used, there shall be comments in each subroutine which references them to denote their scope. If it is possible to show their scope in the name prefix, then do so. (Refer to the "Variable Names" Section of this chapter.)

Rule 2-65: Generally, avoid declaring subroutine parameters or other local variables with the same name as any global, form-scope or module-scope variables. (Using the "g_" prefix in the names of all global variables' names should take care of this for global variables.)

2.7.3 Global Subroutines

2.7.4 Private/Local Subroutines

Rule 2-66: Limit the scope of subroutines to only the modules required to access them. (In VB 3.0, a subroutine is either global or module-scope.) Use the keyword **Private** in the declaration of any subroutine meant to be called only from within its own ".BAS" module. In a form module, this does not apply, since subroutines, variables, and constants in a form cannot be global.

2.7.5 User-Defined Data Types

2.7.6 Constants and Variables -- Scope

2.8 Procedural Coding Standards

2.8.1 Concatenation Operators

2.8.2 Goto Statements

Rule 2-67: **GoTo** statements shall be avoided, except for forward **GoTo** used for error trapping (as in **On Error Go To <label>**). (See the "Database Error Trapping" Section .)

Rule 2-68: Never use a backward **GoTo**.

Rule 2-69: Never use a **GoTo** statement to exit a loop or a process control structure (such as an **if-then-else** or **Select-Case**) -- except for forward **On Error GoTo** statements used for error trapping as referred to above.

Rule 2-70: Never attempt to use a **GoTo** statement to jump between subroutines.

2.8.3 Error Trapping

Rule 2-71: The programmer shall trap each and every database access and file access statement or group of statements using a forward **GoTo** statement pointing to an error-handling routine at the end of the subroutine performing the access (as in **On Error Go To <label>**).

Rule 2-72: Any code sections requiring error trapping via an **On Error Go To <label>** statement preceding it, shall have an **On Error Go To 0** (zero) following it to reset the error-trapping.

Rule 2-73: Each local error handling routine (at the end of a subroutine) should call a common, global error messaging and/or error logging routine for consistency in error reporting, to reduce code size, and to increase maintainability.

Rule 2-74: All error codes without error messages shall be translated through a translation table.

Rule 2-75: End each local error-handling routine with either of the following:

On Error Resume Next
or
On Error GoTo 0
Exit Sub

Rule 2-76: Each source code label (target of **GoTo**, not a label control) in an application should have a name that identifies what object it is reporting an error for, or what kind of error it is, and its purpose or context.

2.8.4 If-Then-Else Structures

2.8.5 Loading Text into Combo Boxes, List Boxes and Grids in VB

2.8.6 AutoRedraw Property

2.8.7 Sending Messages from One Form to Another in VB

2.8.8 Writing Text to Labels in VB

2.8.9 Modularization -- File Organization

2.8.10 Data Management

2.8.10.1 Validating Data Entry

Rule 2-77: Data to be sent to the database shall be validated in the application before being sent to the database, according to known business rules and referential integrity constraints, even though the same rules and constraints are programmed into the database. There are several places in the application where it may be necessary to perform data validation checks:

- On entering a form or record.
- On leaving a form or record.
- On entering a data field entry/edit box.

During each keystroke in a data field entry/edit box.

Some of these validations or "edit-checks" are performed on the immediate field, some are cross-checks between fields, and some are cross-checks between records or with other tables.

Additional data validations might need to be performed before updating/committing the edited record, in a data control's **Update()** event-handler subroutine.

Rule 2-78: Data shall be validated for each record before the user attempts to commit that record to the database. Data validation should be performed in the data-entry/edit window forms on a field-by-field basis and just before the update. Dates and numbers especially should be checked for correct ranges of values. Some dates must be before or after other dates in the database. For example, edit checks should prevent the user from entering an end-date which comes before a start-date for the same activity. The pop-up calendar mentioned elsewhere should be sent begin-dates and end-dates to define a range from which the user can select.

Rule 2-79: Document data validation rules in the source code. The edit-checks performed and the business rules or references to the sections of the requirements specifications containing the pertinent business rules should be listed and described in the descriptive header for each form or validating subroutine. Any ranges of valid values, where static and known, should be defined. Careful analysis of the business rules and good, common sense are necessary to decide which edit-checks to perform where and when.

Rule 2-80: Message boxes should notify the user as soon as possible of invalid data and why it is invalid. If a specific field is invalid or missing, the focus should be set to that field after the error message.

Rule 2-81: Users should be prevented from entering invalid characters or values -- before the edit checks. This can be done by using special controls from Crescent Software for entering numeric, currency, time, and date data, or by calling special keystroke-filter functions in the control's **KeyPress()** event-handler.

Rule 2-82: Provide lookup lists, where applicable, to assist user memory and to limit their choices. Especially where a name or code must be from a fixed, finite list of valid values, such as from a lookup table in the database, the user should not be allowed to enter the value, but should be forced to select from a pop-up list box, a drop-down list box, or a pop-up, modal selection form containing a grid or list box.

Rule 2-83: Lookup lists should not be hard-coded into the applications. They should be loaded from the database, and just once if the same list is used over and over again. (See recommendations for performance improvement for such lookup lists under "Increasing

Database Performance" and in the "Database Access and SQL Coding Standards (SQL)" Sections.)

Rule 2-84: Lookup lists should show descriptive names, not codes; although certain lists can show both if there is a valid reason.

Rule 2-85: If the data in one field is required to be complete and valid before certain other fields should be filled in, then make sure that the required fields come before those needing the required fields in the flow-of-focus among controls and among groups of controls.

Rule 2-86: If the data in one field is required to be complete and valid before certain other fields should be filled in, then make sure that the required fields are correctly filled in before the dependent fields.

Rule 2-87: Control the user's navigation through a data entry/edit form, where some fields are prerequisite to others. While, according to GUI event-driven methods, the user should be able to use the mouse to move at random among fields or to use the *Tab* key and *Shift-Tab* key to move past certain fields or to go back, the reality for data entry/edit forms is that certain fields must be filled in and validated before others (dependent fields) can be accessed.

Based on the above two rules, these corollaries follow.

Corollary 1: If a prerequisite field's data has changed, then the data in the dependent data-field controls should be revalidated and, if not valid, cleared and disabled until re-edited. The user should be notified as to the new editing requirements.

Corollary 2: If a prerequisite field has not been entered and validated, then the dependent data-field controls should be disabled from editing until the prerequisite field has been completed and validated. This can be done by disabling the edit control or by intercepting the mouse clicks and keystrokes for it. The disabling mechanism should also accommodate Corollary 3.

Corollary 3: If the user tries to edit a dependent field before any prerequisite fields have been completed and validated, then the user should be provided with message boxes explaining why one cannot edit the field and what missing fields are required.

2.8.10.2 Debugging SQL

Note: Different rules may be required for databases other than Oracle.

Rule 2-88: Always trap errors for all database accesses and provide an error message that shows the error message, any SQL code associated with the access, and the name of the

subroutine where the error occurred. See the "Database Error Trapping" Section for more details.

Rule -89: When faced with hard-to-debug database access errors involving SQL, temporarily implement a trace mechanism. Either one of the two following mechanisms is recommended:

1. Load an ODBC spy utility before testing the VB application.
2. Capture the translations of SQL code which ODBC sends to a server DBMS, in a local log file called "SQLOUT.TXT." In the <App. Name>.INI file (create one if it does not exist), create an "[ODBC]" section, and under that insert the line, **SQLTraceMode=1**, as in:

```
[ODBC]
SQLTraceMode=1
```

Remove this entry later or set

```
SQLTraceMode=0
```

because it adversely affects performance, and the log file will eventually get too large.

2.8.10.3 Safely Interpreting / Converting Field Values

Rule 2-90: When reading the value of a field of a database table (also called an attribute of a database entity) from any of the database objects in Visual Basic, the access should be performed through a common method (encapsulated in a function-subroutine) in order to test for **NULL** or invalid values and to convert them, where applicable, to default values and datatypes. Also, other specific data validations or conversions specific to a particular data type can be performed there.

Rule 2-91: Field Error-Trapping: Each database field-access subroutine shall have customized error-trapping to trap database access errors and provide a standard format database error message. Error messages shall include the field name, the ODBC or RDBMS error message -- as provided by the VB **Error\$(Err)** function -- and the SQL statement generating the record set. The SQL statement shall be passed to the subroutine as a parameter along with the field name. The calling subroutine shall decide whether to terminate itself, terminate/unload its window-form, or terminate the application. Also, it is up to the calling subroutine to handle or ignore invalid data.

These field-access functions should return an error code and should pass back the field value as

a parameter. The error code shall be a globally-defined constant **DB_SUCCESS=0** (zero) if successful, or one of a number of standard error codes if unsuccessful. These constants and functions shall be defined globally, in a common **.BAS** module.

Rule 2-92: A different set of subroutines must be built (although on the same model) for each of the VB database object types **Dynaset**, **Snapshot**, and **Table**; because the database object must be passed to the subroutine as a parameter; however, part of the error-message handling can be put into a common, lower-level subroutine. These common database field access routines shall exist in a common module maintained and shared among the several Business Area teams.

The conversion rules within these subroutines shall be as follows:

Numeric String Fields: Some numeric fields are implemented as strings for formatting reasons, such as telephone numbers, social security numbers, and zip code numbers. These shall be treated as text strings, but special subroutines can be written to retrieve and validate each of them. See the rules for text fields.

Some other straight numbers might be implemented in database as numeric strings for some reason. Such fields shall be converted to numbers and otherwise treated as ordinary numeric fields. That is, if the field is **NULL**, a zero value shall be passed back.

Rule 2-93: Numeric Fields: Numeric and Boolean fields should be checked for **NULL** values. If the field is **NULL**, a zero value shall be passed back.

Rule 2-94: All Field Types -- Errors: If a field-access subroutine is called expecting a certain data type and the data type of the field is another, incompatible data type, then a message shall be displayed notifying of the difference in data types.

Rule 2-95: For all errors when accessing a field, an error code of **ERR_FIELD_DATA_TYPE** should be returned and a standard-format error message displayed from within the field-accessing function.

Note: If Visual Basic ever allows conditional compilation using compiler directives, then debug code could be stripped out when creating a deliverable, executable application.

Rule 2-96: The user should be given the ability to kill all field-related error messages following the first one for a record. Otherwise, the user is inundated with a barrage of error message windows.

Boolean Fields: If a Boolean field value is **NULL**, the value shall be passed back as **False**

(zero). If the field is numeric and the value is zero, the value should be passed back as **False**; else, if the value is any other numeric value, the value should be passed back as Visual Basic **True**.

If the field type happens to be of type character and the value is a lower- or upper-case "y", "t", "yes", or "true", then value should be passed back as **True**; else the value should be passed back as **False**.

For any other strings or data types which cannot be interpreted as Boolean, the value should be passed back as **False** and an **INVALID_DATA** error code returned. This probably means the program source code is in error and needs to be updated because a field data type changed.

Byte Fields: A Byte data type shall be considered as an unsigned integer of eight bits. If a byte field value is **NULL**, the value shall be passed back as 0 (zero). If the field value is numeric but outside the normal range for an eight-bit unsigned integer (as can happen when the data type of the field is actually an integer, for instance), an error message should be shown, a serious error code returned, and the value should be passed back as either +255 if it can be translated as a positive integer or Zero value if a negative integer. This probably reflects a program logic error, due to its being out of sync with the database.

Integer Fields: If an integer field value is **NULL**, the value shall be passed back as 0 (zero). If the field value is numeric but outside the normal range for a integer (as can happen when the data type of the field is actually a long integer, floating point or currency type), an error message should be displayed in a modal message box, a serious error code returned, and the value should be passed back as either Maximum Integer value if positive or Minimum Integer value if negative.

Long Integer Fields: If a long integer field value is **NULL**, the value shall be passed back as 0 (zero). If the field value is numeric but outside the normal range for a long integer (as can happen when the data type of the field is actually a floating point or currency type), an error message should be displayed in a modal message box, a serious error code returned, and the value should be passed back as either Maximum Long Integer value if positive or Minimum Long Integer value if negative.

Single Floating Point Number Fields: If a single-float field value is **NULL**, the value shall be passed back as 0.0 (zero). If the field value is numeric but outside the normal range for a single floating point number (as can happen when the data type of the field is actually a double floating point), an error message should be displayed in a modal message box, a serious error code returned, and the value should be passed back as either Maximum Floating Point value if positive or Minimum Floating Point value if negative.

Double Floating Point Number Fields: If a **double-float** field value is **NULL**, the value shall be passed back as **0.0** (zero).

Currency Fields: If a **double-float** field value is **NULL**, the value shall be passed back as **0.0** (zero).

String (text) fields: If a **string** field value is **NULL**, the value shall be passed back as "" (empty string).

Memo (Unlimited Size Text) Fields: If a **memo** field value is **NULL**, the value shall be passed back as "" (empty string).

Date Fields: If a **date** field value is **NULL**, the value shall be passed back as Visual Basic **Null**, and no further interpretation shall be attempted. It shall be up to the calling subroutine to handle this.

If a date field is a string, then if it can be converted to a valid date, then this date should be passed back, otherwise a Visual Basic **Null** should be passed back along with an **INVALID_DATA** error code.

Exceptions: The **BLOB** (binary large object) a data type for which no translation rules are set in this standard. Local subroutines can be built to handle specific instances of such fields. Other, specific exceptions to the above data types needed on a local basis can be encapsulated in local subroutines built on the same model.

2.8.10.4 Programming Rules for Database Reliability

Rule 2-97: When setting the **RecordSource** property of a data control programmatically (by assigning a table name, view name, or SQL query) care should be taken to assure that a non-empty recordset is returned, in order to avoid error conditions in bound controls. One method for the programmer to safely handle empty record sets is to first test for the number of records returned using a query like one of the following:

"SELECT COUNT(*) FROM ..."

or

"SELECT COUNT(DISTINCT) FROM ..."

Since this technique increases the size of the executable file and slows down the total time for database retrievals, this technique is recommended only where there is a need for it.

Rule 2-98: When performing or refreshing a query returning a record-set -- either for a database access object of type **Table**, **Dynaset**, or **Snapshot**, or for a data control -- the

record set returned should be tested for zero count.

Rule 2-99: If zero records are returned for a query used for editing data, then one of the following actions should occur, either:

- The user is prompted to create a new record for data entry, or
- A new record is automatically created for the user to edit.

The first is strongly recommended. The edit controls should be disabled until a new record is added or another, non-empty dataset is returned. If the user chooses not to create a new record, then one should be able either to try a new query or to exit gracefully.

Rule 2-100: Likewise, when a user enters a data-edit form with no current record (and where they did not get there as part of their selecting an "Add New Record" function), prompt the user to create a new record or to gracefully exit the form. All error conditions should be trapped and the user offered clear messages and graceful exits. (See the "Database Error Trapping" Section.)

Rule 2-101: A user should always be able to cancel and leave a data edit/entry form without saving the new or changed record. The edit controls should be disabled until a new record is added or another, non-empty dataset is returned.

2.8.10.5 Increasing Database Performance in Visual Basic

Note: This section will require review and update to incorporate new methods and information about problems and solutions with current versions of ODBC drivers and other database access programming tools.

Rule 2-102: In each application, open only one, global database object per database accessed. This saves memory and resources.

Rule 2-103: In each application, use a single set of global string variables, **g_strDatabaseName** and **g_strConnect**, to be shared by the global Database object and any data controls used. This saves programming and minimizes memory used for strings.

Rule 2-104: For data controls, set the **DatabaseName** and **Connect** properties at run-time in their form's **Form_Load()** event-handler subroutine.

Rule 2-105: To make forms with data controls load faster, set the **RecordSource** property of a Visual Basic data control in the **Form_Load()** event-handler to an SQL query which returns either the exact recordset needed -- if that is known at form-load time -- or a minimal number of records. (Care should be taken not to return an empty recordset, if possible.)

2.8.10.6 Database Error Trapping

See the section on "Accessing the Oracle Database" for more general rules about error trapping. (Certain rules and recommendations are the same for other kinds of errors, such as file i/o errors.)

Rule 2-106: Common subroutines shall be built to encapsulate the parsing and presentation of database access error messages. Such error message routines shall parse the VB error messages for DBMS-specific (such as Oracle or SQL Server) error messages or error codes, which the ODBC driver does not translate, and shall translate them further if necessary. All error codes without error messages shall be translated through a translation table.

Rule 2-107: If the database error message is DBMS-specific, the message box title shall indicate this by naming the DBMS. If an Oracle-specific error code is returned, this can be detected by looking for the "ORA-" keyword at the beginning of the VB error string given by `Error$(Err)`. Then the Oracle error number can be parsed out and translated. The message box title should then change from "Database Access Error" to "Oracle Database Access Error"

Rule 2-108: Database access error message routines should name the database object (table, view, field) being addressed. (See the detailed rule for field-access error messages in the section on database field-access routines.) This is necessary for programmers during development, for testers, and for the help-desk in handling problem reports.

Rule 2-109: Database access error message routines and error log files should report the SQL statement which caused the error.

Rule 2-110: Database access error message routines should report the name of the subroutine where the error occurred (not a low-level routine like a common field-access routine, but the subroutine calling that, so that the context of the error is apparent). The user may be given the option to turn this feature on or off, but it should be in any error-log file. It is a necessary aid in software defect reporting and tracking, especially during test.

2.9 Coding to Minimize Memory Use and Executable File Size

Rule 2-111: Load forms only when needed and unload them as soon as possible when no longer needed.

Exception-Rule 2-111a: Where an application's use of memory is not a problem and where a particular form is shown often, keeps the form loaded but hidden (or minimized if an MDI child form) so that it activates faster; otherwise reactivating these forms can become quite annoying to users.

Rule 2-112: Use non-dynamic control arrays for common labels on a form that are not data-bound and which do not need to be addressed programmatically. (This is a particular case of the previous rule.)

Rule 2-113: Avoid using the animated control button, as it uses large resources and slows down execution.

Exception Rule 2-113a: Use it, if at all, only in small applications that will not grow larger and in which the animated button's slowing down of the application is not a problem.

Rule 2-114: Use the image control rather than the picture control, where possible, to save system resources and screen redrawing. (The picture control's **AutoRedraw** property must be turned on, and this redrawing significantly slows down processing.)

Note: This rule is repeated in the coding section under the subsection about saving memory and resources.

Exception Rule 2-114a: Use the picture box control only when its extra functionality is required and that need is greater than the impact of other memory and resource problems in the same application.

Rule 2-115: Load medium-to-large bitmap images from files at run-time -- preferably after a form is loaded but before it is shown.

Rule 2-116: Avoid using 3-D controls (such as those in the VB file THREEED.VBX) where a simpler control can be given 3-D-appearance in a simpler, less resource-intensive and less processing-intensive manner (such as are recommended in the following paragraph). Other disadvantages of most 3-D controls include not being able to use any other background color other than grey.

Rule 2-117: Load data only as needed. Use dynamic arrays rather than fixed size arrays where possible, resizing them to zero immediately when no longer needed.

Caution: In actuality, it may be impossible to resize an array to zero elements, once it has been initialized. The VB statement **Redim Array(0)** actually just resizes the array to one element -- the zeroeth element -- if Option Base zero -- the default -- is used, or causes a program error if Option Base is higher than zero.

Caution: Keep dynamic arrays of data- or record-structures (in Visual Basic, these are called "user-defined types") which use variable length strings out of forms; place them in **.BAS** modules. There is a bug in Visual Basic 3.0 where redimensioning such arrays can cause GPFs (General Protection Faults) due to faulty object-destructor mechanisms.

Rule 2-118: For fixed-size arrays or large data objects used in only one form, keep those data objects in the form where they are used. Avoid making them global, unless necessary. This way, they disappear when their form is unloaded.

Rule 2-119: Call Windows API functions and subroutines where this achieves one of the following and imposes little or no penalty on code readability or maintainability:

- significantly saves memory,
- significantly minimizes executable file size,
- avoids using a custom control,
- significantly speeds processing,
- significantly speeds processing.

Rule 2-120: Combine multiple instances of medium-to-long strings or long sub-strings.

Rule 2-121: Avoid Recursion. See the discussion of this rule which is repeated in the next section on "Coding to Maximize Performance (Execution Speed)".

2.10 Coding to Maximize Performance (Execution Speed)

Rule 2-122: Set the **AutoRedraw** property of forms and picture boxes to **Off**, unless dynamically drawing lines on the form. Avoid drawing lines on forms if performance is a problem. (Using frame panels or Elastic panels might be a better way of grouping related controls.)

Rule 2-123: Use Image controls rather than PictureBox controls where possible. (This rule is repeated in the section on saving memory.)

Rule 2-124: Avoid using animation button controls.

Rule 2-125: Avoid or minimize calls to **DoEvents()** within a loop. Otherwise, place calls to **DoEvents** within a subroutine only where it is necessary.

Rule 2-126: Avoid recursion. Recursion risks running out of stack space--especially in Visual Basic. Whatever can be implemented using recursive function calls can usually be implemented using loops, which will generally run faster.

2.11 Shared, Common, Reusable Source Code

A repository of reusable code shall be created and maintained on a globally accessible server with full configuration management and change control methods used to manage it.

Rule 2-127: The standard VB global constants file CONSTANT.BAS shall be split into multiple files, one each for constants pertaining to OLE, MAPI, MCI (and MIDI), DDE, VB 3D controls, the MSCOMM VBX, and one file called CONST_VB.BAS for all the VB constants left over.

Rule 2-128: All files containing global constants shall be named beginning with "CONS" or "CONST". Thus, the VB database constants file DATACONS.BAS shall be renamed CONSDATA.BAS.

Rule 2-129: All global constants and API function declarations in these files, not commonly used, shall be commented out.

Rule 2-130: Constants and subroutines which are related to database objects, including the Visual Basic database constants, shall be kept in a file named DATA.BAS.

Rule 2-131: Windows API function declarations shall be kept in a file called WINAPI.BAS, combined from WIN30API.TXT and WIN31EXT.TXT, but with all unused API function declarations remarked out.

Rule 2-132: Global variables, constants, and subroutines that are functionally related (such as a VB API to load and manipulate a spreadsheet) shall be grouped together in a separate .BAS file -- especially if these source code objects are expected to be shared with (reused in) another application. One example of this would be where a set of constants and data structures were defined for budget objects in a budget application, where it would be valuable to re-use them in a financial application which referenced the same tables in the database and thus would be expected to reference the same objects and handle the same set of values.

Rule 2-133: Borrow code from the Software Re-Use Repository and from other sources, such as CompuServe, Internet, E-mail, Microsoft Developers Library CD, other knowledgebases on CD-ROM, friends, consultants, and other government agencies.

Rule 2-134: Share code thought to be reusable by sending it to the manager of the Software Re-Use Repository. Before sending it, debug it completely and make it stable, of good quality, and reliable; and make sure it conforms to the Agency software development standards.

3 DATABASE ACCESS AND SQL CODING STANDARDS

This section is concerned only with database access programming methods that result in increased database reliability, performance, and SQL code maintainability. For database standards and guidelines, see the "ORACLE DATABASE ADMINISTRATION STANDARDS" Section of this document.

3.1 SQL Calls to the Database

Rule 3-1: Use database Views where appropriate, to improve performance or decrease programming. Views should not be created dynamically from within a program.

Rule 3-2: Use database stored procedures to retrieve multiple record sets by creating a procedure with a cursor inside of a loop that passes records one at a time through ODBC to an array in VB. After the procedure has finished populating the array, the records in the array can be used for list boxes or other functionality.

Rule 3-3: Convert ALL SQL commands to stored procedures including SQL that:

1. Selects multiple records from a table.
2. Performs database write-operations.
3. Returns SQL aggregate function values such as count(*).
4. Contains write or batch routines that alter the database without user interaction or input.
5. Populate list boxes.
6. Contain database write-operations intended to be completed together within a single COMMIT / ROLLBACK group, rather than the Visual Basic BeginTrans, CommitTrans, and Rollback commands.

3.2 Creating and Using Indexes

Rule 3-4: Concatenate any character to an indexed column to disable the index and force the execution of a full table scan. For example:

```
WHERE SALARY||'' > 12000;
```

Rule 3-5: If more than 25% of the rows in a table are going to be returned, use a full table scan rather than an index.

Rule 3-6: Never do a calculation on an indexed column if the intention is to use the index to

assist with response time. For example:

Inefficient way to code:
WHERE (SALARY * 12) > 12000;

Efficient way to code:
WHERE SALARY > (12000 / 12);

Rule 3-7: Never specify **IS NULL** or **IS NOT NULL** on index columns if the intention is to use the index to assist with response time.

Rule 3-8: Never specify the **SUBSTR** function on a column that has an index because it disables the index. For example:

Inefficient way to code:
WHERE SUBSTR(USER_ID,1,4) = 'OPSS'

Efficient way to code:
WHERE USER_ID LIKE 'OPSS%'

Rule 3-9: In almost all cases the use of **ORDER BY** will disable the use of an index and result in a full table scan. For this reason use a **WHERE** clause condition that uses an index instead of an **ORDER BY**. The records will be ordered the same way the index is ordered. For example:

Inefficient way to code:
ORDER BY EMP_NO

Efficient way to code:
WHERE EMP_NO > 0

3.3 ORACLE SQL Statement Processing Techniques

Rule 3-10: Use **ROWID** as a key for a record when ever possible. The **ROWID** for a record is the single fastest method of record retrieval. **ROWID** is actually an encoded key representing the physical record number within an actual ORACLE database block on the database.

Improvements in performance can be made by selecting a record before updating or deleting it, and including **ROWID** in the initial select list. This allows ORACLE to perform a much more efficient second record access. Remember to select the record **FOR UPDATE** when querying a record prior to updating or deleting. This keeps another process from being able

to update the selected record and change its **ROWID** out from under you. For example:

```
SELECT      ROWID, ...
  INTO :EMP_ROWID
  FROM EMP
  WHERE EMP.EMP_NO = '123'
  FOR UPDATE OF EMP.EMP_NO;
```

```
UPDATE EMP
  SET EMP.EMP_NO = '456'
  WHERE ROWID = :EMP_ROWID;
```

Rule 3-11: Use a where clause that utilizes indexes. For example:

```
SELECT ...
  FROM DEPT
  WHERE EMP_NO > 123;
```

If **EMP_NO** has an index, the index will be used and will return records in **EMP_NO** order.

Rule 3-12: Avoid using **NOT** in any where condition such as "**!=**" or **NOT EQUAL**. For example:

Inefficient way to code:
WHERE AMOUNT != 123

Efficient way to code:
WHERE AMOUNT < 122
AND AMOUNT > 124

Rule 3-13: Avoid the use of **HAVING** in general; use **WHERE** predicates instead.

Rule 3-14: Use table aliases to prefix all column names.

Rule 3-15: Use joins in preference to sub-queries.

Rule 3-16: The ordering of the from clause can in many situations significantly reduces the number of physical reads needed to execute SQL statements. **ORACLE 7** uses a cost based optimizer which in some cases makes its own determination of which table will be the driving table regardless of the order in the **FROM** clause.

The last table name specified in the from clause determines the driving table.

ORACLE creates a set of pointers to records that satisfy the **WHERE** conditions that relate to the last table in the from clause.

Then it eliminates the pointers that don't point to records, that satisfy the **WHERE** conditions, that relate to the second to the last table in the **FROM** clause.

Therefore make sure that the table specified last in the **FROM** clause will return the fewest rows based on its where conditions. This is not always the table that has the fewest rows in it. For example:

```
SELECT ...  
FROM TASKS A  
  WHERE A.EMP_NO in (1,2,3) Pointer set 3  
  AND  A.EMP_NO in (1,2)  Pointer set 2  
  AND  A.EMP_NO = '1';  pointer set 1
```

Set 1 will return the fewest records, set 2 will return more records that set 1, and set 3 will return the most records.

Rule 3-17: When using the **OR** operator be sure to put the column that will return the smallest number of rows first. For example:

```
Where 1 = 1    Should return the least rows.  
or   emp_no < 100  Should return to most rows.
```

Rule 3-18: Specify a sort-order in a SQL query, either by using an **Order By** or specifying the sort-field in the **Where** clause if an index exists on the field or if the required index exists on the set of fields.

The advantage of the **Order By** is that it makes the ordering requirement in the code more obvious and specific for future programmers. Here is an example of using a field that has an index on it to sort the records.

```
Select NAME, POSITION, DEPTNO  
from PERSON  
Where EMP_NO > 0
```

Rule 3-19: Never specify **IS NULL** or **IS NOT NULL** on index columns. It is unnecessary and will disable the use of indexes.

APPENDIX A - Standard VB GUI Object Name Prefixes

Standard VB GUI Object Name Prefixes

For GUI controls which are part of the standard VB set, use the following prefixes to indicate a control's type. These are based on Microsoft's recommended GUI object prefixes.

Separate the prefixes for GUI objects from the rest of the control's name with an underline to make it clear that the object is not a variable. This helps readers in quickly scanning the code to understand it.

(All source code prefixes and examples are in bold .)

| Non-3D VB Controls | | | |
|---------------------------|-------------------------|---------------------|-----------------------------------|
| Control Type | Prefix | Example | Comment |
| Animation button | ani | ani_MailBox | |
| Chart | cht | cht_Sales | |
| Checkbox | chk | chk_ReadOnly | |
| Combo box | cbo or combo | cbo_English | Also used for drop-down list box. |
| Command button | cmd | cmd_Sumdata | |
| Common dialog control | dlg | dlg_FileOpen | |
| Comm. | com | com_Fax | |
| Control | ctl | ctl_Current | |
| Data control | data | data_Biblio | |
| Dir. list box | dir | dir_Source | |
| Drive list box | drv | drv_Target | |
| File list box | fil | fil_Source | |
| Frame | fra | fra_Language | |
| Gauge | gau | gau_Status | |
| Graph | gra | gra_Revenue | |
| Grid | grd | grd_Prices | |

| Non-3D VB Controls | | | |
|---------------------------|--------------------|-------------------------|----------------|
| Control Type | Prefix | Example | Comment |
| Horizontal scroll bar | hsb | hsb_Volume | |
| Image | img | img_Icon | |
| Key state | key | key_Caps | |
| Label | lbl | lbl_HelpMessage | |
| Line | lin | lin_Vertical | |
| List box | list | list_PolicyCodes | |
| MAPI message | mpm | mpm_SentMessage | |
| MAPI session | mps | mps_Session | |
| Masked Edit | medt | medt_Zipcode | |
| MCI | mci | mci_Video | |
| Menu | mnu | mnu_FileOpen | |
| OLE control | ole | ole_Worksheet | |
| Outline control | out or outl | out_OrgChart | |
| Pen Bedit | pbed | bed_FirstName | |
| Pen Hedit | phed | hed_Signature | |
| Pen Ink | ink | ink_Map | |
| Picture | pic | pic_VGA | |
| Picture clip | clp | clp_Toolbar | |
| Report control | rpt | rpt_Qtr1Earnings | |
| Shape control | shp | shp_Circle | |
| Spin control | spn | spn_Pages | |
| Text Box | txt | txt_LastName | |
| Timer | tmr | tmr_Alarm | |

| Non-3D VB Controls | | | |
|---------------------------|---------------|----------------|----------------|
| Control Type | Prefix | Example | Comment |
| Vertical scroll bar | vsb | vsb_Rate | |

| VB Three-D Controls | | |
|----------------------------|---------------|----------------|
| Control Type | Prefix | Example |
| 3D check box | ch3 | ch3_CheckBox |
| 3D command button | cb3 | cb3_Close |
| 3D frame | fr3 | fr3_Shipmethod |
| 3D group push button | pb3 | pb3_Fedex |
| 3D option button | ob3 | ob3_CostPlus |
| 3D panel | pn3 | pn3_Background |

| VB Forms | | | | |
|------------------|--------------------|-------------------------|---------------------|----------------|
| Form Type | Std. Prefix | Alternate Prefix | Example | Comment |
| Form | frm | | frm_Entry | |
| MDI Parent Form | frmMDI_ | MDI_ | frmMDI_ActivityMain | |
| MDI child form | frmmdi_ | mdi_ | frmmdi_Note | |

Third Party VB GUI Object Name Prefixes

Use a two- or three-letter prefix to indicate the vendor of a third party custom control. Common examples for Agency-approved custom controls are:

| Prefix | Custom Control |
|----------------|---------------------------------|
| f sp_ | Far Point's spreadsheet control |
| f ptab_ | Far Point's tab control |
| h e_ | High Edit control |

APPENDIX B - Standard Name Prefixes for Variables and VB Data-Access Objects

Use the following three letter prefixes to indicate a variable's data type. These are based on Microsoft's recommended variable name prefixes.

Do not use an underline to separate the prefix in a variable name.

(All source code is in **bold**, including Visual Basic keywords, prefixes, and examples.)

| Variable Data Name Prefixes | | | | |
|------------------------------------|------------------------|-------------------------|---------------------|--|
| Data Type | Standard Prefix | Alternate Prefix | Example | Comment |
| Boolean | bln | bool | blnFound | Actually an Integer used as a Boolean |
| Currency | cur | | curRevenue | |
| Date (time) | dte | dt | datStart | Microsoft's "dat" conflicted with their own use of "dat" for a data control name prefix. |
| Double | dbl | d | dblTolerance | |
| Error | err | e | errOrderNum | |
| File handle | fil | fi or file | filLogFile | Actually an Integer used as a file handle. |
| Integer | int | i | intQuantity | |
| Long | lng | | lngDistance | A single, lower-case "L" ("l") looks too much like a one. |
| Object | obj | | objCurrent | Future use |
| Single | sng | flo | sngAverage | Single precision floatg point |
| String | str | s | strFName | |
| User-defined Type (struct) | rec | udt | recEmployee | Prefer use a unique, custom prefix to indicate specific data structure type. |
| Variant | vnt | var | vntChecksum | |

| Database Object Name Prefixes | | | |
|--------------------------------------|----------------------|---------------------------|---------------------|
| Object Type | USAID Prefix | Example | (MS Prefix) |
| Database | db_ | db_Main | db |
| Dynaset | ds_ | ds_Temp | ds |
| Field | fld_ | fld_Temp | fd |
| Index | ndx_ | ndx_Temp | ix |
| QueryDef | qdef_ | qdef_SalesByRegion | qd |
| Query | _Qry (suffix) | Result_Qry | Qry (suffix) |
| SnapShot | snap_ | snap_Result | ss |
| Table | tbl_ | tbl_First | tb |
| TableDef | tdef_ | tdef_Temp | td |

The above prefixes differ slightly from the Microsoft internal standard but are more readable.

* Using a suffix for queries allows each query to be sorted with its associated table in Access dialogs (Add Table, List Tables Snapshot).