

PipeLine 5.0

Technical Documentation

PipeLine 5.0

Technical Documentation

USAID | DELIVER PROJECT, Task Order 1

The USAID | DELIVER PROJECT, Task Order 1, is funded by the U.S. Agency for International Development under contract no. GPO-I-01-06-00007-00, beginning September 29, 2006. Task Order 1 is implemented by John Snow, Inc., in collaboration with PATH; Crown Agents Consultancy, Inc.; Abt Associates; Fuel Logistics Group (Pty) Ltd.; UPS Supply Chain Solutions; The Manoff Group; and 3i Infotech. The project improves essential health commodity supply chains by strengthening logistics management information systems, streamlining distribution systems, identifying financial resources for procurement and supply chain operations, and enhancing forecasting and procurement planning. The project also encourages policymakers and donors to support logistics as a critical factor in the overall success of their health care mandates.

Recommended Citation

Blankenship, Lisa, Jeffrey Leiner, and Ashraf Islam. 2010. *PipeLine 5.0: Technical Documentation.* Arlington, Va.: USAID | DELIVER PROJECT, Task Order 1.

Abstract

Develop technical documentation for PipeLine so that the application is well understood by technical persons, easily maintainable and transferrable. This will ensure that this application’s software development life cycle (SDLC) be managed effectively in future.

Cover photo: [Describe the photo, including location, year, and description of the scene or the activity that is taking place. If the photo was not taken by a project staff person, you must obtain permission from the source to use the photo here. Always include the name of the photographer or organization that provided the photo.]

**USAID | DELIVER PROJECT**

John Snow, Inc.

1616 Fort Myer Drive, 11th Floor

Arlington, VA 22209 USA

Phone: 703-528-7474

Fax: 703-528-7480

Email: ask[deliver@jsi.com](mailto:deliver@jsi.com)

Internet: [deliver.jsi.com](file:///C:\Users\lblanken\Documents%20and%20Settings\plancaster\Local%20Settings\Temp\XPgrpwise\www.deliver.jsi.com)

Contents

[Acronyms vii](#_Toc282608517)

[Acknowledgments viii](#_Toc282608518)

[Executive Summary x](#_Toc282608519)

[Introduction 11](#_Toc282608520)

[Overview 11](#_Toc282608521)

[What PipeLine Can Do for You 11](#_Toc282608522)

[Why Use PipeLine? 12](#_Toc282608523)

[Who Should Use PipeLine? 12](#_Toc282608524)

[PipeLine Software Functions 13](#_Toc282608525)

[Pipeline Monitoring 13](#_Toc282608526)

[Procurement Planning 13](#_Toc282608527)

[Technical Architecture 13](#_Toc282608528)

[Process Flows 14](#_Toc282608529)

[Appendices 14](#_Toc282608530)

[The Reddick VBA (RVBA) Naming Conventions, Version 6.01 15](#_Toc282608531)

[System Requirements 15](#_Toc282608532)

[Installation and Configuration 17](#_Toc282608533)

[Installation Instructions 17](#_Toc282608534)

[How to Install PipeLine 17](#_Toc282608535)

[Before You Begin 17](#_Toc282608536)

[Installing PipeLine from a CD 17](#_Toc282608537)

[Installing PipeLine from the Internet 18](#_Toc282608538)

[How to Start PipeLine 18](#_Toc282608539)

[Starting PipeLine from the Windows desktop 18](#_Toc282608540)

[Starting PipeLine from the Windows taskbar 18](#_Toc282608541)

[Reinstalling PipeLine 18](#_Toc282608542)

[Converting Your Existing Data Files 19](#_Toc282608543)

[Run-time installation 19](#_Toc282608544)

[Development environment installation 20](#_Toc282608545)

[Build process 21](#_Toc282608546)

[Installer 21](#_Toc282608547)

[Mechanics for Creating the CD Installer 22](#_Toc282608548)

[Create files with SageKey 22](#_Toc282608549)

[Front-end Application 23](#_Toc282608550)

[Back-end Database 23](#_Toc282608551)

[Edit .msi with InstallShield 23](#_Toc282608552)

[Create Interface with AutoRun Pro 23](#_Toc282608553)

[DownLOAD and INstall the application 23](#_Toc282608554)

[CREATE A NEW PROJECT 24](#_Toc282608555)

[Start THE Application 24](#_Toc282608556)

[CREATE A NEW PROJECT 24](#_Toc282608557)

[CUSTOMIZE THE TEMPLATE 27](#_Toc282608558)

[Apply logo 27](#_Toc282608559)

[Place PipeLine media files in a folder 29](#_Toc282608560)

[Create menu choices 29](#_Toc282608561)

[Create menu options that calls the PipeLine installer to run 29](#_Toc282608562)

[Create CD browse menu option 33](#_Toc282608563)

[Create link to external website menu option 34](#_Toc282608564)

[Create link to next page menu option 35](#_Toc282608565)

[Create second page 36](#_Toc282608566)

[Create open PipeLine User Guide PDF file in Adobe Acrobat 37](#_Toc282608567)

[Create Return to Main Menu 38](#_Toc282608568)

[Create Exit menu option 39](#_Toc282608569)

[Build the Executable AutoRun program and create the master CD 40](#_Toc282608570)

[Test the final master CD and make required number of CD copies 41](#_Toc282608571)

[Create Web Installer 42](#_Toc282608572)

[Web-download 42](#_Toc282608573)

[Application Design 43](#_Toc282608574)

[Overview 43](#_Toc282608575)

[Split Database Architecture 44](#_Toc282608576)

[Naming Conventions 44](#_Toc282608577)

[Data Types 45](#_Toc282608578)

[Graphical User Interface 46](#_Toc282608579)

[Explorer-Style Navigation 46](#_Toc282608580)

[Menu Bar Specification 47](#_Toc282608581)

[Form Conventions 48](#_Toc282608582)

[Control Use 50](#_Toc282608583)

[Command Buttons 50](#_Toc282608584)

[Report Conventions 51](#_Toc282608585)

[Report Layout 51](#_Toc282608586)

[Icons 52](#_Toc282608587)

[VBA Coding Guidelines 52](#_Toc282608588)

[Procedure/Function Declarations 52](#_Toc282608589)

[Variable Declarations 52](#_Toc282608590)

[Comments 52](#_Toc282608591)

[White Space 53](#_Toc282608592)

[Indentation 53](#_Toc282608593)

[Global Variables 53](#_Toc282608594)

[Help 53](#_Toc282608595)

[Database Schema 55](#_Toc282608596)

[Overview 55](#_Toc282608597)

[Entity Relationship Diagrams 55](#_Toc282608598)

[Table and Column Descriptions 55](#_Toc282608599)

[Other Data Structures 55](#_Toc282608600)

[Program Units 57](#_Toc282608601)

[Overview 57](#_Toc282608602)

[Source Code CD content 57](#_Toc282608603)

[Program name 57](#_Toc282608604)

[Menu Structure 59](#_Toc282608605)

[Overview 59](#_Toc282608606)

[Program calls by Menu choices 59](#_Toc282608607)

[Archiving and Backup 61](#_Toc282608608)

[Interfaces with Other Systems 63](#_Toc282608609)

[Overview 63](#_Toc282608610)

[System A 63](#_Toc282608611)

[System B 63](#_Toc282608612)

[References 64](#_Toc282608613)

[1 Intro 102](#_Toc282608614)

[2 General Properties 102](#_Toc282608615)

[2.1 Fonts 102](#_Toc282608616)

[2.1.1 Face 102](#_Toc282608617)

[2.1.2 Size 102](#_Toc282608618)

[2.1.3 Type 102](#_Toc282608619)

[2.1.4 Color 102](#_Toc282608620)

[2.1.5 Background Color 102](#_Toc282608621)

[2.1.6 Case 103](#_Toc282608622)

[2.2 Color 103](#_Toc282608623)

[2.2.1 Palette 103](#_Toc282608624)

[2.2.2 Text 103](#_Toc282608625)

[2.3 Dimensions 104](#_Toc282608626)

[2.3.1 Screen 104](#_Toc282608627)

[2.3.2 Regions 104](#_Toc282608628)

[2.3.3 Controls 104](#_Toc282608629)

[2.4 Flow 105](#_Toc282608630)

[2.5 Usability 105](#_Toc282608631)

[2.5.1 Keyboard vs. Mouse 105](#_Toc282608632)

[2.5.2 Section 508 105](#_Toc282608633)

[2.5.3 Performance 105](#_Toc282608634)

[2.6 Internationalization 105](#_Toc282608635)

[2.6.1 Multi-Language support 105](#_Toc282608636)

[2.6.2 Date & Time 105](#_Toc282608637)

[3 Applications 106](#_Toc282608638)

[3.1 Desktop 106](#_Toc282608639)

[3.1.1 Page 106](#_Toc282608640)

[3.1.2 Pop Up Dialogs 109](#_Toc282608641)

[3.1.3 Reports 111](#_Toc282608642)

[3.1.4 Controls 111](#_Toc282608643)

[3.2 Web 115](#_Toc282608644)

[3.2.1 Page 115](#_Toc282608645)

[3.2.2 Fonts 121](#_Toc282608646)

[3.2.3 Content 121](#_Toc282608647)

[3.2.4 Performance 122](#_Toc282608648)

Figures

[1. CPR by Method for Married Women (1992–2004) [EXAMPLE] **Error! Bookmark not defined.**](#_Toc232309872)

Tables

[1. Pallet Size and Pallet Volume [EXAMPLE] **Error! Bookmark not defined.**](#_Toc232309903)

Acronyms

PipeLine Pipeline Monitoring and Procurement Planning System

LMIS logistics management information system

USAID U.S. Agency for International Development

The list above is a sample. List only the acronyms that you use in your manuscript.

For help with acronyms, log in to [deliver.jsi.com](http://www.deliver.jsi.com)—

1. Select *Staff Resources.*
2. Select *Communications Resources.*
3. Select *Acronyms List*.

Acknowledgments

[Body text starts here. Use only one space between sentences.]

Sample text: Ut laore feugiam quiscipit, sit, commodo euguercin ulput velendre ting eum volortie vullan heniamet luptat. Duis ea faci te exer alit niamet nos nosto etuerat.

Obor si. Unt praessed tie dolendreet lore del ut doloreratue velisim zzrilit num etuerostrud tet prat dolestrud delit praesequis augiat. Gait aliquat lan volore exercipit et, vulluptat lobore veriuscing eum veraese doloborem ate mincipsum ipit. Etueriustrud te magna conse facinci et, sismodiat, si ex er adio odit wisl ipismodolor incilit iure ea facil ulluptat dipit, consequisi.

Lesenis modolestis ea faccum quatet, sustie facilit nostis nibh et loreet wisim velit, sum voloborem quisse cor am vent ercipisl erosto od tatummy. Niscil ut aliquam iurem veliquat, suscidui enim vel eugait augiam, sequate magna consectem zzrilis cilisi.

Vel el delese feugait venibh ea accum vel utpatem delese dolobor acil dui bla aliquat. Ibh et lut ing el ex et lortincin volobor adit ad tem zzriure te consequam vulland ipsumsandip eu faccum ero cons nullandreet incin veriure ea consendre tie consequam. Quat alis ationsecte veliquamet lore mod tem num vent ea accum numsan ullam nos nim aciduisl do consed tatie del ectem iustrud euiscip essi blaorerostin exero con ut ulluptat nosto. Dolortie modit nos et irit velessim zzril il il eu faccum dolore endre veliquis aliquatis alisis nostionulla feugue core eros non hendrem zzril ea at lobore mincill amcore dolore vullutate tatio dolorerat. Minci bla consequ iscidunt augue velit in et la conse eu facil irit lamcons equipsu mmodignissi.

Rat velit aut in esed euis dip et ulluptat.

This blank page is intentional. All chapters start on a right-hand side.

Executive Summary

[Body text starts here. Use only one space between sentences.]

Lesenis modolestis ea faccum quatet, sustie facilit nostis nibh et loreet wisim velit, sum voloborem quisse cor am vent ercipisl erosto od tatummy niscil ut. Aliquam iurem veliquat, suscidui enim vel eugait augiam, sequate magna consectem zzrilis cilisi.

Vel el delese feugait venibh ea accum vel utpatem delese dolobor acil dui bla aliquat. Ibh et lut ing el ex et lortincin volobor adit ad tem zzriure te consequam vulland ipsumsandip eu faccum ero cons nullandreet incin veriure ea consendre tie consequam. Quat alis ationsecte veliquamet lore mod tem num vent ea accum numsan ullam nos nim aciduisl do consed tatie del ectem iustrud euiscip essi blaorerostin exero con ut ulluptat nosto dolortie modit nos et irit. Velessim zzril il il eu faccum dolore endre veliquis aliquatis alisis nostionulla feugue core eros non hendrem zzril ea at lobore mincill amcore dolore vullutate tatio dolorerat. Minci bla consequ iscidunt augue velit in et la conse eu facil irit lamcons equipsu mmodignissi.

This blank page is intentional. All chapters start on a right-hand side.

Introduction

Overview

The Pipeline Monitoring and Procurement Planning System (PipeLine), a software tool, was designed to help program managers monitor the status of their product pipelines and product procurement plans. PipeLine provides information needed to initiate and follow-up actions to ensure the regular and consistent stock of products at the program or national level. Consistency of stock is the first step in meeting the basic objective of any logistics system, which is to provide—

* the *right* quantities
* of the *right* commodities
* in the *right* condition
* in the *right* place
* at the *right* time

for the *right* cost.

These are the *six rights* of logistics management.

What PipeLine Can Do for You

PipeLine helps you achieve the *right* quantities at the *right* time.

For each product, PipeLine monitors—

Total quantities *consumed* (i.e., amounts dispensed to users or sold to clients)

*Shipments* of new products (planned, ordered, shipped, or received) into your program

and the values of your products

*Inventory levels* for each product in your program's logistics system (desired and actual)

*Inventory level changes* (e.g., product losses or transfers out of or into your program)

With these data and an understanding of the *lead time* required for each step in the procurement process, PipeLine can—

1. Show what actions you need to take for procurement planning and management, and when these actions should be taken.

Identify impending problems (i.e., surpluses, shortfalls, or stockouts) *before* they occur.

Calculate procurement quantities needed to keep your pipeline in balance.

Calculate the estimated value of shipments or maintain the actual value (if known).

You can use this information with program policymakers, product suppliers, and donors to provide a rational basis for planning future product needs.

PipeLine is *not* the answer to every logistics question. It helps monitor the *aggregate* quantity of each product entering and leaving your program's distribution system (preferably using data from a logistics management information system [LMIS]).

PipeLine's utility is enhanced if your program has a well-functioning LMIS and forecasting procedures. Even without these underlying systems, use PipeLine with whatever data are available. By beginning a rational and systematic product monitoring and planning process, you take the first step toward ensuring consistent stock levels.

Why Use PipeLine?

Ensuring adequate supplies of commodities is difficult for most programs. As a program manager, you face a complex procurement planning environment, characterized by—

1. Multiple suppliers of many products (local and private suppliers, bilateral and multilateral donors, etc.), each with its own products, lead times, costs, information needs, and bureaucratic constraints

Proliferation of service delivery points, in many cases in an integrated service delivery setting, and/or with multiple service delivery organizations served by a single logistics system

Increasing volume (and costs) of commodities, which must be managed and moved through complex distribution channels

Increasing emphasis on accountability, cost-effectiveness, and sustainability from donors who fund product procurement and from policymakers.

You need to monitor the quantity and timing of multiple products entering your logistics system from multiple suppliers. Because procurement lead times may be long—years, in many cases— you need to take action months or years before commodities are needed to receive them on time.

You may need to negotiate with many different suppliers and donors to obtain the quantities you require. Such negotiations are best accomplished when specific data on product requirements are available. You must know when you will stock out of each product, how much must be procured to meet future needs, and when you should receive it. To prevent overordering, you must also know what quantities would exceed your storage capacity or risk wastage due to expiry. PipeLine can provide this information.

Who Should Use PipeLine?

In a multi-product, multi-supplier environment, procurement planning and pipeline monitoring functions cannot be donor driven. It is increasingly necessary that local program managers be empowered to do their own forecasting, pipeline monitoring, and procurement planning; they must also take charge of coordinating the activities of donors and local suppliers, as well as those of their own logistics management staff. Donor staff often have other priorities and little time to devote to the details of logistics management. Commercial suppliers have interests that may or may not correspond to the interests of your organization.

If you are the logistics manager or program manager for your organization, you should manage your own pipeline. PipeLine can help.

While your managers and decision makers will be the primary users of PipeLine, the system can provide information to—

**Suppliers of commodities**PipeLine provides reports on the current status and the cost of pending shipments from aspecific supplier, which that supplier can use to monitor product flow.

**Purchasers/donors of commodities**Staff who finance the purchase of commodities can use PipeLine reports and graphs to understand the current pipeline status and future requirements.

**Host-country policymakers**PipeLine reports and graphs can be used to help policymakers understand issues with the levels of particular commodities and the implications of different decisions on the availability of the product.

PipeLine Software Functions

PipeLine can help you with pipeline monitoring and procurement planning functions, as explained below.

Pipeline Monitoring

Pipeline monitoring functions include—

* Monitoring stock balances, in terms of quantities and months of stock on hand in the entire program (aggregate of stock at all levels)
* Comparing stock balances to maximum and minimum stock policies
* Automating the identification of pipeline problems (quantities needed, stockouts, balances below minimum or above maximum)

Providing couple-years of protection (CYP) conversion graphs.

Procurement Planning

Procurement planning functions include—

* Calculation of shortfalls/surpluses and quantities needed to maintain the program’s desired stock levels
* Automated calculation and tracking of pending pipeline actions, based on lead times (shipments to plan, order, ship, and receive)
* Application of USAID’s contraceptive procurement tables (CPT) format for the computation of calendar year quantities required and the generation of data for USAID’s planning requirements
* Calculation of estimated costs of shipments and freight
* Comparison of alternative procurement scenarios and analysis
* Alternative unit of measure calculation displays products in Basic Units. Basic Units are used to quantify patient or consumer needs and usually refers to tablets, capsules, or milliliters, rather than packs or bottles.

Technical Architecture

The PipeLine software is developed using following programming language, tools and techniques:

* Microsoft Access 2003 and 2000. The front end database is based on Access 2003 and the back-end database is based on Access 2000.
* On-line help was developed using Robohelp X5
* Automated installable version was created using InstallShield 12 and SageKey for Access 2003 version 2.0.9
* Data export/import interface between Supply Chain Manager and PipeLine is build using XML 1.1
* CD auto-run was build using AutoRun Pro Enterprise II
* User Guide and Technical documentation was developed using combination of Microsoft Visio, Powerpoint and Word
* For source control, Microsoft Visual Source Safe was used.

For bug tracking and issue tracking, open source bugzilla application was used.

Process Flows

**Update**

**Background  
Data**

**Run Imports**

(i.e. Product  
Catalog from  
ORION, Shipment,  
and Consumption  
Data)

**Update  
Shipment Data**

**Update/Import  
Consumption Data**

**Run Forecast**

**Balance PipeLine**(i.e. adjust   
shipment schedules

as needed)

**Run Reports**

**Update Stock Data**

**PIPELINE PROCESS**

Appendices

|  |  |  |
| --- | --- | --- |
| Appendix | Title | Description |
| A | The Reddick VBA (RVBA) Naming Conventions, Version 6.01 | Industry standard naming conventions for Access applications |
| B | The Reddick VBA (RVBA) Coding Conventions (version 0.90) | Industry standard coding conventions for Access applications |
| C | Microsoft Application User Interface Guidelines | JSI GUI standards |

System Requirements

The following resources are recommended for use with PipeLine—

|  |  |
| --- | --- |
| Component | Hardware/Software Requirements |
| CPU | Pentium IV or higher |
| Operating System | Windows XP or above |
| Memory | 1 GB or higher |
| Hard Disk Space | 500 MB of free space or higher |
| Video Adapter | SVGA with at least 800 X 600 resolution |
| Microsoft Office | Microsoft Office 2003 or higher |

Installation and Configuration

Installation Instructions

Microsoft Office® 2003

Although PipeLine will run without Microsoft Office® 2003 installed on your computer, having Office 2003 installed will enhance PipeLine’s usefulness by allowing PipeLine to export data files to Word® or Excel®.

How to Install PipeLine

PipeLine can be installed from a CD-ROM or the Internet.

Before You Begin

You can run PipeLine 2 and PipeLine 5. on the same computer, but we recommend that you uninstall PipeLine 2 BEFORE installing PipeLine 5.

To uninstall PipeLine 2—

1. Click on Start.

Previous Versions

All other previous versions of PipeLine cannot be run on the same computer as PipeLine 5. When you install PipeLine 5, the installer will automatically remove the previous version of PipeLine prior to installing PipeLine 5.

Click on the Settings option.

Click on the Control Panel option.

After the Control Panel window opens—

Click on the Add/Remove Programs option.

Locate and click on PipeLine in the Currently Installed Programs list.

Click on the Change/Remove button.

The PipeLine 2 setup program will start, and will prepare your computer to uninstall PipeLine 2.

Click on the Remove All Button.

A message is displayed asking if you want to remove PipeLine.

Data Files

Data files created with previous versions of PipeLine are not removed from your system. See Converting Your Existing Data Files on page 18 for information on converting your existing PipeLine 2 data files.

Click on the Yes button to begin the uninstall procedure.

When the uninstall procedure is completed, you will be prompted to restart windows.

Click on the Restart Windows button.

Installing PipeLine from a CD

1. Start Microsoft Windows.

Insert the PipeLine CD.

The PipeLine installation should begin automatically.

Follow the on-screen instructions.

If the installation does not begin automatically—

Click on Start on the Windows Taskbar.

Click on Run from the pop-up menu.

In the Command Line box, type *x:setup* ("x" is the letter of your CD-ROM drive).

Click on the OK button, and follow the on-screen instructions.

After PipeLine is successfully installed, the PipeLine shortcut (shown below) will be displayed on your desktop.

Installing PipeLine from the Internet

PipeLine CD-Rom

If your Internet connection is slow and/or unreliable, order the PipeLine CD-ROM.

PipeLine is available on the USAID | DELIVER PROJECT website at the following web address:

http://deliver.jsi.com.

To download PipeLine—

1. Access the Internet, and enter the USAID | DELIVER PROJECT web address.

Locate the PipeLine download page, and follow the on-screen instructions to download PipeLine.

How to Start PipeLine

PipeLine can be started from the Windows desktop or the Windows taskbar.

Starting PipeLine from the Windows desktop

From the Windows desktop—



1. Locate and double-click on the PipeLine icon to start the application.

Starting PipeLine from the Windows taskbar

From the Windows taskbar—

1. Click on Start.

Click on Programs.

Locate and click on the PipeLine 5.0 link.

Reinstalling PipeLine

To reinstall PipeLine—

Password

The Internet version of PipeLine requires a password to start the install process. That password was sent to you by email when you downloaded PipeLine. If you no longer have the email containing the PipeLine password, redownload PipeLine from the USAID | DELIVER PROJECT website: (http://deliver.jsi.com).

1. Place the PipeLine CD in your CD-ROM drive.

If you do not have the PipeLine CD, you can use the copy of PipeLine you downloaded from the USAID | DELIVER PROJECT website.

Start the install process, and follow the instructions on your screen.

During the process, a message box is displayed instructing you to remove PipeLine from your computer.

Click on the Remove button to remove PipeLine from your computer.

Click on the Finish button when prompted.

After PipeLine has been removed—

Repeat the PipeLine installation procedure.

See page 16 for information on installing PipeLine.

Converting Your Existing Data Files

This version of PipeLine allows you to convert data created with previous versions of PipeLine.

Upgrading

The current program’s data file is not the current version. You can allow PipeLine to upgrade the file now. If you do not, some of PipeLine’s features may not work properly.

From the Program Data screen—

1. Click on the File Menu Bar option, and select the Open option from the pull-down menu.

PipeLine opens a window so you can locate the data you need to convert.

Locate and select the data file you need, and click on the Open button.

After you select the data file you need to convert, PipeLine displays a message similar to the one in the text box below.

Click on the Yes button to convert the selected data.

PipeLine opens a window, and allows you to rename the file you selected to upgrade. This safeguards the original data by saving the upgraded data under a different name.

Original Data

Remember, your original data remains in its original directory with its original name. The converted data is a copy of the original.

Type the new name in the File Name field, and click on the Open button.

PipeLine converts the selected data, renames the file, and displays its associated program data on the Program Data screen. You can now work with the converted data file.

Run-time installation

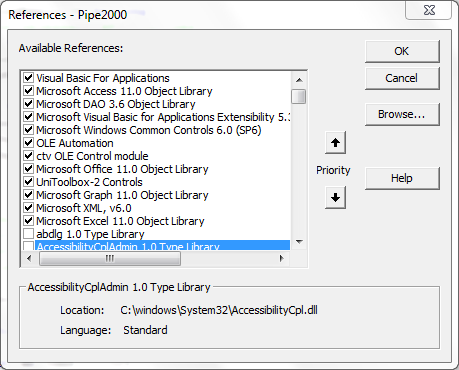
|  |  |
| --- | --- |
| Installation location | |
| Directory/File | Purpose |
| **PipeLine** | **Parent directory** |
| **/ANYMOH** | **Directory for sample database** |
| /globalmoh.MDB | Sample Database |
| **/Data** | **Directory for live databases** |
| **/Graphics** | **Directory for application graphics** |
| /SplashNewT.avi | Splash screen movie |
| /PL40.ico | PipeLine taskbar icon |
| /Pipeline\_ICON-xx.ico | PipeLine Desktop icon |
| **/Import** | **Directory for imported files** |
| **/Summary** | **Directory for PipeLine Summary** |
| /Roboex32.dll | Dll required for PipeLine Summary to run properly |
| /Proc2000.mdb | PipeLine Summary frontend |
| /Proc\_BE.mdb | PipeLine Summary backend |
| /Prog2000.mdb | PipeLine Summary program list |
| /Summary.ico | PipeLine Summary icon |
| /Sumv2.cnt | PipeLine Summary help cnt file |
| /SUMv2.hlp | PipeLine Summary help file |
| **/XML** | **Directory for xml files** |
| /ECatalog\_Live\_Final\_Generic\_20100701.xml | E-Catalog file distributed with application |
| /SCMS Product\_ARV\_TEST.xml | SCMS ARV file distributed with application |
| /Contraceptives.xml | Contraceptives file distributed with application |
| /e-help.cnt | PipeLine (English) help cnt file |
| /e-help.HLP | PipeLine (English) help file |
| /E-PL-help.cnt | PipeLine help cnt file |
| /E-PL-help.hlp | PipeLine help file |
| /Pipeline\_ICON-xx.ico | PipeLine icon |
| /Pipeline2000.MDB | PipeLine frontend file |
| /PLFix1.reg | Registry fix for graphs |
| /PLFix2.reg | Registry fix for graphs |
| /PLFix3.reg | Registry fix for graphs |
| /pmp\_mpty.mdb | Empty PipeLine backend file |
| /posttransform.xslt |  |
| /ProgV4.mdb | PipeLine program list |
| /ReadMe.txt | Readme file for installation issues and known issues |
| /Roboex32.dll | Dll required for PipeLine to run properly |

Development environment installation

The Development Environment requires the installation of Microsoft Office 2003 (MSACCESS and Excel are required). Some other tools are suggested as well

|  |  |  |
| --- | --- | --- |
| Software/Tool | Required | Description |
| Microsoft Access | Y | The Main Development Tool for PipeLine |
| Microsoft Excel | Y | Required for the creation of the Output to Excel reports |
| Microsoft Visual SourceSafe (VSS) | N | The software code repository for PipeLine. Required for checking in/out the source code files. Optional (but suggested) if development is being done in a stand alone environment |
| Microsoft Access Plug-In: Source Code Control | N | odc\_accscc.exe, this allows VSS integration into Access. Access .mdb files can be stored in VSS as individual components allowing multi-user development on a single .mdb file). |
| FMS Total Visual Code Tools | N | This plugin/toolbar provides the developer with the ability to quickly stub in new procedures and functions and to cleanup existing code modules with proper development standards. |
| Microsoft Windows Common Controls 6.0 (SP6) | Yes | This provides the Treeview control. |

The Access References should look like this:



Build process

The Build Process for PipeLine involves checking the code into VSS (if necessary), removing the application for VSS, compact the application, and updating the tblSysParameters. In the tblSysParameters, update the AsOf date, the Version number, and set InitialInstall to True. Please note that the application will not relink properly if InitialInstall is not set to true since this flag informs the application to look for the default data file distributed with PipeLine.

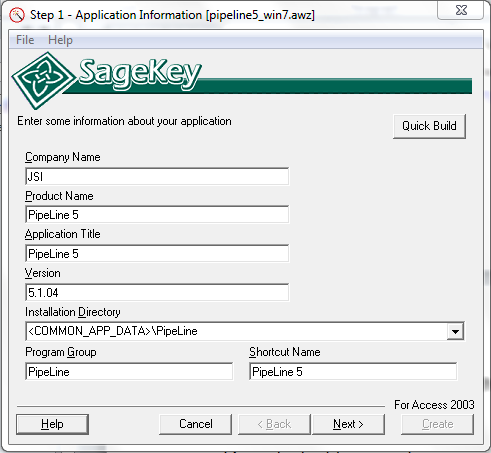
Installer

PipeLine is widely used in many countries. It is also downloaded through the web. A majority of the users install the PipeLine application on their own. The installer is first created using SageKey MSI Wizard 2003 along with Installshield 12. In order to then provide a simple, user friendly and self-installable interface, a tool called AutoRun Pro Enterprise II was used. (This tool is available at this website <http://www.longtion.com>.) These file create the cd image and the files are zipped using WinZip Self Extractor to create a single downloadable file used for the web distribution.

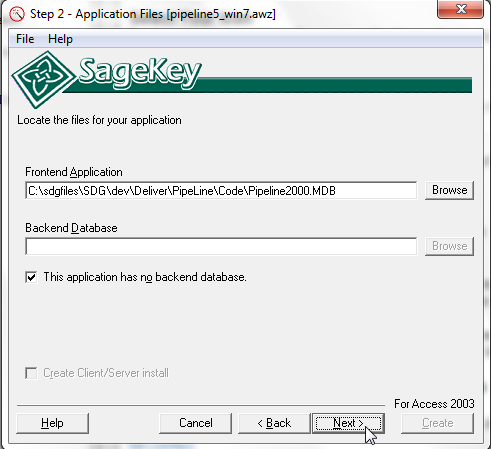
Mechanics for Creating the CD Installer

Create files with SageKey

Once the build is complete, open the file PipeLine5\_win7.awz.



Update the basic information about PipeLine here. For more information on each field, right click the corresponding label and select 'What's this?" Click Next to go to next screen.



Select your database project files here and click Next to go to next screen.

Front-end Application

Click 'Browse' to locate the PipeLine2000.mdb file.  The files must exist in the place specified or you can not proceed to the next step.

Back-end Database

Since we connect to the backend dynamically, check the box next to 'This application has no back-end database.' The text field will go blank and the browse button will no longer be available.

Edit .msi with InstallShield

Create Interface with AutoRun Pro

DownLOAD and INstall the application

Download the application from the link below:

<http://www.longtion.com/autorunenterpriseii/autorunpro.htm>

Install the application. The installation steps are simple, like any other windows application, wizard driven, and takes under 2 minutes.

Once the application is installed, start the application and start building a project.

CREATE A NEW PROJECT

Download the application from the link below:

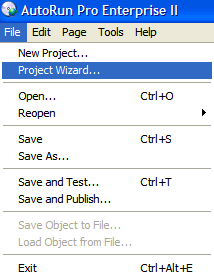
Start THE Application

Find and click on the AutoRun Pro icon on your desktop

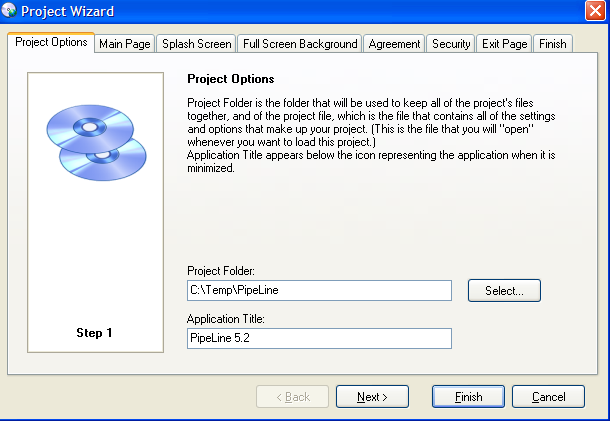


CREATE A NEW PROJECT

Create a new project to build the PipeLine CD auto-run program. See following illustrations for the steps.

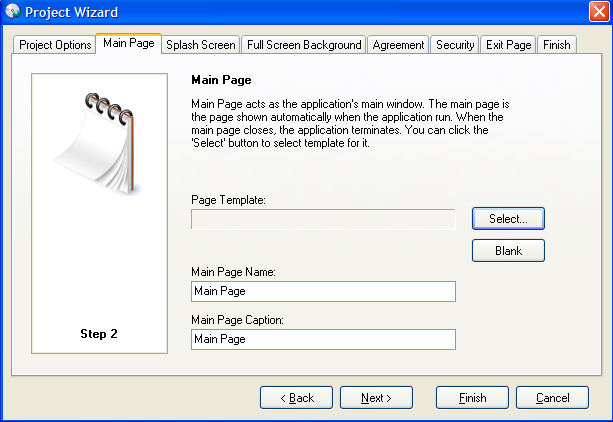


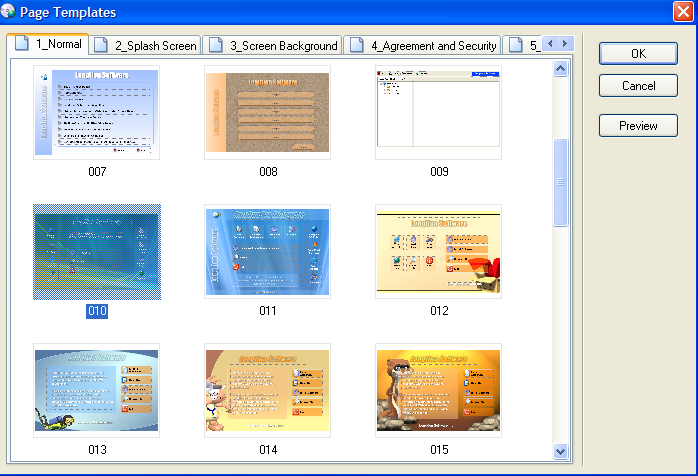
Select File > Project Wizard. The Project Wizard starts.



Select the folder on which to save the project files. Give a name to the project.

CD installer can display installation screens through multiple menu/pages. In next screen of the Project Wizard select a suitable template to base the look and feel of the installation screen.





Template number 010 was used as the base template for PipeLine.

Next couple of screens of the wizard asks about whether to use splash screen, what kind of background graphics to use, whether to ask the user to agree to a user agreement legal statement, a security page to enter password to activate the product, an exit page to display to the user. None of these were used for the PipeLine project. Click finish to complete the project creation steps.

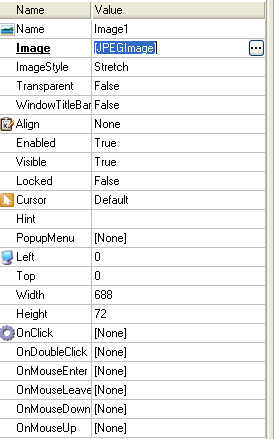
CUSTOMIZE THE TEMPLATE

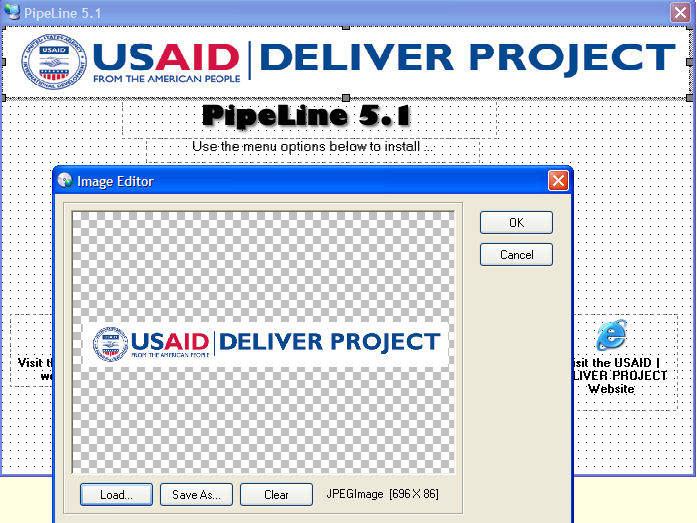
Customize the template number 010 that was used as the base. The process of customizing is basically:

* **Select a screen background image and color**. No background image was use. Color was set to white.
* **Set screen size**. Screen size was kept to default 697 X 480
* **Apply logo**. The USAID | DELIVER PROJECT logo was used. See screenshot below.
* **Create required pages**. After finalizing the page 1, a second page was created through Page > Duplicate Page menu option. The first page is meant to display program installations, links to website for further resources and a link to go to next screen. The second page displays the PipeLine documentation related menu choices.
* **Create menu options**. The template number 010 comes with default menu option choices. Those menu options were customized to build PipeLine related menu choices. The associated steps are illustrated below.

Apply logo

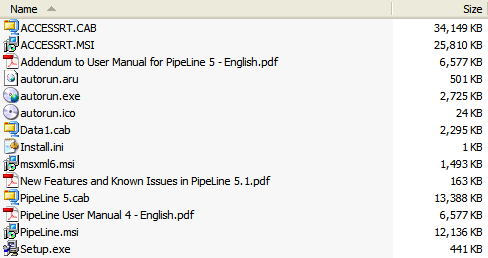
On the page 1 properties window, selecting the image item and select USAID logo. See illustration below:





Place PipeLine media files in a folder

Create a folder to place all files related to PipeLine program. The AutoRun program allows creating folder and sub-folder structure for complex projects. However for PipeLine project, all files were kept on a single folder, as illustrated below.

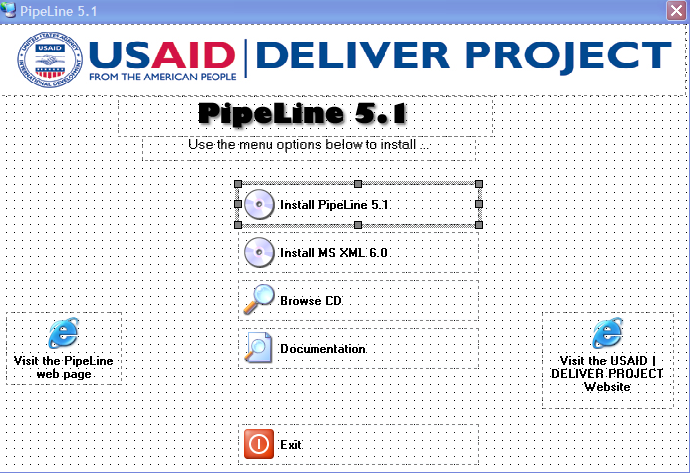


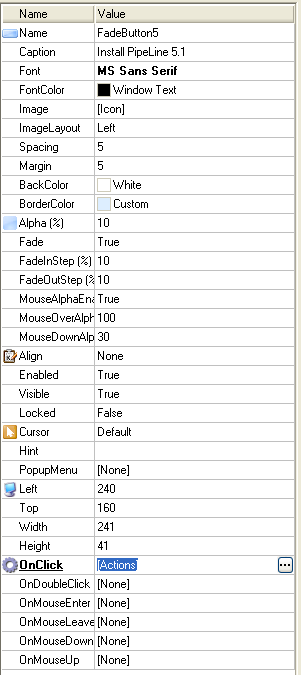
Create menu choices

The template 010 comes with various menu choices. The following steps illustrate the process of building the PipeLine menu choices. Instead of repeating the steps for identical items, only the unique items were illustrated below.

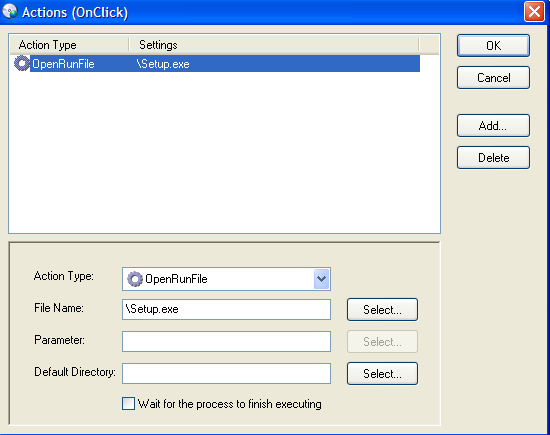
Create menu options that calls the PipeLine installer to run

The menu options are defined through the properties window. The visual layout and the selected properties to define the PipeLine installation main menu option are illustrated below.

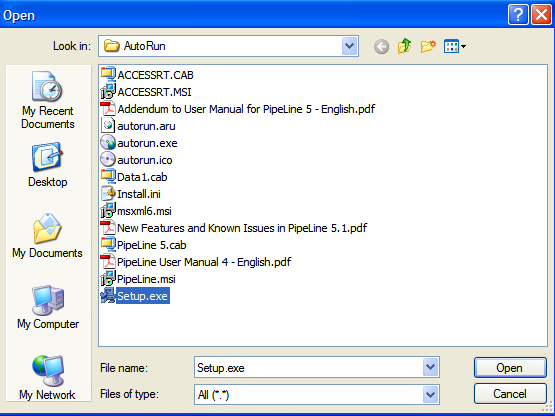




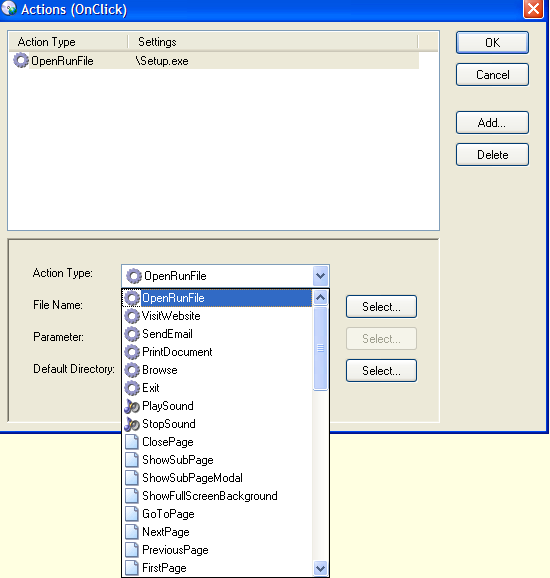
Through the properties window above, text, label, icon, formatting and onclick events were defined.



The onClick event executes the Setup.exe file placed at the root folder. Using the select button, select the Setup.exe file and click Open.

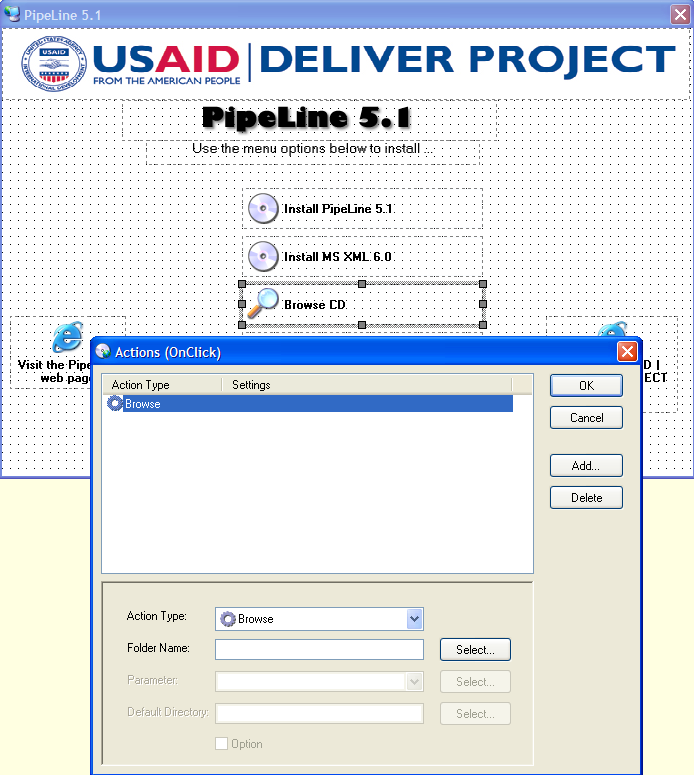


Other available Onclick events are illustrated below



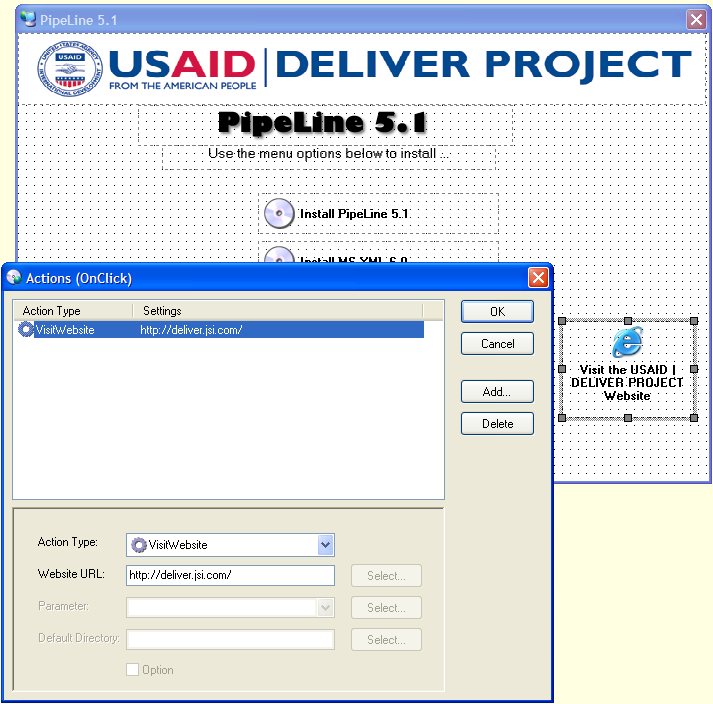
Create CD browse menu option

Using Onclick > Browse event, the CD browse option was defined. See illustration below.



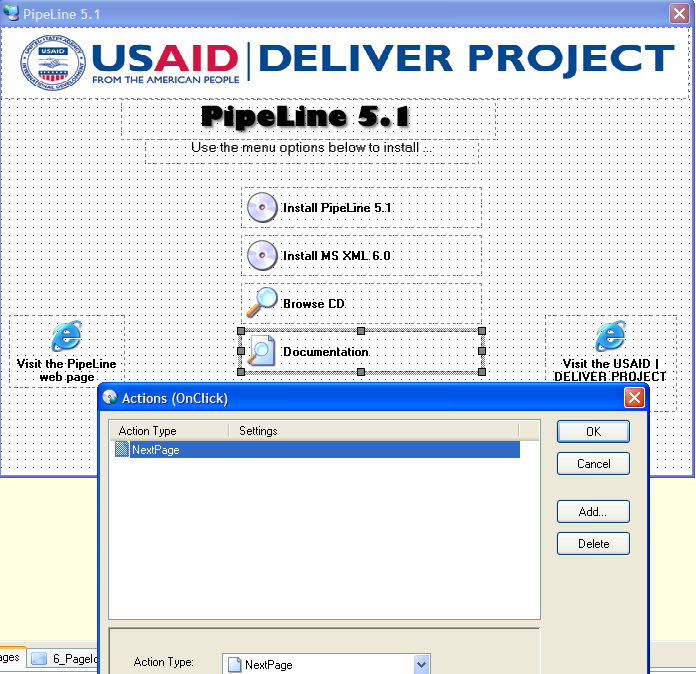
Create link to external website menu option

Using Onclick > VisitWebsite event, the navigation to the USAID | DELIVER PROJECT website and Visit PipeLine website links were created. This click will go to respective website, using the users’ default browser.



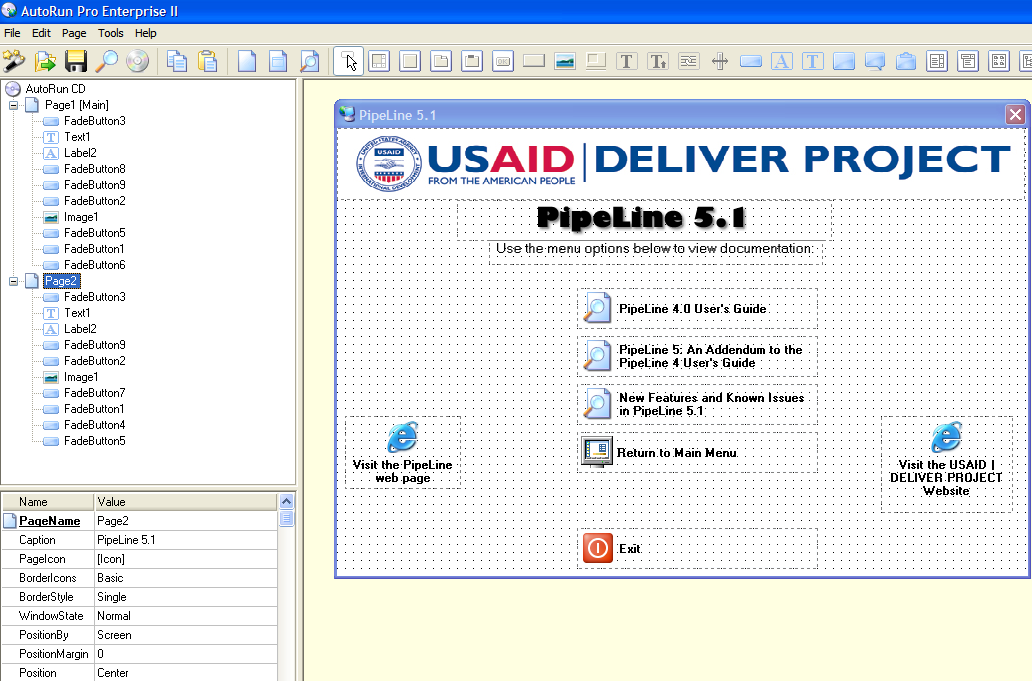
Create link to next page menu option

Using Onclick > NextPage event, the navigation to next page was defined. See illustration below.



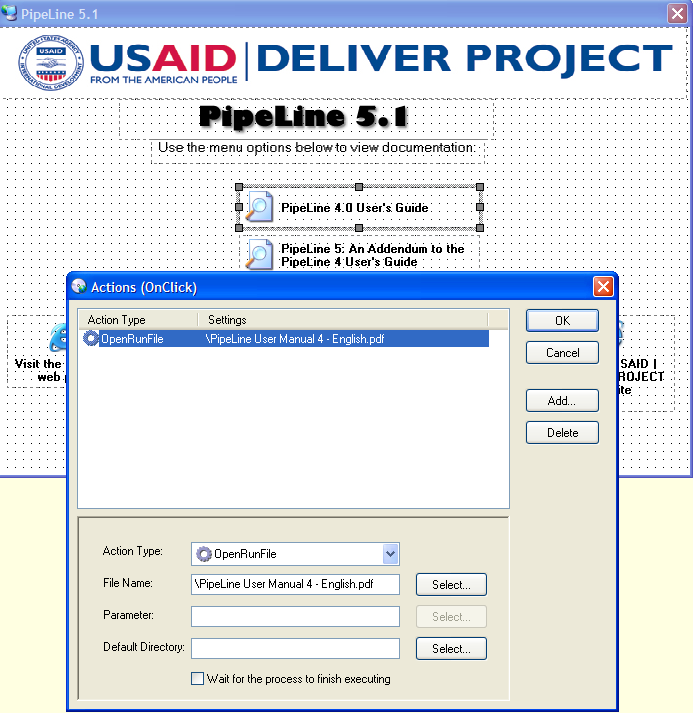
Create second page

Using Page > Duplicate Page menu option second page was created.



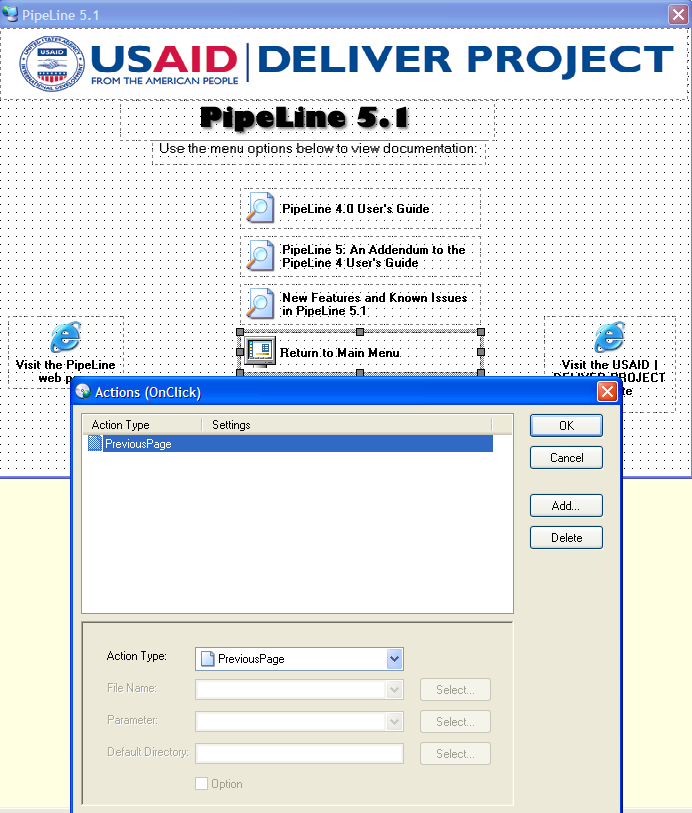
Create open PipeLine User Guide PDF file in Adobe Acrobat

Using OnClick > OpenRunFile click event, the open PDF file was defined.

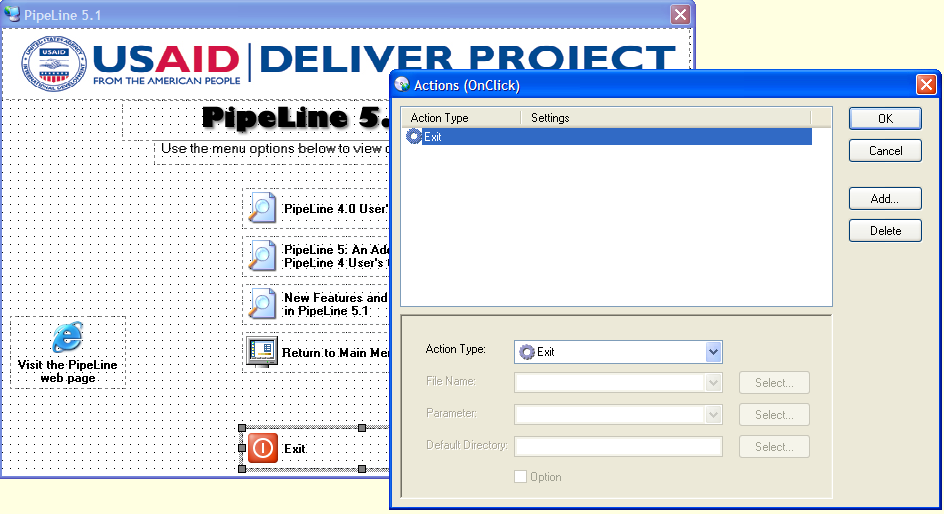


Create Return to Main Menu

Using OnClick > PreviousPage click event, return to previous page was defined.

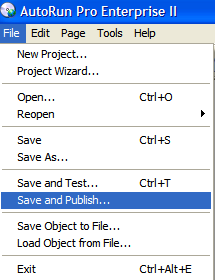


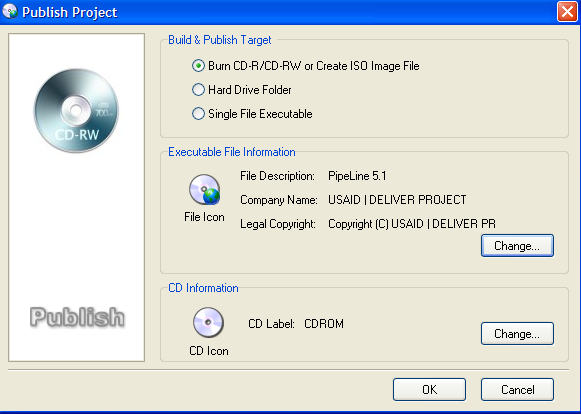
Create Exit menu option

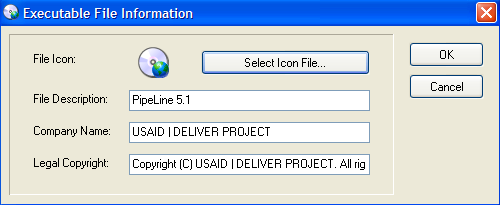
Using OnClick > Exit click event, exiting the CD Run program was defined.

Build the Executable AutoRun program and create the master CD

After the full two page menu choices were defined, it is time to test the project. Using File > Save and Test menu option, test the Auto install program. Once every element is tested and found acceptable, build the final version using File > Save and Publish menu option.



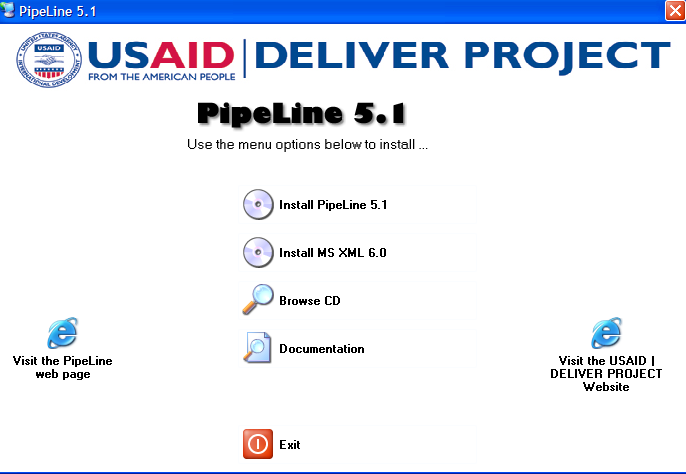




Test the final master CD and make required number of CD copies

After the master CD is build, test the CD under following operating environments:

* Windows XP
* Windows Vista
* Windows 7 32 bit
* Windows 7 64 bit



Final version of the Auto-run CD should look like above. After testing and acceptance, using a mass CD burner, publish required number of CDs. For professional quality CD label and high speed publishing, contact professional firms; provide master CD and related instructions.

Create Web Installer

Web-download

The full installer is downloadable from the SDG website (<http://sdg.jsi.com>). Upon downloading, user will getbe requested to fill out a information form. The password for unzipping the web installer will then be emailed to the address provided on this form.

Application Design

Overview

Access stores all database tables, queries, forms, reports, macros, and modules in the Access Jet database as a single file.

For query development, Access offers a "Query Designer", a graphical user interface that allows users to build queries without knowledge of the SQL programming language. In the Query Designer, users can "show" the data sources of the query (which can be tables or queries) and select the fields they want returned by clicking and dragging them into the grid. One can set up joins by clicking and dragging fields in tables to fields in other tables. Access allows users to view and manipulate the SQL code if desired. Any Access table, including linked tables from different data sources, can be used in a query.

When developing forms and reports that are linked to queries placing or moving items in the design view, Access runs the linked query in the background on any placement or movement of an item in that Form or Report. If the form or report is linked to a query that takes a long time to return records this means having to wait until the query has run before you can add/edit or move the next item in the form or report (this feature cannot be turned off).

Non-programmers can use the macro feature to automate simple tasks through a series of drop-down selections. Macros allow users to easily chain commands together such as running queries, importing or exporting data, opening and closing forms, previewing and printing reports, etc. Macros support basic logic (IF-conditions) and the ability to call other macros. Macros can also contain sub-macros which are similar to subroutines. Macros however, are limited in their functionality by a lack of programming loops and of advanced coding logic. PipeLine only uses macros for populating the custom menubar. This allows for ease in creating the menubar at runtime and in opening the proper dialog box.

The programming language available in Access is, as in other products of the Microsoft Office suite, Microsoft Visual Basic for Applications, which is nearly identical to Visual Basic 6.0 (VB6). VBA code can be stored in modules and code behind forms and reports. Modules can also be classes.

To manipulate data in tables and queries in VBA, Microsoft provides two database access libraries of COM components:

* Data Access Objects (DAO) (32-bit only), which is included in Access and Windows
* ActiveX Data Objects ActiveX Data Objects (ADO) (both 32-bit and 64-bit versions)

PipeLine uses DAO objects and so DAO must be a registered reference for the application.

Many Access developers use the Reddick naming convention, though this is not universal; it is a programming convention, not a DBMS-enforced rule. It is particularly helpful in VBA where references to object names may not indicate its data type (e.g. tbl for tables, qry for queries).

Split Database Architecture

Microsoft Access applications, like PipeLine, adopt a split-database architecture. The database is divided into a front-end database that contains the application objects (queries, forms, reports, macros, and modules), and is linked to tables stored in a back-end shared database containing the data. The "back-end" database can be stored in a location shared by many users, such as a file server. The "front-end" database is distributed to each user's desktop and linked to the shared database. Using this design, each user has a copy of Microsoft Access installed on their machine along with their application database. This reduces network traffic since the application is not retrieved for each use, and allows the front-end database to contain tables with data that is private to each user for storing settings or temporary data. This split-database design also allows development of the application independent of the data. When a new version is ready, the front-end database is replaced without impacting the data database.

Linked tables in Access use absolute paths rather than relative paths, so the development environment either has to have the same path as the production environment or a "dynamic-linker" routine can be written in VBA. In PipeLine, the links to the backend are stored in the ProgV4.mdb file. This also allows us to populate the Window menubar option.

This is not an economical setup across slow networks, or in large organizations separated by great distances, as it will result in excessive lag to database users. Therefore, when users are installing PipeLine, consideration needs to be made for their network speed.

Naming Conventions

Conventions used

Naming and coding conventions were not originally used. With version 3.0, these conventions have been used. All future programming should follow these guidelines.

The conventions that follow were developed to promote uniformity and consistency in naming various program modules. Variables and objects with familiar labels make source code easier to read.

All MSAccess databases should utilize Reddick VBA (RVBA), Version 6.01 (see Appendix A). Table 1 summarizes the conventions used from the RVBA for database window objects.

|  |  |  |
| --- | --- | --- |
| Object | Prefix | Description |
| Table | tbl | Data Table |
|  | tmp | Temporary Table |
| Query | qsel | Select Query |
|  | qupd | Update Query |
|  | qdel | Delete Query |
|  | qapp | Append Query |
| Forms | frm | Form |
|  | fsub | Subform |
| Reports | rpt | Report |
|  | rsub | Subreport |
| Macros | mcr | Macro |
| Modules | bas | Module |

Table 4.1 – Prefixes for Access Database Window Objects

Table 2 summarizes the conventions for object types found in form and reports.

|  |  |
| --- | --- |
| Prefix | Object Type |
| chk | Checkbox |
| cbo | Combobox |
| cmd | Command Button |
| lbl | Label |
| lst | listbox |
| ole | ObjectFrame |
| opt | OptionButton |
| pge | Page |
| Tab | Tab Control |
| Txt | Text Box |
| Tgl | Toggle Button |

Table 4.2 – Access Object Variable Prefixes

A prefix should first be added to each field in a table reflecting a two-digit abbreviation of the table name followed by an underscore and the descriptive naming convention for the object type. The first part of this prefix should reflect the table where it is located or the table name of the parent table if the field is a foreign key (i.e. ea\_datUpdated could be included in a table named tblExpectedActions and fr\_datReceived could be included in the same table thus showing its link to the parent table tblFundsReceived). Table 3 summarizes the naming conventions for the object types in the tables.

|  |  |
| --- | --- |
| Prefix | Object Type |
| bin | Binary |
| byt | Byte |
| lng | Long |
| cur | Currency |
| dat | Date/Time |
| dbl | Double |
| int | Integer |
| mem | Memo |
| sng | Single |
| str | Text |
| f | Yes/No |

Table 4.3 – Field Name Conventions

Data Types

The following table contains the data types in the database tables along with other common data formats.

|  |  |  |  |
| --- | --- | --- | --- |
| Type  Field Size | Description | Format | Prefix |
| Yes/No | Used to setup fields containing boolean values. The default value is set to False. | True/False | f |
| Currency | Used to setup fields containing numeric values referring to US dollars. | 2 decimal places | cur |
| Number |  |  |  |
| Long Integer | Used to setup fields containing long integer values. |  | lng |
| Single | Used to setup fields containing single-precision floating-point values. |  | sng |
| Double | Used to setup fields containing double-precision floating-point values. |  | dbl |
| Byte | Used to setup fields containing byte values. |  | byt |
| Integer | Used to setup fields containing generic integer values. |  | int |
| Memo | Used to setup character fields spanning multiple lines. |  | mem |
| Date | Used to setup fields containing date values. | dd-mmm-yyyy | dat |
| Text  1 - 255 | Used to setup fields containing generic single-line text values. |  | txt |
| Zip Codes  10 | Used to setup fields containing zip codes. | 00000\-9999;0;\_ | txt |
| Phone #/Fax #  16/10 | Used to setup fields containing phone/fax numbers. | \(000”) “000\-0000\ a#####;1;\_ | txt |
| Percent | Used to setup fields holding percent values. | 2 decimal places | per |

Table 4.4 – Data Types

Graphical User Interface

The following conventions were developed to promote uniformity and consistency in appearances. This MSAccess database utilizes Microsoft Application User Interface Guidelines created by JSI. (See Appendix C).

Explorer-Style Navigation

Resolution

The standard screen resolution has been updated to 1024x768. Please note that when applying this upgrade, the treeview, list view and detail view regions were increased proportionally.

Explorer-style navigation provides for easiest navigation of the application’s screens. It contains a treeview list of all screens available in the application, a list view of the records available to view and the detail view of the selected record. (For Reports the list view region will display the parameters and the detail view will display the report results based on selected parameters.) The following drawing defines the recognized screen regions and their sizes.

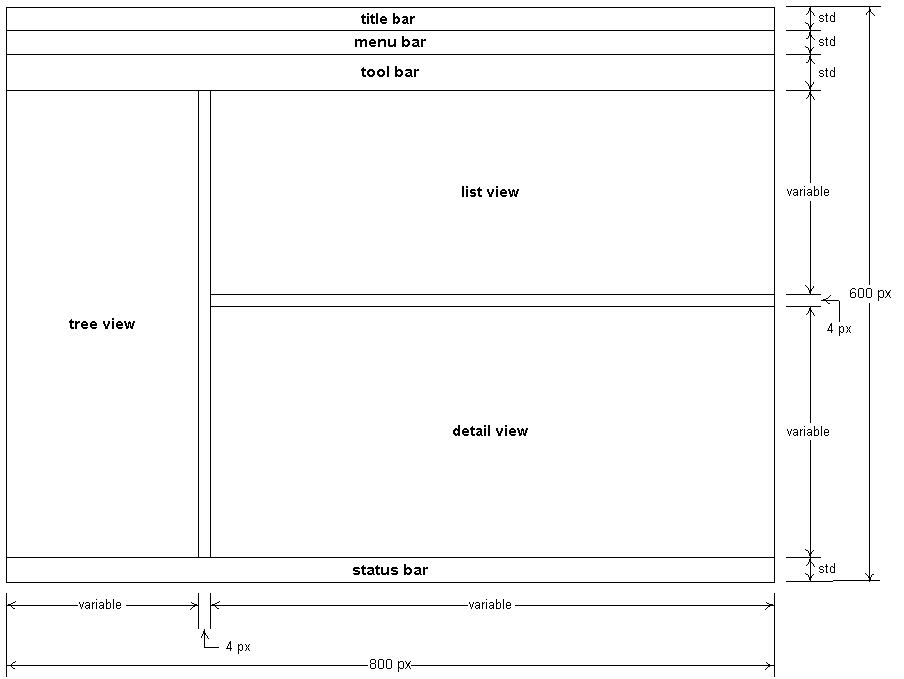


Figure 4.1 –Sample Menu

|  |  |
| --- | --- |
| Element | Comments |
| Title Bar | Should be set with the System Title and selected tree view title |
| Menu Bar | Should contain the minimum options needed for the form. |
| Tool Bar | (Optional) Should contain icons for quick access to various tools/features. |
| Tree View | On click should repopulate List View and Detail View with appropriate information. |
| List View | Should list the various record available in the detail view. In the case of reports, this should list the various parameters available for the report. |
| Detail View | Should display detailed information for the item selected in the list view or the parameters selected for the report. |
| Status Bar | Should provide quick simple explanation for the current field on the form. |

Table 4.5 – Menu Element Specification Table

Menu Bar Specification

In PipeLine the Menu Bar contains the following options.

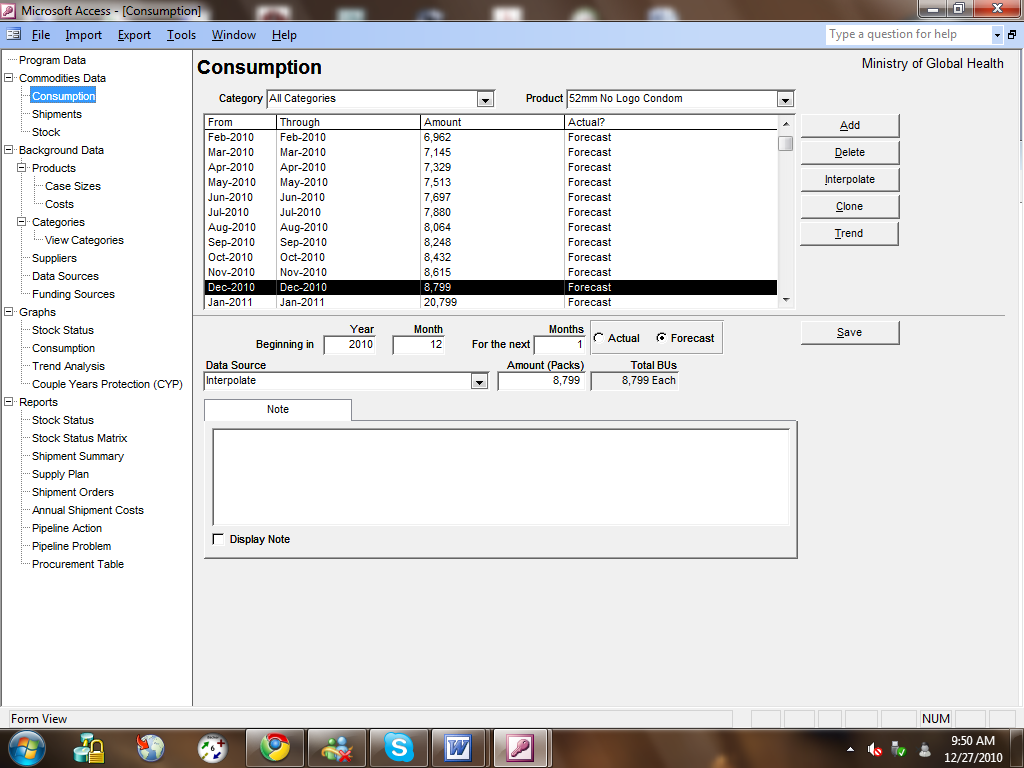
|  |  |  |
| --- | --- | --- |
| Main Choice | Sub Choice | Description |
| File | Exit | Close application and disconnect from the database |
| Import |  |  |
| Export |  |  |
| Tools |  |  |
| Window |  |  |
| Help | Help | Open online help for application. If no online help has been developed, should open Access help. |

Table 4.6 – Standard functionality of Switchboard Strip Menus

Form Conventions

As shown in the figure below, the standard form lets users view record details and all updatable fields are enabled. Upon selecting an item in the list view, the form will filter to the correct record. (User may also select new to be taken to a new record where all fields are empty.)User may then click on SAVE to save the data. If the user modifies the data and tries to leave the form without clicking save, they will be prompted to save. To delete data from the application, the user will select the record in the list view and click the DELETE button. They will be prompted to verify the deletion.

The sample below depict standard for the list and detail screen.

Figure 4.2 – Sample Detail Form

For each control in PipeLine a specific style must be applied. This is controlled by the tblStyle table in the application. The tlkTranslationText table contains the labels for all controls on all forms in the application. In this table, a style form the tblStyle is selected. When opening a form, the styles and labels are then applied to the control. This allows the entire application to be uniform and if the style requirements change, the developer only needs to modify the style table to apply the change to all forms. Below is the list of styles for each element:

| Control Type | Font | Font Size | Fore Color | Font Weight | Underline? | Special Effect | Border Style | Back Color |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Title | Arial | 14 |  | 700 | No | Flat | transparent |  |
| ProgramName | Arial | 10 |  | 400 | No | Flat | transparent |  |
| Main Selection | Arial | 10 |  | 400 | No | Sunken | solid |  |
| General Text | Arial | 8 |  | 400 | No | Sunken | solid |  |
| RptTitle1 | Arial | 12 |  | 700 | No | Flat | transparent |  |
| RptHeader | Arial | 8 |  | 400 | No | Flat | transparent |  |
| RptTitle2 | Arial | 10 |  | 400 | No | Flat | transparent |  |
| RptLabel | Arial | 8 |  | 400 | No | Flat | transparent |  |
| Cmd Button | arial | 8 |  | 400 | No |  |  |  |
| Checkbox |  |  |  |  | No | Sunken | solid |  |
| TabCtrl | Arial | 8 |  | 400 | No | Sunken |  |  |
| Frame |  |  |  |  | No | Sunken | solid |  |
| graph |  |  |  |  | No | Sunken | solid |  |
| Subform |  |  |  |  | No | Sunken | solid |  |
| RptGeneralHead w/o Border | arial | 8 |  | 400 | No | Flat | transparent |  |
| Hidden (fc=bc) | arial | 8 |  | 400 | No | Flat | solid |  |
| RptGroup1 | Arial | 8 |  | 700 | No | Flat | transparent |  |
| RptGroup2 | Arial | 8 |  | 400 | No | Flat | transparent |  |
| RptGroup3 | Arial | 8 |  | 400 | No | Flat | transparent |  |
| RptGroup4 | Arial | 8 |  | 400 | No | Flat | transparent |  |
| General Label | Arial | 8 |  | 400 | No | Flat | transparent |  |
| Main Select Label | Arial | 10 |  | 400 | No | Flat | transparent |  |
| General sfr Text | Arial | 8 |  | 400 | No | Flat | solid |  |
| RptGeneral w/Border | Arial | 8 |  | 400 | No | Flat | solid |  |
| RptGeneral w/o Border | Arial | 8 |  | 400 | No | Flat | transparent |  |
| RptGeneralHead w/Border | arial | 8 |  | 700 | No | Flat | solid |  |
| RptHidden (fc=bc) | arial | 8 |  | 400 | No | Flat | transparent | 10092543 |
| MoveButton | arial | 10 |  | 400 | No |  |  |  |
| Hypertext | Arial | 8 | 255 | 400 | No | Flat | transparent |  |
| RptTitle2\_Highlighted | Arial | 10 |  | 400 | No | Flat | transparent | 10092543 |
| RptGeneral w/Border Highlighted | Arial | 8 |  | 400 | No | Flat | solid | 10092543 |

Control Use

MSAccess features several different types of standard controls. It’s often possible to use two different controls to achieve the same general functionality. For example, a list of static values could be represented as a set of radio buttons or as pop list. The following guidelines should be used to determine what the best control for the job is.

|  |  |  |
| --- | --- | --- |
| Control | # | Rules |
| Check Boxes | 1 | Check Boxes should always be used for True/False fields unless the True/False paradigm does not obviously apply in which case two Radio Buttons should be used |
| Combo Box | 1 | Combo Boxes should be used in cases where a list may change or exceeds 8 values |
|  | 2 | The user should always select on a descriptive value rather than a code |
| List Box | 1 | List Boxes should be used in cases where a list may change and it’s important for the user to be able see the other options |
|  | 2 | The user should always select on a descriptive value rather than a code |
|  | 3 | List Boxes should be used in cases where it’s necessary for the user to keep track of multiple selected/ deselected options (i.e., Countries to include on a report) |
| Command Buttons | 1 | Command Buttons should be used to indicate a decision made by the user. |
| Radio Buttons | 1 | Radio Buttons should be used in cases where there are less than 8 choices and these choices are static. |
|  | 2 | Text Labels should be placed to right of Radio Buttons |
| Tab | 1 | Tab canvases should be used to model forms that would exceed a single screen. Also, tab canvases can be used to incorporate affiliated processes into a single module. |

Table 4.7 – Control rules

Command Buttons

PipeLine contains thirteen standard command buttons that provide consistent functionality throughout the application. The buttons should only be used to provide the following functionality. Additional or different functionality should be provided by another command button.

When a command button is selected, the ON CLICK event procedure is fired, which calls the appropriate code. The table below describes the functionality of standard buttons.

|  |  |
| --- | --- |
| Button | Description |
| Add | Allow the user to enter a new record in the current recordset |
| Delete | Allow the user to delete the current record from the current recordset. |
| Save | Saves all changes to the current record since edit button was selected |
| Cancel | Cancels all changes to the current record since edit button was selected |
| Close | Closes form. Returns to calling form |
| Print | Sends the report directly to the printer. |
| Preview | Shows the print preview of the report. |
| PDF | Sends the report directly to the pdf printer specified. |
| ShowData | Shows data for report based on selected parameters in the detail view of the form. |
| Hide Data | Hides data for report in the detail view of the form. |

Table 4.8 – Functionality of standard buttons

Report Conventions

Report Layout

The conventions on the following pages are to be used when designing and laying out reports:

|  |  |  |
| --- | --- | --- |
| Group | Items | Specification |
| General | Report Output | Should be horizontally centered on the page. |
|  | Default Paper Size (8.25 x 11) | Reports should be created to fit on both A4 and Letter sized paper. |
|  | Parameters | if parameters are required, they should be called from the list view section of the form so that the parameters can be validated. |
| Report Margins | Portrait Reports | .5" Top  .75 Bottom  .5" Left and Right |
|  | Landscape Reports | .5" Top and Bottom  .5" Left  .75" Right |
| Report headings | Report header including system title, report title, any other centered columns, run date, run time, page number | Set by tblStyle |
|  | 1st line of the report title | Report Name |
|  | 2nd line of the report title | SubReport Name |
|  | 3rd line of the report title | Description of Report including filtering summary (if applicable) |
|  | Report title spacing | Report titles should be stacked. There shouldn’t be a line or half line between titles. |
|  | Application Name, Report Display Name, and Program Name | Should be aligned with the left margin |
|  | Application Name | PipeLine 5.1 |
|  | Report Display Name | As specified on Program Form |
|  | Program Name | As specified on Program Form |
|  | Run Date, Run Time, and Page Number | Should be aligned with the right margin |
|  | Run Date | DD-MMM-YYYY Format |
|  | Run Time | Format as short time |
|  | Page Number | =”Page “ & [Page] & “ of “ & [Pages]) |
| Report Body | Report body text | Set by tblStyle |
|  | Report body labels | Set by tblStyle |
| Column Headings | Column headings | Justification of columns should match headings. Right justified amounts should have right justified headings, etc. |
|  | Column headings | Should appear on every page - but only once. |
| Totals | Totals | Should not appear by themselves on a page. |
|  | Total labels | Should appear to the left of the total row. The total label should be formatted bold and include the value of the group being totaled - i.e. Total Kenya or Total AVSC. Don’t use “Subtotal”. |

Table 4.9 – Report Checklist

Icons

The standard MSAccess icons should be used.

VBA Coding Guidelines

VBA can be difficult to read when coded badly. PipeLine follows the Reddick VBA (RVBA) Coding Conventions (see Appendix B). The following outlines the proper way to document and structure VBA program units.

Procedure/Function Declarations

If an event procedure is more than 10 lines of code or is used more than once in the system, then a module should be created containing that public function. Any subroutine of the function should be included in the module and should be private if not being called by any other function.

Variable Declarations

Variables should also follow the naming conventions outlined in this chapter.

Comments

Every event procedure and functions should have a header comment. The header comment should include the following:

The purpose of the procedure or function

The purpose and definition of input parameters

The definition of any returned parameter

The date created and name of the primary developer

The dates, initials and technical notes of any subsequent developers

The header comments should appear in the following format.

‘ Comments:

‘ Parameters:

‘ Returns:

‘ Created:

‘ Modified:

‘––––––––––––––––––-

Any section of code that needs further explanation or is based on an external assumption should be commented. These comments should appear in the following format.

‘ This is the comment

White Space

Wherever the readability of the code will be enhanced (i.e. Around IF statements, FOR Loops, etc.) developer’s are encouraged to use white space.

Indentation

Developers should follow standard indentation practices when writing code.

Global Variables

Unless absolutely necessary, Global Variables should not be used. Instead use passed parameters or form level parameters

Help

Database Schema

Overview

Entity Relationship Diagrams

Table and Column Descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Table: ABC** | | | |
| **Column Name** | **Data Type** | **Not Null?** | **Comments** |
| **Column A** | INTEGER |  | Id. Primary Key |
| **Column B** | NUMBER |  |  |
| **Column C** | VARCHAR2(60) |  | is required |
| **Relations:** | **Primary key (Column A)**  **Parent (Column A)**  **Child (Column C)** | | |
| **Purpose:** | **Stores data for … Maintains relationship with … through foreign key …** | | |
| **Notes:** |  | | |

Other Data Structures

Program Units

Overview

Source Code CD content

Program name

|  |  |
| --- | --- |
| **Purpose:** | |
| **Source code listing to be found on file name …** |

Menu Structure

Overview

Program calls by Menu choices

Archiving and Backup

Interfaces with Other Systems

Overview

System A

System B

References

[EXAMPLES]

Allen, Michael T. 2005. *Health, Nutrition, and Population Sector Programs.* Washington, DC: World Bank.

Health II Project. 2009. *Medical Supplies for Malawi.* Boston: Health II Project (accessed on May 3, 2009: http://health II project/portal/page/portal/604BBCDC49C97).

Moore, Allen, Barbara Jahari, and David Klem. 2006. Vitamin A–Fortified Grain: A New Idea for an Old Problem. *American Journal of Clinical Nutrition.* 48:1265–70.

Office of Technology Assessment (OTA). 2009. *Technology Transfer to the Middle East*. Publication no. OTA-ISC-173. Washington, DC: Government Printing Office.

USAID | DELIVER PROJECT, Task Order 1. 2007. *Implementing Multiple Health Commodities Logistics Management Information Systems.* Arlington, Va.: USAID | DELIVER PROJECT, Task Order 1.

Examples above include references for one author, the Internet, multiple authors, a government publication, and the project.

All references include the—

* author
* date of publication
* full title
* place of publication and publisher.

For more information, log on to [deliver.jsi.com](http://www.deliver.jsi.com).

1. Select *Staff Resources.*
2. Select *Communications Resources.*
3. Select *Documenting Sources*.

APPENDIX A: The Reddick VBA (RVBA) Naming Conventions, Version 6.01

*Copyright © 1992-1999 by Greg Reddick*

The purpose of the Reddick VBA (RVBA) Naming Conventions is to provide a guideline for naming objects in the Visual Basic for Applications (VBA) language. Having conventions is valuable in any programming project. When you use them, the name of the object conveys information about the meaning of the object. These conventions attempt to provide a way of standardizing that meaning across the body of VBA programmers.

VBA is implemented to interact with a host application-for example, Microsoft Access, Microsoft Visual Basic, AutoCAD, and Visio. The RVBA conventions cover all implementations of the VBA language, regardless of the host application. Some of the tags described in this document may not necessarily have an implementation within some of the particular host programs for VBA. The word *object,* in the context of this document, refers to simple variables and VBA objects, as well as to objects made available by the

VBA host program.

While I am the editor of these conventions, they are the work of many people, including Charles Simonyi, who invented the Hungarian conventions on which these are based, and Stan Leszynski, who co-authored several versions of the conventions. Many others, too numerous to mention, have also contributed to the development and distribution of these conventions, but I’d especially like to thank Paul Litwin and Ken Getz who have made substantial contributions over the years.

These conventions are intended as a guideline. If you disagree with a particular part of the conventions, simply replace that part with what you think works better. However, keep in mind that future generations of programmers may need to understand those changes, and place a comment in the header of a module indicating what changes have been made. To be concise, the conventions are presented without rationalizations for how they were derived although each of the ideas presented has a considerable history

to it.

**Changes to the Conventions**

Some of the tags in the version of the conventions presented here have changed from previous versions. Consider all previous tags to be grandfathered into the conventions--you don’t need to go back and make changes. For new development work, I leave it up to you to decide whether to use the older tags or the ones suggested here. In a few places in this document, older tags are shown in {braces}. As updates to this document are made, the current version can be found at the Xoc Software web site, [http://www.xoc.net.](http://www.xoc.net/)

**An Introduction to Hungarian**

The RVBA conventions are based on the Hungarian conventions for constructing object names, named for the native country of the inventor, Charles Simonyi. The objective of Hungarian is to convey information about the object concisely and efficiently. Hungarian takes some getting used to, but once adopted, it quickly becomes second nature. The format of a Hungarian object name is

[prefixes]tag[BaseName[Suffixes]]

The square brackets indicate optional parts of the object name. These components have the following meanings:

Component Meaning

Prefixes Modify the tag to indicate additional information. Prefixes are all lowercase. They are usually picked from a standardized list of prefixes, given later in this document.

Tag Short set of characters, usually mnemonic, that indicates the type of the object. The tag is all lowercase. It is usually selected from a standardized list of tags, given later in this document.

BaseName One or more words that indicate what the object represents. Capitalize the first letter of each word in the BaseName.

Suffixes Additional information about the meaning of the BaseName. Capitalize the first letter of each word in the Suffix. They are usually picked from a standardized list of suffixes, given later in this document.

Notice that the only required part of the object name is the tag. This may seem counterintuitive; you may feel that the BaseName is the most important part of the object name. However, consider a generic procedure that operates on any form. The fact that the routine operates on a form is the important thing, not what that form represents. Because the routine may operate on forms of many different types, you do not necessarily need a BaseName. However, if you have more than one object of a type referenced in the routine, you must have a BaseName on all but one of the object names to differentiate them. In addition, unless the routine is generic, the BaseName conveys information about the variable. In most cases, a variable should include a BaseName.

**Tags**

Use the techniques described in the following sections to construct tags to indicate the data type of an object.

**Variable tags**

Use the tags listed in Table 1 for VBA data types. You can also use a specific tag instead of *obj* for any data type defined by the host application or one of its objects. (See the section “Host Application and Component Extensions to the Conventions” later in this document.)

*Table 1: Tables for VBA Variables*

Tag Object Type bool {f, bln} Boolean

byte {byt} Byte

cur Currency date {dtm} Date

dec Decimal dbl Double int Integer lng Long

obj Object sng Single

str String

stf String (fixed length)

var Variant

Here are several examples:

lngCount intValue strInput

You should explicitly declare all variables, each on a line by itself. Do not use the old-type declaration characters, such as %, &, and $. They are extraneous if you use the naming conventions, and there is no character for some of the data types, such as Boolean. You should always explicitly declare all variables of type Variant using the *As Variant* clause, even though it is the default in VBA. For example:

Dim intTotal As Integer Dim varField As Variant Dim strName As String

**Constructing Properties Names**

Properties of a class present a particular problem: should they include the naming convention to indicate the type? To be consistent with the rest of these naming conventions, they should. However, it is permitted to have property names without the tags, especially if the class is to be made available to customers who may not be familiar with these naming conventions.

**Collection Tags**

You treat a collection object with a special tag. You construct the tag using the data type of the collection followed by the letter *s*. For example, if you had a collection of Longs, the tag is lngs. If it was a collection of forms, the tag for the collection is frms. Although, in theory, a collection can hold objects of different data types, in practice, each of the data types in the collection is the same. If you do want to use different data types in a collection, use the objs tag. For example:

intsEntries frmsCustomerData objsMisc

**Constants**

Constants always have a data type in VBA. Because VBA will choose this data type for you if you don’t specify it, you should always specify the data type for a constant. Constants declared in the General Declarations section of a module should always have a scope keyword of Private or Public, and be prefixed by the scope prefixes *m* or *g*, respectively. A constant is indicated by appending the letter *c* to the end of the data type for the constant. For example:

Const intcGray As Integer = 3

Private Const mdblcPi As Double = 3.14159265358979

Although this technique is the recommended method of naming constants, if you are more concerned about specifying that you are dealing with constants rather than their data type, you can alternatively use the generic tag *con* instead. For example:

Const conPi As Double = 3.14159265358979

**Menu Items**

The names of menu items should reflect their position in the menu hierarchy. All menu items should use the tag mnu, but the BaseName should indicate where in the hierarchy the menu item falls. Use *Sep* in the BaseName to indicate a menu separator bar, followed by an ordinal. For example:

mnuFile (on menu bar)

mnuFileNew (on File popup menu) mnuFileNewForm (on File New flyout menu) mnuFileNewReport (on File New flyout menu) mnuFileSep1 (first separator bar on file popup menu) mnuFileSaveAs (on File popup menu)

mnuFileSep2 (second separator bar on file popup menu)

mnuFileExit (on File popup menu)

mnuEdit (on menu bar)

**Creating Data Types**

VBA gives you three ways to create new data types: enumerated types, classes, and user-defined types. In each case, you will need to invent a new tag that represents the data type that you create.

**Enumerated types**

Groups of constants of the *long* data type should be made an enumerated type. Invent a tag for the type, append a “c,” and then define the enumerated constants using that tag. Because the name used in the Enum line is seen in the object browser, you can add a BaseName to the tag to spell out the abbreviation indicated by the tag. For example:

Public Enum ervcErrorValue ervcInvalidType = 205 ervcValueOutOfBounds

End Enum

The BaseName should be singular, so that the enumerated type should be ervcErrorValue, not ervcErrorValues. The tag that you invent for enumerated types can then be used for variables that can contain values of that type. For example:

Dim erv As ervcErrorValue

Private Sub Example(ByVal ervCur As ervcErrorValue)

While VBA only provides enumerated types of groups of the long type, you can still create groups of constants of other types. Just create a set of constant definitions using an invented tag. For example:

Public Const estcError205 As String = "Invalid type"

Public Const estcError206 As String = "Value out of bounds"

Unfortunately, because this technique doesn’t actually create a new type, you don’t get the benefit of the VBA compiler performing type checking for you. You create variables that will hold constants using a similar syntax to variables meant to hold instances of enumerated types. For example:

Dim estError As String

**Tags for classes and user-defined types**

A class defines a user-defined object. Because these invent a new data type, you will need to invent a new tag for the object. You can add a BaseName to the tag to spell out the abbreviation indicated by the tag. User-defined types are considered a simple class with only properties, but in all other ways are used the same as class modules. For example:

gphGlyph edtEdit

Public Type grbGrabber

You then define variables to refer to instances of the class using the same tag: For example:

Dim gphNext As New gphGlyph Dim edtCurrent as edtEdit Dim grbHandle as grbGrabber

**Polymorphism**

In VBA, you use the *Implements* statement to derive classes from a base class. The tag for the derived class should use the same tag as the base class. The derived classes, though, should use a different BaseName from the base class. For example:

anmAnimal (base class)

anmZebra (derived class of anmAnimal)

anmElephant (derived class of anmAnimal)

This logic of naming derived classes is used with forms, which are all derived from the pre-defined Form base class and use the frm tag. If a variable is defined to be of the type of the base class, then use the tag, as usual. For example:

Dim anmArbitrary As anmAnimal

Dim frmNew As Form

On the other hand, if you define a variable as an instance of a derived class, include the complete derived class name in the variable name. For example:

Dim anmZebraInstance As anmZebra

Dim anmElephantExample As anmElephant

Dim frmCustomerData As frmCustomer

**Constructing Procedures**

VBA procedures require you to name various items: procedure names, parameters, and labels. These objects are described in the following sections.

**Constructing Procedure Names**

VBA names event procedures, and you cannot change them. You should use the capitalization defined by the system. For user-defined procedure names, capitalize the first letter of each word in the name. For example:

cmdOK\_Click GetTitleBarString PerformInitialization

Procedures should always have a scope keyword, Public or Private, when they are declared. For example:

Public Function GetTitleBarString() As String

Private Sub PerformInitialization

**Naming Parameters**

You should prefix all parameters in a procedure definition with ByVal or ByRef, even though ByRef is optional and redundant. Procedure parameters are named the same as simple variables of the same type, except that arguments passed by reference use the prefix “r.” For example:

Public Sub TestValue(ByVal intInput As Integer, ByRef rlngOutput As Long) Private Function GetReturnValue(ByVal strKey As String, \_

ByRef rgph As Glyph) As Boolean

**Naming Labels**

Labels are named using upper and lower case, capitalizing the first letter of each word. For example:

ErrorHandler: ExitProcedure:

**Prefixes**

Prefixes modify an object tag to indicate more information about an object.

**Arrays of Objects Prefix**

Arrays of an object type use the prefix “a.” For example:

aintFontSizes astrNames

**Index Prefix**

You indicate an index into an array by the prefix *i*, and for consistency the data type should always be a long. You may also use the index prefix to index into other enumerated objects, such as a collection of user-defined classes. For example:

iaintFontSizes iastrNames igphsGlyphCollection

**Prefixes for Scope and Lifetime**

Three levels of scope exist for each variable in VBA: Public, Private, and Local. A variable also has a lifetime of the current procedure or the lifetime of the object in which it is defined. Use the prefixes in Table 2 to indicate scope and lifetime.

*Table 2: Scope prefixes*

Prefix Object Type

(none) Local variable, procedure-level lifetime, declared with “Dim”

s Local variable, object lifetime, declared with “Static”

m Private (module) variable, object lifetime, declared with “Private”

g Public (global) variable, object lifetime, declared with “Public”

You also use the “m” and “g” constants with other objects, such as constants, to indicate their scope. For example:

intLocalVariable mintPrivateVariable gintPublicVariable mdblcPi

VBA allows several type declaration words for backward compatibility. The older keyword “Global” should always be replaced by “Public,” and the “Dim” keyword in the General Declarations section should be replaced by “Private.”

**Other Prefixes**

Table 3 lists and describes some other prefixes:

*Table 3: Other commonly-used prefixes*

Prefix Object Type

c Count of some object type

h Handle to a Windows object

r Parameter passed by reference

Here are some examples:

castrArray hWndForm

**Suffixes**

Suffixes modify the base name of an object, indicating additional information about a variable. You’ll

likely create your own suffixes that are specific to your development work. Table 4 lists some generic VBA

suffixes.

*Table 4: Commonly-used suffixes*

Suffix Object Type

Min The absolute first element in an array or other kind of list

First The first element to be used in an array or list during the current operation

Last The last element to be used in an array or list during the current operation

Lim The upper limit of elements to be used in an array or list. Lim is not a valid index. Generally, Lim equals Last + 1

Max The absolutely last element in an array or other kind of list

Cnt Used with database elements to indicate that the item is a Counter. Counter fields are incremented by the system and are numbers of either type Long or type Replication Id.

Here are some examples:

iastrNamesMin iastrNamesMax iaintFontSizesFirst igphsGlyphCollectionLast lngCustomerIdCnt varOrderIdCnt

**File Names**

When naming items stored on the disk, no tag is needed because the extension already gives the object type. For example:

Test.Frm (frmTest form)

Globals.Bas (globals module) Glyph.Cls (gphGlyph class module)

**Host Application and Component Extensions to the**

**Conventions**

Each host application for VBA, as well as each component that can be installed, has a set of objects it can use. This section defines tags for the objects in the various host applications and components.

**Access 2000, Version 9.0 Objects**

Table 5 lists Access object variable tags. Besides being used in code to refer to these object types, these same tags are used to name these kinds of objects in the form and report designers.

*Table 5: Access object variable tags*

Tag Object Type aob AccessObject

aops AccessObjectProperties aop AccessObjectProperty app Application

bfr BoundObjectFrame chk CheckBox

cbo ComboBox

cmd CommandButton ctl Control

ctls Controls

ocx CustomControl dap DataAccessPage dcm DoCmd

frm Form

fcd FormatCondition fcds FormatConditions frms Forms

grl GroupLevel hyp Hyperlink img Image

lbl Label lin Line

lst ListBox bas Module

ole ObjectFrame opt OptionButton

fra OptionGroup (frame)

brk PageBreak

pal PaletteButton prps Properties

shp Rectangle ref Reference refs References rpt Report

rpts Reports scr Screen sec Section sfr SubForm

srp SubReport tab TabControl txt TextBox

tgl ToggleButton

Some examples:

txtName lblInput

For ActiveX custom controls, you can use the tag ocx as specified in Table 5 or more specific object tags that are listed later in this document in Tables 14 and 15. For an ActiveX control that doesn't appear in the Tables 14 or 15, you can either use ocx or invent a new tag.

**DAO 3.6 Objects**

DAO is the programmatic interface to the Jet database engine shared by Access, Visual Basic, and Visual

C++. The tags for DAO 3.6 objects are shown in Table 6.

*Table 6: DAO object tags*

Tag Object Type cnt Container cnts Containers db Database

dbs Databases dbe DBEngine doc Document docs Documents err Error

errs Errors fld Field flds Fields grp Group grps Groups idx Index idxs Indexes

prm Parameter prms Parameters pdbe PrivDBEngine prp Property

prps Properties qry QueryDef qrys QueryDefs rst Recordset rsts Recordsets rel Relation rels Relations tbl TableDef tbls TableDefs

usr User usrs Users

wrk Workspace wrks Workspaces

Here are some examples:

rstCustomers idxPrimaryKey

Table 7 lists the tags used to identify types of objects in a database.

*Table 7: Access Database Explorer object tags*

Tag Object Type tbl Table

qry Query frm Form rpt Report mcr Macro bas Module

dap DataAccessPage

If you wish, you can use tags that are more exact or suffixes to identify the purpose and type of a database object. If you use the suffix, use the tag given from Table 7 to indicate the type. Use either the tag or the suffix found along with the more general tag, but not both. The tags and suffixes are shown in Table 8.

*Table 8: Specific object tags and suffixes for Access Database Explorer objects*

Tag Suffix Object Type tlkp Lookup Table (lookup) qsel (none) Query (select) qapp Append Query (append) qxtb XTab Query (crosstab) qddl DDL Query (DDL) qdel Delete Query (delete) qflt Filter Query (filter) qlkp Lookup Query (lookup)

qmak MakeTable Query (make table)

|  |  |  |
| --- | --- | --- |
| qspt | PassThru | Query (SQL pass-through) |
| qtot | Totals | Query (totals) |
| quni | Union | Query (union) |
| qupd | Update | Query (update) |
| fdlg | Dlg | Form (dialog) |
| fmnu | Mnu | Form (menu) |
| fmsg | Msg | Form (message) |
| fsfr | SubForm | Form (subform) |
| rsrp | SubReport | Form (subreport) |
| mmnu | Mnu | Macro (menu) |

Here are some examples:

tblValidNamesLookup tlkpValidNames fmsgError mmnuFileMnu

When naming objects in a database, do not use spaces. Instead, capitalize the first letter of each word. For example, instead of Quarterly Sales Values Table, use tblQuarterlySalesValues.

There is strong debate over whether fields in a table should have tags. Whether you use them is up to you. However, if you do use them, use the tags from Table 9.

*Table 9: Field tags (if you decide to use them)*

Tag Object Type

lng Autoincrementing (either sequential or random) Long (used with the suffix Cnt)

bin Binary byte Byte

cur Currency date Date/time dbl Double

guid Globally unique identified (GUID) used for replication AutoIncrement fields int Integer

lng Long mem Memo ole OLE

sng Single str Text bool Yes/No

**Visual Basic 6.0 Objects**

Table 10 shows the tags for Visual Basic 6.0 objects.

*Table 10: Visual Basic 6.0 object tags* Tag Object Type app App

chk CheckBox clp Clipboard cbo ComboBox

cmd CommandButton ctl Control

dat Data

dir DirListBox drv DriveListBox fil FileListBox frm Form

fra Frame glb Global

hsb HScrollBar img Image

lbl Label

lics Licenses lin Line

lst ListBox mdi MDIForm mnu Menu

ole OLE

opt OptionButton pic PictureBox

prt Printer

prp PropertyPage scr Screen

shp Shape

txt TextBox tmr Timer

uctl UserControl udoc UserDocument vsb VscrollBar

**Microsoft ActiveX Data Objects 2.1 Tags**

Office 2000 provides version 2.1 of the ActiveX Data Objects library. Table 11 lists the recommended tags for this version of ADO.

Note: Many of the ADO, ADOX, and JRO tags overlap with existing DAO tags. Make sure you include the object library name in all references in your code, so there’s never any possibility of confusion. For example, use

Dim rst As ADODB.Recordset

or

Dim cat As ADOX.Catalog

rather than using the object types without the library name. This will not only make your code more explicit and avoid confusion about the source of the object, but will also make your code run a bit faster.

*Table 11: ADO 2.1 Object tags*

Tag Object Type cmn {cmd} Command cnn {cnx} Connection err Error

errs Errors fld Field flds Fields

prm Parameter prms Parameters

prps Properties prp Property rst Recordset

**Microsoft ADO Ext. 2.1 for DDL and Security (ADOX) Tags**

In order to support DDL and security objects within Jet database, Microsoft provides ADOX, an additional

ADO library of objects. Table 12 lists tags for the ADOX objects.

*Table 12: ADOX Object tags*

Tag Object Type cat Catalog

clms Column clm Columns cmd Command grp Group

grps Groups idx Index idxs Indexes key Key keys Keys

prc Procedure prcs Procedures prps Properties prp Property

tbl Table tbls Tables usr User usrs Users vw View vws Views

**Microsoft Jet and Replication Objects 2.1**

In order to support Jet’s replication features, ADO provides another library (JRO). Table 13 lists suggested tags for the JRO objects.

*Table 13: JRO object tags*

Tag Object Type flt Filter

flts Filters

jet JetEngine rpl Replica

**Microsoft SQL Server and Microsoft Data Engine (MSDE) Objects**

Table 14 lists RVBA tags for Microsoft SQL Server and the Microsoft Data Engine (a limited-connection version of SQL Server 7) objects.

*Table 14: SQL Server/MSDE object tags* Tag Object Type tbl table

proc stored procedure trg trigger

qry view

dgm database diagram pk primary key

fk foreign key

idx other (non-key) index rul check constraint

def default

**Microsoft Common Control Objects**

Windows 95 and Windows NT have a set of common controls that are accessible from VBA. Table 15 lists the tags for objects created using these controls.

*Table 15: Microsoft Common Control Object tags.*

Tag Object Type ani Animation

btn Button (Toolbar)

bmn ButtonMenu (Toolbar) bmns ButtonMenus (Toolbar) bnd Band (CoolBar)

bnds Bands (CoolBar)

bnp BandsPage (CoolBar)

btns Buttons (Toolbar)

cbr CoolBar

cbp CoolBarPage (CoolBar) hdr ColumnHeader (ListView) hdrs ColumnHeaders (ListView) cbi ComboItem (ImageCombo) cbis ComboItems (ImageCombo) ctls Controls

dto DataObject

dtf DataObjectFiles dtp DTPicker

fsb FlatScrollBar imc ImageCombo iml ImageList

lim ListImage lims ListImages

lit ListItem (ListView)

lits ListItems (ListView)

lsi ListSubItem (ListView) lsis ListSubItems (ListView) lvw ListView

mvw MonthView

nod Node (TreeView)

nods Nodes (TreeView)

pnl Panel (Status Bar) pnls Panels (Status Bar) prb ProgressBar

sld Slider

sbr StatusBar

tab Tab (Tab Strip) tabs Tabs (Tab Strip) tbs TabStrip

tbr Toolbar tvw TreeView udn UpDown

**Other Custom Controls and Objects**

Finally, Table 16 lists the tags for other commonly used custom controls and objects.

*Table 16: Tags for commonly-used custom controls*

Tag Object Type

cdl CommonDialog (Common Dialog) dbc DBCombo (Data Bound Combo Box) dbg DBGrid (Data Bound Grid)

dls DBList (Data Bound List Box)

gau Gauge (Gauge) gph Graph (Graph) grd Grid (Grid)

msg MAPIMessages (Messaging API Message Control) ses MAPISession (Messaging API Session Control) msk MaskEdBox (Masked Edit Textbox)

key MhState (Key State)

mmc MMControl (Multimedia Control) com MSComm (Communication Port) out Outline (Outline Control)

pcl PictureClip (Picture Clip Control)

rtf RichTextBox (Rich Textbox)

spn SpinButton (Spin Button)

**Summary**

Using a naming convention requires a considerable initial effort on your part. The payoff comes when either you or another programmer has to revisit your code later. Using the conventions given here will make your code more readable and maintainable.

*Greg Reddick is the President of Xoc Software, a software development company developing programs in Visual Basic, Microsoft Access, C/C++, and for the web. He leads training seminars in Visual Basic for Application Developers Training Company (AppDev). In a previous life, he worked for four years on the Access development team at Microsoft. Greg can be reached at* [*mailto:grr@xoc.net*](mailto:mailto:grr@xoc.net) *or from the Xoc Software web site,* [*http://www.xoc.net.*](http://www.xoc.net/)

APPENDIX B: The Reddick VBA (RVBA) Coding Conventions (version 0.90)

*Copyright © 1999 by Greg Reddick*

What follows are the Reddick VBA (RVBA) Coding Conventions. The objectives of the conventions are to make code:

* More readable: Conventions allow a reader to understand the meaning of the code with less effort.
* More maintainable: The code can be more reliably changed to fix bugs and enhance functionality.
* More reliable: The code is more likely to perform as expected.
* More efficient: The code performs faster or consumes fewer resources.

These conventions are separate from the RVBA Naming Conventions and may be adopted without adopting the naming conventions. The current version of these conventions can always be found on the Xoc Software web site: http://www.xoc.net.

More rational is provided for these recommendations than is given in the RVBA Naming Conventions. In most cases, there are good rationales for the given conventions. However, in some cases an arbitrary decision was made to select one convention from a set of reasonable alternatives. The other reasonable alternatives to the conventions placed in {braces} at the end of a section. In some cases, a topic only relates to the Visual Basic 6.0 development environment, as opposed to VBA in general. In those cases, the topic is marked with [VB6] after the topic heading.

No set of conventions can cover every case or every consideration. The general rule is that exceptions to the conventions can be made with the approval of the programming team after careful consideration.

The sections are listed in alphabetical order to facilitate their use as a reference work. However, this makes the flow of the document unusual for casual reading as some topics are much more technical than others.

**Arrays**

Always specify the both the lower and upper bound of an array. This makes explicit whether element zero of the array is a valid element or not. For example:

Dim astrValue(1 To 10) As String

By convention the index variable used to walk an array should always be a Long data type. This assures that if the array size grows past 32767 elements when maintaining the program that the index variable can still address all elements in the array

When walking an array, always use the VBA LBound and UBound functions to visit each item. This makes sure that every item in the array is visited. For example:

Dim iastrValue As Long

For iastrValue = LBound(astrValue) To UBound(astrValue)

MsgBox astrValue(iastrValue)

Next iastrValue

**Assertions**

VBA provides a built-in assertion mechanism through Debug.Assert. If the expression following the Debug.Assert evaluates to True, the code continues. If the expression evaluates to False, VBA enters Break mode as if a breakpoint had been set on that line. The line shown here acts as a hard coded breakpoint:

Debug.Assert False

Assertions that do not include a function call in the expression are removed by the compiler when an executable is made, so they only apply to debugging inside the VBA environment. Assertions with a function call in the expression will remain in the executable, but the resulting value of the expression is discarded. VBA doesn't remove function calls because they may have side effects, but discards the return value from the function.

Any time that there is an assumption in the code about the state of the program, there should be an assertion that states the assumption. For example, suppose that a procedure includes this code:

Select Case intValue

Case 1

MsgBox "Aircraft"

Case 2

MsgBox "AutoMobile"

Case 3

MsgBox "SnowMobile"

End Select

This code assumes that the value of intValue is between one and three. However, if through some bug, intValue had the value of zero or four, this code doesn't work right. The result is that no MsgBox appears at all. Tracking down why the MsgBox doesn't appear is time consuming. Instead, the code could be written one of two other ways. Either:

Debug.Assert intValue >= 1 And intValue <= 3

Select Case intValue

Case 1

MsgBox "Aircraft"

Case 2

MsgBox "AutoMobile"

Case 3

MsgBox "SnowMobile"

End Select

Or

Select Case intValue

Case 1

MsgBox "Aircraft"

Case 2

MsgBox "AutoMobile"

Case 3

MsgBox "SnowMobile"

Case Else

Debug.Assert False

End Select

In general, every Select/End Select block should have a Case Else to trap unexpected values. If the Case Else should never occur, then a Debug.Assert False should be inserted into the block. If the code is correctly written to handle 1 To 3, but zero and four are allowed values, the code should be written with a comment in the Case Else block to indicate that this is expected, like this:

Select Case intValue

Case 1

MsgBox "Aircraft"

Case 2

MsgBox "AutoMobile"

Case 3

MsgBox "SnowMobile" Case Else 'Do nothing End Select

Assertions trap logic errors early. Rather than waiting to see the results of a bug in the use interface, there is immediate feedback that the bug has occurred. Assertions are only effective if they are present, which means that they have to be added when writing the code. Any logic error that is fixed in the code is a good indication that some additional assertions need to be added.

**Comments**

A comment in VBA starts with an apostrophe and ends at the end of the line. Comments may be placed on a line by themselves or at the end of a line. A comment starts with the apostrophe followed immediately by the text with no space between the two.

The comment at the end of a line should be used in only a few places:

* At the end of a declaration line
* On a Case line
* On the line that ends a block to indicate what block is being ended. For example on a set of nested If/End If blocks, a comment on the End If line may say what If block is completed. This is especially useful if the block spans several screens.

Examples:

Dim dateUTC As String 'time in Univeral Coordinated Time Case 11 'Division by Zero

If the end of a line comment line exceeds the 80 characters line limit, continue the comment on the next line indented by one tab stop. For example:

Case 35602 'This key is already associated with an element of this 'collection Set nodChild = tvw.Nodes.Item(cci.Guid)

All other comments should be placed on a separate line above the line they are documenting and indented to the same level. A comment of this sort is generally preceded by a blank line unless it is the first line of an indented block. For example:

vt = vti.VarType

'Special hack for analyzing my code If LCase$(Left$(strParamName, Len(strcDecPrefix))) = strcDecPrefix Then strDataType = strDecimal End If

If it is the first line of an indented block, it is not preceded by a blank line. For example:

If mboolShowProperties Then 'Show properties for each member For Each mi In ci.Members

Comments should state the intention of the code not how it performs the task. This is an example of a worthless comment:

'Place the VarType into the vt variable vt = vti.VarType

It is worse than no comment at all. The comment is wrong if the code changes to use the variable name vtCur instead of vt without changing the comment. When reading a comment that doesn't match the code, the question becomes whether the comment is correct or the code is correct. Usually it is the comment that is wrong, but it may take some time to prove that. A wrong comment can be worse than no comment at all. A comment that says the same thing as the following line of code is worthless. In general, don't write comments that have to be maintained, because in the real-world comments frequently aren't maintained.

A comment that states the intention of the code, though, may be useful. For example:

'Store VarType for recovery in error condition. vt = vti.VarType

However, use these comments only when the intention is not immediately clear when reading the code. Instead strive to make the code self explanatory, through good naming and coding conventions.

**Constants**

Always give constants an explicit data type. For example:

Private Const dblcPi As Double = 3.14159265358979

If a literal value other than zero or one appears in the code, consideration should be given to whether it makes things more readable and maintainable to replace it with a constant. Replace a magic number used more than once in the code with a constant.

Global constants are allowed and encouraged. Replace sets of constants of the data type long with enumerated types using Enum.

**Date Functions and Date Variables**

Be careful about using the VBA date functions: Date, DateAdd, DateSerial, DateValue, and Now. These functions return a variant containing a date. If implicit type conversion to turns the return value into a string, the string representation of the date displays a two-digit year number. That year number is, of course, not Y2K compliant. This also applies to allowing a variable of type Date to be converted into a string. Instead, use the Format$ function to convert the date into a string. For example:

strValue = Format$(Date, "mm/dd/yyyy")

**Default Properties**

Using default properties makes code difficult to read. VBA allows you to just use the name of a textbox and looks up the default property, Text. For example:

MsgBox txtValue

This prints the value of the txtValue textbox. On the other hand, it is much clearer to say:

MsgBox txtValue.Text

To even be more explicit, it could even be expressed as:

MsgBox Me.txtValue.Text

This, however, does not add any additional worth because all references to a control in a module from a form are implicitly on Me.

The reason to be explicit about default properties is to keep the programmer from having to figure out what property is being referenced. This is especially true when referencing ActiveX controls and ActiveX DLLs where the default properties are obscure. For example, when an ADO field is referenced, you are allowed to say:

varValue = rst!strFirstName

This references the Value property of the strFirstName field. However, it is much clearer to say:

varValue = rst.Fields.Item("strFirstName").Value

This code doesn't use any default properties and retrieves the same value.

**Deprecated Features**

Avoid using features Visual Basic supports only for backwards compatibility. Avoid using undocumented features. Also, avoid using functionality that VBA has replaced with functionality that is more modern. Some examples of these kinds of features:

* %, &, $, Etc. in declaration of variables
* Rem statements
* Line numbers (except in conjunction with the Erl function in special error handling situations)
* Single line If statements (use If/End If blocks instead). For example, don't use:

If boolValue Then MsgBox "Hi There"

* While/Wend loops (replace with Do While/Loop)
* Variables declared with Global (use Public instead). Using Dim in the General Declarations section (use Private instead)
* Using user defined types except in the case of Windows API calls or reading fixed width record files (use Class modules instead)
* Gosub
* The End statement in most cases (simply unload the last form in a standard EXE instead)

**Disambiguation**

When referencing classes from an ActiveX library, always use the library name to explicitly tell VBA from what library to get the class. If you don't, then VBA will use the order of the libraries in the References dialog to determine from which library it gets the class. The library name always appears in the upper left-hand listbox of the VBA object browser. For example, if there are references to both the Access and Excel object libraries, then this is ambiguous:

Dim appObj As Application

Because both the Access and Excel libraries include an Application class, which Application class is referenced depends on which one appears first in the References dialog. Instead, it should be declared like this:

Dim appObj As Excel.Application

Microsoft refers to this as "disambiguation". With this declaration, it does not matter what the order of the libraries is inside the References dialog, as appObj will always refer to the Excel Application object. All references to class names in libraries should include the disambiguating library name.

**DLL Base Address [VB6]**

The base address is the location that the DLL is loaded into memory. If two DLLs are loaded into the same base address, then VBA moves the second DLL to a new address. VBA then has to modify the binary code within the DLL's address space to reflect the new address. This slows down loading the second DLL.

Libraries used together should start at different base addresses. In Visual Basic, enter the Base Address for a library in the Project Properties dialog Compile tab. Enter a random number base address different than any other used at the same time.

**Dollar Sign ($) Functions**

If the result of a function is used as a string or assigned it to a string variable, use the $ form of the function. This results in faster executing code, because a conversion from a variant to a string is unnecessary. For example, this is proper usage of dollar sign functions:

If LCase$(Left$(strParamName, Len(strcDecPrefix))) = strcDecPrefix Then

This example calls the LCase$ and Left$ functions instead of the LCase and Left functions because the result is used as a String. If the result is used as a Variant, then call the LCase and Left functions instead.

The $ version of the function returns the same value as the Variant version. The one except to the rule is the VBA Date function. The Date function should always be used because the Date$ function doesn't behave correctly. The Date$ always returns information in mm-dd-yyyy format regardless of the Windows localization settings, whereas the Date function uses the localization settings.

**Error Handling**

A procedure should always include runtime error handling. In general, Error handling should be blocked out the same way in every procedure, as shown in this example:

Private Sub Test() On Error GoTo ErrorHandler

'Code for the procedure goes here

ExitProcedure:

On Error Resume Next

'Cleanup code for the procedure goes here Exit Sub ErrorHandler:

Select Case Err.Number

'Case statements for expected errors goes here

Case Else

Call UnexpectedError(Err.Number, Err.Description, Err.Source, \_

Err.HelpFile, Err.HelpContext)

End Select

Resume ExitProcedure End Sub

Use the label names shown in the example, although the label names have been arbitrarily chosen. Notice that the Exit Sub and ErrorHandler label are left justified making them easily findable. Case statements for expected errors should be given with the error number and a comment with the error message. For example:

Select Case Err.Number

'Case statements for expected errors go here

Case 11 'Division by zero

MsgBox "Zero isn't a valid divisor", vbExclamation, Me.Caption

Case Else

The UnexpectedError routine is a global routine that is only called in a condition where a runtime error that isn’t expected is received, so that there is a bug in the problem. This procedure should log the error message. At the absolute minimum it should just look like this, but ideally it should do a lot more to log the error:

Public Sub UnexpectedError(ByVal lngNumber As Long, \_

ByVal strDescription As String, ByVal strSource As String, \_ ByVal strHelpfile As String, ByVal lngHelpContext As Long) On Error Resume Next MsgBox "[" & strSource & "]" & vbCrLf & "Run-time error '" \_

& CStr(lngNumber) & "':" \_ & vbCrLf & vbCrLf & strDescription, vbExclamation, App.Title, \_ strHelpfile, lngHelpContext

Debug.Print "Case " & CStr(lngNumber) & " '" & strDescription Debug.Assert False End Sub

The first executable line of every procedure should be the On Error GoTo ErrorHandler line. The only exception to the rule is when a procedure checks the values of its arguments and generates a runtime error when they are invalid. In this case, the checking code comes before the On Error GoTo line. For example

Public Sub Test(ByVal intValue As Integer) If intValue < 1 Or intValue > 10 Then Call Err.Raise(Number:=lngcInvalidValue, \_

Description:=strcInvalidValue) End If On Error Goto ErrorHandler

**Exiting a Procedure**

In general, a procedure should only have one exit point. Having one exit point makes it easier to read the code and understand when and where it exits. If you use the code mentioned in the Error Handling code section, that exit point is the Exit Sub, Exit Function, or Exit Property line at the top of the error handling. The only other way to exit the procedure should be through using Err.Raise. These Err.Raise lines should occur either before the On Error GoTo line when validating the parameters (see Error Handling) or inside the error handler.

In a few cases, there may be a need to raise an error inside the body of the procedure. In such cases, you should explicitly set any object variables to Nothing (see Nothing), and then exit the procedure. In such cases, the exiting the procedure should be explicitly detailed by a comment that shows the exit, consisting of an arrow stretching to 80 character right margin. The On Error GoTo 0 statement has to be used to turn off error handling for this procedure before executing the Err.Raise. For example, if this code appears somewhere after the On Error GoTo line, it should be written like this to make it explicit that there is an exit point in the middle of the procedure:

If intValue > 1000 Then 'Raise an Error-------------------------------------------------------> Set rst = Nothing On Error Goto 0 Call Err.Raise(Number:=lngcInvalidValue, \_

Description:=strcInvalidValue) End if

**For/Next and For Each/Next Loops**

The index variable used in the For/Next loop should be specified on the Next line. This makes it explicit which For loop is being completed. For example:

Dim iastrValue As Long For iastrValue = LBound(astrValue) To UBound(astrValue) MsgBox astrValue(iastrValue) Next iastrValue

The object variable used to walk the collection should be placed on the Next line in a For Each/Next loop. For example:

Dim frm As Form For Each frm in Forms

If Not (frm Is Me) Then Unload Me End If Next frm

**GoTo Statements**

You can usually avoid using GoTo statements in VBA code. Use GoTo statements only when the alternative code is not as clear as the GoTo statement. A common reason to use a GoTo statement is to jump out of nested loops. For Example:

For iastrOuterLoop = 1 To 10

For iastrInnerLoop = 1 To 100 'some other code If astr(iastrOuterLoop, iastrInnerLoop) = "Done" Then

GoTo ExitNestedLoops End If 'some other code

Next iastrInnerLoop Next iastrOuterLoop ExitNestedLoops: 'More code here

**Headers**

Each module should start with a header code that looks something like this:

'$Header: $ '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Option Explicit 'This module includes definitions of Windows API calls

The line of asterisks is an apostrophe followed by 79 asterisks. See the section on Long Lines.

Each Public procedure should begin with a header block that looks something like this:

Public Sub Almanac(ByVal lngTrecena As Long, ByVal vein As veinc, \_ ByVal lngRows As Long, ByRef alngBlack() As Long, \_ ByRef alngRed() As Long, ByRef aveinRowStart() As veinc, \_ ByRef aveinAlmanac() As veinc, ByRef lngComplete As Long) 'Generates a Maya almanac

|  |  |
| --- | --- |
| 'lngTrecena [in] | Upper left corner trecena |
| 'vein [in] | Upper left corner veintena |
| 'lngRows [in] | Number of rows in the almanac |
| 'alngBlack() [in] | Black distance numbers across almanac |
| 'alngRed() [out] | Calculated Red trecena numbers across almanac |
| 'aveinRowStart() [out] | Calculated Leftmost shown veintenas in almanac |
| 'aveinAlmanac() [out] | Actual veintenas implied by almanac |
| 'lngComplete [out] | Number almanac misses completing by. |
| 'Return value: | None |

'If lngComplete returns zero then it is an almanac, if it is non-zero, 'then it misses completing and you'll need to report that. You will still 'need to handle the error encNotAnAlmanac because the black numbers 'in alngBlack must wrap back to the starting lngTrecena. On Error GoTo ErrorHandler

Read/write values allowed are [in], [out], and [inout].

Event procedures do not need a header unless the scope is changed to Public. Private procedures may need the header depending on the context. Note that the name of the routine is not referenced in the comments, making it possible to change the name of the procedure without changing the comments. No change history or coding history is included. Histories should be maintained by source code control systems, not by programmers since they are rarely properly kept up to date.

The comments are addressed to the person calling the procedure, and should include just enough information to tell the person how to call the procedure and use the returned values. After the On Error GoTo, other comments can be placed describing algorithms and other implementation details, if needed (although see the section on Comments).

**Indenting**

Tab stops should be set at four spaces. No member of a programming team should vary this number, as it makes editing other members of the team's code difficult.

All code inside a block should be indented one tab stop from the surrounding code, with exceptions noted elsewhere in this document. Indenting blocks makes finding the start and end of the block easy. A block is defined as the code that falls between the following keywords:

* Do/Loop
* Enum/End Enum
* For/Next
* For Each/Next
* Function/Exit Function/End Function
* If/Else/ElseIf/End If
* #If/#Else/#ElseIf/#End If
* Property/Exit Property/End Property
* Sub/Exit Sub/End Sub
* Type/End Type
* With/End With

For example:

For Each ci In tlio.Constants

Set nodChild = tvw.Nodes.Add(Relative:=nod.Key, \_ Relationship:=tvwChild, \_ Key:=ci.Guid & ci.Name, Text:=ci.Name, Image:=strcEnum)

nodChild.EnsureVisible DoEvents If mboolShowProperties Then

For Each mi In ci.Members

Set nodEnumChild = tvw.Nodes.Add(Relative:=nodChild.Key, \_ Relationship:=tvwChild, Text:=mi.Name & strcEquals & \_ mi.Value, Image:=strcConstant)

nodEnumChild.EnsureVisible DoEvents Next mi End If Next ci

See also the section on Select/End Select Blocks.

{Alternative: The entire programming team may standardize on another number of spaces.}

{Alternative: Exit Function, Exit Property, and Exit Sub statements may be indented to the level of the surrounding code.}

**Instantiation**

An object variable should not be declared with New on the line it is declared on, unless there is a good reason to do so. The declaration should instead be broken into two lines. For example:

Dim rst As ADODB.Recordset Set rst = New ADODB.Recordset

Not this:

Dim rst As New ADODB.Recordset

Breaking it into two lines causes each reference to the rst variable to execute slightly faster. In addition, the object variable can be tested to see if it contains the value Nothing. For example:

If rst Is Nothing Then MsgBox "rst not initialized" End If

If a one-line declaration is used, the above code would never execute the MsgBox because the reference to the rst variable in the If statement causes the object to be instantiated before the Is operator is evaluated. For Private and Public object variables, occasionally the convenience of using the New keyword outweighs the performance benefit, so the one-line declaration may still be used.

**Labels**

Labels in the code should be left justified, regardless of the indenting level of the surrounding code. They should appear on a line by themselves. For example:

ExitProcedure: On Error Resume Next

**Long Lines of Code**

VBA code editors will scroll a line of code to make the end visible. However, this makes it difficult to read the code quickly. It also means the code is not understandable if placed into a media that doesn't scroll, such as a paper print out or a book. For these reasons, the length of lines should be restricted.

A physical line of code should not exceed 80 characters. If a logical line of code exceeds 80 characters, then the line should be broken into two or more physical lines using the underscore line continuation character. All physical lines in the logical line following the first physical line should be indented one tab stop (four spaces) from the first physical line.

It may help to place a line at the top of the module with an apostrophe followed by 79 asterisks. Then the code window of the VBA editor can be sized to barely make the last asterisk visible. A fixed width font, such as Courier New, should be used to display the code in the VBA code window.

You should choose an appropriate place to break the line to enhance the maximum readability of the remaining code. When breaking lines that have a list separated by commas, you should break the line after a comma and before the next non-space character. For example:

Private Sub GetFiles(ByRef fso As Scripting.FileSystemObject, \_ ByRef fld As Scripting.Folder)

When breaking a line that is an expression built by operators, break the line before an operator of the expression. For example when the expression is built of string concatenation operators, break it like this:

strParameters = strParameters & strAdd \_ & strPassingConvention & pmi.Name & strArray \_ & strcAs & strDataType & strDefault

The next line becomes more readable this way.

If you have a long literal string, you may have to break the line like this:

strValue = "This is a very, very long string that will cause the code " \_ & "to wrap. Because of this, you will need to break it."

In such cases, break it before the start of a word. Note that VBA performs the string concatenation at runtime, so this has performance considerations. In many cases, the string should be placed into a constant, an entry in resource file, or a database field and retrieved from there.

Comments should never be continued. When a comment exceeds 80 characters, continue the comment on the next line preceded by another apostrophe. See the section on comments.

Don't overly indent lines. Move overly indented code to a new procedure and call it from the original. In general, code should not need to be indented more than eight tab stops.

{Alternative: Place operators at the end of the line before the line continuation character instead of on the next line.}

**Nothing**

Explicitly set Object variables to Nothing before allowing the variable to be destroyed. This is especially true of object variables declared with the Dim keyword. For Example:

Public Sub Test(ByVal intValue As Integer)

'error handling omitted for clarity Dim rst As ADODB.RecordSet Set rst = New ADODB.RecordSet 'More code here Set rst = Nothing

End Sub

Setting the object variable to Nothing is not just good programming practice. If the rst object has code in its Class\_Terminate event handler, that code can mess with global variables and objects.

In addition, set the object variable to Nothing is before exiting the procedure with Err.Raise. For example:

Private Sub Test(ByVal intValue As Integer)

'error handling omitted for clarity Dim rst As ADODB.RecordSet Set rst = New ADODB.RecordSet 'More code here If intValue > 1000 Then

'Raise an Error------------------------------------------------------> Set rst = Nothing On Error Goto 0 Call Err.Raise(Number:=lngcInvalidValue, \_

Description:=strcInvalidValue) End if Set rst = Nothing

End Sub

In the example just shown, if the rst object variable is not set to Nothing before performing the Err.Raise, the Class\_Terminate of the rst object likely will change the properties of the Err object so that it no longer reflects the number given in lngcInvalidValue. This Class\_Terminate code executes before the calling routine's error handler is invoked. This weird flow of execution has caused a number of very difficult to track down bugs.

**Parameters to a Procedure**

Every parameter to a procedure should be given an explicit data type, including variants. Every parameter should be passed by value using the ByVal keyword, with a few exceptions. These are:

* VBA doesn't allow certain data types to be passed by value, such as arrays, user-defined types, and objects.
* You specifically want to allow the changed value of the parameter to be passed back to the calling routine.
* The parameter to event procedure is specified as being by reference when VBA creates it.
* The arguments to a Declare statement must match the definition in the DLL.

Even in the cases where the argument should be passed by reference, you should explicitly prefix the parameter with ByRef, even though this is the default in VBA. This makes it explicit that you meant to pass that parameter by reference.

After VBA inserts an event procedure, the parameters to the event procedure should be changed to include ByVal and ByRef keywords, and change the parameter names to use the appropriate naming conventions. For example, VBA inserts the event procedure like this (with the line wrapped in this document):

Private Sub Form\_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)

End Sub

This should be changed to read like this:

Private Sub Form\_MouseMove(ByRef intButton As Integer, \_ Byref intShift As Integer, ByRef sngX As Single, ByRef sngY As Single)

End Sub

By changing it to read like this, the naming conventions indicate the data type and the ByRef keywords indicate that VBA may see the changes to the parameters.

**Parentheses**

Always use parentheses where the reading of the line may be unclear. For example, suppose that a line is written:

If Not frmTest Is Nothing Then

It may not be clear that the Is operator has higher precedence than the Not operator in this line. Recode it to read:

If Not (frmTest Is Nothing) Then

This makes it clear what order the operators are evaluated. The general rule is that if there is any question what the operator precedence is, use parentheses to make it clear.

**Procedure Scope**

Always use the Private scope on a procedure unless you need to expose the procedure outside the current module. In a library, use the Friend scope when you need a larger scope. Use Public only when access to the procedure is required outside the library. For Example:

Private Sub Test()

**Project Properties [VB6]**

In Visual Basic, the Project Properties dialog should always be filled in. These values may not apply to VBA hosts other than Visual Basic. Most of these fields can be retrieved from the EXE, DLL, or OCX file by right clicking on it in the Windows Explorer, then selecting Properties, then clicking on the Version tab in the dialog that appears. The values can be retrieved from within the program by getting properties of the App global system object. The following fields should be always be filled in:

* Project Name: The name of the library or standard EXE name. The library name should always start with a short word or abbreviation indicating the company or organization that is developing the library. For example, the Maya Calendar engine library from Xoc Software, might be named XocEngine or XocMayaEngine. This term is used for disambiguation of libraries and shows in the Object Browser. See the Disambiguation section. This is the internal name of the library. This may have abbreviations in it.
* Project Description: This should be the same name as the Project Name, except with spaces between the words. Abbreviations and the company or organization name should be spelled out. For example, use Xoc Maya Engine. These words show up in the VBA References dialog.
* Major/Minor/Revision number: These should be filled in with appropriate values. The version number should never be set to a smaller value as installation programs depend on it to determine if they should overwrite an older version with a newer one.
* Auto Increment: In most cases this should be checked. This automatically increments the revision number by one every time the project is compiled to a file.
* Application Title: The application title should be the name of the product that you expect to show externally, on the Windows Start menu, the Windows task list, the Windows Task Bar, and should be copied to the Caption of the main form in the application when the program starts. For example: Xoc Maya Engine.
* Comments: If Visual SourceSafe is used to maintain the project, this should be filled in with $Header: $. If keyword expansion in files is used, then Visual SourceSafe will place the expansion into the comments section of the executable. This gives the source name of the project is, the SourceSafe version number of the VBP file, the date and time the project file was changed, and by whom. This helps roll back the project to a given release to test for bugs. See the section on Source Code Control to configure SourceSafe.
* Company Name: Should be filled with your company or organization name. For example: Xoc Software. This is used on splash screens and about dialogs. Therefore, if your company is XYZ Software, Inc., you probably want to use XYZ Software.
* File Description: This is the description of how this file fits into the entire package. For example: Xoc Maya Calendar calculation engine or Xoc Maya Calendar UI.
* LegalCopyright: Enter the copyright notice for the program. For example: Copyright © 1999 by Xoc Software. You may find it useful to type Alt+0169 on the keypad (not the main keyboard) to get the © symbol in the dialog.
* LegalTrademarks: Enter any trademarks or registered trademarks for the company or product. For example: Xoc™ is a trademark of Xoc Software. You may find it useful to type Alt+0153 on the keypad (not the main keyboard) to get the ™ symbol and Alt+0174 to get the ® symbol. Note that these symbols may or may not show correctly in the application depending on the font you choose to display them.
* Product Name: This is the name of the product, without the company name. Therefore, if the name of the product elsewhere is Xoc Maya Calendar, the name here should be just Maya Calendar. This value may be used in splash screens and about dialogs.

**Raising Errors**

When you raise a runtime error from a component, to be trapped in the calling code, the error number that you raise should have a unique error number. For this purpose, VBA defines a constant vbObjectError that guarantees that errors that you generate will not conflict with ones that VBA defines. However, all libraries use errors in the range larger than vbObjectError, so you should strive to be different from the other libraries with your numbers. There is no way to guarantee this; the chances can be reduced by starting your errors at a random number in the range 512 to 32767 larger than vbObjectError. No library that an organization produces should ever have conflicting error numbers with another library from the same organization. For example: XYZ Software might start numbering its errors at vbObjectError + 4096. The first library produced from XYZ software might generate errors in the range from vbObjectError + 4096 to vbObjectError + 4146, the second library from vbObjectError + 4147 to vbObjectError + 4196, etc.

**Select/End Select Blocks**

The Select/End Select block is indented differently from other blocks (see Indenting). The Case blocks within the Select/End Select are lined up with the Select/End Select keywords. Code within a Case block is indented one tab stop from the Case statement. For Example:

Select Case Err.Number

Case tliErrCantLoadLibrary

Err.Raise Number:=Err.Number, Description:=Err.Description, \_

Source:=Err.Source

Case 35602 'This key is already associated with an element of this

'collection

Set nodChild = tvw.Nodes.Item(cci.Guid)

nodChild.Image = "InstClass"

Resume NextItem

Case Else

Call UnexpectedError(Err.Number, Err.Description, Err.Source, \_

Err.HelpFile, Err.HelpContext)

End Select

In non-RVBA coding standards, it is more common to indent Case blocks one tab stop from the surrounding Select/End Select. However, this causes the actual executing code to be indented two tab stops from the surrounding Select/End Select. The readability of the code is just as good, if not better with this scheme, although it takes some getting use to the first Case block being indented to the same level as the Select line.

See also the note about Case Else blocks in the section on Assertions.

{Alternative: Indent the Case blocks one tab stop from the surrounding Select/End Select. Then indent the code in the case blocks one more tab stop.}

**Source Code Control [VB6]**

Code should be maintained using some sort of Source Code Control. Microsoft Visual SourceSafe is the most common product used for this. When using Visual SourceSafe, the Administrator should configure it to expand keywords in files in the SourceSafe Administrator Options dialog. The following files should be expanded: \*.bas,\*.cls,\*.ctl,\*.frm,\*.pag,\*.vbp. Entries such as $Header: $ can then be placed into the code and are expanded automatically. See the SourceSafe documentation on keyword expansion. Also, see the use of the Comments entry in the section on Project Properties in this document.

**Type Conversion**

VBA is considered a weakly typed language. You can construct expressions such as this one:

strValue = "Your order came to " & intQuantity \* curPrice

VBA will automatically convert the result of the expression into a string to make the expression work. However, it is better programming practice to make explicit what VBA is doing using the type conversion functions: CBool, CInt, CLng, CStr, etc., plus the Format$ function. For example:

strValue = "Your order came to " \_ & Format$(CCur(intQuantity) \* curPrice, "$#,###.00")

The RVBA naming conventions will help point out possible bugs. If you see a line that looks like this, you may have a potential bug:

intValue = lngInput

If the value in the variable lngInput is 90,000, this line will cause an Overflow runtime error. The fact that the types of the variables are different is a clear warning sign. If, however, you knew the value in lngInput could only be in the range 1 to 1000, it might be acceptable to do the assignment like this:

Debug.Assert lngValue >= 1 And Debug.Assert lngValue <= 1000 intValue = CLng(lngInput)

See also the section on Assertions.

**Variable Declaration**

Every variable should be explicitly declared. Using the Option Explicit keyword at the top of the module will have VBA enforce that. The VBA editor's Tools Options dialog has a setting that will make this be automatically inserted in all new modules.

Every variable should be given an explicit data type. This includes variants, which are the default. For example, a variant should be declared as:

Dim varValue As Variant

Rather than letting it be implicitly defined or declaring it as:

Dim varValue

Every variable should be declared on a line by itself. This precludes running into this bug:

Dim intValue, intTest As Integer

That declaration makes it obvious that the two variables were meant to be declared as integers, but the first variable is defined as a variant. If instead the declarations were made on a line by themselves, the problem goes away. For example:

Dim intValue As Integer Dim intTest As Integer

In addition, by declaring each variable on a line by itself, you can use the Ctrl+Y keyboard shortcut to cut the declaration to the clipboard regardless of where the caret is on the line, then paste it somewhere else. If there are multiple declarations on a line then editing is not as easy.

**Variable Initialization**

Some languages allow declaring a variable and giving it an initial value at the same time. Visual Basic doesn't allow that. A variable has a default value at the time that it is declared based on its data type. However, a syntax variation can be used inside of a procedure to give the feel of initializing it and assigning the default value.

Visual Basic allows you to place multiple logical lines of code on the same physical line if you separate them with colons. It also allows you to mix the declaration of variables with executable lines of code. Therefore, inside a procedure you can initialize a variable and give it a default value in one physical line like this:

Public Sub Test()

'Error handling omitted for clarity

Dim intValue As Integer: intValue = 7

Dim strTest As String: strTest = "Default Value"

'other code End Sub

**Variables Scope and Lifetime**

Variables should always be declared with the smallest level of scope and the shortest lifetime possible. Thus, you should declare variables with Dim inside of procedures by preference. If you need a longer lifetime, then use Static. If you need a wider scope, use Private. Only use Public as a last resort. Public variables declared in standard modules are global and can be changed by any piece of code throughout the entire project. This makes debugging changes to their value very difficult. Global variables of this sort should only be used in the context where they are set during initialization of the program, then remain static for the rest of the time the program executing.

Global variables should never be changed in one part of the program to be retrieved in another part of the program. In such cases, you should use parameters of procedures or properties of forms or objects to pass the information. If there are more than 20 global variables in the program, it is a warning sign that the program design is wrong.

Having Public constants are allowed and encouraged. See the section on constants.

**Version Compatibility [VB6]**

After building an ActiveX control or ActiveX DLL for the first time in Visual Basic, the project compatibility should be set to Binary Compatibility in the Project Properties dialog. Each time the component is "released," a copy of the component should be made to the same directory, but with the filename extension set to CMP. The binary compatible file entry should point to this file. That means that you can modify the interface to the file within a release as long as you are still backwards compatible with the last release. The CMP file should be checked into the Source Code Control project, whereas the current copy of the component itself probably should not be checked in (see the section on Source Code Control).

*Greg Reddick is the President of Xoc Software, a software development company developing programs in Visual Basic, Microsoft Access, C/C++, and for the web. He leads training seminars in Visual Basic for Application Developers Training Company (AppDev). In a previous life, he worked for four years on the Access development team at Microsoft. Greg can be reached at mailto:grr@xoc.net or from the Xoc Software web site, http://www.xoc.net.*

APPENDIX C: Microsoft Application User Interface Guidelines

# Intro

By using established guidelines we can ensure that we build our software products with appropriate and consistent appearances.

We will follow the standard Windows user interface guidelines as described in “*Windows User Interface Guide*”, Microsoft Press, 1999, and as described in the three documents that can be found at <http://sdg.jsi.com/standards/style-guides/user-interface/windows>. This document further specifies the guidelines set forth in those documents.

# General Properties

Some properties are common to pages, controls, and elements. The rules set forth in this section should suffice in most cases.

## Fonts

### Face

When not defaulting to the system font, use Arial.

### Size

#### Text

Use 8 point, normal.

#### Customer/Client

Use 12 point, normal.

#### Page Title

Use 14 point, bold.

### Type

Use boldface type for titles. Use normal type elsewhere. Never use underlined or italic typefaces.

### Color

All text should be black.

### Background Color

The font background should always be transparent, thereby defaulting to the background color of the font’s parent form or control.

### Case

Use title case for all labels, names, and titles. Do not use either all UPPERCASE or all lowercase.

## Color

### Palette

Deliver has selected a palette of five colors: red, orange, gold, green, and blue.

|  |  |  |
| --- | --- | --- |
| **Red** | HSL: 234, 195, 85  PMS: 194  RGB (decimal): 164, 17, 40  RGB (hexadecimal): #A41128  Windows: 2625956 |  |
| **Orange** | HSL: 20, 40, 120  PMS: 152  RGB (decimal): 255,125,0  RGB (hexadecimal): #FF7D00  Windows: 32255 | deliver-orange |
| **Gold** | HSL: 25, 240, 129  PMS: 137  RGB (decimal): 255, 168, 20  RGB (hexadecimal): #FFA814  Windows: 1353983 |  |
| **Green** | HSL: 74, 143, 74  PMS: 363  RGB (decimal): 47, 126, 32  RGB (hexadecimal): #2F7E20  Windows: 2129455 |  |
| **Blue** | HSL: 138, 229, 64  PMS: 541  RGB (decimal): 3, 76, 133  RGB (hexadecimal): #034C85  Windows: 8735747 |  |

### Text

All text should be black. For further info on text properties, see the section on “Fonts”.

## Dimensions

### Screen

#### Height

All screens will fit onto a medium resolution terminal, i.e., 600 pixels high. If at all possible, no data will be displayed below the viewable area of the screen. In other words, avoid vertical scrolling.

#### Width

All screens will fit onto a medium resolution terminal, i.e., 800 pixels wide. No data should ever be displayed to the right of the viewable area of the screen. In other words, horizontal scrolling is forbidden.

### Regions

#### Header & Primary Navigation

The header and primary navigation area will span the entire width of the page, occupying no more than the top 75 pixels of the screens height.

#### Secondary Navigation

The secondary navigation area will run down the left side of the screen, below the header. Of this space, the secondary navigation bar will occupy no more than 15% of the screen’s width.

#### Content

The content area will fill the area from the bottom of the header to the top of the footer, and from the right edge of the secondary navigation area to the right edge of the screen.

Ordinarily, this region will be subdivided into other regions dictated by the application and the application’s purpose.

#### Footer

The footer will span the breadth of the page, occupying no more than the bottom 40 pixels of the screen’s height.

### Controls

Controls should be of uniform dimensions.

#### Buttons

(Borrowed from GUI LNF Standards – DENR (Interact))

If the length of text for a series of command buttons in a dialog box is similar, make all the buttons in the dialog box the size of the largest button

If the text length for a series of command buttons in a dialog box varies, use two button sizes—one for shorter text and another for longer text. Do not use more than two different button sizes in a dialog box.

#### Text Boxes

(Borrowed from GUI LNF Standards – DENR (Interact))

Size text boxes to indicate the approximate length of the field. If you have text boxes of similar length, make them the same length unless you need to show the exact size of the field. If the length of the field can vary, use text boxes of the same length to minimize the number of unique margins on the screen.

Left align text boxes on the screen to minimize the number of different margins. If a particular text box has a long label, use a different margin for that text box. Limit the number of unique margins to two.

#### List Boxes

Show at least three, but no more than eight items in a list box at a time. If you have more items use a scroll bar to view the rest of the items.

## Flow

Tabbing from control to control will go from top to bottom, then left to right.

## Usability

### Keyboard vs. Mouse

All applications will be fully usable without a mouse, i.e., all functionality will be readily accessible from the keyboard.

### Section 508

All applications will be Section 508 compliant.

### Performance

The system will always provide some visual feedback to the user as soon as possible. For desktop applications, this feedback must occur within one second. For web applications, the feedback must occur within five seconds.

## Internationalization

### Multi-Language support

All apps will provide Unicode (wide character) multi-language support. This may mean that dialogs & screens have to be somewhat auto-sizing.

### Date & Time

All applications will use the system-defined date and time formats.

# Applications

Aside from the general, overarching principles used to govern screen layout and design, the various types of applications have their own special constraints.

## Desktop

Desktop applications are applications that can be run in standalone mode without the use of a browser. These are traditional Windows applications.

### Page

#### Splash Screen

Each Deliver app will feature a splash screen of uniform layout (actual layout is to be determined by the communications group). This splash screen will be 300 pixels by 400 pixels. The splash screen will persist for 5 seconds. The splash screen will be available from Help 🡪 About <appname>.

#### Framework

Each application will feature a title bar, a menu bar, a tool bar, and a status bar.

(Microsoft Access-based applications will use the framework provided by Access.)

##### Title Bar

The title bar is the line at the very top of the application or dialog, just above the menu bar (if the menu bar is present). The title bar contains the minimize, maximize, and restore buttons.

###### Font

The font should be Arial 12, bold.

###### Color

The title bar should be either the system color, or the appropriate color from the Deliver palette. Which one?

###### Content

The title bar should read “Page Name – Application Name”.

###### Case

Titles should be in Proper Case, except in the case of logos.

##### Menu Bar

Standard menu bar names and positions will be used.

##### Tool Bars

Tool bars will use standard Windows icons.

###### Icon Size

There are two sizes, small and large. Small is 16x16 pixels, large is 20x20 pixels. Prefer the small icons.

###### Spacing

For each size, there should be 3 pixels between a toolbar button and its text label.

##### Status Bar

*The status bar will be the Windows standard 1 line tall. It should be used to provide the user with information about the status of the application.*

##### Sizing

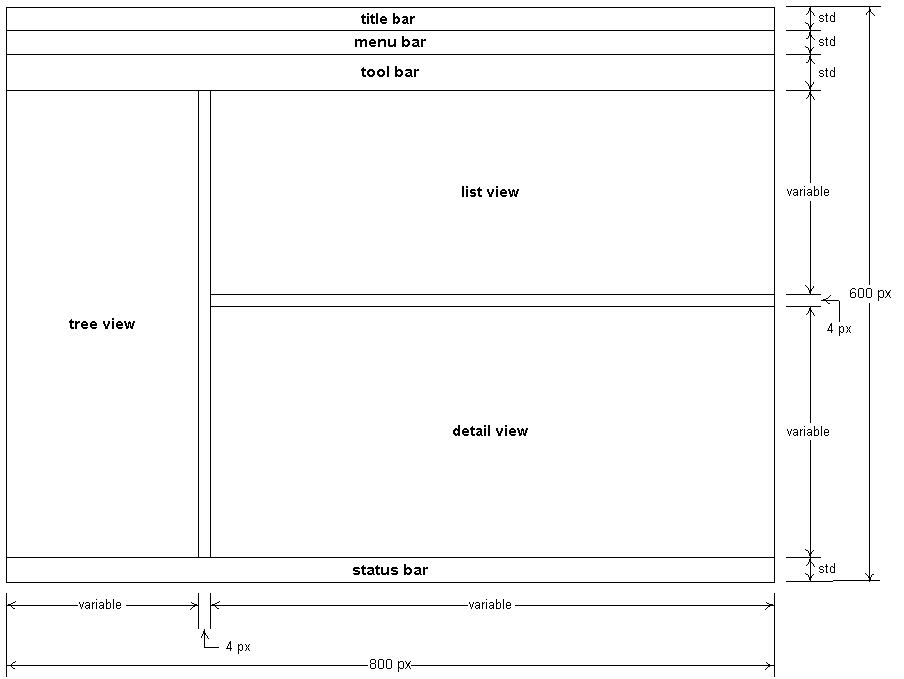
All screens will support minimizing, maximizing, and custom sizing.

#### Layout

##### General

Deliver uses “explorer” style applications. This provides for easiest navigation of the application’s screens.

The following drawing defines the recognized screen regions and their sizes.



##### Size

Target screen resolution: all apps should be presentable at 600 x 800.

Each pane in the window can be resized.

Windows will control the heights of standard Windows elements such as title bar, menu bar, tool bar, and status bar.

##### Font

Windows will control the fonts in the title bar, menu bar, tool bar, and status bar.

##### Color

Windows will control the colors in the title bar, menu bar, tool bar, and status bar.

##### Graphics

###### Splash Screen

Has the app logo, product version info, JSI logo, funding source (customer) logo, and application image.

###### About

Use the same info as the splash screen, add links to websites, email address.

#### Regions

The screen is divided into number of regions, each of which has a purpose.

##### Tree View

The Tree View is used to navigate the application.

###### Icons

We will use the following standard icons for the following purposes:

Form

Forms will be represented by a little picture of a form. (need sample)

Report

Reports will be represented by a piece of paper with writing on it. (need sample)

Question

Question marks are commonly used for this. (need sample)

Configuration

A picture of tools will represent configuration screens. (need sample)

Group/Section

This could be either plus sign for closed groups, and a minus sign for open groups, or opened and closed folders. Either set of icons is commonly used in Windows apps. (need sample)

##### Forms

Forms are essentially dialogs that do not pop up, but remain embedded in the application framework. Whereas dialogs will contain a title bar and minimize/maximize/close buttons, a forms do not.

###### Layout

For controls that do not contain their own labels, the label should be placed to the left or above the related control. This makes it easier for users to associate the label with the corresponding control.

The logo should go in the upper right corner.

All forms will support resizing, minimizing, and maximizing.

###### Logos

All Deliver sw has a logo area in top right area of each screen. This logo area will be used to brand the product as a Deliver app. This area will likely contain only the application’s logo. The JSI and customer logos would appear on the splash screen and in the “About” dialog.

###### Style (Special Effect)

Dialogs and forms should be flat.

###### Elements

Screen elements are the things that go in the regions. These are things like tool bars, labels, and titles.

Title Bar

This is the same as standard definition for a title bar, except that the dialog title bar should not include the “ – Application Name”.

Title

If a title is present it should be bold, 12pt, default font, proper case. No italics; no underline.

Menu Bar

Use standard menu bar.

Tool Bar

Use standard tool bar.

Labels

Use the default font.

### Pop Up Dialogs

#### Modal

##### When to use

Modal dialogs are used whenever the application absolutely cannot continue without user input. In most cases, though, this can be handled with an “Apply” button.

##### Buttons

As a minimum, modal dialogs will have “OK”, “Cancel”, and “Help” buttons.

#### Modeless

##### When to use

Modeless dialogs are to be used unless the criteria for using a modal dialog are met.

##### Buttons

As a minimum, modeless dialogs will have “OK”, “Cancel”, and “Help” buttons. “Apply” is almost meaningless – the user must hit either “OK” or “Cancel” to close the dialog, and the functionality of “OK” is a superset of that of “Apply”.

#### Tabbed

##### When to use

Use tabbed dialogs when the dialog uses more controls than can be fit on a single screen.

##### Layout

Each tab should contain controls that relate to a particular dialog subtopic. If a control does not pertain to the topic of the overall dialog, then it should not be included on any of the tabs.

##### Buttons

Use consistent tab width, allowing the longest tab label dictate the tab width.

Each page of the tabbed dialog will feature, as a minimum, “OK”, “Cancel”, and “Apply” buttons. The buttons will follow the properties set forth in the General Properties section of this document.

#### Elements

##### Title Bar

Same as standard definition for a title bar, except that the dialog title bar should not include the “ – Application Name”.

##### Title

If a title is present it should be bold, 12pt, default font, proper case. No italics; no underline.

##### Menu Bar

Do not use menu bars in dialogs except in exceptional circumstances.

##### Tool Bar

Do not use tool bars on dialogs except in exceptional circumstances.

##### Labels

Use the default font.

#### Style (Special Effect)

Dialogs should be flat.

### Reports

#### Size

Must be compatible with both A4 (8.27” x 11.69”) and US 8 ½ x 11 paper. Therefore, reports should fit onto a space not larger than 8.25 x 11 (unless the software is going to be smart enough to re-format the reports based on paper size).

#### Format

##### Paper Size

Reports should be formatted for A4 paper.

##### Margins

Use ½” on the sides, and ¾” on the bottom.

##### Layout

Use the Supply Chain Manager reports as a guide.

### Controls

Controls are the buttons and cells used on the tool bars, forms, reports, and dialogs.

#### General

##### Dialog Base Units

Dialog Base Units are device-independent units of measure, and keyed to the size of the current system font. A horizontal dialog base unit is defined as one-fourth of the width of the current system font. A vertical base unit is equal to one-eighth of an average character height for the current system font.

##### Size

Sizing of controls, etc. is based on dialog units (DLUs). DLUs are not the same as pixels – they are device-independent measures to use for layout purposes. Pages 448 to 457 of the Microsoft Windows User Experience manual (Microsoft Press) provide good information on the layout, size, spacing and positioning of DLUs.

###### Examples:

Command buttons (14 DLUs); the width of these should never be more than 3.5 times the height

Check boxes (10 DLUs)

Drop-down list (10 DLUs)

Option buttons (10 DLUs)

Text boxes (14 DLUs)

Text labels (8 DLUs) – per line

Other screen text (8 DLUs) – per line

##### Spacing

Align related items.

###### Controls

4 DLUs between buttons or controls, if related; 7 DLUs between unrelated controls.

The absolute smallest space between controls is 2 DLUs.

###### Dialogs

There should be a 7 DLUs between the edge of the dialog box and the text or frame.

###### Paragraphs

There should be 7 DLUs between paragraphs of text.

###### Text Labels

There should be 3 DLUs between text labels and their controls.

###### Group Boxes

The first control in a group box should be 11 DLUs down from the top of said group box.

Controls in a group box should be aligned vertically to the group box title.

The last control in a group box should be 7 DLUs above the bottom of the group box.

###### Buttons

If a text label is beside a button, it should be 3 DLUs down from the top of the button.

A check box, list box, or option button beside a button should be 2 DLUs down from the top of the button.

##### Grouping

The following rules apply to grouping:

* Group related components
* Group box controls
* Use separator lines on menus

Much more comprehensive discussions of grouping are available in the resources cited in the introduction to this document.

##### Color

Color can be used, but is not recommended. Allow the user to change color schemes.

##### Other considerations

Main command buttons in a secondary window should be stacked in the upper right corner or in a row along the bottom. If there is a default button, it should always be the first one in the set. OK and Cancel buttons should be placed next to each other.

##### Alignment

In group boxes, controls should be left-aligned with the text label of the group.

Command buttons in the group should be right-aligned.

In toolbar arrangements, buttons and other controls are typically left-aligned or top-aligned

#### Text box

##### Layout

Use a consistent width between boxes. Flow should be from top to bottom, left to right.

##### Unlocked

###### Font

Use the standard default.

###### Color

Text

Use the standard text color.

Background

White is the standard Windows color for the background of unlocked text boxes. We will let the system control this color.

###### Size

Text boxes should be of uniform length, unless there is a requirement to show the user how large the data field is.

##### Locked

###### Font

Use Deliver default.

###### Color

Text

Use the standard text color.

Background

The Windows standard color for locked text boxes is light gray, but we will let the system control this color.

###### Size

Text boxes should be of uniform length, unless there is a requirement to show the user how large the data field is.

#### List box

##### Font

Use default.

##### Color

Use default.

##### Size

Display from 3 to 8 rows. Allow vertical scrolling if there are more than 8 items on the list.

Avoid horizontal scrolling.

#### Combo box

##### Font

Use default.

##### Color

Use default.

##### Size

Display from 3 to 8 rows. Allow vertical scrolling if there are more than 8 items on the list.

Avoid horizontal scrolling.

#### Radio buttons

##### When to use

Use option buttons when users should pick one mutually exclusive choice from a list of options, for example, choosing a pay period in a personnel application.

##### Colors

Let the system decide.

##### Size

Use default.

##### Arrangement

Lay these out vertically, using an outline to group them.

#### Check boxes

##### When to use

Use check boxes when users can choose one or more options, but these choices are not mutually exclusive.

Use check boxes for toggling a single value on or off. It is okay to have just one check box.

##### Colors

Let the system decide.

##### Size

Use default.

##### Arrangement

Lay these out vertically, using an outline to group them.

#### Command Buttons

##### Size

See the description of Buttons in General/Dimensions/Buttons.

##### Behavior

###### Ok

Saves settings and closes screen.

###### Cancel

Discards changes and closes screen.

###### Apply

Saves changes and keeps user on current screen.

###### Clear

Discards changes and keeps user on current screen.

###### Next

Takes user to the next logical screen. Does not implicitly save changes.

###### Previous

Takes user to the next logical screen. Does not implicitly save changes.

###### Help

Displays help text.

## Web

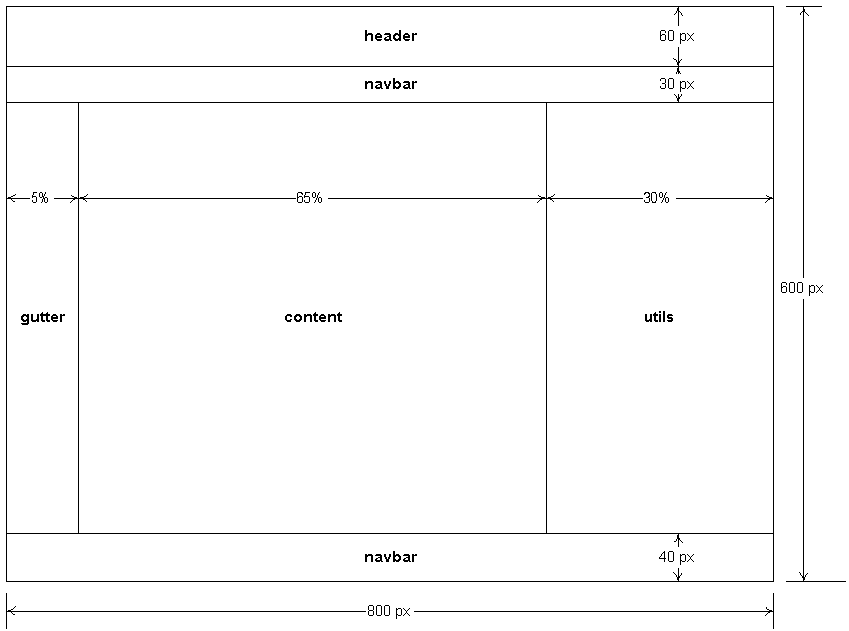
Here we describe the basic styles for Deliver’s websites.

### Page

This section discusses the properties of a page

#### Layout

Here is a diagram that clearly describes the standard layout for a page on a Deliver website.



##### This page illustrates a page laid out in conformance with the above spec (although this example does not comply with the 600px x 800px requirement):



#### Size

Our target resolution is 600 x 800. All data should appear on the screen. In rare situations, some vertical scrolling may be tolerated, but we will make all attempts to avoid this.

#### Regions

##### Header

###### Dimensions

Width

100%

Height

60 pixels.

###### Elements

Deliver logo

Filename and location

tbd

Size

40 pixels x 40 pixels

Example:

tbd

USAID logo

Filename and location

tbd

Size

40 pixels x 40 pixels

Example:

tbd

JSI logo

Filename and location

tbd

Size

40 pixels x 40 pixels

Example:

tbd

##### Navbar

###### Dimensions

Width

100%

Height

30 pixels.

###### Elements

Do we have standard elements that go here? Do we use the tabbed look that Joel tends to favor, or do we use “::”, “|” or other delimiters?

Home

The “home” link takes the user to the site’s main entry point.

My Account

The “My Account” link takes registered users to a page where they can manage their personal data and preferences.

About us

The “About Us” link takes the user to a page where they can learn a little bit about Deliver, USAID, and JSI, and how to contact each.

Other

What are the rules determining whether something gets a link on the navbar? A pod? A type of info? A type of service? A region?

###### Appearance

Fonts

Face

Use default.

Size

size = 1

Color

Black

Effects

Bold

Colors

Choose colors from the approved Deliver palette.

Layout

Horizontal, tabbed.

###### Example

*Paste picture here*

##### Footer

###### Dimensions

The footer will span the full page width, and be 40 pixels tall.

###### Elements

Do we have standard elements that go here?

Home

My Account

About us

Other

What are the rules determining whether something gets a link on the navbar? A pod? A type of info? A type of service? A region?

###### Appearance

Fonts

face = sans-serif

size = -1

effects = bold

Colors

The footer will feature black text on a white background. There will be

###### Example

*Paste picture here*

##### Content

###### Dimensions

The content region will occupy 65% of the screen’s width.

The content region will be 470 pixels tall.

###### Appearance

Space the sub regions 5 pixels apart. If a table is used for layout, this means using “cellpadding=”5””.

###### Example

*Paste picture here*

### Fonts

#### Faces

Use Arial for everything.

#### Styles

##### Bold

Titles.

##### Underline

Links.

##### Italics

Never.

#### Sizes

##### -2

-2 is used for the footer.

##### -1

-1 is to be used for normal text, labels, and the contents of text controls.

##### 1

1 is used for sub region titles.

##### 3

3 is to be used for Page Titles only.

### Content

#### Writing for the Web: Quick Tips

1. **Be succinct**: Keep paragraphs and sentences short. Think in “chunks” of no more than 75 words each. You want the reader to skim over short chunks of information that either convey an idea or direct the reader to another location for more information.
2. **Use bullet lists:** Break long paragraphs into bullets. This is an easy way to deal with long paragraphs that contain lists or a series of short phrases. Readers are much more likely to take in a bulleted list and retain the ideas presented than to read and remember a long paragraph.
3. **Use strong verbs**: For example, use “decide” instead of “make a decision;” and “use” instead of “make use of.” This will help command the reader’s attention and will reduce the number of words you need.
4. **Use active voice**: Although there are times when the passive voice is useful, on Web sites text should be in the active voice. Passive voice sounds too academic and stuffy.
5. **Limit idioms and jargon:** Because our audience is global you should limit use of idiomsfrom any culture. Also avoid logistics jargon. Instead, use simple, everyday language to describe logistics ideas.
6. **Use links:** Ifyour text requires more than one screen, include anchor links at the top that jump to information further down on the page or break up the text and have it link to other pages.
7. **Look at examples**: To see good examples of websites that make effective use of short chunks of information, links, sidebars, graphics, and PDFs, go to—

[www.familyplanet.org](http://www.planet.org) or [www.globalhealth.org](http://www.globalhealth.org).

(Some of these guidelines are taken from *Writing for the Web* by Crawford Kilian, Self-Counsel Press, 1999.)

#### Style Sheet for DELIVER Website

All regular DELIVER style rules apply. Here are some additional guidelines to remember:

##### Spell out all acronyms upon first use on each web page

##### Contact line example: “For more information, contact Heather Davis at heather\_davis@jsi.com.”

##### Correct spellings:

DELIVER

email

Internet

NEWVERN

online

PipeLine

stock out

supply chain

Web

web page

website

World Wide Web

### Performance

Because many of our users do not have access to high-speed internet connections, we must tune our sites such that they perform acceptably across a 28.8k bps dial-up modem connection. This means that site pages should average 34kb (including both text and graphics), and that no page should be larger than 100kb. Data should be cached and delivered to the user in screen-sized increments.

The system will always respond to user action within 5 seconds of user input, even is this means simply posting a “wait” message.

APPENDIX D: Building the PipeLine Installation CD

This blank page is intentional. All chapters start on a right-hand side.

For more information, please visit deliver.jsi.com.

USAID | DELIVER PROJECT

John Snow, Inc.

1616 Fort Myer Drive, 11th Floor

Arlington, VA 22209 USA

Phone: 703-528-7474

Fax: 703-528-7480

Email: ask[deliver@jsi.com](mailto:deliver@jsi.com)

Internet: deliver.jsi.com