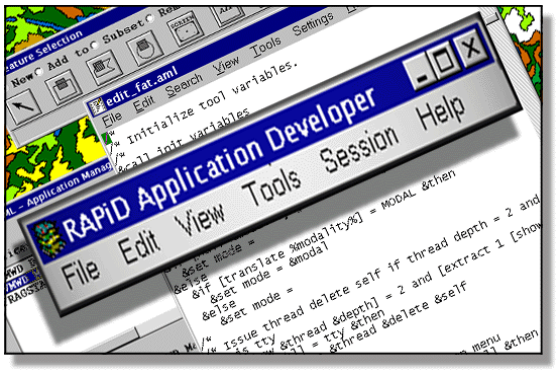


The ARC/INFO™ Application Development Primer

An Overview of Application Development Methodologies for ARC/INFO™

This workbook provides an overview of application development methodologies available for the ARC/INFO™ GIS.[1] The primer reviews different approaches and issues related to developing custom applications with the ARC/INFO platform. This book is intended for existing ARC/INFO users who have some experience with application development and programming. It will be particularly useful for application developers familiar with the Arc Macro Language (AML) and ARC/INFO command operation. It is not intended for the casual or novice ARC/INFO user.

GIS Solutions Inc.
INNOVATIVE

Suite 300
2000 S. College Ave.
Fort Collins, CO 80525

Telephone: (970) 490-5900
FAX: (970) 490-2300
E-mail: igis@innovativegis.com

Visit us on the Web at **http://www.innovativegis.com**

[1] ARC/INFO, Arc Macro Language, AML are registered trademarks of Environment Systems Research Institute, Inc, (ESRI), Redlands, CA.

Table of Contents

List of Tables and Figures

Preface

This workbook was developed to support AML based application development workshops that I presented in the late 1990's. With my colleagues at Innovative GIS Solutions in Fort Collins, CO, we became leaders in the ArcInfo application development arena and were well ahead of our time, especially with the Rapid AML development environment for AML. But alas, technologies change, and AML went out of favor to be replaced with more complex and powerful development languages and environments.

At the time this workbook was written only the ESRI ArcTools framework compared, although I believe that few in the industry were able to leverage the extensive framework that Matt and his group at ESRI were able to develop. The tools we developed, as explained in this textbook, significantly leveraged ESRI's early efforts in ArcTools and acknowledgement must be given to this group.

In our time at Innovative GIS in the 1990's we continued to use the approaches described in this workbook to develop some pretty cool applications for our clients. Ultimately these approaches and techniques have been taken by Dave Bouwman (dbouwman@dtsagile.com) and enhanced and developed into the outstanding work we see from Dave and his team today. I like to think that our early work, which Dave was a key part of, influenced some of the achievements we see today.

Recently at the 2010 ESRI BPC after renewed discussions with Peter Eredics (peredics@esri.com), a colleague of whom I have known for 20 years, I remembered this workbook and decided to dig it up and create a PDF for folks to use. Hopefully, for some, it provides some utility.

Many of the technical references in the book are dated, however the core content remains valid. In fact, I must admit to still using the AML approaches to prototype analytical models with my colleague Dr. Joe Berry on a fairly regular basis. Old habits die hard.

I have provided the document "as is" with original content, examples and references. The author can be reached at dbuckley14@live.com. Enjoy!

David Buckley, March 2010

This book is a result of several years experience in applying ARC/INFO and related ESRI products to typical land management problems. Many of the terms and concepts introduced in this book have been compiled from a variety of different sources. These include some of the prevalent text books in the field, and ESRI documentation, but most have been firmly implanted from practical experience with ARC/INFO since version 3.0. This workbook is intended to provide existing users a review of practical issues and methods for building robust applications with ARC/INFO.

This workbook was prepared for the *Application Development Methodologies for ARC/INFO* workshop originally presented at the GIS'98 – Resource Technology conference in Toronto, Ontario, Canada in April 1998. Accordingly, many of the concepts and approaches discussed in this book are intended to be supplemented with real time examples by IGIS staff. This one day workshop presents examples applications and programs that illustrate the methodologies endorsed in this book.

GIS is in essence, an applied science. The more *hands-on* experience you have the more entrenched the methodologies and techniques become for effectively applying the technology. ARC/INFO is a classic example in this regard. Most GIS software offerings exist as command sets that require application development before they can really be applied in an operational

sense. With ARC/INFO, AML is the primary glue that connects the extensive commands sets into applications.

The sample methodologies and code presented in the ARC/INFO Application Development Primer have been compiled from our experiences representing real operational requirements of selected IGIS clients, past and present. These sources include provincial and federal natural resource agencies in Canada and the United States, and private companies across North America. In most cases the code templates and examples used in the workbook have been simplified in an effort to focus on procedures and techniques for applying ARC/INFO. They are not intended to represent operational programs or tools.

In addition, many of the AML development methodologies presented in this workbook have been developed from the practical application of approaches initially developed and promoted by Environmental Systems Research Institute, Inc, (ESRI). In particular, the ArcTools methodologies that promote the multiple entry point, *toolset* approach are the basis for many of the methods and techniques presented. The toolset approach presented in this workbook is simplified, based on our experience of practical application for clients, for an implementation and usage that we believe is more in line with the average AML developer's skills for most ARC/INFO customers. This modified approach is embodied in the RAPiD AML development environment used for the presentation of the workshop that accompanies this workbook.

Readers are urged to review the description of the ArcTools toolsets and API usage that can be found in the ARC/INFO help and manual documentation set. As well, the approaches presented in this workbook are not intended to replace formalized AML training. They are intended to supplement the ARC/INFO user and applications developer who has taken proper ARC/INFO and AML training courses. ESRI provides an excellent array of courses related to these topics. Some references are provided at the back of the workbook.

The ever increasing rate of technological change prompts us to update the Primer to reflect the new and exciting capabilities that GIS technology provides. Recent changes to the ESRI product line and ARC/INFO's movement into the Windows NT platform has extended the alternatives for building applications. This has resulted in more alternatives, and decisions, for the average GIS manager on how to build, and deploy applications. We hope this workbook helps address these issues and provide some clarity to available options. Comments and suggestions are always welcome in this regard.

GIS has changed so drastically since ARC/INFO was originally released in the early 1980's; but in so many ways remains the same. Changing technology has provided many more capabilities and possibilities, often beyond the comprehension of the average user. Nonetheless, ultimately the tool is only as good as the craftsman using it. Consistently it is the imagination and creativity of the *domain specialist* using GIS, that makes GIS work, not the new technological capabilities or increased performance. I believe that while the GIS vendor community, hardware and software vendors alike, provide us with newer, better and faster technological tools, it is in the end, the domain specialists applying the tool that define *state-of-the-art*. The heartbeat of GIS still lies in the field and district offices, the logging divisions, the engineering offices, and with the small GIS entrepreneurs in offices everywhere.

Many thanks to our colleagues, past and present, who have supported and worked with us over the years. This book reflects our collective experiences and knowledge. A special thanks to David Bouwman at IGIS for compiling the ODE overview, and preparing many of the figures for this workbook. It is always the technical complement of skills that makes GIS work.

The GIS Primer is provided by *Innovative GIS Solutions, Inc.* as the workbook and reference document for our ARC/INFO application workshop. If you'd like more copies please give us a call.

David J. Buckley

Fort Collins, Colorado. **March 1998**
e-mail: http://buck @ innovativegis.com

1.0 Overview of the ARC/INFO Product Line

The current version of ARC/INFO and related ESRI products provides several options for developing custom applications. ARC/INFO is the core product of the ESRI product line. It should be considered the *GIS engine* for applications development. However, because not all user's needs dictate analytical capabilities other options do exist for building and deploying applications to the user. This workbook focuses on methods and techniques for building applications using the ARC/INFO engine.

This section of the workbook provides a brief overview of the ARC/INFO product line and developments related to application development. It is intended to provide basic information for other chapters that focus on more specific methods and techniques for building and deploying applications with ARC/INFO.

A number of terms are used throughout this workbook that refer to specific application development activities or conceptual organization of components. Table 1 provides a summary of these terms in order to clarify them for use with this workbook. These terms are described in the context of the ARC/INFO and related product line, and may have different meanings in other software environments or disciplines.

Table 1 : Summary of Terms

Term	Description
Application	A collection of functions, macros, menus, programs and tables which operate as a discrete software system presented in a customized form for addressing user needs and functional requirements. Applications are typically created through Application Programming Interfaces (API), eg. Arc Macro Language (AML), that utilize standard GIS software toolkits as development platforms such as ARC/INFO. With AML applications are usually comprised of several toolsets bundled together through a GUI.
Development Environment	A software development product that facilitates the programming of specific programs, macros, menus etc. A development environment usually involves a set of operational parameters defining various application management, display, editing, and data manipulation conditions that remain active during a development session until explicitly changed by the user. A development environment provides a framework for building applications utilizing a structured, standard, and flexible methodology. Visual Basic is a good example of a Windows NT based development environment.
Component (ARC/INFO)	A functional subsystem of ARC/INFO that operates with a separate set of commands for a specific purpose, such as LIBRARIAN for spatial data library management, or ArcPlot for display, query and

Term	Description
	plotting. Components are fundamental operational parts of ARC/INFO.
Component (application)	Applications are usually broken down into functional components represented by separate toolsets. An application component will address a specific functional purpose such as data/file management, querying, display, plotting, etc.
Toolset (tool)	A toolset is a discrete functional combination of macro(s), menu(s), related data tables, and help file(s) that perform a specific task. Toolsets are the fundamental building blocks of AML based applications in ARC/INFO. A tool performs an operation or a set of operations for the application developer based on a specific functional capability. It may be a single AML that performs a simple operation, such as checking if the user has write access to a directory, or more typically it includes a menu interface that allows the user to specify the parameters needed to perform an operation. In general, a tool consists of a menu interface, a driver program (AML), and a help file.
Module Extension	A module extension is a separate software product that works in combination with ARC/INFO to provide extended or enhanced capabilities. Examples include the ARC GRID module for raster modeling, ARC TIN for surface (3-D) analysis and display, ARC NETWORK for linear network analysis and path/route optimization. ARC/INFO modules work seamlessly with AML and provide more comprehensive application development capabilities. The use of a module is dependent on specific user needs.
Model	A model is commonly defined as an abstraction of reality that is reflected in a procedure run on a database to derive a measure or set of measures. Often a model is referred to as a set of clearly defined analytical procedures used to derive information. A model is structured as a set of rules and procedures to derive information that can be analyzed to aid in problem solving and planning. Most spatial modeling in ARC/INFO is done with the GRID module. Models can include a combination of logical expressions, mathematical procedures, and criteria, which are applied for the purpose of simulating a process, predicting an outcome, or characterizing a phenomenon.

Term	Description
Macro	A text file containing a sequence of ARC/INFO and/or AML commands, functions or directives. A macro file can be executed with a single command in ARC/INFO using the "&run" AML directive. Macros are sometimes referred to as *scripts*, however in UNIX systems the term *script* is typically refers to UNIX command macros. Macros can contain any combination of ARC/INFO commands (component or module), and AML commands (directives, functions). Macros in ARC/INFO have an ".aml" file extension.
Menu	A text file containing AML menu control commands that are interpreted by the AML processor to display a X-11 Window menu (UNIX) or a Windows standard menu (Windows NT). Menu files are the primary interfaces tools available within AML. Menus in ARC/INFO have a ".menu" file extension.
Developer and Operator	A basic distinction between the type of user for ARC/INFO applications. An *Operator* simply runs the application. There is no requirement to program application code. A *Developer* needs to run an application but also have capabilities to modify application programs/code in a WYSIWYG manner.
Application Desktop	A default or standard GUI template to be used for all new applications. An application desktop should include a standard main menu interface, graphics environment window, toolbars and/or display pan/zoom tools. Using a standard application desktop helps to ensure a common interface (look-and-feel) for applications on a site, and usually results in better usage and productivity. Application desktops are inherent for most Windows based application development environments, but are not for UNIX and/or AML based applications.
Application Architecture	A standard subdirectory organization for application code and files. Such a subdirectory must accommodate standard file extensions, e.g. .AML, .menu, etc., and support files. No standard architecture exists, or is promoted, for ARC/INFO AML based applications.
Application Development Methodology (ADM)	Refers to a formalized approach for managing and developing applications on a site. An ADM involves procedures for deploying applications to users, managing development and production versions of applications, programming and coding standards for macros and menus (GUI), and code management standards for toolset development and maintenance.

The following table provides summary information on ESRI products and their primary application development platforms. This table also includes products not considered part of the ARC/INFO product suite to illustrate the varying languages and tools used across the entire product line. Understanding the role and requirements of product components and modules is a critical prerequisite for application development.

Table 2 : ESRI Product Line API Characteristics

Product / Module	Programming /Macro Languages	GUI Builder	Primary Data Formats	Operating Platforms
ARC/INFO 7.1.2+ **Open Development Environment (ODE)**	• AML • AML IAC - C • ActiveX (NT) • Motif, Tcl/Tk (UNIX)	• AML • Any ActiveX Programming Tool (NT) • Motif, Tcl/TK (UNIX)	• Coverage • Library • Shape File	• Windows NT • UNIX
ARC/INFO 7.1.1-	• AML • AML IAC - C	AML	• Coverage • Library	• UNIX • Windows NT
Spatial Database Engine (SDE)	SDE API (C)	UNIX (Tcl/Tk)	SDE format	UNIX
ArcView	Avenue	Avenue	• Shape File • Coverage	Windows
MapObjects	ActiveX (Visual Basic, etc)	ActiveX (Visual Basic, etc)	• Shape File • Coverage	Windows
PC ARC/INFO	SML	NONE	Coverage	Windows
ArcCAD	AutoLISP	AutoCAD	• Coverage • AutoCAD DWG	Windows

1.1 AML – ARC/INFO's Glue

The ARC Macro Language (AML) is the primary programming language available in ARC/INFO

to program and develop applications. AML allows you to automate frequently performed actions, create your own commands, provide startup utilities to help new or inexperienced users perform operations that require specific command settings, and develop menu-driven user interfaces to meet the needs of end users.

Through AML, programs and menus can be shared between ARC/INFO operating platforms with little or no modification. Because AML is exclusive to ARC/INFO it is platform independent. In addition, because AML is part of ARC/INFO, it recognizes ARC/INFO data objects (such as coverages and libraries) and provides information about these objects as well as information about specific ARC environments. AML contains a suite of data management functions to aid in custom application development.

Simply said, AML is the glue that binds ARC/INFO and its many components/modules together. It is the foundation for building applications in ARC/INFO with all its components and module extensions.

It is the core component to using ARC/INFO effectively. Accordingly it has been designed to be totally integrated with all the ARC/INFO modules (such as GRID, TIN, COGO, etc.) and operational components, (such as ArcPlot, ArcEdit, Librarian, etc.).

Through all the recent enhancements and developments with the ARC/INFO product line AML continues to be the primary programming language for automating ARC/INFO tasks and interacting with other development tools. Since it has been such a fundamental component of ARC/INFO it is not expected that this will change.

1.2 Module languages

ARC/INFO is a command toolkit organized into operational components and module extensions. *Components* are functionally separate subsystems of ARC/INFO, but are included as part of the core product. *Module extensions* are additional modules that provide extended capabilities for ARC/INFO. They are not included as part of the core ARC/INFO product and must be purchased separately. Each of the fundamental components and optional modules operates as a series of independent commands. Each command will operate as a function that will execute a specific data, display or analysis task. Table 3 provides a summary of ARC/INFO components.

Table 3 : Summary of ARC/INFO's Primary Subsystems and Modules

ARC/INFO Component / Module	Description
Components	
Arc	Primary data input and manipulation subsystem. The Arc subsystem contains all data conversion and file manipulation functions including INFO database access, and spatial data library management.

ARC/INFO Component / Module	Description
ArcPlot	Primary display and plotting subsystem. ArcPlot includes commands to display data coverages, compose maps and plot files, and query data coverages.
ArcEdit	Primary interactive editing graphics subsystem. ArcEdit includes capabilities for data input, digitizing, and graphics editing.
Database Integrator (DBI)	Series of commands to link to external relational databases such as ORACLE, Informix, etc.
Librarian	Subsystem for creating and managing spatial data libraries based on a tile management approach.
INFO	Primary internal relational database provided with ARC/INFO. INFO is considered an internal DBMS for managing attribute data tables that link seamlessly to spatial data coverages. The INFO subsystem operates as an independent set of commands.
Module Extensions	
ArcStorm 	A spatial database manager that provides enhanced capabilities for linking to RDBMS, referential integrity, and feature management.
ArcPress 	A graphics file rasterization and plotting extension for ArcPlot that supports the generation of production quality cartographic maps.
ArcScan 	Raster data interactive editing module required for editing of raster data (GRID) formats.
GRID 	The primary raster analysis module that operates a linear modeling language. GRID is also required to manage 3-D raster (regular interval) data.
TIN 	The primary module for managing, displaying and analyzing 3-D surface data formats, GRID's and TINs for ARC/INFO. TIN is required to manage 3-D irregular point data.

ARC/INFO Component / Module	Description
COGO 	Module for coordinate geometry input and data manipulation. Integrated for use with ArcEdit.
NETWORK 	Module for the analysis and management of routing, allocation, address matching, and dynamic segmentation of linear data networks.

Some of the components and module extensions listed in Table 3 are inherently macro development languages. In particular, the GRID module operates as a modeling language for raster coverage file management, display and analysis. GRID contains functions and commands that support typical programming capabilities such as looping. GRID is seamlessly integrated with AML so that functions, directives or commands in either language can be combined in any order to develop macro programs, and applications.

2.0 The Arc Macro Language – AML

This chapter provides a review of AML focusing on issues and techniques related to developing custom applications. This chapter identifies key requirements for managing and building applications, including techniques and methods for code programming. However, this chapter does not provide a comprehensive overview of AML functions. It is assumed that readers have a working knowledge of AML and it's capabilities. Readers are referred to the References section for sources of more detailed information about AML. Nonetheless, the next section provides a brief overview of AML's basic operating principles prior to more detailed sections.

2.1 Basic Operating Principles

AML is an interpreted language that lets you write programs containing sequences or combinations of ARC/INFO commands, host operating system commands, and AML directives, functions, and variables. An *interpreted language* translates each command line (typically from a script or macro file) to machine language and executes the command line before moving to the next line. No compiling of scripts or macros is required. Accordingly, AML macros and menus exist as textual source code, and are transferable across different operating platforms, e.g. AML macros developed on UNIX will work on Windows NT. This is an inherent advantage of interpreted languages.

AML is similar in principle, form and functionality to scripting languages available in many computing operating systems such as Bourne or C shells in UNIX. AML has its roots in the command processing languages of earlier supported mini-computer operating systems such as Prime Computer's Inc.'s Command Procedure Language (CPL) and the VAX/VMS Digital Command Language (DCL). While these systems are rarely used for ARC/INFO systems anymore they had rich operating system scripting languages and provided excellent foundations for AML's design. AML supports sophisticated actions such as branching, variable manipulation, and argument transfer.

Every command line, either entered from the keyboard or an AML file, is interpreted by the AML processor before the command is executed by the current ARC/INFO program. Thus, the actual command executed in ARC depends on how the command line is interpreted by the AML processor. ARC/INFO never sees directives, functions or variables; it sees only the results of their interpretation. Every time an AML program is executed, the AML processor evaluates each component of a command line in a specific order. The final results of the evaluation are passed to the ARC program being run where the command line is executed. AML statements are executed in the order in which they appear unless you explicitly specify a change through a branching function.

AML macros and menus are text files that contain a sequence of ARC/INFO commands (component or module), and/or AML directives, functions and controls. Simple macros typically contain a list of ARC/INFO commands, such as ArcPlot commands to display data or create a plot file. ARC/INFO commands and AML statements can be combined in any order in a macro file.

The following is an example of a simple macro comprised of ArcPlot commands used to display selected polygons from a forest cover data coverage:

```
MAPEXTENT FOREST
RESELECT FOREST POLY SPECIES EQ 'Pine' AND AGE GT 200
POLYGONSHADES FOREST 2
```

ARCLINES FOREST 1

Macros must consider the ARC/INFO subsystem that is active when the macro is executed. This is the responsibility of the programmer. For example, the above macro containing ArcPlot commands will not operate in the ArcEdit subsystem since these commands will not be recognized by the specific subsystem command interpreter. ARC/INFO subsytsems are atomic in this regard. Typically, AML directives and functions are used to *trap* for subsystems within macros. All AML macros are executed by using the AML "&run" directive.

More complex macros may contain AML directives and functions that accept keyed in arguments from the user, check for existence of data, and branch to specific parts of the macro to undertake actions based on input criteria. This is the norm for most AML programs.

AML is comprised of directives, variables, functions and menu controls.

- Directives begin with **&**
- Variables are surrounded by **%** symbols for interpretation; and
- Functions are surrounded by **[ ]**.

A brief description of each is provided.

2.1.1 Directives

Directives are AML commands; they instruct AML to perform specific tasks. If a command begins with an ampersand (&), it is recognized as an AML directive. AML directives perform actions that never pass directly to a program. Instead, directives instruct AML to perform a specific operation. For example, they can be used to run an AML program, using "&run", or type a message to the screen. Some commonly used AML directives are shown below.

Directive Example	Description of Action
&station 9999	sets up the ARC/INFO environment
&workspace /disk1/demos/data	changes to another workspace
&run my.aml	runs an AML program
&type Hello there	displays a message on the screen
&menu example.menu	displays a menu on the screen
&return	ends the program and returns control

2.1.2 Variables

Variables are a fundamental part of AML. *Variables* are named storage entities whose value can be used, and changed, as an AML program executes. A variable is a means of storing dynamic information. AML directives manage AML variables by performing such tasks as assigning, listing and deleting variable values. The most common variable related directive is the "&set variable" directive. Using this directive the user can create variables within an AML macro that can be utilized by other macro statements.

AML variables can be assigned a wide variety of data types:

- Character strings such as file names, directories, etc.
- Integers
- Real numbers
- Boolean expressions, such as .TRUE. or .FALSE. commonly used to store the result of another AML function such as a coverage existence check
- Expressions that evaluate to any of the above

Variables can be interpreted by other AML statements by surrounding the variable name with the "%" characters. In this way variables values can be change dynamically.

The ability of AML variables to be dynamic is perhaps the most important aspect of AML for operational application development. This allows for changing variables values based on data scenarios resulting in the capability to build robust applications.

AML variables can be either *local* or *global*. *Local* variables are only valid in the AML macro where they are set and used. Their existence and value is unknown to ARC/INFO or other AML programs. *Global* variables, however, are maintained outside of the AML program they are created and/or used in. In this regard a global variable can be used by other AML programs and ARC/INFO commands.

Global variables are the fundamental building blocks of AML macro development allowing AML macros and menus to utilize common variables across toolsets and application components.

2.1.3 Functions

A function, like a variable, performs text substitution, although it is a slightly more complex type of substitution. The name of a function always appears within square brackets [ ].

When AML encounters the square brackets, it evaluates the function contained in the brackets and returns the value of the function.

The value an AML function returns depends upon the function being used. Functions can return a number, a character string, or a Boolean value. A function does not usually act as a command by itself. Normally, the returned value is assigned to a variable or used as part of a command line. For example, the following AML lines illustrate how directives, variables and functions can be used together to determine whether to BUILD an input coverage.

AML Statement	Description of Statement
&args cover	Accept a data coverage as input from the

	user
&if [exists %cover% -cover] &then	Use [exists] function to ensure that the coverage exists for the current workspace
BUILD %cover% POLY	Build topology for the input coverage
&else	Branch to following statement since coverage does not exist
&type Coverage %cover% does not exist.	Echo message to screen telling user coverage does not exist
&return	Return control back to command line

2.1.4 Functional Groups

AML is organized into a number of functional groups. Each functional group is comprised of a number of directives, functions and/or menu controls that satisfy specific programming requirements. Functional groupings include :

- User environment directives
- Input source directives
- Controlling statement execution
- Variable manipulation directives
- Help directives
- Program testing and monitoring directives
- Dialog management directives
- Coordinate input directives
- Mathematical and trigonometric functions
- String manipulation functions
- Reporting functions
- User-file input/output functions
- File management functions
- User input functions
- Inter-Application Communication (IAC)
- Form menu controls (widgets) and options

These groupings reflect the comprehensive nature of AML. AML provides a suite of basic building commands that satisfy virtually any programming requirement. By combining AML statements with ARC/INFO component and module commands an unlimited array of applications can be created.

2.2 Organizing Applications – An Application Development Methodology

The foundation for developing custom applications is the organization and management of applications across a site. This is often referred to as an *Application Development Methodology* (ADM). Defining a methodology for organizing and managing applications on a site will help

AML application developers in building AML programs that are re-usable and easily maintained by an organization. While AML provides excellent application building blocks it does little to aid the GIS system manager, or ARC/INFO application developer, in establishing a framework for continued and flexible application development. At most sites this task is inherently undertaken by the AML programmer or applications developer based on experience, or using the basic ATOOL capabilities of ARC/INFO. Typically, this does not satisfy application management requirements.

The establishment of an ADM is essential for the successful development and deployment of ARC/INFO applications. A simple ADM, embodied in the RAPiD AML software product, is employed and presented in this workbook.[2] Other common approaches are also presented as a framework for comparison.

2.2.1 RAPiD AML – An Example ADM

The RAPiD*AML application development is used as an example in this workbook to illustrate the capabilities and requirements for a proper ADM for ARC/INFO applications. Other solutions and approaches do exist and are viable. The most common are also reviewed in this chapter.

Overall, a comprehensive application development environment provides the following capabilities:

Capability	Description
Application Architecture	A predefined subdirectory structure for applications that will accommodate development and user requirements, multiple development versions, and a production version.
Application Management	Capabilities for managing multiple applications on a site across networks. Typically, a meta-data approach is used to manage applications, versions, users/developers, etc.
Version Control	Management of multiple versions for applications including date stamping, development versions, and a production user version. Ability to create standalone AML applications that do not require the development environment active.
User Access	Setting permissions for application access for users including end users (production version) and programmers

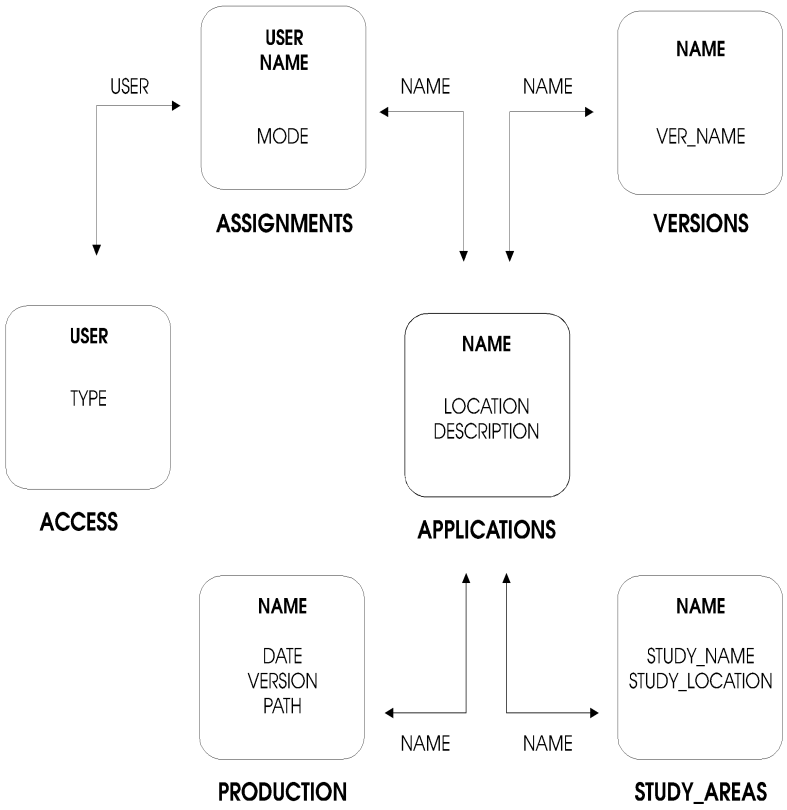
[2] RAPiD AML is a commercial software product developed and sold by Innovative GIS Solutions, Inc., Fort Collins, CO. RAPiD AML is an application development platform built on the ESRI AML platform. RAPiD AML provides a standard application architecture for ARC/INFO application development that enhances the AML programmers ability to develop and maintain applications for a site. For more information on RAPiD*AML visit the IGIS web site at http://www.innovativegis.com.

Capability	Description
	(development versions).
Integrated Development Tools	Application development tools for file management for individual applications including capabilities for macro, menu, script (UNIX or NT), programs (C, FORTRAN, etc), icons/bitmaps, help files, and application support files (lookup tables, key files, etc).
Programming Aids	Typical programming aids including file editors, find/search and replace tools, syntax and spell checking, AML parameter management, session management, and individual user profiles.
GUI and Macro Templates	Provision of a default and standard application interface for new applications including site customizable AML macro and menu templates (files). These templates should be tightly integrated with the programming tools.

Meta-Data Tables

RAPiD*AML uses a simple approach of meta-data files, stored as INFO tables in an ARC/INFO workspace, to manage applications, versions, users, and application assignments (permissions). The following figure reflects the typical relationships between meta-data tables required to support simple application management capabilities.

Figure 1 : A Typical Meta-Data Definition for Application Management



In the meta-data definition illustrated above the following tables exist:

Table Name	Description and Purpose
APPLICATIONS	List of applications organized by a textual name. May also include a path location for the application structure, a longer description, and or a creation date.
VERSIONS	List of development versions for each application. Primary key would be the application name and the version name. May also include a version date (for date stamping), and a version description.
ASSIGNMENTS	List of which applications are assigned to which users. This is the permissions table that defines which users can access which applications. Assignments may distinguish between Developers versus Operators.
ACCESS	List of valid application users. May distinguish between Developers versus Operators.
PRODUCTION	List of production application versions including date stamp, source development version.

Table Name	Description and Purpose
STUDY_AREAS	List of study areas or workspaces valid for an application. Since most workspace definitions are inherently coded into the application programs, and/or utilize environment variables, this table may not be required.

Meta-data tables provide the primary data requirement to develop a suite of simple AML macros and menus that will provide a standard interface for Developers and Operators running an application. These tools can be written in any language, however an AML interface for managing applications is simple to implement based on the defined meta-data tables. Building an application management interface in AML also ensures user access to, and proper configuration of the user account, for operating ARC/INFO applications. Tools are required to :

- Select an existing application and version (within permission definitions);
- Open the application for use (Operator) –or- for development (Developer);
- Create new applications and/or modify existing applications by:
    - Application name:
    - Version name;
    - Source path;
    - Description.
- Manage user access and permissions; and
- Perform software administrative functions like meta-data table backup.

Application management capabilities also require other functional components to provide a comprehensive ADM. In particular, a standard application subdirectory structure, often referred to as an *application architecture*, is required as well as a suite of application development tools. The use of default application templates, often referred to as an *application desktop*, can also be used to kick start a new application for Developers. Default and standard application desktops ensure a consistent look-and-feel for applications on a site, and also address initial code pathing and setup requirements. This leaves the application developer free to focus on the procedural flow of the application, and the specific toolsets required to satisfy application functional specifications. Using a standard application desktop simply mimics conventional Windows application approaches.

Figure 2: The main application development interface for RAPiD*AML presents existing applications, and their versions, assigned for the current user. Applications are assigned based on user name and access is inherent in the list of available applications.

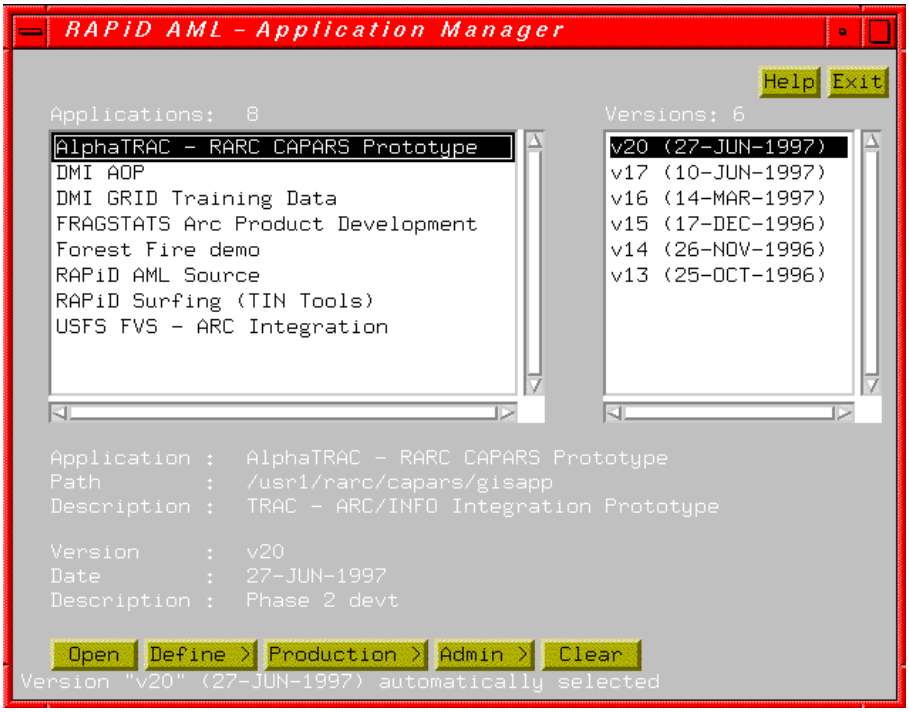


Figure 3: Additional tools are required to help Developers manage existing applications, and versions, or create new applications.

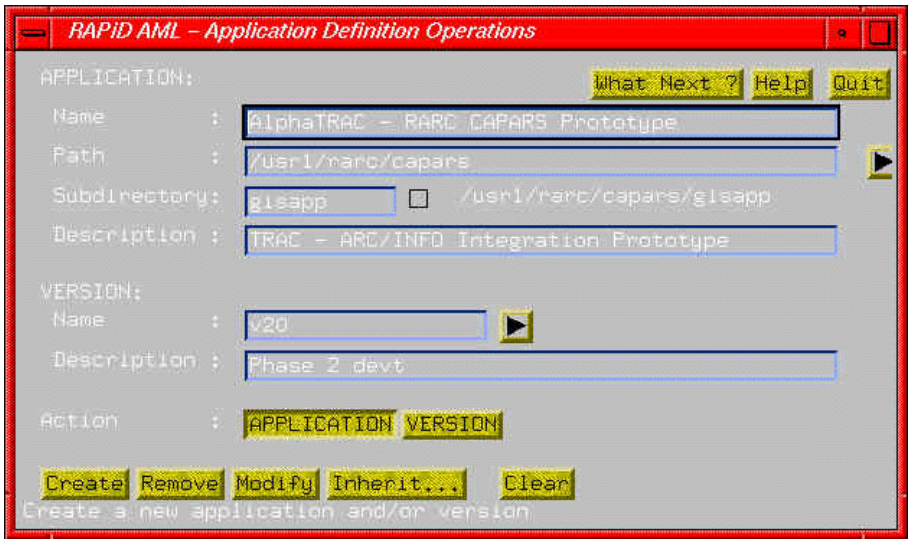
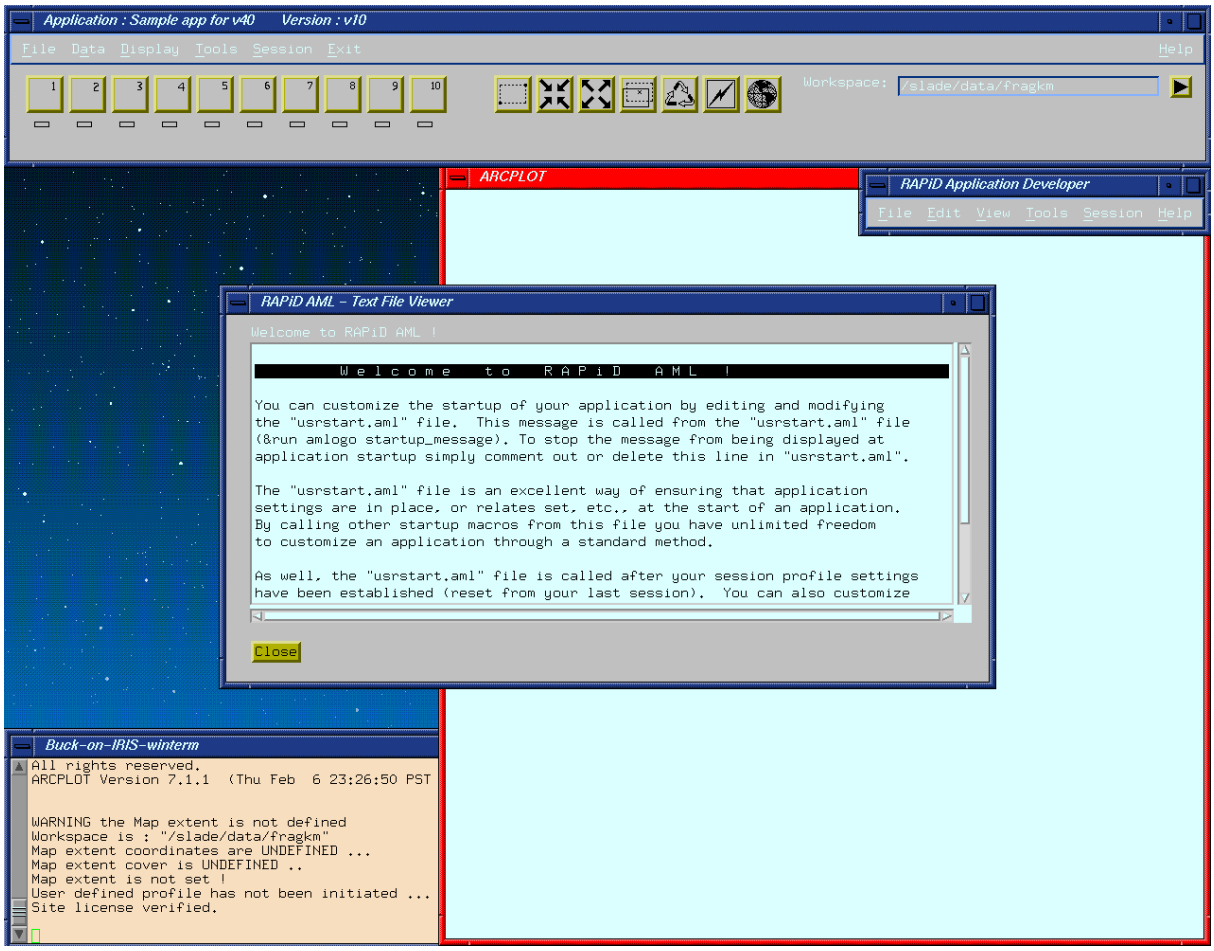


Figure 4: The default application desktop with RAPiD*AML is comprised of a main form menu, with 10 assignable buttons, an embedded pulldown menu for generic tools, a graphics display window, and the application developer tools. Upon entering a new application for the first time a welcome message is displayed reviewing options

for customizing the startup of the application using the a startup macro.



2.2.2 The ATOOL Approach

An approach many AML developers use to customize ARC/INFO for applications utilizes the ATOOL methodology provided with ARC/INFO. This approach allows AML programs to be interpreted as ARC/INFO commands when these files are placed in a specific ATOOL directory. For example, when the “createmaps.aml” file is placed in the proper subsystem ATOOL subdirectory it will be available as the “createmaps” command for that subsystem. Simply keying in the command “createmaps” will execute the AML program.

This is an effective method for starting AML programs, or applications, using a single command. By default a series of ATOOL subdirectories are provided with the ARC/INFO program directories to accommodate this approach. They typically include a single subdirectory for each subsystem, such as:

ATOOL Subdirectory	Description
$ARCHOME/atool/arc	ATOOL commands for the Arc subsystem
$ARCHOME/atool/arcplot	ATOOL commands for the ArcPlot subsystem
$ARCHOME/atool/arcedit	ATOOL commands for the ArcEdit subsystem
$ARCHOME/atool/grid	ATOOL commands for the GRID subsystem

While this approach does accommodate the startup of AML programs and applications, it does not accommodate application management or development requirements, nor consider the pathing requirements (&AMLPATH and &MENUPATH) for other application macros, menus and other required files. The ATOOL approach is also restricted to separate subsystems. In addition, these ARC/INFO ATOOL subdirectories are located within the ARC/INFO software programs, and are typically protected from *write* access from most users.

To offset this issue an AML directive, &ATOOL, allows the user to customize the path location for ATOOL subdirectories. With this approach the AML developer, or system administrator, can define custom ATOOL subdirectories, not located within the ARC/INFO programs, that developers can write AML programs to. In this manner ATOOL commands can easily be added to ARC/INFO, allowing for custom commands for a site. This is often effective for simple AML tools used by many end users such as map creation and data management. However, for operational applications the ATOOL approach suffers from the lack of management and development tools required to support on-going development and maintenance of applications.

2.3 File Management Issues

Most operational ARC/INFO applications are comprised of a number of AML macros, menus and other assorted files. These files must be organized in a manner that allows the AML processor to execute them without specific hardcoding of subdirectory names or paths. This section of the workbook reviews different issues and methods for addressing file management issues.

2.3.1 Pathing & Environment Variables

AML contains two primary directives that allow the developer to customize the paths to be searched by the AML processor for AML programs and files during execution of an application. &AMLPATH and &MENUPATH directs the AML processor to search for macros and menus in the named directories if they cannot be found in the current working directory. These directives operate much like the PATH statements under DOS and UNIX where directories listed in the path are searched in order for the program file, and if found, are executed.

The &AMLPATH and &MENUPATH commands are critical tools in defining

code locations for AML based applications. These path statements should reflect the application architecture that is defined for applications on a site.

The setting of AML paths allows the programmer to embed &RUN statements in an AML program without needing to explicitly define a path for the AML program. For example, by setting the &AMLPATH in advance to :

&amlpath /disk1/apps/tools

&menupath /disk1/apps/tools

an application's programs will properly find and execute an embedded &RUN command, such as running the INIT routine of the "myamltool.aml" program :[3]

&run myamltool init

Without an AML path setting the programmer would have to explicitly define the path for the AML program, such as :

&run /disk1/apps/tools/myamltool init

The &MENUPATH is also required if the AML program to be run utilizes a menu file (.menu or .emf), and/or any UNIX icon or bitmap files (.icon and .bm). Since icons/bitmaps are only used in AML menus they are included by the AML processor in the &MENUPATH.

A maximum of 30 paths can be defined in any AML path directive. An important consideration in setting AML paths is the order in which subdirectories are defined. Remember that the AML processor will search directories in the order that they are defined.

The following priorities are general guidelines for defining AML paths with applications:

Priority	Description of Path
1. Custom user profiles or application customizable code	Complex applications sometimes allow the user/operator to customize desktop or operating preferences. These preferences are often stored as AML programs in a user specific application subdirectory.
2. Primary Application code	The primary application code.
3. Local site specific AML tool directories	Some ARC/INFO sites create generic tools or ATOOL directories to store standard site-specific tools accessible for all users.
4. Application development AML tools	If an application development product is used, e.g. such as RAPiD*AML, it is often useful to utilize tools

[3] The use of routines in AML programs is described in Section 2.5.

	provided with this software.
5. ARC/INFO support tools such as ArcTools paths	The ARC/INFO support directories, and the ArcTools subdirectories in particular, provide an useful suite of toolsets that are often invaluable for many applications.
6. Any other existing path	Any paths existing when the application is started should be maintained by placed last in the path search order so that no possibility exists for application files to be superseded by user programs.

Paths in AML are always superseded by the directory that you start ARC/INFO in. This can cause problems with AML program execution because any AML programs with the same name as a program in your application, that is located in your startup directory, will take precedence over your defined AML paths. Accordingly, it is good practice to start ARC/INFO and any applications from a directory that does not contain AML programs, menus or icons/bitmaps.

Another method is often used to define path locations within application AML programs. The use of a system environment variable, like the $ARCHOME variable, is very useful for applications that meet the following criteria:

- The path for the application code may change frequently;
- The application code may be installed on several different systems across a site; or
- The application will be distributed to many different sites where application and data organization methodologies are not known.

For example, for the FRAGSTATS*ARC software application developed by Innovative GIS Solutions, Inc. we needed to accommodate the distribution of the software on many different systems, running different UNIX or Windows NT operating systems.[4] To satisfy this the application uses a system environment variable, $FRAGHOME, to *point to* (locate) the source path for the application code. Once the application starts up it decodes the $FRAGHOME environment variable and uses this path to define the priority &AMLPATH and &MENUPATH. With this approach the application user merely has to have the environment variable set prior to running the application. This is a standard approach for most commercial software, especially with UNIX systems, and is utilized directly by the ARC/INFO software with the $ARCHOME and $ATHOME environment variables. To use an environment variable on UNIX you would need to define the environment variable first, such as:

setenv FRAGHOME /disk1/apps/fragstats/pro

and then have &RUN or &MENU statements use the environment variable in your AML

[4] FRAGSTATS*ARC is a landscape structure and spatial pattern analysis application developed for use in landscape ecology and bio-diversity disciplines. More detailed information can be obtained from the IGIS Web Site at http://www.innovativegis.com

programs, such as :

&run $FRAGHOME/code/myamltool init

For UNIX systems the environment variable should be defined in the individual user's C-shell resource file ".cshrc" located in the home directory. For Windows NT systems the environment variable is defined in the System Environment settings tool found in the Control Panel.

2.3.2 Individual User Startup Files

Another method sometimes used by system administrator's to customize an individual user's startup of ARC/INFO, and its subsystems, is the ".<subsystem>" file. By creating a file in a user's home directory using the ".<subsystem>" naming convention where <subsystem> is arc, arcplot, arcedit, etc., you can customize how an ARC/INFO subsystem startups for a user. By default ARC/INFO will check for, and execute as an AML program, a ".<subsystem>" startup file for each user when it starts up an ARC/INFO subsystem. Many users will utilize this method for setting &AMLPATH and &MENUPATH automatically so system wide and custom AML tools are available for use at the command line. This is a good method for individual customization, but each startup file must be duplicated for all users if you want system wide application.

Startup File Name	When File is Executed
.arc	Runs every time ARC is started for the user
.arcplot	Runs every time ArcPlot is started for the user
.arcedit	Runs every time ArcEdit is started for the user
.grid	Runs every time GRID is started for the user
Etc.	

In addition, many users customize the ".<subsystem>" startup file with explicit &AMLPATH and &MENUPATH statements. This can often cause serious operational problems with an AML application. For example, a user may execute an AML application which internally sets appropriate paths. In that application it might start ArcPlot for graphics display. If the individual user has a custom ArcPlot startup file that explicitly sets AML paths, without respecting existing paths, it would get executed automatically when ArcPlot starts by ARC/INFO. This could change AML paths resulting in the AML processor not being able to find application programs. In this scenario each time the application attempts to execute an AML program you would receive an error.

Two simple methods exist to address this. Ideally, any AML paths set in ".<subsystem>" files should respect existing AML paths. Secondly, it is good practice to account for such individual user customization by checking AML paths in your application code to ensure that paths have

not been modified by ".<subsystem>" files. This solution requires more AML programming in the application, but will ensure operation. The following example illustrates the proper AML path settings for a by ".<subsystem>" file.

Proper AML Path Setting	Improper AML Path Setting
&amlpath [show &amlpath] /disk1/apps/tools	&amlpath /disk1/apps/tools
&menupath [show &menupath] /disk1/apps/tools	&menupath /disk1/apps/tools

2.3.3 System Startup Files

The ARC/INFO software also provides AML programs that are used in the startup of all subsystems on a global wide system basis. The subsystem startup AML programs are stored in the *$ARCHOME/startup* directory located with the ARC/INFO software programs. Typically this directory is protected for general access and restricted to the system administrator for *write* access.

System Startup File Name	When File is Executed
$ARCHOME/startup/arc.aml	Runs every time ARC is started by a user
$ARCHOME/startup/arcplot.aml	Runs every time ArcPlot is started by a user
$ARCHOME/startup/arcedit.aml	Runs every time ArcEdit is started by a user
$ARCHOME/startup/grid.aml	Runs every time GRID is started by a user
$ARCHOME/startup/librarian.aml	Runs every time LIBRARIAN is started by a user
Etc.	

These startup files can be modified to customize the startup of each subsystem on a system wide basis. In other words, changes to these files will affect all users. It is important that any modification to these files be undertaken by an experienced ARC/INFO system administrator. Inappropriate or incorrect modifications may cause fatal errors and subsystems may not start.

In most cases these files are only modified to reflect a site's notice and identification, such as "Company X GIS Environment", or to provide system wide messages about new AML programs or tools that are available using the ATOOL approach. Providing a list of custom site tools is ideal with subsystem startup. It helps notify users of changes in available tools.

2.3.4 Version control

Version control and maintenance of application code is often a problem with ARC/INFO applications. Since no standard application development platform is provided with ARC/INFO, or readily promoted in the commercial software marketplace, most applications are maintained by the resident AML developer or programmer using self defined approaches. At most sites this is a concern and may result in lost code, or cumbersome updates for production applications. It

is typically not an issue for sites where most applications are operated by a single user. In any event application version control should be considered mandatory, especially for larger sites where applications are used to support production activities.

Development versus Production versions

The ability to maintain and distinguish between *development* versions of an application, and a *production* version is often critical. Version control techniques allow developers to maintain multiple versions of an application, yet establish a single production version for operational use. With non-interpretive languages this is inherently accommodated by *source code and compiled executables*. Such is not the case with AML, but can be achieve through using file encryption techniques available with the &ENCODE directive. However, this does not negate the need for formalized version control capabilities.

Tools exist within most version control software that will update the production version at any time with a new development version once sufficient testing and quality assurance of the application has been completed. Version stamping and date stamping is commonly used to guide the developer in maintaining lineage for a production version of an application. This approach ensures that production users are not affected by on-going development of an application, and provides a structured mechanism for fixing errors and bugs that arise in an application.

While many commercial version control software offerings exist in the marketplace these are typically considered overkill for an interpretive language like AML. Few ARC/INFO sites make use of these commercial options with AML.

It is desirable for the version control capabilities to be integrated with the application management system in use.

Encrypting AML Programs

At some sites it is desirable for production applications to be encrypted into binary format so they are protected from inappropriate modification. An AML directive, &ENCODE, allows the developer to encrypt AML programs (macros and menus) into a binary format. Encoded programs have an ".eaf" extension for ".aml" macros, and an ".emf" extension for ".menu" files. The file extension is automatically assigned by the &ENCODE directive, and is recognized by the AML processor. Changing the file extension by renaming an encrypted file may cause the AML processor to not recognize the file.

It is relatively simple to write an AML program to encrypt your application code, however it is desirable to have these capabilities within the version control capabilities of the application management system that is utilized within your site's ADM. Developer's should have the option to encrypt AML programs when creating a production version of an application. It is important to note that the encryption process is not reversible, and original AML source code should always be kept in a safe location if encryption techniques are used.

In case you may want to utilize the &ENCODE encryption capability it is good practice to not include AML file extensions in your "&RUN" and "&MENU" statements in AML application programs. In particular, with "&RUN" (macro) and "&MENU" (menu) directives ensure that you do not define the file extension unless you always want the source code to execute.

The following example illustrates this:

Restrictive Statement	Flexible Statement
&run myamltool.aml	**&run myamltool**
&menu mymenutool.menu	**&menu mymenutool**

If the file extension if not defined in an AML statement, the AML processor will automatically search the defined AML paths for encrypted programs first, and source code programs second. By simply following this methodology you can provide the flexibility for using encrypted code if the need warrants it.

Remember, that encrypted AML programs (".eaf" and ".emf" files) will always take precedence over source AML programs (".aml" and ".menu" files) with the AML processor. Accordingly, ensure that source and encrypted AML programs are <u>not</u> maintained in the same directory. Encrypted programs are <u>not</u> automatically updated when you modify source AML programs. You must explicitly encrypt each source program. Accordingly, unpredictable results will occur if you are modifying source AML programs in a directory where encrypted files of the same name exist.

2.4 Workspace / Data Management Issues

It is common practice that workspace and data management occur within specific application programs. ARC/INFO operates around the use of a single active workspace, or opened data libraries. Most AML programs, especially toolsets, are written to operate on data within any active workspace. If specific workspace definitions are required in AML programs it is desirable to not hardcode workspace or directory names. Hardcoding will greatly restrict the ability to maintain code, and make it difficult to transfer applications across different sites, computers, and/or users.

2.4.1 Environment Variables

The use of system environment variables is an effective method of defining a data subdirectory structure without hardcoding explicit paths. If a specific data directory architecture is utilized for application specific data an environment variable should be used to define the root path for this structure. The environment variable can then be utilized in AML programs and menus. If the data location is ever changed application code would not need to be modified, simply the user's environment variable setting instead.

The definition of library paths is explicit in the LIBRARIES INFO table used by the LIBRARIAN subsystem for managing libraries. When creating a library ensure that you use an environment variable in the definition of the DATABASE directory in the LOCATION item value for the library. This will ensure that you do not suffer from hardcoding with your ARC/INFO library configuration. Hardcoding can also affect the operation of an application, especially if libraries are moved to other storage device (disk) after initial library configuration.

2.4.2 Data and Code Separation – Workspace Independence

It is desirable to separate AML programs from data workspaces. It is consider bad practice to mix application programs with data in the same workspace directory. When this does occur it provides opportunities for AML programs that reside in an active workspace to supersede AML programs of the same name the reside in a valid AML paths defined with the &AMLPATH or &MENUPATH directives.

2.5 Macro Development

AML is a rich and robust programming language whose strength lies in the ability to easily create macros. The AML macro, identified by the ".aml" file extension, is the cornerstone of building robust ARC/INFO applications. This section reviews some important aspects of macro programming focused on utilizing effective and efficient coding methodologies.

2.5.1 Coding Methodologies

To make best use of ARC/INFO and the powerful AML language, formal programming (coding) methodologies are recommended. Formalizing standards for AML macro development, and promoting techniques for re-using AML programs are critical for the effective use of ARC/INFO. By adopting AML coding standards early, individual programmers and organizations, are assured of a better life cycle for AML based applications.

ArcTools Coding Standards

ESRI developed, and promotes, a standard AML coding methodology through the ArcTools component of the ARC/INFO software.[5] ArcTools is a menu interface for ARC/INFO subsystems that is provided as a set of AML-based *tools*. ArcTools provides a menu interface for the most commonly used components of ARC/INFO, that is Arc, ArcPlot, ArcEdit, and GRID. In addition, and perhaps most importantly, ArcTools provides a set of AML macros, menus and icon/bitmaps that can be used as source programs for developing your own custom applications. ArcTools provides a rich source for building your own toolsets. Quite often much of the source programs required to undertake functional tasks in ARC/INFO already exist within ArcTools.

ArcTools is organized into modules. They include :

ArcTools Module	Description
Map Tools	An interface for ArcPlot used to create map displays and query data.
Edit Tools	An interface for ArcEdit used for interactive editing, data conversion, digitizing, and data manipulation.
Grid Tools	An interface for the GRID module used to undertake raster analysis and modeling.

[5] Environmental Systems Research Institute, Inc., Redlands, CA is the company that develops and sells the ARC/INFO software line.

ArcTools Module	Description
Command Tools	An interface for Arc commands used for data manipulation, conversion, and geo-processing functions.
Land Records	A generic application that uses the COGO module for the management of land parcels. This application is provided as a generic municipal parcel mapping and data management application.

Each of the ArcTools modules and application are comprised of toolsets. These toolsets can be utilized as generic tools, in the ArcTools module, or in the case of Land Records as a generic application.

The ArcTools system promotes the development of re-usable *toolsets* through a standard coding methodology. This methodology focuses on using *multiple entry point* AML programs in an effort to make functions atomic in nature. These AML programs are referred to as *toolsets*. This approach is consistent with conventional programming techniques of using *subroutines* with non-object oriented languages such as FORTRAN.

Using Toolsets – AML Lego

A toolset is a discrete functional combination of a macro, menu(s), related data tables, icon/bitmaps, and help file(s) that perform a specific task.

Toolsets are the fundamental building blocks of AML based applications in ARC/INFO.

A *toolset* is comprised of :

- A primary AML program (macro);
- A primary AML menu;
- Icons (SUN) or bitmaps (X-11); and
- Help files.

A tool performs an operation, or a set of operations, for the application developer based on a specific functional capability. It may be a single AML that performs a simple operation, such as checking if the user has write access to a directory, or more typically it includes a menu interface that allows the user to specify the parameters needed to perform an operation.

This approach affords a very modular design and results in independent tools that can be utilized in combination, and re-used for different applications. For example, one tool might manage workspaces, another symbology sets, another coverage display, etc. This approach affords the development of tools, and applications, in short timeframes.

The following conventions and guidelines are recommended for programming toolsets:

1) All files associated with a tool, such as the macro, menus, and help files, should have a common name prefix, e.g. buffer.aml, buffer.menu, buffer.help.;
2) Tool names should describe the tool's purpose such as "buffer", "getcover", "select", "createmaps", etc.
3) Global variables used within a tool should have a standard naming convention that can be referenced explicitly to the tool, such as:

Variable	Descritpion
.buffer$incover	Input coverage to be buffered
.buffer$outcover	Output coverage to be created
.buffer$distance	The buffer distance

   Often the tool name is used as the prefix for the global variables and is separated from the suffix by a "$" sign. There is no interpretive meaning to the "$" separator.
4) All menus initialized by a tool should be created as individual *threads*.[6]
5) Standard approaches should be used in all tools for headers, comments, routine names, thread management, and argument passing. A *header* is the commented description that should reside at the top of each AML program. Appendix A provides an example of a typical file header for an AML toolset.

Multiple Entry Point Approach

The multiple entry point approach is based on the development of individual routines within an AML program. Each routine performs a specific task required for the tool. This is modeled after conventional linear programming techniques common to FORTRAN and other similar languages that use *subroutines*. Each routine serves a separate purpose, and routines are used together to execute specific tasks for the tool.

Routines

Each routine in an AML program is encapsulated by the &ROUTINE and &RETURN statements. &ROUTINE identifies the beginning of the routine while &RETURN identifies the end of the routine. Routines are typically listed sequentially in an AML program in a logical manner, and/or order of use. Routines are atomic, that is, they are not dependent on other routines. However, routines can include any valid ARC/INFO or AML statement, and hence can call other routines (&CALL) or run other AML programs (&RUN). The following example illustrates the basic &ROUTINE block for a routine called "buffer_cover". The ability to call another routine in the same AML toolset, or run a routine from another AML toolset is also

[6] Threads are the mechanism that delivers input to the AML processor. ARC/INFO is automatically started with a thread. AML threads are the mechanism used to display and manage multiple menus in an application. With threads, tools can have submenus and an AML application can manage the input through these threads. Threads are the primary building blocks for AML applications using a Graphical User Interface (GUI).

illustrated.

AML Statements	Description
&routine buffer_cover	Start of routine block
&call get_item	Call another routine in same AML program
&run buffer init %cover% %item%	Run a routine in a different AML program
&type All done.	AML statement to echo a message
&return	Exit the routine and return control

The ArcTools toolset approach provides *template* macro and menu files to support toolset creation. A standard set of routines are defined in each toolset (AML program) to address tool initialization, menu threading, help, and orderly exit and error *bailout*. Based on our experience with applying these templates, we promote the use of a modified set of standard routines that better address application requirements. This reflects a minor change from the ArcTools standards, but endorses the same coding methodology.

The most noticeable change is the addition of the BACK and NEXT routines. In some application scenarios it is desirable to present the user with a *procedural interface* that *walks* the user through a series of steps, with each step coded as a separate tool. This is often referred to as the Windows Wizard approach. With this approach menus typically have [< Back] and [Next>] buttons to guide the user through steps. IGIS has implemented this approach successfully in several applications, and has modified the toolset standard routines to accommodate this approach. Of course, this approach requires the definition of menu widgets to match. These widgets are provided with the menu template.

The standard routines in a toolset are described below. Deviations from the standard ArcTools routines are identified by a *****.

Standard Routine	Description
INIT	This is the routine first called to initialize the tool. The INIT routine often includes optional arguments depending on the task the tool performs. This routine calls the INITIALIZE and MENU routines.
INITIALIZE *	This routine performs any data or lookup table initialization that is required for the toolset. This routine can be called at anytime to reset, or re-initialize the tool, or during the initial startup of the tool.

MENU *	This routine starts a thread for the tool main menu. Arguments are accepted for the menu position on the screen, and the stripe (title).
USAGE	This routine echoes back to the screen the calling usage for the tool if the AML program is called without a routine.
HELP	This routine will display the tool's help file.
BACK *	This routine is optional. It allows the tool to utilize a Wizard like approach for the specific toolset. This routine will return the user to the previous tool, and removes the current tool thread. This mimics the [< Back] action in Wizard Windows menus.
NEXT *	This routine is optional. It allows the tool to utilize a Wizard like approach for the specific toolset. This routine move the user to the next logical menu, and removes the current tool thread. This mimics the [Next >] action in Wizard Windows menus.
APPLY	This routine is used to accept the input and quit the tool.
CANCEL	This routine will cancel the tool without accepting tool input.
QUIT	This routine will exit the tool by deleting the thread.
BAILOUT	This routine is called if an error occurs during execution of any tool routines. It is a method for exiting the tool orderly during error scenarios, and provides a standard error message identifying line number that the AML program failed in.

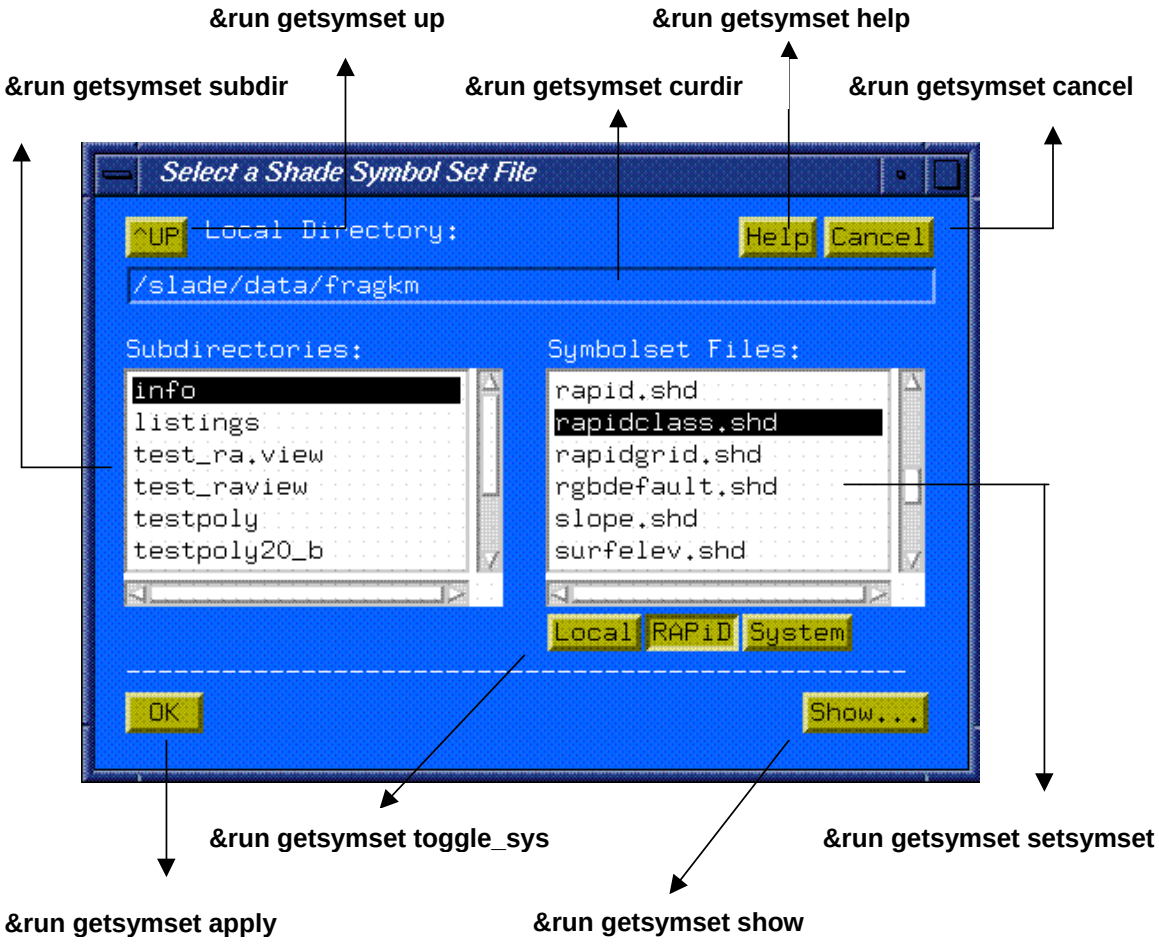
Appendix B presents a standard toolset template macro to illustrate the organization of routines.

An Example Toolset – getsymset.aml

The "getsymset.aml" toolset example is presented to illustrate how tool components, that is the macro and menu files, work together. This example represents a standard ArcTools tool provided with ARC/INFO that has been slightly modified to address local site requirements. In this case, the purpose of the tool is to support the user in selecting an active symbolset file. The tool has been modified to accommodate RAPiD*AML symbol sets, as well as local workspace, and ARC/INFO symbolsets. Figure 5 illustrates the routines called by menu widgets.

This coding approach utilizes the toolset macro file for all actions. No actions other than direct macro calls are executed in the menu. The menu widgets simply call routines in the toolset macro. With this approach all action or processing code is located in a single location, the macro toolset.

Figure 5: Routines Used by the "getsymset" toolset.



Appendix C presents the macro code associated with this toolset.

All actions that a particular tool executes are contained within a single AML and represented as routines in that AML. To activate a tool, run it with the initialization routine passed as an argument, e.g., &RUN toolname INIT. The initialization routine displays the tool's menu and may initialize variables used by that menu (see the previous description of standard routines such as INIT, INITIALIZE and MENU). Subsequent actions invoked from the menu are accomplished by running the tool's *driver AML* with the appropriate routine name as an argument, e.g. &RUN toolname <routine>). There may be other arguments in addition to the routine name.

The main body of tool driver simply calls the requested routine, or calls the USAGE routine if no routine has been specified. The example below shows the main body of a typical tool driver:

AML Statement	Description
&args routine	Accept the routine name as the only argument
&severity &error &routine bailout	Define the BAILOUT routine as the source for error

AML Statement	Description
	situations
&if [null %routine%] &then	Check for a null routine name
&call usage	Call the USAGE routine to echo tool usage in case of no routine being passed
&else	A routine was passed as an argument
&call %routine%	Call the routine
&return	Return control to the calling thread (menu)

Every tool driver has a routine for initializing the tool menu. The INIT routine initializes any default settings that will be used by the tool, then displays the tool menu. The modified toolset template presented in this document uses a separate routine, INITIALIZE, to setup variables etc., and a separate routine, MENU, to actually initialize the menu thread. The conventional ArcTools approach uses a single routine, INIT, to accomplish these tasks. The properties for the tool are stored as global variables that follow coding standards and naming conventions. The tool menu contains widgets which may set some parameter, or result in the execution of some operation, e.g. a routine. Any place where an operation results from a menu widget selection, the driver is executed with the appropriate routine. In other words, for each widget that performs an operation, there is an associated routine in the driver.

The EXIT routine of standard ArcTools toolsets will delete the thread, which results in the menu being removed. In addition, this routine also *cleans up* all variables used in the tools. A standard approach is to delete all global variables used in the tool. Because these global variables are removed, and not maintained after use of the tool, they are referred to as *local globals*. Depending on the need for tool global variable settings to interact with other tools for your application you may not want to use this approach. It is our experience that in many cases variable settings are useful for the initialization, and configuration, of other tools. If this is the case, you may want to modify this action for individual toolsets.

Using AML Templates

To aid programmers and developers in building their own custom toolsets *template* macro and menu files are typically used. The template macro should include the standard set of routines. The template menu should include widgets for standard APPLY, HELP and QUIT buttons. Buttons for BACK and NEXT are optional and only required if the menu is to be used as part of a step-by-step Wizard interface.

Templates are an excellent approach for kick-starting the toolset development process. ArcTools provides standard toolset templates in the *$ATHOME/util* directory. The template files presented in Appendix B presents a simplified version of this approach with minor additions to accommodate for the BACK and NEXT routines.

The use of template toolset files should be tightly integrated with the application development

methodology employed at your site. In particular, the template files should be automatically used by any integrated development tools or programming aids employed in the ADM.

Application Startup and Exit

A common requirement for applications is the ability to customize the startup and/or exit of an application. A development environment should facilitate a standard method for application developers to implement startup and exit customization. This can be achieved simply by using an AML program with a standard name, such as “appstart.aml” or “appexit.aml”. These macros should be called by the macros that manage the application.

2.5.2 Using Variables

Global variables are the glue that bind AML toolsets into applications. In most cases it is global variable management that is most critical to the operation of an AML application.

Accordingly, care should be taken in managing the use of global variables. This involves several issues including naming conventions, error trapping and handling, and techniques for overcoming variable limitations.

The toolsets programming methodology revolves around the use of global variables. Variables are defined as needed to accommodate the requirements of the individual toolset. A standard naming convention is typically used with toolset variables that helps to organize and manage variables for use. In general, global variables should utilize a prefix (following the period) that matches the toolset name, e.g. .buffer for the buffer.aml toolset. The actual name of the variable should follow a “$” character that is commonly used to separate the prefix from the name. This name should be fairly descriptive such as the examples shown earlier, .buffer$incover, .buffer$outcover etc.

Objects

In some instances, especially for more complex toolsets, it may be desirable to add another level in the naming convention. For example, it situations where global variables are used to store an array of values, such as characteristics of several coverages defined as part of a display theme or view, variable naming may be more complex. This approach is utilized by ArcTools to maintain definitions of *objects*. This approach mimics object oriented programming techniques by storing *object* properties in an array of global variables. By definition, an object is an entity which encapsulates both a definition of classes and properties, and code to represent the object.

Even though AML is not an object oriented language, ArcTools programming standards follow object oriented methodology where possible, including encapsulation of code and the definition of classes for creating and storing objects.

ArcTools uses object oriented programming techniques for views and themes, map layouts and map objects (Map Tools), and grid models and model steps (GRID Tools). Each of these types of objects has its own set of *properties*. The different classes of themes and map objects each have unique properties. Global variables are used to mimic conventional variable arrays. The following example illustrates how global variables are used to store properties about map layout

objects. Two objects are shown. Note that the "**->**" separator is used by ArcTools to define an object property global variable. These properties of the *map object* are defined using a standard toolset approach of macros and menus. In fact, an *object manager* toolset is used to manage the properties of defined objects, through a *property sheet* menu, so the user can modify object definitions.

```
/*
/* Object 1 properties
/*
&set .map_object1->outline = 2.180 9.183 4.144353084564 9.4593156
&set .map_object1->justification = ll
&set .map_object1->symboldef = 1000,1,proportional,0,0,0,1,~
  0.2763,0,0,0,butt,miter,none,0,0,horizontal,simple
&set .map_object1->class = text
&set .map_object1->string = parcel map
&set .map_object1->locy = 9.183
&set .map_object1->locx = 2.180
&set .map_object1->driver = class_text
&set .map_object1->drawfull = .true.
&set .map_object1->identifier = title
&set .map_object1->outlinetxtloc = 3.16 9.32
/*
/* Object 2 properties
/*
&set .map_object2->outline = 0.05 0.05 8.45 10.95
&set .map_object2->class = neatline
&set .map_object2->outside_size = .1
&set .map_object2->maxy = 10.95
&set .map_object2->maxx = 8.45
&set .map_object2->miny = 0.05
&set .map_object2->minx = 0.05
&set .map_object2->layers = 1
&set .map_object2->driver = class_neatline
&set .map_object2->linesize = 0.0099
&set .map_object2->drawfull = .true.
&set .map_object2->offset = .1
&set .map_object2->identifier = border
&set .map_object2->symboldef9 =
&set .map_object2->symboldef8 =
&set .map_object2->symboldef7 =
&set .map_object2->symboldef6 =
&set .map_object2->symboldef5 =
&set .map_object2->symboldef4 =
&set .map_object2->symboldef3 =
&set .map_object2->symboldef2 =
&set .map_object2->symboldef1 = 1000,1,hardware,0,none,~
  1,0.0099,0,0,'0',0,0,0,0.0099,round,round,0,0,0
&set .map_object2->outlinetxtloc = 4.25 5.5
```

&set .map_object2->inside_size = .01

The use of objects with AML is a level of complexity higher than the use of toolsets. The use of objects, even though used extensively in ArcTools for the Map Tools and GRID Tools modules, is typically not required for the average AML programmer. While it is a sophisticated and elegant approach to utilize global variable arrays, it is complex and confusing to most AML programmers, and in most cases unnecessary. An important aspect of employing different coding approaches is the ability to easily maintain the application code for an organization. Complex approaches, while being elegant operationally, often required more seasoned skills to maintain. This should always be considered when deciding on *utility* versus *object* toolset coding approaches.

Error Handling and Trapping

Like all programming languages AML requires extensive error trapping and handling to build robust applications. In many cases the amount of error trapping for possible error scenarios exceeds the operational statements in an AML routine. Unfortunately, this is the downfall of conventional linear programming languages. However, several approaches will enhance your ability to trap for errors and handle these errors in an orderly fashion.

The &SEVERITY directive allows you to specify actions to be taken should an error occur in an AML program. With AML toolsets the &SEVERITY setting for errors is to *drop into* the BAILOUT routine and echo the routine name, the error message, and line number of the error. This provides an orderly mechanism for addressing errors, and provides a source code line number to check. This is typically sufficient information for correcting errors.

Most AML functions provide a mechanism to trap for errors by checking logical values prior to executing an AML statement, or by checking AML$ reserved variables. In the first case the following guidelines should be used for error trapping your routines:

- Always check for null values in global variables prior to using them in a routine. This is especially true if the variable is a passed argument. Never assume that previous routines, or user interaction, will set a global variable value. ;
- When using AML functions, such as [EXISTS], always provide a warning or informational message to the user if the function fails, e.g. the coverage does not exist.;
- When using AML functions, such as [OPEN] or [READ], that contain a status variable, always check the status variable. UNIX or external settings often cause errors with AML functions pertaining to file management.;
- Whenever possible, trap for the impossible especially if the tool requires user key-in input !

Limits

There is a virtually unlimited number of global variables allowed for use in ARC/INFO at any one time. Nonetheless, it is important to cleanup and manage your global variables in an application. Adhering to ArcTools coding standards, e.g. the *local* global approach of deleting globals upon exiting a tool, will help to automatically minimize the amount of variables active.

Each global variable is restricted to 1024 characters in value length. This is often a problem when using global variables to store the results of an AML function, such as LIST* functions.

The LIST* functions in AML will populate a global variable, or ASCII file, with a listing of selected entities defined by the specific LIST* function. For example, the LISTUNIQUE function is often used to create a list of unique INFO item values to be used in a processing loop. In some scenarios the returned list of unique item values exceeds the 1024 character limit for AML variables. When this occurs an error is echoed, but it is not easily trapped for. If subsequent code is dependent on the returned value to be store in the AML variable numerous processing errors will occur. The primary problem with this limit is that it is impossible to trap for the error.

A solution for this problem is to *pipe* the output from AML functions to an ASCII file whenever possible. For example, many of the LIST* functions have an optional argument that will create an ASCII text file and populate the file with the results of the function. With this there is no character limit. A LISTUNIQUE function that would fail due to the 1024 character limit when output into a global variable, will successfully return the results to an ASCII file. This is potentially a problem if the results of the function are to be used in a processing loop, or in a similar fashion, within your AML program, since they are no longer locally stored in global variables. In these scenarios you must [OPEN] the ASCII file and loop through and [READ] each line. This adds another level of complexity to your AML program, yet will ensure you are not affected by the dreaded 1024 limit.

Unfortunately, the use of the ASCII file technique, is fully dependent on your data and the complexity of your AML programs. Some users never run up against the 1024 limit. However, it is our experience in building operational applications that the limit is often reached when generating lists, typically using AML functions, and especially when dealing with INFO 'C' type items. The use of the ASCII file technique should be considered when data lists are involved.

2.5.3 ArcTools - Source Toolsets

The ArcTools code is a rich source for developing your own custom toolsets. We have found that in most situations ArcTools code already exists for virtually every function within ARC/INFO. It is an excellent starting point for development of a custom toolset.

Two types of tools exist within ArcTools, *utility* tools and *object* tools. For both types, AML code is organized into modules, or building blocks, which are used to create the overall system, and can be used to embed into your applications.

Utility tools are designed to perform simple operations, such as browsing the file system and returning the name of a selected coverage, or issuing the identify command on a specified coverage and feature class.

Object tools allow the user to specify certain properties for an object, store these properties, perform some action based on these properties, edit the properties, and recall these properties for later use. For example, a line coverage object tool would allow the user to specify the drawing properties for the coverage, draw the coverage, and store these properties as global variables.

The object tools are more complex in design, and consequently are more difficult to implement into a custom application for the average AML programmer. They also require more seasoned programming skills to

maintain.

Utility tools are grouped by the ARC/INFO module for which they provide functionality within ArcTools, and are located in the various directories under $ATHOME (the environment variable set to locate the ArcTools directory). There are ARCEDIT tools located in aelib, ARCPLOT tools in aplib, GRID tools in gridlib, and so forth. There are also general tools that can be used in any module, including messaging tools, file browsers, and simple data managers, located in *$ATHOME/lib*. The following table identifies the general utility tools available within ArcTools that are easily used in custom application development.

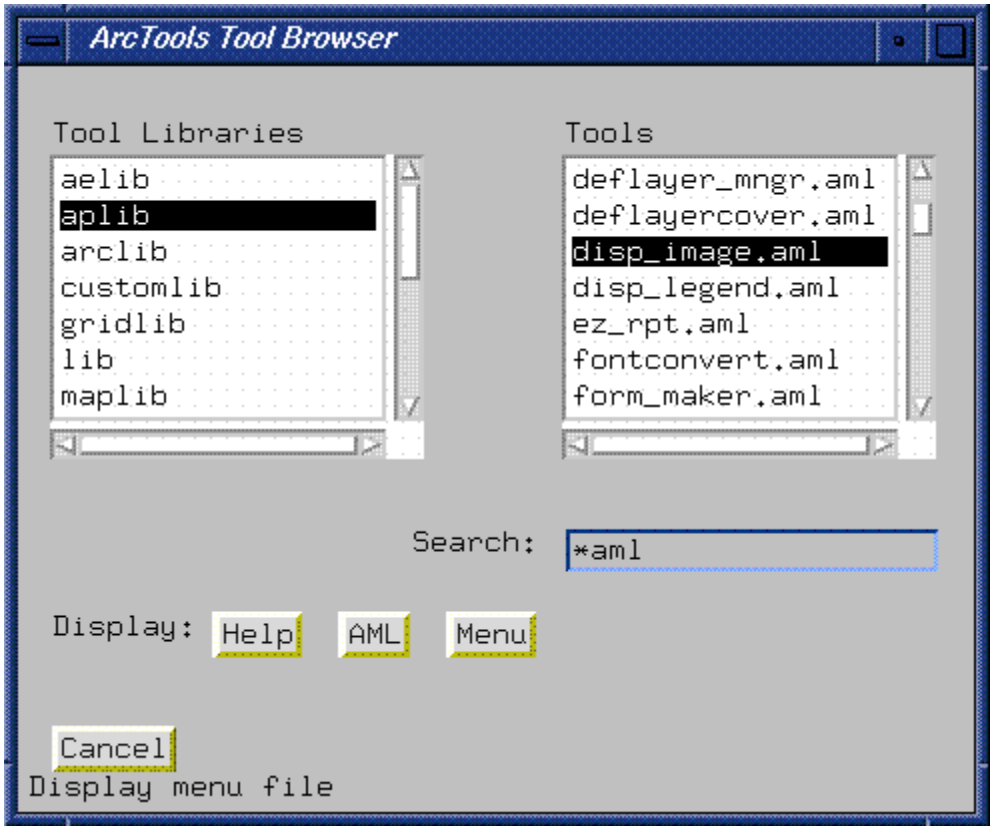
Table 4 : General ArcTools Utility Tools

Type of Tools	Description
Messaging tools	Messaging tools provide popup form menus containing programmer specified text. These tools are used to inform the user about the current status, or to get input from the user to decide what subsequent actions will be taken. These tools include: • msconfirm • msinform • msrespons • msworking
Data managers	The data managers are simple file browsers that also allow the user to describe, copy, move, delete, and sometimes display the data. These include: • cover_mngr • eventsource_mngr • grid_mngr • infofile_mngr • relate_mngr • tin_mngr • workspace_mngr

Type of Tools	Description
File browsers	File browsers allow the user to browse the system for a particular type of file, and then select the needed file. The selected file is returned to the calling program via a programmer specified variable. The list of file browsers includes: • getaelayer • getannocover • getconfig • getcover • getdirectory • geteventsource • getfile • getgrass • getgrid • getimage • getitem • getlayer • getlayout • getlib • getlibrary • getmany • getregioncover • getrelate • getroutecover • getstack • getsymset • getsymsetae • gettable • gettemplate • gettin • getview • getworkspace

To help support users with browsing ArcTools a *tool browser* tool is provided within ArcTools. This tool traverses the ArcTools directory organization and provides a mechanism for AML programmers to review tools. As a general guideline you should never modify the source tool (within ArcTools). Always copy the tool files into your application directory before customizing it.

Figure 6 : The ArcTools Tool Browser provides a mechanism for programmers to review available tools. Tools are organized by functional libraries (subdirectories).



A concern often raised by application developers is the confusing nature of many of the ArcTools menus. Most ArcTools menus reflect a *functional capability* design rather than a *procedural* design. This is to be expected considering their source as core tools. In most cases ArcTools toolsets can become procedurally useful for specific sites by simply modify the toolset menu. This may include re-organizing the menu widgets, or it may involve adding new menus to the tool to reflect procedural implementation of toolset routines and tasks. In either case, the toolset macros typically contain most of the functionality required for use in custom applications.

More advanced application development environments, like RAPiD*AML, automatically include the ArcTools paths in application paths. This provides the technical foundation for utilizing ArcTools toolsets directly in custom applications, or modifying toolset macros or menus locally. This approach allows the developer to utilize toolsets as is appropriate without the overhead of the ArcTools interface. By simply browsing the toolsets, copying the toolset menu (or macro) to the local application, and customizing as desired, a custom toolset can quickly be built. This is a highly productive technique for building applications and leveraging the wealth of capabilities within ArcTools, without the restriction of the interface.

2.5.4 Key AML Functions and Directives

A wide range of AML directives and functions exist within AML. However, certain directives and functions are especially effective in building custom applications. The following table reviews key AML constructs.

Directive or Function	Description and Usage
[token]	This function allows the manipulation of variable lists comprised of strings. This function is powerful for managing custom variable lists such as changing the order of lines in the list, removing lines, replacing lines, etc. It is the key function for manipulating variable lists that populate scroll lists for a menu tool.
[list*]	The LIST* functions allow you to generate lists of data files etc. and populate these into a global variable for ASCII file. These functions are often used prior to processing data in a looping manner.
[show]	The SHOW functions are perhaps the most powerful suite of functions within AML.. They allow you to obtain virtually any setting or property within ARC/INFO and AML.. Specific show functions are only available within subsystems.
&DATA	The &DATA block allows you to execute a discrete ARC/INFO session within an AML program. DATA blocks are often used to issue a series of INFO or TABLES commands.
&RUN	This directive is the primary mechanism to run an AML program.
&CALL	This is the method to execute a routine within an AML program without leaving the program thread. CALL is used to nest routine calls within routines.
&THREAD	This directive is the key for all thread management within AML..
&ROUTINE	Identifies the beginning of a routine block.
&RETURN	Returns control back to the calling thread. Also required to end a routine block.
&ARGS	Primary directive for accepting arguments in an AML program.
&DO	Several different types of DO blocks exist for processing data in a loop, and supporting actions for IF…ELSE blocks.
[AFTER] **[BEFORE]** **[EXTRACT]**	Key functions for extracting subsets of string values with global variables. Usually used to parse input arguments.
[NULL [VALUE]]	Nested use of functions to check for existence, and null value, of a variable. This nesting is often used at the beginning of a routine to check for variables and/or arguments.
&IF … &ELSE	Conventional IF … ELSE block logic for branching code.

Directive or Function	Description and Usage
&SYS … &	Allows the execution of a system command (UNIX or NT) from within an AML program. The "&" end statement can be used to spawn the command as a separate process. &SYS is also used to start external programs in separate windows using the "xterm" (UNIX) or "cmd" (NT) system commands.

2.6 GUI Development

Unlike other conventional scripting languages AML also provides a suite of graphic user interface menu creation commands and functions. These menu capabilities allow you to create menus based on either the SUN Open Look or X-11 Windows standards for UNIX, or Windows NT. Since AML is platform independent programs that utilize menu functions will operate on any platform, yet will appear different on SUN versus X-11 versus Windows NT. This section reviews the menu and interface options available within AML with a focus on methodologies for employing menus in custom applications.

2.6.1 Menu Options

In addition to providing programming tools, AML provides tools for creating a menu and *form* based interfaces for your toolsets and applications. Menus can help new users get started with ARC/INFO right away, minimize errors by providing only valid choices, and reduce the number of commands you have to give by grouping actions into one selection. Menus provide the primary interface for toolsets and applications.

Three different types of menus are available through AML; screen, key, and paper menus. Each group has its own look and method of selection. There are four types of screen menus, referred to by how they look, or how they present and reveal choices: **matrix, pulldown, sidebar, and forms.**

A *form* menu is perhaps the most dynamic of the screen menus. Not only can each form have a completely individual look, but the display of the form can change as the user interacts with it.

While menus are used in a GUI presentation, they are actually ASCII text files created with a text editor using AML directives and functions. In this regards, menu programs are no different from macro programs. Menu files utilize a ".menu" file extension while AML files utilize a ".aml" extension.

Forms are text files but should be created with the ARC/INFO FormEdit program to ensure proper control positioning.[7] The format of the text in the file is important because it determines which choices display and which actions are taken when a selection is made. The first line of a file identifies the type. This is a number from one to eight represents the following menu types

[7] FormEdit is a GUI editor that allows you to create form menus by dragging widgets into a menu canvas. By double clicking on a menu widget you can then define the properties of the widget. FormEdit is a quick and easy way to create form menus.

supported within AML.. The primary menus used for application interfaces are identified by **bold** text.

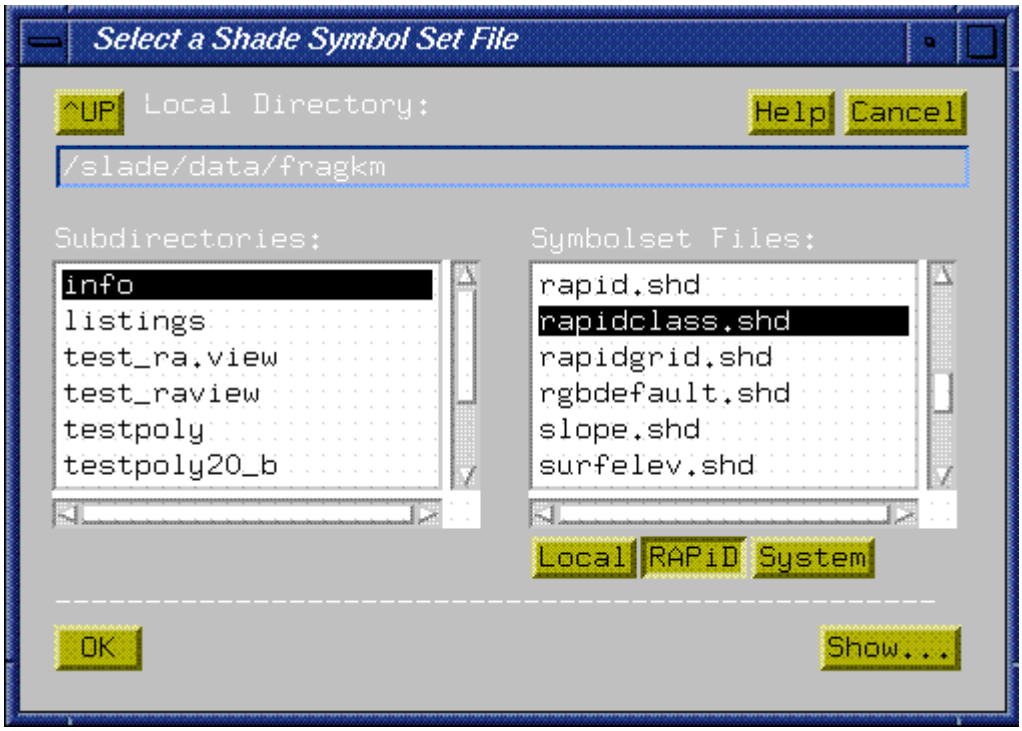
1 = Pulldown menu
2 = Sidebar menu
3 = Matrix menu
4 = Key menu
5 = Tablet menu
6 = Digitizer menu
7 = Form
8 = Pulldown with pullright

The following example presents the “getsymset” sample tool menu in both ASCII file format, e.g. the AML menu program, and the GUI menu that is displayed.

Figure 7: Sample menu tool in both source ASCII format and GUI form.

```
7 getsymset.menu
/* $Id: getsymset.menu,v 2.0 1996/01/24 23:46:36 markz Exp $
/* ------------------------------------------------------------------------
/*          Environmental Systems Research Institute
/* ------------------------------------------------------------------------
/*    Menu: GETSYMSET.MENU
/* Purpose: Display subdirectories and symbolset files in the current
/*          directory.
/*          Allow user to move up one directory, or specify any
/*          directory in an input field.  ARC/INFO symbolsets can also be
/*          specified.  Returns the name of the selected symbolset
/*          file.
/* ------------------------------------------------------------------------
/* Globals:
/* ------------------------------------------------------------------------
/*   Calls:
/* ------------------------------------------------------------------------
/*   Notes: Since .getsymset$symsetdir needs to be set, this menu does not
/*          call get_routines.aml directly.
/* ------------------------------------------------------------------------
/* History: Matt McGrath     - 02/25/92 - Modified the GETMANY tool.
/*          Mark D Zollinger - 11/01/94 - VAXinate.  Centralize routines.
/*          Ian DeMerchant   - 11/02/94 - Added help button.
/* ========================================================================
/* ^Symbol type:
/* %wild
/* %wild CHOICE symtype PAIRS ~
/*    HELP 'Select symbol set type' ~
/*    RETURN '&if [LOCASE %symtype%] = line &then; &set wildcard = *.lin; ~
/*            &if [LOCASE %symtype%] = marker &then; &set wildcard = *.mrk; ~
/*            &if [LOCASE %symtype%] = shade &then; &set wildcard = *.shd; ~
/*            &if [LOCASE %symtype%] = text &then; &set wildcard = *.txt; ~
/*            &set cursymset' ~
/*    Marker marker Line line Shade shade Text text
```

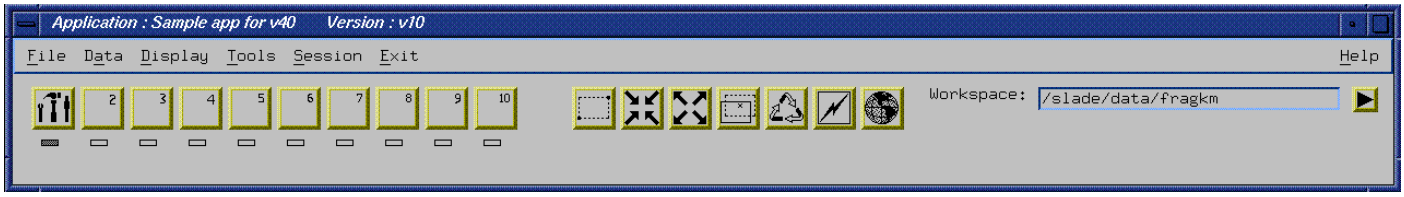
```
/* %sys BUTTON HELP 'List ARC/INFO default symbol sets' ~
/*    'System symbol sets' &set .getsymset$symsetdir = $ARCHOME/symbols/
 %up  Local Directory:                %b4  %cancel
 %curdir

 Subdirectories:          ^Symbolset Files:
 %subdir                %fil




                      %sys
 ------------------------------------------------
 %ok                                  %ok_1
%up BUTTON KEEP RETURN ~
  HELP 'Move up one directory' ~
  '^UP' &run getsymset up
%b4 BUTTON KEEP RETURN ~
  HELP 'Display help about this tool' ~
  'Help' &run getsymset help
%cancel BUTTON CANCEL ~
  HELP 'Quit from this menu' ~
  'Cancel' &run getsymset cancel
%curdir INPUT .GETSYMSET$CURDIR 50 TYPEIN YES SCROLL NO ~
  HELP 'Directory path' ~
  RETURN '&run getsymset curdir' ~
  CHARACTER
%subdir INPUT .GETSYMSET$SUBDIR 25 TYPEIN NO SCROLL YES ROWS 6 ~
  RETURN '&run getsymset subdir' ~
  FILE ~
  %.getsymset$availdirs% -DIRECTORY
%fil INPUT .GETSYMSET$FILE 25 TYPEIN NO SCROLL YES ROWS 6 ~
  REQUIRED ~
  RETURN '&run getsymset setsymset' ~
  FILE ~
  %.getsymset$symwild% -FILE
%sys CHOICE .GETSYMSET$SYSTEMFILES SINGLE ~
  HELP 'List Local, RAPiD or ARC/INFO symbol sets' ~
  RETURN '&run getsymset toggle_sys' ~
  'Local' 'RAPiD' 'System'
%ok BUTTON ~
  HELP 'Select symbol set and quit menu' ~
  ' OK ' &run getsymset ok
%ok_1 BUTTON ~
  HELP 'Show selected symbol set' ~
  'Show...' &run getsymset show
%FORMOPT SETVARIABLES IMMEDIATE MESSAGEVARIABLE .getsymset$msg
```



The form menu is the most dynamic and commonly used within AML. Virtually all ArcTools toolsets utilize a form menu as the primary interface. Recent enhancements with ARC/INFO have added an extended pulldown menu type (type 8 menu) which allows multiple levels of cascading pulldown bars. Previously the pulldown menu (type 1) was limited to a single pulldown bar. This provides much greater flexibility in creating effective pulldown menus with multiple cascading submenus.

In addition, capabilities now exist to embed a cascading pulldown menu (type 8) into a form menu (type 7). This provides the application developer the ability to combine different interface approaches. Menu interfaces can now be created reflecting a more intuitive look-and-feel with the ability to mimic the widely used, and standard, Windows application interface that combines pulldown menus with toolbars. The following figure presents an integrated type 7 and 8 menu used as the standard application main menu within RAPiD*AML.. This menu is utilized as the primary main menu for the *application desktop* for RAPiD*AML based applications. The ability to create embedded menus provides the functionality required to standardize menu interfaces for all applications on a site. Previously this was cumbersome with AML menuing capabilities.

Figure 8: An example form menu with an embedded type 8 menu. Using such a nested main menu provides application developers with options to build application interfaces without using multiple menus. Application capabilities can now be categorized using standard Windows conventions.



2.6.2 Creating Menus

Formedit (type 7 menus)

FormEdit is a graphical editing tool, executed as an Arc subsystem command (UNIX), or as a separate standalone program in Windows NT, designed to create AML forms. FormEdit combines the graphical component of form layout and design with the descriptive or syntactical component that defines the operation of the individual objects, or controls, in the form. FormEdit creates new forms or modifies existing ones. FormEdit helps new users create forms quickly, without having to learn form syntax, e.g. AML directives and functions for &MENU definitions. For experienced users, FormEdit provides the capability to create a form with full functionality, but simplifies the creation and editing process, allowing the user to concentrate on form design rather than on syntax. In essence, FormEdit is a GUI to create GUI's.

FormEdit is used typically by the applications programmer, to create the forms for toolsets and applications. FormEdit is used to define how the form operates in the ARC environment much like you use a text editor to create an AML program. While FormEdit helps you to build the forms needed for an application, it is not itself an application builder. FormEdit is one of many tools that you'll use when building an application for the user who will use the application.

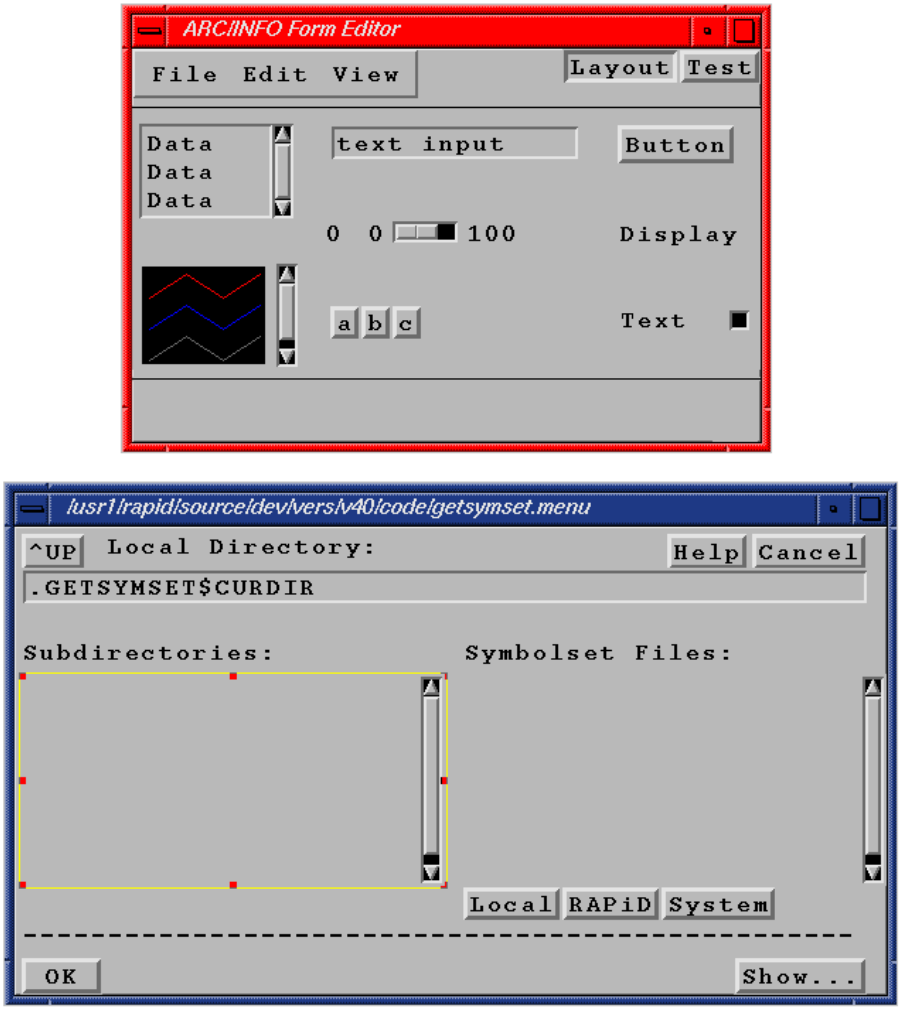
FormEdit should be considered a programming aid to be used in the application development process. It's used should be tightly tied to the development of application code, much like text editors are used for AML macro development.

AML forms provide a broad range of user interface tools that allows you to create effective user interfaces to ARC/INFO processes. Forms are perhaps the most versatile of the screen dialogs. They present graphic controls that when manipulated, dynamically define the action to be performed. The dynamic nature of forms is what sets them apart from the other types of dialogs.

Like all AML dialogs, forms provide structure to complex processes or operations. A carefully designed form helps the user by grouping individual operations and arranging them in a manner that is logical and consistent with the action that needs to be performed. The 'free form' layout of controls within the form facilitates the logical organization of operations and imparts an individual, but recognizable, look to every form.

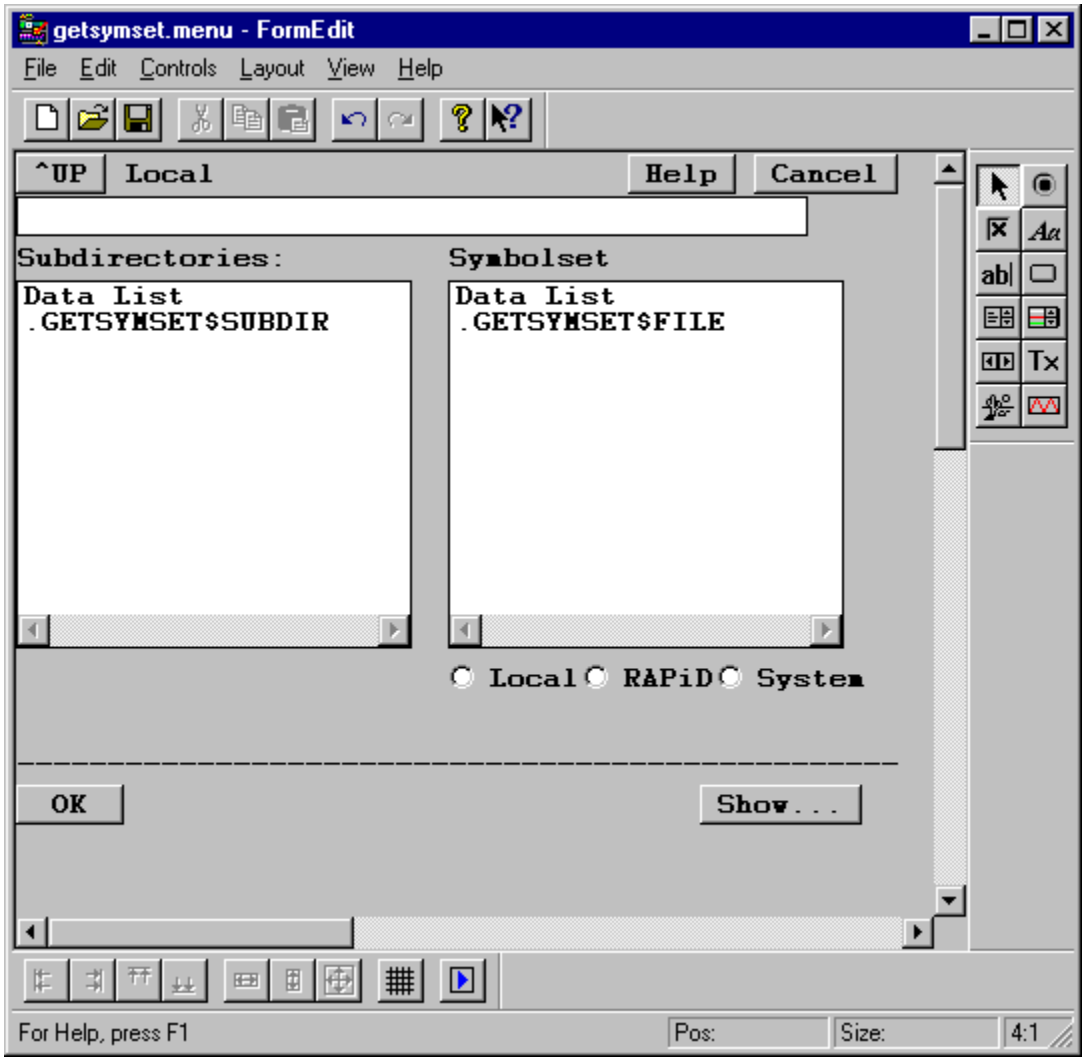
FormEdit is a critical component in the arsenal of the application developer for creating robust toolsets. The following figure illustrates a FormEdit session. The FormEdit command menu allows you to manage the ".menu" file that is open using pulldown menu options, while the *control* form is the source for menu *widgets*. A menu widget is a menu control that is presented in a specific operational look-and-feel. FormEdit supports 9 different menu widgets/controls with the DISPLAY widget extendable to display icons and graphic symbols. Users build a menu by *dragging* widgets into the *menu canvas.* After a widget is placed on the canvas you can set properties by simply clicking on the widget. This is a graphical drag-and-click layout approach to building menus.

Figure 9: A typical UNIX FormEdit session. The top window is the FormEdit widget control menu, while the lower window is the custom menu canvas for the menu being edited. The example toolset "getsymset" is shown.



The recently released Windows NT version of ARC/INFO provides a more enhanced FormEdit program that is consistent with Windows GUI editing capability. With UNIX, FormEdit operates as an ARC/INFO command. With Windows NT, FormEdit is a standalone program that allows the user to edit menus without using an ARC/INFO license. This is advantageous.

Figure 10: The Windows NT version of FormEdit provides a more intuitive drag-and-click interface as a standalone program. A standard Windows program interface is utilized with menu controls available as toolbars.

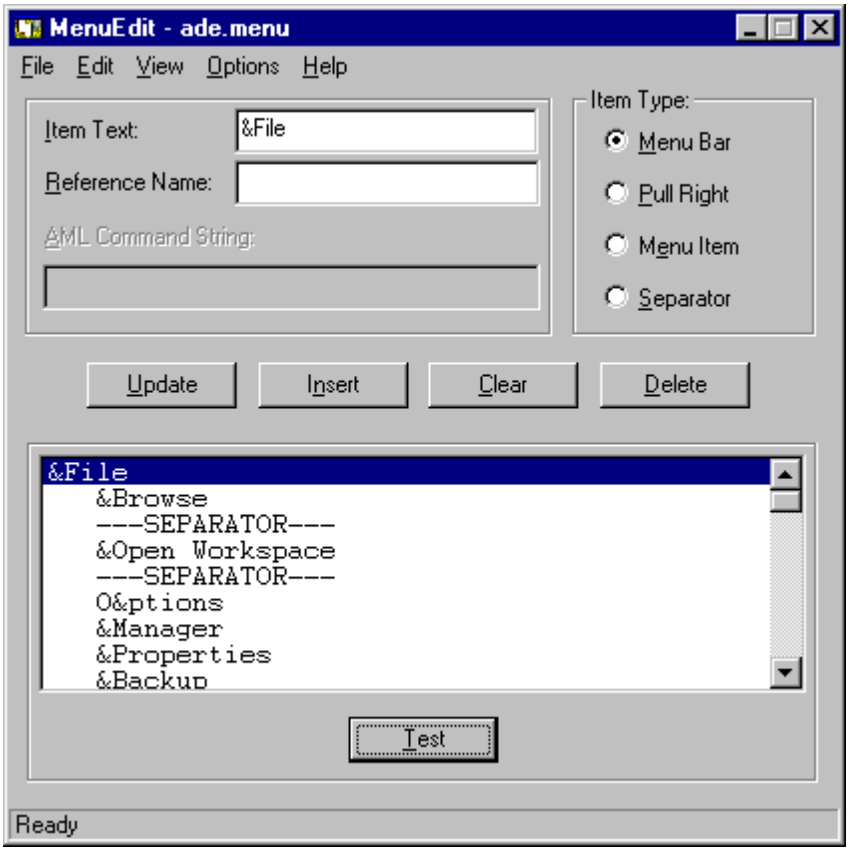


Menuedit (type 8 menus)

Type 8 pulldown menus represent an enhanced menu option with ARC/INFO. These menus allow the application developer to mimic Windows cascading pulldown menus. A new programming tool, MenuEdit, is provided with ARC/INFO 7.1.1+ to help the application developer in building type 8 menus. MenuEdit enhances the GUI development options within ARC/INFO to accommodate complex menus. In addition, type 8 menus can be embedded within type 7 form menus. The Preferences option within FormEdit contains a direct link to MenuEdit to accommodate embedded menu editing.

Figure 11: The MenuEdit program is a GUI interface for creating type 8 menus. It

operates as a standalone program, or from within FormEdit to supported embedded menus in type 7 form menus.



2.6.3 Menu Standards

Individual toolset menu standards are promoted within the ArcTools software in order to ensure a consistent tool look-and-feel, as well as support the toolset programming approach. This section reviews guidelines for building AML toolset menus, and describes the fundamental requirement for orderly thread management within AML.

Menus and Threads

A robust application allows the user to interact with more than one menu at a time. With ARC/INFO, threads are the mechanism used to *carry* input to the AML processor, and to allow multiple menus to interact with the AML processor. AML threads are the mechanism used to display and manage multiple menus in an application. With threads, tools can have submenus and an AML application can manage the input through these threads. Threads accept input from three sources: AML programs, the command dialogue keyboard, and AML menus. ARC/INFO is automatically start as a single thread typically named “thread001”. All input uses this thread until such times as other threads are created with the &THREAD directive in AML. Threads provide for :

- More than one input source for an application;

- Endorse the logical and consistent grouping of operations into toolsets;
- A visually pleasing application interface comprised of multiple menus. Menus can be utilized when necessary by the user through the use of new threads; and
- Multiple menus can be displayed simultaneously.

The &MENU directive will start an AML menu and maintain control of the active input thread until such times as the menu is removed. With this approach only a single thread, and menu, can be active. No other input source is available. However, the use of the &THREAD directive in combination with the &MENU directive allows the application developer to create multiple threads for multiple menus.

Threads are the primary building blocks for AML applications using a Graphical User Interface (GUI).

The design of the AML toolset is inherently based on the use of multiple threads. The INIT routine (and subsequent MENU routine in the toolset template presented in this document) create a thread for each new menu used by the toolset. In most cases this is a single menu. By creating a new thread for the menu, control can be returned to the calling program. Menus appears as independent threads that can operate, and must be managed, separately. The toolset must explicitly manage the thread by creating it for menu startup, and removing it upon exiting the tool and menu.

The following example illustrates the standard approach for creating a thread for a new menu with AML toolsets.

AML Statement	Description of Action
&if not [show &thread &exists myamltool] &then &do	Check to ensure that the thread does not already exist
&thread &create myamltool ~	Create a new thread …
&menu myamltool ~	Start the menu in the thread with …
&pinaction '&run myamltool myamltool quit;' ~	A pinaction action define, and
&pos [unquote %position%] ~	A defined screen position, and
&str %stripe%;	A menu stripe.
&end	End of &DO block

A Thread Manager

Since thread management is such a key component to building useable toolsets and applications, it is desirable to have a *thread manager* tool available. With RAPiD*AML we have created a generic thread manager toolset that is used by all other toolsets to :

- Create threads;
- Synchronize threads; and
- Delete threads.

The use of a generic thread manager removes the thread management code that would be duplicated in each toolset, and provides a simpler and consistent approach for the AML programmer to manage threads in any application. To illustrate the importance of this approach the following table presents the standard thread creation code used in the ArcTools INIT routine.

```
/*-----------
&routine INIT   /*  {'position'} {'stripe'} {MODELESS | MODAL}
/*-----------
/* Initialize tool interface
/*
&set position = [extract 1 [unquote %arglist%]]
&set stripe   = [extract 2 [unquote %arglist%]]
&set modality = [extract 3 [unquote %arglist%]]
&if [null %position%] or %position%_ = #_ &then
  &set position = &cc &screen &cc
&if [null %stripe%] or %stripe%_ = #_ &then
  &set stripe = Put Your Menu Stripe Here
&if [null %modality%] or %modality%_ = #_ &then
  &set mode =
&else
  &if [translate %modality%] = MODAL &then
    &set mode = &modal
  &else
    &set mode =
/*
/* Issue thread delete self if thread depth = 2 and input is tty
&if [show &thread &depth] = 2 and [extract 1 [show &thread &stack]] = tty &then
  &set launch = &thread &delete &self
&else
  &set launch
/*
&if [show &thread &exists tool$tmplt_util] &then
   &thread &delete tool$tmplt_util
&thread &create tool$tmplt_util %mode% ~
  &menu tmplt_util ~
  &position [unquote %position%] ~
  &stripe [quote [unquote %stripe%]] ~
  &pinaction '&run tmplt_util exit'
%launch%
/*
&return
```

Using a standard thread manager tool the menu initialization routine can be simplified to the setting of global variables. In addition, the thread manager can be enhanced, as in this case, to provided Windows-like minimize capabilities, and explicit menu sizing. This modified approach is illustrated below using a standard thread manager tool “rathread”.

```
/*-----------
&routine MENU
/*-----------
/*    Accept as 1st two args the position of the menu
/*    and the stripe (title text) of the menu
/*
&set position = [EXTRACT 1 [UNQUOTE %arglist%]]
&set stripe   = [EXTRACT 2 [UNQUOTE %arglist%]]
&if [NULL %position%] or %position%_ = #_ &then
 &set .ra$thr$pos [quote &ul &thread menubar &ll];
&else
 &set .ra$thr$pos %position%
/*
&if [NULL %stripe%] or %stripe%_ = #_ &then
 &sv .ra$thr$str template
&else
 &set .ra$thr$str [quote [unquote %stripe%]]
/*
/*    minimize stripe var (default is first 10 characters)
/*    CHANGE THIS TO YOUR CHOICE (< 10 CHARACTERS)
&sv .ra$template$stripe_min [substr [quote %.ra$thr$str%] 1 10];
/*
/* set a specific size for the menu, or leave null to default to menu extent
&sv .ra$thr$size ;
/*
/*    Set thread and menu specifications here first
&sv .ra$thr$menu template              /* name of menu file (no extension)
&sv .ra$thr$pin '&run template quit'   /* routine for thread cleanup quit
/*    set the next global to 'modal' if you want the menu modal
&sv .ra$thr$modal ;
/*    Spark the main menu using the RAPiD AML thread manager tool
&run rathread launch
&return
```

Modality

In most scenarios toolsets are intended to operate as free form standalone tools. In other words the user is free to start any menu, or select from any menu at any time, since all menus are independent threads. However, in some cases it is appropriate to ensure that a user finishes a particular task with a toolset before undertaking any other action. This is referred to as a *modal* menu. With modal menus all input is forced to the current (named) thread. When the modal thread is started no other thread can be used until the modal thread is deleted. This forces the user to complete actions with the modal thread before continuing on.

Modal threads are typically used to confirm user input, such as action confirmation. The standard “msconfirm” message tool available with ArcTools is created modal. For example, the following menu is created as a modal thread to ensure that the user confirms the action before processing is continued. This action is confirming the removal of an application using the

RAPiD*AML ADM.

Figure 12: Example modal thread.

Checking the return status from the “Yes“ and/or “No“ button allows the toolset to proceed accordingly. If “Yes“ is returned the tool will continue processing by calling a REMOVE_APPLICATION routine. If “No” is returned the tool with call the EXIT routine. Removing the modal thread returns control to the calling thread, which is typically non-modal.

Thread management is a critical component of a robust application. Wherever possible AML developers should attempt to standardize the use, and techniques , for managing thread within an application.

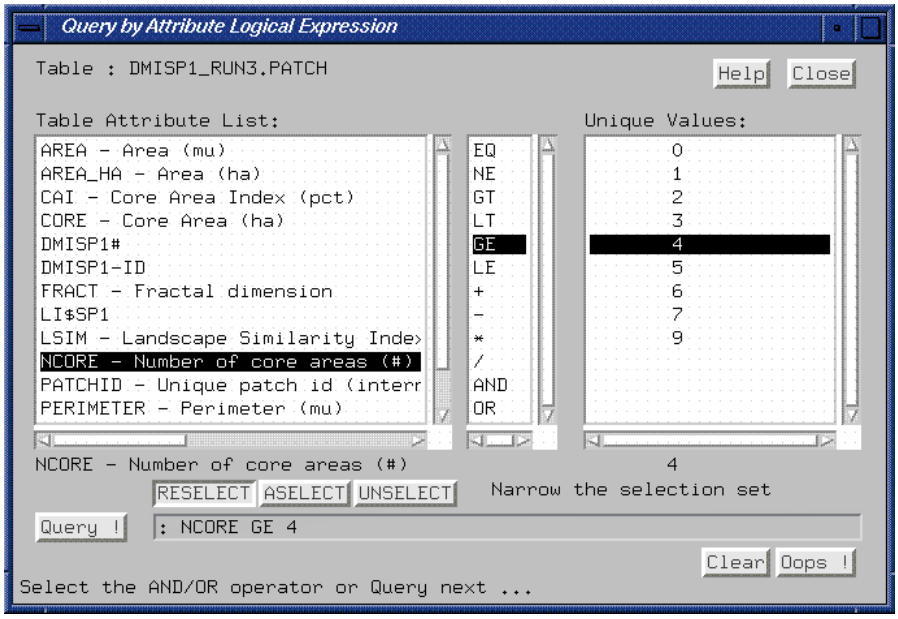
Scroll Lists

Scroll lists are also a key component of most toolsets. Scroll list map controls (widgets) are commonly used to list available data, e.g. coverages, items, item values, INFO files, etc., in a menu for selection by the user. Other map controls undertake actions using the values selected from the scroll lists. AML menus support a variety of ways to populate values into scroll lists. Most scroll lists utilize direct listings of data files using standard LIST* function calls. The scroll list map control is populated directly.

However, in some scenarios it is desirable to have more understandable lists available. When this is required ASCII files can be used to populate scroll lists. This approach is used when the application developer wishes to *hide* the explicit data naming convention from the user. This also allows the developer to provide more understandable, English or domain specific, descriptions in a scroll list.

The example illustrated below presents a menu where an ASCII file is used in the left scroll list to present the user with a more understandable list of attributes (items). This menu is the primary attribute query selection menu from a landscape structure analysis application. The left scroll lists represents calculated landscape metrics for a polygon coverage. Rather than present these to the user as a list of item names, it was desirable to present the list of items as a combination of item name, and metric description.

Figure 13: A typical toolset that uses scroll lists populated by ASCII files, and global variables, to present selection options to the user. This tool allows the user to build a logical attribute expression query by selecting coverage attributes from a scroll list populated by an ASCII file.



While it is straightforward to create an ASCII file with descriptive text for the item, the internal routines of the tool still require the explicit item name for populating the unique value scroll list (far right scroll list), and issuing the query to the database. To accomplish this, a technique is used with the ASCII file that bundles the description with an item definition. This approach also accommodates the use of related tables. An ASCII lookup table is created as illustrated below and the FILE option is used to populate the map control. The contents of the ASCII file are shown below for the menu show above.

AREA - Area (mu)	::PATCHREL//AREA,4,12,F
AREA_HA - Area (ha)	::PATCHREL//AREA_HA,12,12,N
CAI - Core Area Index (pct)	::PATCHREL//CAI,12,12,N
CORE - Core Area (ha)	::PATCHREL//CORE,12,12,N
DMISP1#	::PATCHREL//DMISP1#,4,5,B
DMISP1-ID	::PATCHREL//DMISP1-ID,4,5,B
FRACT - Fractal dimension	::PATCHREL//FRACT,6,6,N
LI$SP1	::PATCHREL//LI$SP1,3,3,I
LSIM - Landscape Similarity Index (pct)	::PATCHREL//LSIM,8,8,N
NCORE - Number of core areas (#)	::PATCHREL//NCORE,8,8,I
PATCHID - Unique patch id (internal #)	::PATCHREL//PATCHID,8,8,I
PERIMETER - Perimeter (mu)	::PATCHREL//PERIMETER,4,12,F
SHAPEI - Shape index	::PATCHREL//SHAPEI,6,6,N
SP1	::PATCHREL//SP1,2,2,C

Since each line of the ASCII file is much longer than the width of the scroll list only the textual description is shown. The entire line can be viewed by simply using the arrow control on the scroll list, but by default it is left justified and the item info is hidden. This trick is used throughout the ArcTools objects definitions. It is an effective, yet simple, technique to enhance the presentation of scroll lists.

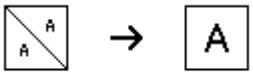
With this technique you must have a routine in your toolset that *parses* the item name, and/or definition, into variables from the line of the ASCII file that is selected from the scroll list. Note that the "::" characters are used to separate the descriptive text from the item information. Typically the [EXTRACT], [BEFORE], [AFTER] and [TRIM] functions are used to undertake the parsing actions.

Widgets Organization

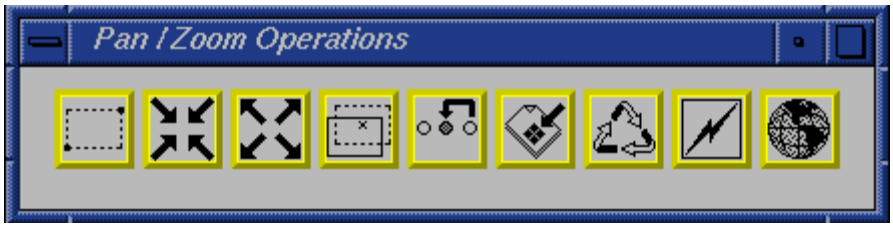
The ArcTools coding standards promote a standard look-and-feel organization of widgets (controls) on toolset menus. By convention APPLY (OK), QUIT and/or CANCEL, and HELP buttons are required. Additional buttons are required as dictated by the purpose of the specific toolset. The template promoted in this workbook also accommodates a NEXT and BACK button for optional Wizard tools. However, typically these buttons are not required.

Icons and Bitmaps

Icons and bitmaps are useful controls to use in AML menus to provide a pictorial illustration of the action for the tool, or as buttons in a toolbar like menu. The following example illustrates how bitmaps can be used as DISPLAY controls in a menu to represent the tool action. This example is from the ArcTools Dissolve tool.



Icons are also useful for BUTTON controls. These are ideal for creating point-and-click toolbars. The following example uses icon files assigned to menu BUTTONs for a generic pan and zoom menu *toolbar*.



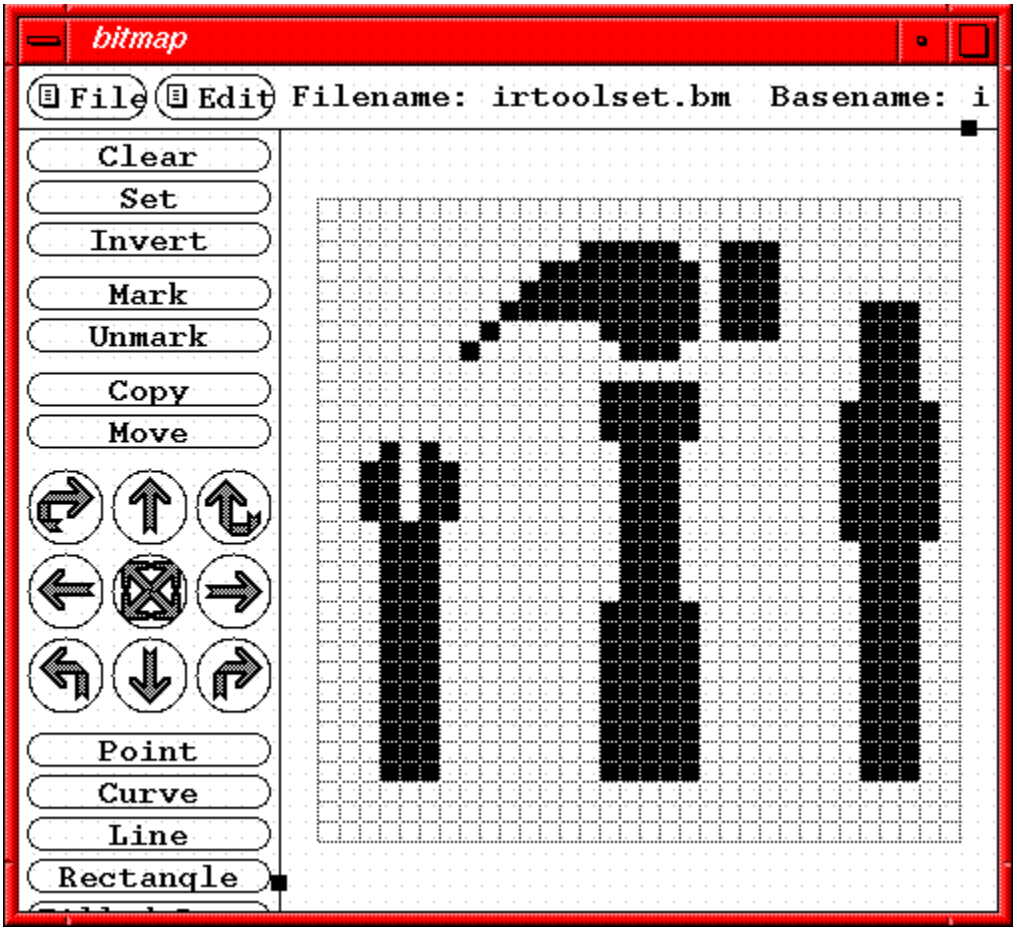
ARC/INFO AML supports two different type of icon files;

- SUN ".icon" files created with the "icondedit" program; and
- X-11 Windows ".bm" files created with the "bitmap" program.

Both these file formats and programs are UNIX programs. Icons and bitmaps can be used interchangeably by AML menus. However, these files can only be edited by their respective UNIX operating system programs. In other words, "iconedit" does not edit ".bm" files are vice versa. Some public domain and commercial tools do exist that allow you to convert, and/or edit these files.

SUN icons and X-11 bitmaps are both supported by AML menus on the Windows NT platform. However, no capability is provided within ARC/INFO to create or edit ".icon" or ".bm" format files on Windows NT. Windows Bitmap ".bmp" file are also not supported in ARC/INFO AML menus at this time.

Figure 14: This figure illustrates the X-11 Windows "bitmap" program used to create and edit bitmaps on UNIX platforms.



2.7 Linking ARC/INFO to External Programs

Linking ARC/INFO AML to external programs is often a requirement for many applications. Most

database software (DBMS) can be linked to the INFO database using the ARC/INFO Database Integrator (DBI) component, or via custom programming with the DBI Cookbook library. However, quite often there is a need to link other external software, such as statistical programs, S-PLUS, etc., or programs coded in FORTRAN or C. Techniques do exist using AML tools to provide linkages between external programs and ARC/INFO. This section provides an overview of approaches that can be used to satisfy most external linkage requirements.

2.7.1 System Calls - &SYSTEM

The &SYSTEM directive is explicitly used to execute system calls from within AML programs. &SYSTEM allows you to execute operating systems commands in two modes:

- As modal commands where the system command must finish before the AML program will continue; and
- As an independent program that will immediately return control to the AML program before the system command processing is finished.

Either technique is useful for running external software, especially custom programs like FORTRAN or C executables, from within AML. Often if an independent *process* is desired the &SYSTEM directive will be used in conjunction with and "**xterm**" command (UNIX) or "**start**" command (NT) to initialize a separate window for the program to run in. This is required for FORTRAN or C executables. The standard UNIX qualifier "**&**" can be used at the end of the &SYSTEM line to spawn the command as a separate process returning control immediately to ARC/INFO. The following examples illustrate the use of &SYSTEM for spawning external programs. Use of the "**xterm**" command illustrates a UNIX example. Use of the "**start**" command illustrates a NT example.

Example &SYSTEM Statements	Description of Usage
&system	Starts a operating system command dialog (modal). Type "exit" to return to AML or ARC/INFO command line.
&system ls -al	Execute a system command and when finished return to AML or ARC/INFO
&system xterm –e $ARCHOME/bin/arc &	Start an xterm window and execute Arc as an independent process in the xterm window. Returns control immediately to ARC/INFO.
&system xterm –e arc	Start an xterm window and start in the xterm window. This is modal, so control is not returned to ARC/INFO until the xterm is exited.
&system xterm –e myCprogram	Start an xterm and run the C executable program "myCprogram". Without the terminating character

Example &SYSTEM Statements	Description of Usage
	"**&**" control will be returned to ARC/INFO once the program is finished (UNIX).
&system start myCprogram	Start a NT command window and run the C executable program "myCprogram". Control is returned immediately to ARC/INFO. No termination character is needed using "start" under NT.

2.7.2 IAC – Inter-Application Communication

A elegant approach for linking ARC/INFO to external programs is by using the IAC functions available within AML. IAC allows the developer to create external, standalone, independent ARC/INFO *server* processes which can be used by different *client* processes on the system. An ARC/INFO server is an independent process that can operate in the background. ARC/INFO IAC servers can operate in any subsystem, e.g. ArcPlot, Arc, GRID, etc. Other ARC/INFO processes can *connect* to the server, and send command *requests* to the ARC/INFO server with control immediately returning to the local ARC/INFO process.

IAC techniques are an excellent way of offloading ARC/INFO processing to background server processes running in memory while maintaining an active ARC/INFO session. In addition, other programs, such as UNIX and C programs can also *connect* to an ARC/INFO *server* and act as *client* processes sending *requests* to the server. With this approach an ARC/INFO IAC server can act as a *GIS processing engine* facilitating data conversion, analysis, and even piping ArcPlot displays to different devices on a network.

The following table illustrates the basic AML procedural steps for using IAC functions.

AML IAC Statements	Description
&sv status = [IACOPEN connect.file]	Create a server and store process information in a connect file.
&sv iacid = [IACCONNECT connect.file]	Connect to the open server using the connect file information and store the server id in the "iacid" variable. You can now send multiple requests.
&sv status3 = [IACREQUEST %iacid% 1 &amlpath / tools]	Now that you have connected send a request to the server to set the

	&AMLPATH to my tools directory.
&sv status4 = [IACREQUEST %iacid% 1 &run myamltool create_map]	Send another request to run the “myamltool” tool “create_map” routine.
&sv status5 – [IACDISCONNECT connect.file]	Finished with commands so disconnect from the server, leaving it for future use.

3.0 Open Development Environment – ODE

The ARC/INFO ODE is a new method by which developers can access ARC/INFO functionality. Its purpose is to deliver the complete functionality of ARC/INFO through a *object oriented* interface on both UNIX and Windows NT platforms. This allows developers to add ARC/INFO functionality to applications using the development environment of their choice instead of being restricted to AML. This interface, however, is not to low-level ARC/INFO commands but rather to the command line. The ODE approach is clearly focused on utilizing the capabilities available within different programming environments on the Windows platform.

The use of the Open Development Environment (ODE) involves a paradigm shift from the *toolset* application design methodology. With this new environment comes a new, and as yet undefined methodology for developing your applications. The standard object-oriented programming approach with the creation of discrete objects, which have properties and methods, is likely the best methodology to follow. However, to build robust objects, which will be easily re-useable requires significant time and expertise. While it is possible to develop applications that are not composed of atomic objects and totally re-useable code, it is significantly more difficult than using AML to do the same tasks. The ideal situation is to take the time to build robust objects, and over the course of a number of applications, build up a library of objects. At this point you can then start to re-coup your investment in the building of those robust objects by rapidly building applications from them.

3.1 Role of AML functions and code

At this point, ODE application development involves a combination of object-oriented code in conjunction with regular ARC/INFO AML commands and functions. There is no clear distinction as to when either method should be used, except to check the object diagrams. If your function can not be found there, then you must issue the ARC/INFO command and parse the response. It is also possible to run standard AML's, with a few caveats:

1) AMLs can not change sub-systems, this includes the use of &DATA <ARC|INFO>, &SYSTEM Arc, or issuing an Arc command in another subsystem (e.g. Arcplot: arc build mycov arcs).
2) AMLs are also restricted in that they cannot pop form menus, or use AML functions or toolsets that require an interactive response from the user, e.g. [RESPONSE] or "msresponse" toolset.

Below is an example of Visual Basic code which runs a sample routine "test" in the AML program "odetest.AML".

```
Private Sub Command1_Click()
    Dim Severity As Long
    Dim Result As New ESRIutil.Strings
    Severity = Arcedit1.Command("&run odetest.aml test", Result)
End Sub
```

The use of AML's is limited to linear data processing procedures. At this time, existing AML's can be used as starting points for the creation of an ODE application, but ODE is not simply another method of building a GUI for an existing AML toolset based application. You will need to rebuild most, if not all of application, depending on how much toolset actions are tied to AML GUI interaction. The exception to this might be very simple tools such as a GUI for the

PROJECT command.

3.2 Direct ARC/INFO Command Calls

While many features can be accessed through the object model, it is still necessary to use Arc/Info commands to accomplish some tasks. This is done through the Command method of the objects.

Below is a small section of code that shows the use of some traditional ArcEdit commands.

```
    Select Case Button.Key
       Case "open"
         frmOpenDialog.aicontrol = frmArcedit.Arcedit1
         frmOpenDialog.Show vbModal
       Case "zoom"
         Severity = Arcedit1.Command("mape *;draw", Result)
         Call MapScale
       Case "pan"
         Severity = Arcedit1.Command("dynpan", Result)
       Case "fullextent"
    Severity = Arcedit1.Command("mape def;de nodes off;draw",
Result)
         Call MapScale
       Case "select"
         Severity = Arcedit1.Command("select one", Result)
       Case "split"
         Severity = Arcedit1.Command("split;de arc nodes;draw",
Result)
     End Select
```

Basically, the control (Arcedit1 in this case) is passed a command. A success flag is passed back in the Severity variable, and any results are put into the Result variable. You then write code to take actions depending on the contents of these two variables. This is much like adding an abstraction layer on top of AML, and thus it makes debugging of the code more difficult.

In ARC/INFO Version 8.0+, the underlying code is completely available as components, for use by both standard development environments and AML. ESRI notes that AML will continue to be fully supported under the new architecture in order to support the significant investment many users have made in legacy applications. The exact methodology of how this will operate is yet to be identified.

3.3 GUI Options

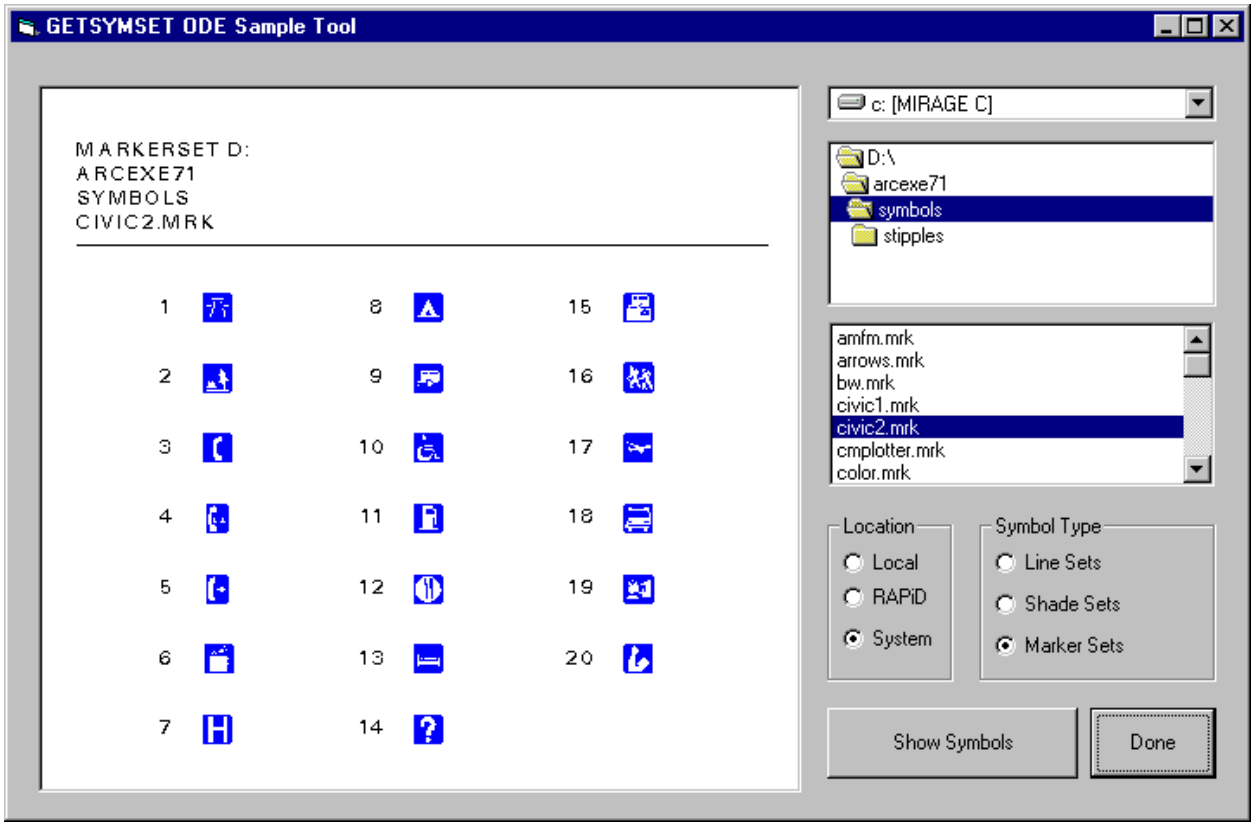
A promise of the Open Development Environment is the ability to standardize GUI's based on conventional Windows based GUI controls. Several different programming environments are available that support the development of GUI's using standard object based controls. The Visual Basic software is used as an example to illustrate some simple samples of GUI possibilities.

3.3.1 Visual Basic

Visual Basic and other Windows programming environments offer the developer a wide range of GUI controls (widgets in UNIX speak). Visual Basic comes with 21 controls to start with, and you

have access to another 40, or so, that are a part of the Windows operating system. In addition to these, there are many, many companies and individuals that have created custom controls which can also be used. In comparison, FormEdit has 11 controls, with no 3rd party extensions.

Figure 15: This example illustrates a re-coding of the "getsymset" AML toolset reviewed earlier in the AML chapter of the workbook. Note how the tool is now configured with an ArcPlot canvas to display symbol sets.



Another example of GUI differences is illustrated in the Multi-Window tool. This tool is provided with RAPiD*AML and allows the user to partition the ArcPlot graphics window in order to display different data, and/or maps, on different areas of the graphic canvas. The toolset is often used directly from other display oriented toolsets by calling routines without the need for a toolset menu. However, in this example we are presenting the toolset menu.

Figure 16: The standard UNIX AML version of the toolset menu utilizes BUTTON widgets to select the number of partitions, and the active partition to display in. Icons are used as DISPLAY fields to echo the current active partition for the ArcPlot window.

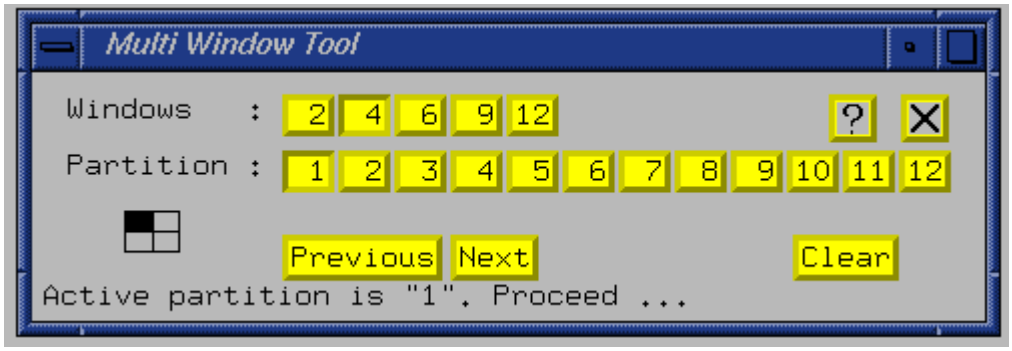


Figure 17: This menu illustrates the Multi-Window AML menu code displayed under Windows NT.

Note that some of the controls, noticeably the CHOICE control, are presented with more standard Windows look-and-feel. As well, all menus inherently have the standard Windows 'minimize' capabilities for the window. No changes have been made to the AML code.

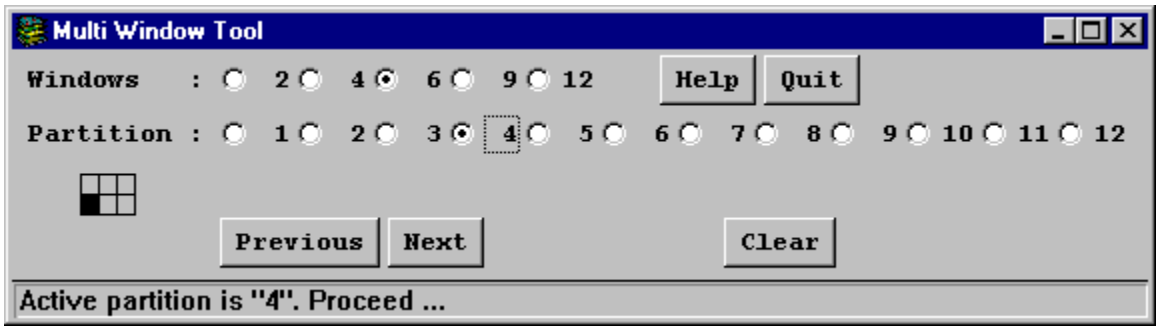
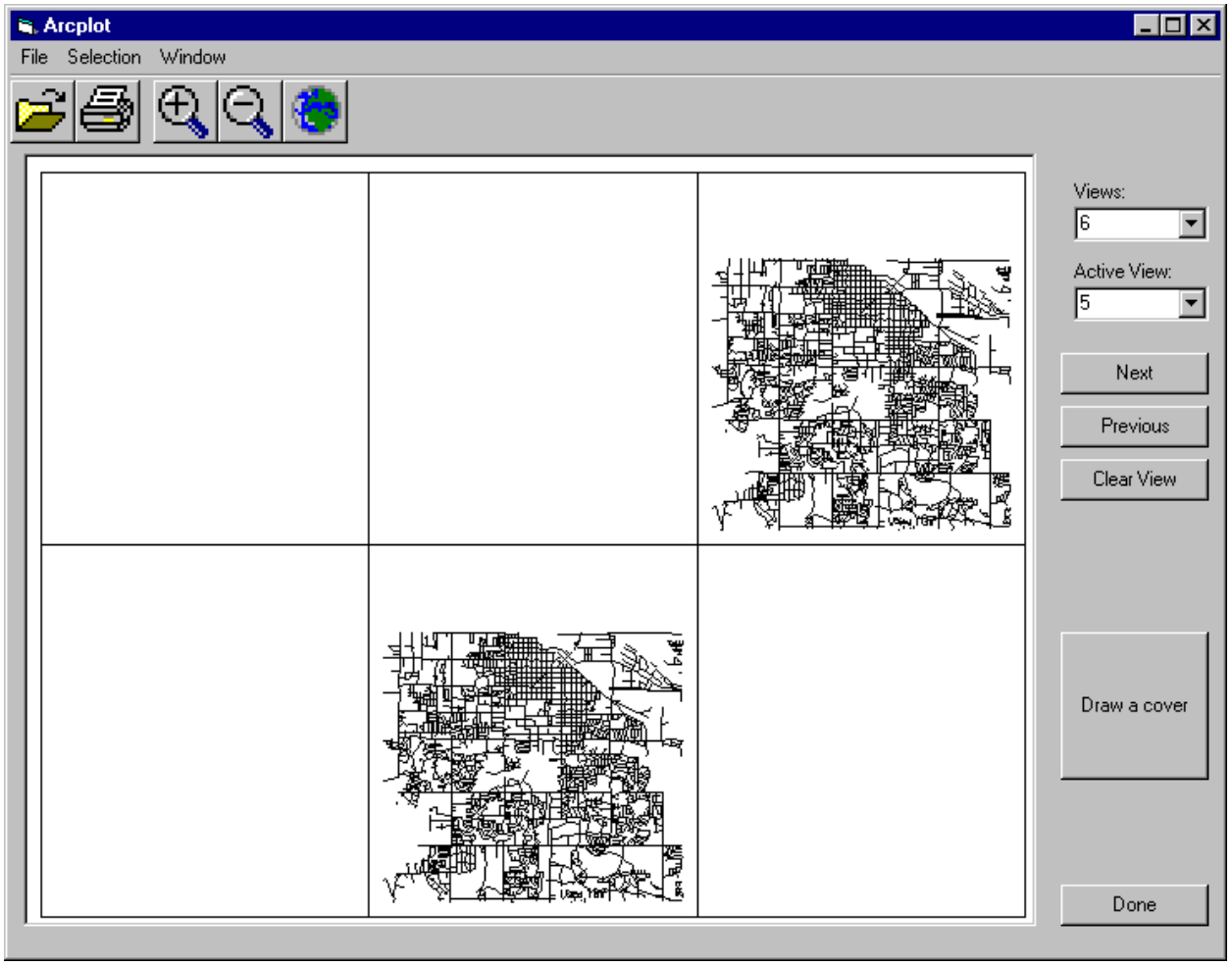


Figure 18: This GUI represents a re-coding of the Multi-Window toolset menu using Visual Basic and ODE techniques.

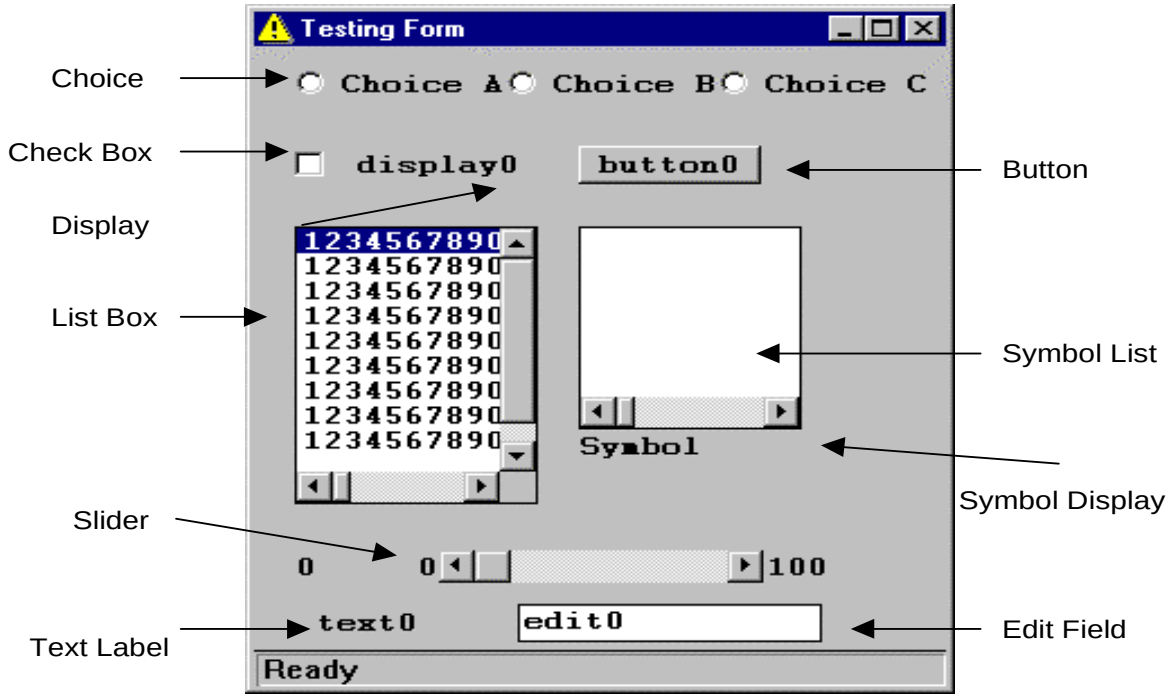


Some minor modification was required with the toolset routines to accommodate communication between the AML and the GUI. Ideally, this tool would be developed as a re-usable object, which could be plugged into existing ArcPlot applications. However, as a standalone tool we needed to wrap the tool around an ArcPlot canvas. Using Visual Basic programming techniques does provide a much wider range of objects and functionality for the programmer. The embedding of pan and zoom objects, with automatic redisplay, is an obvious advantage in the ArcPlot subsystem.

3.3.2 Windows Controls vs AML Widgets

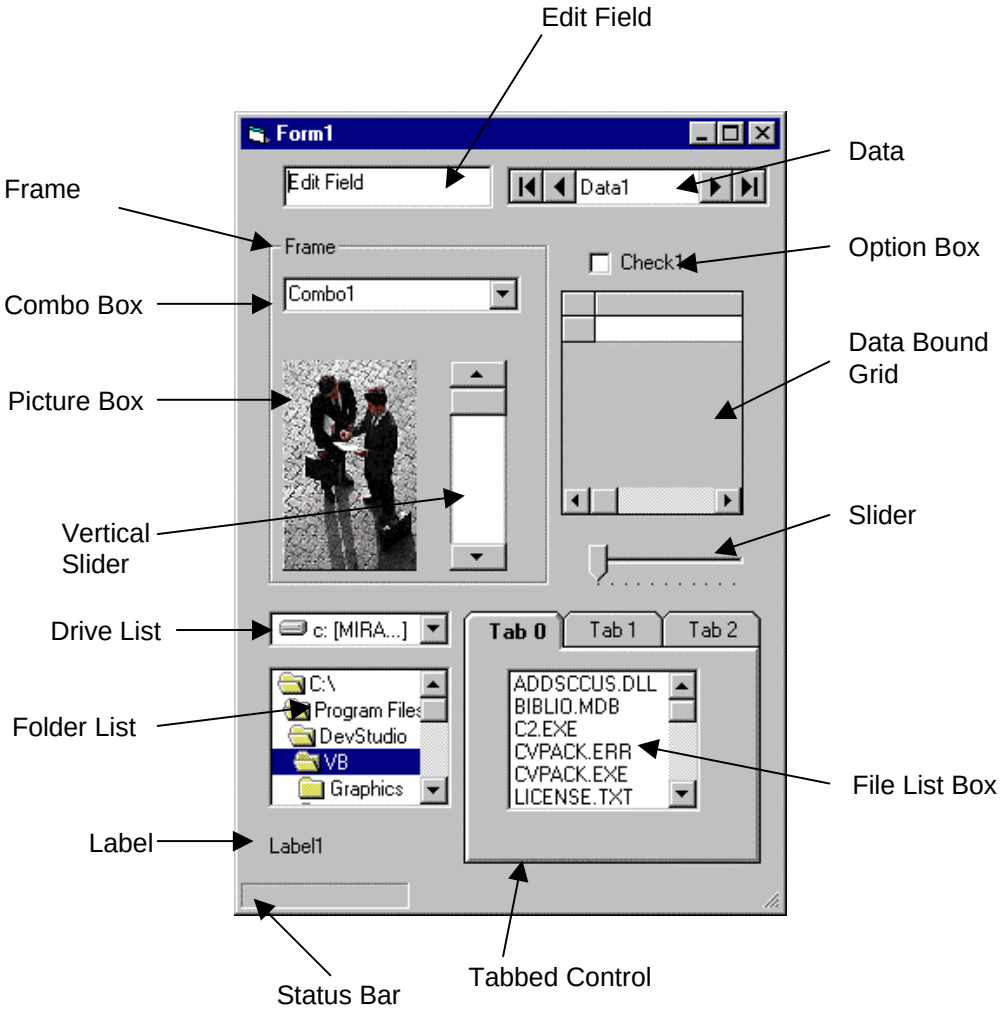
A criticism with AML menu widgets has been the lack of controls available compared to other operating system development environments. AML as traditionally only included a subset of the windowing widgets available under the SUN Open Look environment or the X-11 Windows environment for UNIX. However, with the advent of ARC/INFO migrating to the Windows NT platform AML has been extended slightly to provide new menu controls. A good example is the new type 8 cascading pulldown menu.

Figure 19: This menu illustrates the current menu widgets/controls available with AML on either UNIX or Windows NT (shown).



Visual Basic and other Windows programming environments offer the developer a wide range of GUI controls (widgets in UNIX speak). Visual Basic comes with 21 controls at start with, and you have access to another 40 or so that are a part of the Windows operating system. In addition to these, there are many, many companies and individuals that have created custom controls which can also be used. In comparison, FormEdit has 11 controls, with no 3rd party extensions.

Figure 20: This menu illustrates the menu controls available under Visual Basic, and hence available to ARC/INFO applications under ODE.



In the UNIX world, the GUI options are also very wide. A common choice for the GUI is Tcl toolkit. While this tool kit is distributed by Sun Microsystems, varieties exist for most UNIX platforms. TCL/TK is a programming system developed by John Ousterhout at the University of California, Berkeley which is easy to use, and which has very useful graphical interface facilities. TCL is the basic programming language, while TK is a ToolKit of widgets, which are graphical objects similar to those of other GUI toolkits, such as Xlib, Xview and Motif. Unlike many of the other toolkits, it is not necessary to use C or C++ in order to manipulate the widgets, and useful applications can be built very rapidly once some expertise of the TCL/TK system has been gained. It is important to note that the UNIX options for using the ODE require significantly more programming background and knowledge than for Windows, with the possible exception of

TCL/TK which is on par with the learning curve of Visual Basic.

C - API

While ESRI promotes the use of Visual Basic for use with the ODE, they have also provided header files and libraries for linking with C/C++. At version 7.1.2, libraries are provided for Arcplot, Arcedit and Grid.

3.4 OLE / ActiveX Development Strategies

The ability to build GIS applications that are truly integrated within the process flow of general office use is now a reality with ActiveX. This workbook will not try to explain all the features and options when developing ActiveX applications - much more information can be found at Microsoft's web site (www.microsoft.com). Basically, through the use of ActiveX based applications, a developer can imbed applications into any Microsoft product. An example would be a simple spatial selection application which generates mailing lists, that could be run within Word.

From a strategic standpoint, integration into other existing applications can be desirable when the main function of an application can best be accomplished within the existing application, and the GIS analytics are necessary, but secondary. Why try to build a fully featured word processor in a GIS application when you can embed the GIS portion into an existing word processor?

4.0 References

This section contains list of references used in preparation of this workbook. Additional information is also provided to help the reader in obtaining more information on related topics.

General Information

Additional copies of this workbook are available from Innovative GIS Solutions, Inc. They can be reached via telephone at (970) 490-5900 or by email at igis@innovativegis.com.

The ESRI User List (ESRI-L) available on the Internet is an excellent source of information and a forum for exchanging problems, solutions and issues about using ARC/INFO and related products with other ARC/INFO users. Many of the problems are AML based, or require an AML solution. Information on subscribing to this list can be found on the ESRI Web Site.

Books

Many of the books identified here can be obtained from the ESRI Web Site at http://www.esri.com

Understanding GIS – The ARC/INFO Method. ESRI, Inc. Redlands, CA.

ARC Macro Language – Self Study Workbook (with CD). Developing ARC/INFO Menus and

Macros with AML.. ESRI, Inc. Redlands, CA.

AML and Formedit. ARC/INFO User Manual. ESRI, Inc. Redlands, CA.

AML Commands. ARC/INFO User Manual. ESRI, Inc. Redlands, CA.

Courses

Course information and schedules can be obtained from the ESRI Web Site at http://www.esri.com

Customizing ARC/INFO with AML. Regular course offered by ESRI, Redlands, CA.

Using ARC/INFO ODE with Visual Basic. Regular course offered by ESRI, Redlands, CA.

Appendix A – Sample AML Toolset Header

This appendix illustrates a sample header used by the RAPiD*AML software.

```
/*-------------------------------------------------------------------------------------------
/*              I N N O V A T I V E   G I S   S O L U T I O N S ,  I N C.
/*                                RAPiD*AML - Toolset Header
/*-------------------------------------------------------------------------------------------
/* Name:           template.aml
/* Author:         <name_here>
/* Date:           <creation date here>
/* Changes:        <modification history here>
/* Purpose:        <describe purpose here>
/* Discussion:
/*         <describe toolset here>
/*
/* Standard Routines:
/*
/*    INIT             The main tool startup routine
/*    INITIALIZE       Initialize toolset
/*    MENU             Spark the main menu.
/*    INIT_SUBMENU     Example routine to initialize a submenu for the toolset.
/*
/*    BACK             Back up to previous menu (optional)
/*    NEXT             Proceed to next menu  (optional)
/*
/*    APPLY            Apply and quit
/*    CANCEL           Cancel current menu selections and exit
/*    QUIT             The quit routine for the menu thread.
/*    HELP             The default help routine.
/*    USAGE            Echo usage back to user if no routine input.
/*    BAILOUT          The orderly exit routine for AML errors in your routines.
/*-------------------------------------------------------------------
/* Calls Macros/Menus/Scripts: <lists other toolsets called here>
/*
/*          rathread.aml      RAPiD AML macro for thread management.
/*          template.menu     Default main menu for this toolset.
/*
/* Custom Routines:  <list routines used here>
/*
/* Globals:          <list globals used here>
/*
/* Syntax:           <list possible usage here>
/*          Usage: template INIT {'position'} {'stripe'}
/*          Usage: template <routine> {args}
/*    where
/*          routine  - name of the routine to be called.
/*          position - (quoted string) opening menu position.
/*          stripe   - (quoted string) menu stripe displayed.
/*===========================================================================
```

Appendix B – Example Toolset Template Files

The files presented in this appendix represent sample templates for toolset macros and menus. These templates are utilized as generic templates in the RAPiD*AML product. Developers are urged to customize these files for use at their site. The macro template presented here utilizes a standard thread manager tool, called “rathread.eaf”. This tool is a core routine of the RAPiD*AML software, but could easily be replaced by any thread manager tool.

```
/*-----------------------------------------------------------------------------------------------
/*              I N N O V A T I V E   G I S   S O L U T I O N S , I N C.
/*                                RAPiD*AML - Toolset Header
/*-----------------------------------------------------------------------------------------------
/*
/* Author:     D. Buckley - Innovative GIS
/* Date:       <creation date here>
/* Changes: <modification history here>
/* Purpose:
/*          This AML macro is provided as a standard file for creating your own "macro' toolsets.
/*
/*    YOU SHOULD ADD A DESCRIPTION OF YOUR TOOLSET HERE
/*
/* Discussion:
/*
/*    The RAPiD thread manager has been expanded to provide menu
/*    minimized capabilities 'al la Windows 95'.  The new template
/*    menu provides the required widgets to utilize this
/*    new functionality.  If you wish to make use of this capability
/*    you must use the new 'rathread' thread manager. The required RAPiD
/*    files are:
/*
/*    Thread manager:          rathread.eaf
/*    Thread minimizer:        ramenumgr.eaf
/*
/*     The thread manager automatically creates the thread name by adding
/*    a 'ra$' as a prefix to the menu file name.  In this manner, thread
/*    naming is standardized and removed from the users discretion.
/*    By doing this, we can provide standard thread tools, e.g. like minimize !
/*
/*    To provide this functionality you must have a copy of RAPiD AML 4.0
/*    and include the Minimize '_' and Exit 'x' widgets in your tool menu.
/*    The template menu (template.menu) includes these.
/*    The two specific widget calls that are required to support this
/*    minimize capability are:
/*
/*    Minimize widget : rawin95min.bm
/*    Minimize call   : &run ramenumgr minimize <current_thread> <min_stripe>
/*
/*    Exit widget     : rawin95exit.bm
/*    Exit call       : &run ramenumgr remove <current_thread>
/*
/*    The 'ramenumgr remove' routine simply calls your 'template quit'
/*    routine after it has cleaned up some global lists it uses to
```

```
/*    manage the minimize menus.
/*
/* Standard Routines:
/*
/*    INIT          The main tool startup routine
/*    INITIALIZE  Initialize toolset
/*    MENU        Spark the main menu.
/*    INIT_SUBMENU
/*                Example routine to initialize a submenu for the toolset.
/*
/*    BACK        Back up to previous menu (optional)
/*    NEXT        Proceed to next menu  (optional)
/*    APPLY       Apply and quit
/*
/*    CANCEL      Cancel current menu selections
/*    QUIT        The quit routine for the menu thread.
/*    HELP        The default help routine.
/*    USAGE       Echo usage back to user if no routine input.
/*    BAILOUT     The orderly exit routine for AML errors in your routines.
/*-----------------------------------------------------------------------
/* Called By:
/*
/* Calls Macros/Menus/Scripts: (lists other toolsets called here)
/*
/*          rathread.aml      RAPiD AML macro for thread management.
/*          template.menu   Default main menu for this toolset.
/*
/* Routines: (list routines used here)
/*
/* Globals: (list globals used here)
/*
/*          .ra$samglob       Example global documentation line
/*
/* Syntax: (list possible usage here)
/*          Usage: template INIT {'position'} {'stripe'}
/*          Usage: template <routine> {args}
/*    where
/*          routine   - name of the routine to be called.
/*          position - (quoted string) opening menu position.
/*          stripe    - (quoted string) menu stripe displayed.
/*
/*=======================================================================
/*




&args routine arglist:rest
/*
```

```
&severity &error &routine bailout
/*
/* Check arguments
&if ^ [NULL %routine%] &then
  &call %routine%
&else
  &call init
&return

/*-----------
&routine INIT
/*-----------
&call initialize
&call menu
&return

/*-----------
&routine INITIALIZE
/*-----------
/*       INSERT ANY TOOL INITIALIZATION STATEMENTS HERE
&return

/*-----------
&routine MENU
/*-----------
/*    Accept as 1st two args the position of the menu
/*    and the stripe (title text) of the menu
/*
/*     To launch a menu simply set the .ra$thr$* variables below.
/*     The thread manager will manage the thread startup etc.
/*
/*     The following variables are required:
/*     .ra$thr$pos        Position
/*     .ra$thr$str        Menu stripe
/*     .ra$thr$size       Size (null is default – extent of menu)
/*     .ra$thr$menu       AML menu file (no extension)
/*     .ra$thr$pin        PIN action
/*     .ra$thr$modal      Set modality (null for no, 'modal' for modal menu)
/*     .ra$template$stripe_min   10 character name for menu minimize (UNIX)
/*
&set position = [EXTRACT 1 [UNQUOTE %arglist%]]
&set stripe   = [EXTRACT 2 [UNQUOTE %arglist%]]
&if [NULL %position%] or %position%_ = #_ &then
 &set .ra$thr$pos [quote &ul &thread menubar &ll];
&else
 &set .ra$thr$pos %position%
/*

&if [NULL %stripe%] or %stripe%_ = #_ &then
 &sv .ra$thr$str template tool
&else
 &set .ra$thr$str [quote [unquote %stripe%]]
```

```
/*    The following variable defines the name for the menu when it is minimized.
/*    minimize stripe var (default is first 10 characters)
/*    CHANGE THIS TO YOUR CHOICE (< 10 CHARACTERS)
&sv .ra$template$stripe_min [substr [quote %.ra$thr$str%] 1 10];
/* set a specific size for the menu, or leave null to default to menu extent
&sv .ra$thr$size ;
/*    Set thread and menu specifications here first
&sv .ra$thr$menu template             /* name of menu file (no extension)
&sv .ra$thr$pin '&run template quit'   /* routine for thread cleanup and exit

/*    set the next global to 'modal' if you want the menu modal
&sv .ra$thr$modal ;
/*    Spark the main menu using the RAPiD AML thread manager tool
&run rathread launch
&return

/*-----------
&routine INIT_SUBMENU
/*-----------
/*    This is an sample routine to show you how to start a submenu from your
/*    toolset main menu.  This routine may be called by pressing a button
/*    from the main menu, e.g. 'Submenu...', with an action definition of
/*    '&run template init_submenu'
/*
/*    Set thread and menu specifications here first
/*
&sv .ra$thr$menu submenu1 /* menu file
/*    spark the submenu relative to the lower right corner
/*    of the main menu
&sv .ra$thr$pos '&cc &thread ra$template &lr' /* menu position
/*    Pressing the pin in the top left of the submenu will
/*    remove itself
&sv .ra$thr$pin '&thread &delete &self' /* pin action
&sv .ra$thr$str Submenu Example Menu    /* menu stripe
/* set a specific size for the menu, or leave null to default to menu extent
&sv .ra$thr$size ;
/*    set the next global to 'modal' if you want the menu modal
&sv .ra$thr$modal ; /* modality ('modal' or ' ')
/*    Spark the menu thread here
&run rathread launch
&return




/* ================================================================
/*    INSERT YOUR TOOLSET ROUTINES HERE
```

```
/* ===================================================================
/*-----------
&routine BACK
/*-----------
/*    if you are using a wizard approach modify, and uncomment, the
/*    next line to reflect the last tool to be called
/*
/**&run lasttool init
/*    now delete this tool thread
&run rathread delete ra$template

&return

/*-----------
&routine NEXT
/*-----------
/*    if you are using a wizard approach modify, and uncomment, the
/*    next line to reflect the next tool to be called
/*
/**&run nexttool init
/*    now delete this tool thread
&run rathread delete ra$template
&return

/*-----------
&routine APPLY
/*-----------
/*    insert your code required to apply the tool action here

/*    now quit
&call quit
&return

/*-----------
&routine CANCEL
/*-----------
/*
&run rathread delete submenu1 template
&return

/*-----------
&routine QUIT
/*-----------
/*
&run rathread delete submenu1 template
&return
/*-----------
&routine HELP
/*-----------
&run rathread viewhelp template.help [quote &cc &screen &cc]
&return
```

```
/*------------
&routine USAGE
/*------------
&type Usage: template INIT {"'position'"} {"'stripe'"}
&type Usage: template <routine> {args}
&return &inform

/*--------------
&routine BAILOUT
/*--------------
&severity &error &fail
&echo &off
&messages &on
&type *************************
&type Bailing out of tool "template" ...
&run msworking close
&sv cls [close -all]
&return &warning An error has occurred in routine: [translate %routine%] (template.aml).
```

The following file is a sample template form (type 7) menu file (template.menu)

```
7 template.menu
/*-------------------------------------------------------------------------------------------
/*              I N N O V A T I V E   G I S   S O L U T I O N S ,  I N C.
/*                                RAPiD*AML - Toolset Header
/*-------------------------------------------------------------------------------------------
/*
/* Author:     D. Buckley - Innovative GIS
/* Date:       <creation date here>
/* Changes: <modification history here>
/* Purpose:
/*
/*-------------------------------------------------------------------------------------------
                                          %help  %exit

  ____________________________________________
                                %back  %next  %appl
%help BUTTON KEEP ~
  HELP 'Review Help' ~
  'Help' &run template help;
%exit BUTTON KEEP ~
  HELP 'Quit' ~
  'Quit' &run template quit
%back BUTTON KEEP ~
  HELP 'Return to previous menu' ~
  '< Back' &run template back
%next BUTTON KEEP ~
  HELP 'Proceed to next menu ...' ~
  'Next >' &run template next
%appl BUTTON KEEP ~
  HELP 'Apply and quit' ~
  '  Apply  ' &run template ok
%FORMOPT SETVARIABLES IMMEDIATE MESSAGEVARIABLE .template$mess
%FORMINIT &sv .template$mess Proceed ...
```

Appendix C – Example AML Toolset – getsymset.aml

This appendix presents an ArcTools toolset, getsymset.aml, that is reviewed in the previous chapters. This toolset was developed by ESRI, Inc. and is provided as part of the ArcTools menu system. It illustrates a simple toolset that is designed to satisfy a specific purpose, the selection and review of a toolset.

```
/* $Id: getsymset.aml,v 2.0 1996/01/24 23:46:33 markz Exp $
/*-----------------------------------------------------------------------
/*           Environmental Systems Research Institute
/*-----------------------------------------------------------------------
/*   Program: GETSYMSET.AML
/*   Purpose: Tool for browsing the file system to select a symbolset file.
/*
/*-----------------------------------------------------------------------
/*     Usage: getsymset INIT <LINE | MARKER | SHADE | TEXT> <variable_name>
/*           {'position'} {'stripe'}
/*     Usage: getsymset <routine_name>
/*
/* Arguments: routine - name of the routine to be called.
/*
/*          These arguments are used with the INIT routine:
/*            varname - variable name to receive file name returned from the
/*                      menu
/*            symtype - the symbolset type to display
/*                      defaults to marker
/*            position - (quoted string) opening menu position.
/*            stripe - (quoted string)  menu stripe displayed.
/*
/*   Globals:
/*-----------------------------------------------------------------------
/*-----------------------------------------------------------------------
/*     Notes: If the menu is canceled, the variable that was passed in is
/*           set to null.
/*-----------------------------------------------------------------------
/*     Input:
/*    Output:
/*-----------------------------------------------------------------------
/*   History: Matt McGrath - 12/15/92 - Original coding (modified getsymset)
/*           Matt McGrath - 07/06/93 - replace modal.aml with &modal,
/*                     update EXIT routine
/*           Mark D Zollinger - 11/01/94 - VAXinate.  Centralize routines
/*           Ian DeMerchant - 11/02/94 - Added help routine
/*=======================================================================
/*


&args routine symtype varname position stripe
```

```
&severity &error &routine bailout


/* Check arguments
&if [null  %routine%] &then
 &set routine = init
/*
&call %routine%
/*
&return

/*------------
&routine USAGE
/*------------
/*
&type Usage: getsymset INIT <LINE | MARKER | SHADE | TEXT> <variable_name>
&type                {"'position'"} {"'stripe'"}
&type Usage: getsymset <routine_name>
&return &warning

/*-----------
&routine INIT
/*-----------
/* Display the menu
/*
/* Check arguments that should accompany the INIT routine
&if [null %varname%] &then &call usage
&if [keyword [locase %symtype%] line marker shade text] lt 1 &then
  &call usage
/*
/* Set the names of the variables to receive the values set in the menu
&set .getsymset$filevarname = %varname%
/*
/* Initialize other variables
/*
/* Current directory
/* Set path to previous one if this tool has already been used
&if not [variable .arctools$get_savepath] &then
  &set .arctools$get_savepath
&if [null %.arctools$get_savepath%] &then
  &set .getsymset$curdir = [show &workspace]
&else &set .getsymset$curdir = %.arctools$get_savepath%
/*&set .getsymset$symsetdir = %.getsymset$curdir%
/*
/*
/* Save the current symbol set.  This is done so that the selected symbol
/* sets can be displayed in the menu.  The original symbolset is then reset
/* upon exiting this tool


&set symtype = [translate %symtype%]
```

```
&select %symtype%
  &when MARKER
    &do
      &set .getsymset$symtype = %symtype%
      &set .getsymset$origmarkerset = [show markerset]  /* Save current set
      &if [exists .xxat_symset -FILE] &then
        &set mmc [delete .xxat_symset -FILE]
      MARKERCOPY 1000 999     /* Save the current symbol environment
      MARKERSAVE .xxat_symset /* Make a backup of the current symbols
      MARKERDELETE ALL        /* Clear the plate
      &set .getsymset$wildcard = [joinfile * mrk -ext]
    &end
  &when TEXT
    &do
      &set .getsymset$symtype = %symtype%
      &set .getsymset$origtextset = [show textset]  /* Save current set
      &if [exists .xxat_symset -file] &then
        &set mmc [delete .xxat_symset -file]
      TEXTCOPY 1000 999     /* Save the current symbol environment
      TEXTSAVE .xxat_symset /* Make a backup of the current symbols
      TEXTDELETE ALL        /* Clear the plate
      &set .getsymset$wildcard = [joinfile * txt -ext]
    &end
  &when LINE
    &do
      &set .getsymset$symtype = %symtype%
      &set .getsymset$origlineset = [show lineset]    /* Save current set
      &if [exists .xxat_symset -FILE] &then
        &set mmc [delete .xxat_symset -FILE]
      LINECOPY 1000 999     /* Save the current symbol environment
      LINESAVE .xxat_symset /* Make a backup of the current symbols
      LINEDELETE ALL        /* Clear the plate
      &set .getsymset$wildcard = [joinfile * lin -ext]
    &end
  &when SHADE
    &do
      &set .getsymset$symtype = %symtype%
      &set .getsymset$origshadeset = [show shadeset]  /* Save current set
      &if [exists .xxat_symset -FILE] &then
        &set mmc [delete .xxat_symset -FILE]
      SHADECOPY 1000 999     /* Save the current symbol environment
      SHADESAVE .xxat_symset /* Make a backup of the current symbols
      SHADEDELETE ALL        /* Clear the plate
      &set .getsymset$wildcard = [joinfile * shd -ext]
    &end
  &otherwise
    &call usage
&end /* Select block
/*
/* set up scrolling lists of directories and files
&run get_routines subdir .getsymset
/* Initialize list to ARC/INFO symbolsets
```

```
&set .getsymset$symsetdir = $RAPIDHOME/main/samples/symbols
&set .getsymset$systemfiles RAPiD
&set .getsymset$symwild ~
     = [joinfile %.getsymset$symsetdir% %.getsymset$wildcard% -file]
/*
&if [null %position%] or %position%_ = #_ &then
  &set position = &cc &screen &cc
&if [show &thread &exist ra$fs_pref] &then
  &set position = &ul &thread ra$fs_pref &ll
/*
&if [null %stripe%] or %stripe%_ = #_ &then
  &set stripe = 'Select a Shade Symbol Set File'
&if [show &thread &exists tool$getsymset] &then
  &thread &delete tool$getsymset
&thread &create tool$getsymset &modal ~
  &menu getsymset ~
  &position [unquote %position%] ~
  &stripe [quote [unquote %stripe%]] ~
  &pinaction '&run getsymset cancel'
/*
&return

/*-------------
&routine CURDIR
/*-------------
/* Change the current directory to the specified directory and update
/* the list of subdirectories and coverages
&run get_routines curdir .getsymset
&set .getsymset$symwild = %.getsymset$curwild%
&set .getsymset$systemfiles = .FALSE.
/*
&return

/*-------------
&routine SUBDIR
/*-------------
/* Change the current directory to the specified subdirectory
&run get_routines subdir .getsymset
&set .getsymset$symwild = %.getsymset$curwild%
&set .getsymset$systemfiles = .FALSE.

/* Apply the selected subdir value
/*&set [value .getsymset$subdirvarname] = %.getsymset$subdir%
&return




/*---------
&routine UP
/*---------
```

```
/* Move up one directory.
&run get_routines up .getsymset
&set .getsymset$symwild = %.getsymset$curwild%
&set .getsymset$systemfiles = .FALSE.
/*
&return

/*-----------
&routine FILE
/*-----------
/* Apply selected file name
&set [value .getsymset$filevarname] = %.getsymset$file%
/*
&return

/*-----------------
&routine TOGGLE_SYS
/*-----------------
/* Toggle between system and local symbolsets
&if [quote %.getsymset$systemfiles%] eq 'System' &then
  &set .getsymset$symwild ~
   = [joinfile [joinfile $ARCHOME symbols -sub] %.getsymset$wildcard% -file]
&if [quote %.getsymset$systemfiles%] eq 'RAPiD' &then
  &set .getsymset$symwild ~
   = $RAPIDHOME/main/samples/symbols/%.getsymset$wildcard%
&if [quote %.getsymset$systemfiles%] eq 'Local' &then
  &set .getsymset$symwild = %.getsymset$curwild%
&return

/*----------------
&routine SETSYMSET
/*----------------
/* update scrolling list of symbols to reflect selected symbolset
%.getsymset$symtype%DELETE ALL
%.getsymset$symtype%SET %.getsymset$file%
&set .getsymset$cursymset = [ENTRYNAME %.getsymset$file%]
&return

/*-----------
&routine SHOW
/*-----------
&if [null [value .getsymset$file]] &then
```

```
 &sv .getsymset$msg Symbol Set is not defined. Cannot show set.
&else
&do
 &if [exists %.getsymset$file% -file] &then
 &do
  &sv .getsymset$msg Displaying current symbol set "[entryname %.getsymset$file%]"...
  &run rathrwin sync tool$getsymset
  &call setsymset
  &if [show &thread &exists ra$fs_class] &then
    &run symboldump shade screen 1 10;
  &else
    &run symboldump shade screen 1 100
  &messages &on
 &end
 &else
 &do
   &sv .getsymset$msg Symbol set does not exist. Cannot show set.
 &end
&end
&return

/*---------
&routine APPLY
/*---------
/* Set values and clean up
/*
/* If a system symbolset has been selected, don't include the full path
&if [quote %.getsymset$systemfiles%] eq 'Local'  &then
  &set [value .getsymset$filevarname] = %.getsymset$file%
&if [quote %.getsymset$systemfiles%] eq 'System'  &then
  &set [value .getsymset$filevarname] = [entryname %.getsymset$file%]
&if [quote %.getsymset$systemfiles%] eq 'RAPiD'  &then
  &set [value .getsymset$filevarname] = ~
$RAPIDHOME/main/samples/symbols/[entryname %.getsymset$file%]
/*

/* &type Shadeset is now : [show shadeset]
/* Save the path for future use of this tool
&set .arctools$get_savepath = %.getsymset$curdir%
&call exit
/*
&return




/*-------------
&routine CANCEL
/*-------------
/* Set values to null to indicate to calling program that nothing was
/* selected
```

```
&set [value .getsymset$filevarname]
/*    reset fragclass.shd
shadedelete all
&sv dummy %.am$slash%main%.am$slash%samples%.am$slash%symbols%.am$slash%
shadeset [pathname $RAPIDHOME]%dummy%rapidclass.shd
&call exit
/*
&return

/*-----------
&routine HELP
/*-----------
/* Display help about this menu
&run disp_help getsymset
&return

/*-----------
&routine EXIT
/*-----------
/* Quit from menu, clean up
/*
&select %.getsymset$symtype%
  &when MARKER
    &do
      MARKERDELETE ALL
      MARKERSET .xxat_symset /* Reset the saved symbolset
      MARKERSET %.getsymset$origmarkerset% /* Reset individual symbolset
      /* Reset the current symbol environment
      MARKERCOPY FROM .xxat_symset 999 1000
      /* Clean up temporary symbolset
      &if [exists .xxat_symset -FILE] &then
        &set mmc [delete .xxat_symset -FILE]
    &end
  &when LINE
    &do
      LINEDELETE ALL
      LINESET .xxat_symset /* Reset the saved symbolset
      LINESET %.getsymset$origlineset% /* Reset individual symbolset
      /* Reset the current symbol environment
      LINECOPY FROM .xxat_symset 999 1000
      /* Clean up temporary symbolset
      &if [exists .xxat_symset -FILE] &then
        &set mmc [delete .xxat_symset -FILE]
    &end
  &when SHADE
    &do
     &if [null [value .getsymset$file]] &then
     &do
      SHADEDELETE ALL
      SHADESET .xxat_symset /* Reset the saved symbolset
      SHADESET %.getsymset$origshadeset% /* Reset individual symbolset
      /* Reset the current symbol environment
```

```
      SHADECOPY FROM .xxat_symset 999 1000
      /* Clean up temporary symbolset
     &end
     &else
     &do
      SHADEDELETE ALL
      SHADESET %.getsymset$file%
      /* Reset the current symbol environment
     &end

      &if [exists .xxat_symset -FILE] &then
        &set mmc [delete .xxat_symset -FILE]
    &end
 &when TEXT
    &do
      TEXTDELETE ALL
      TEXTSET .xxat_symset /* Reset the saved symbolset
      TEXTSET %.getsymset$origtextset% /* Reset individual symbolset
      /* Reset the current symbol environment
      TEXTCOPY FROM .xxat_symset 999 1000
      /* Clean up temporary symbolset
      &if [exists .xxat_symset -FILE] &then
        &set mmc [delete .xxat_symset -FILE]
    &end

&end  /* End select block
/*
&dv .getsymset$*
&if [show &thread &exists tool$getsymset] &then
  &thread &delete tool$getsymset
/*
&return
/*--------------
&routine BAILOUT
/*--------------
/*
&severity &error &ignore
&severity &warning &ignore
&return &warning An error has occurred in routine: %routine% (GETSYMSET.AML)
```