Read This First !

Contents of Package II
Clascal and ToolKit

This guide will serve as an overview of the documents contained in Package II. Please read this entire guide before you start reading the documents.

Contents:

Summary of Contents

Packaging and Release Information

1. In this package you should have the following pieces:
   a. Installation Instructions
   b. The Introduction to Clascal Manual.
   c. Clascal Syntax Diagrams.
   d. The Lisa ToolKit Self-Paced Training.
   e. The Lisa Applications ToolKit Reference Manual.
   f. More on Debugging.
   g. A Phrase File Document
   h. A Printing Document
   i. A Class Hierarchy Document
   j. UObject, UDraw, and UABC interfaces.
   k. UDialog documentation and interface.
   l. UText documentation, and interface for UText and UUnivText.
   m. Listings of Sample ToolKit programs
   n. Documentation on converting from older versions of the ToolKit to ToolKit 3.0.
   o. Icon editor documentation.
   p. Lisa User Interface Guidelines.
   q. Lisa Internals.
   r. A Flow of Control Diagram (poster).
   s. Five (5) ToolKit Release Diskettes.

2. To use the ToolKit, you need to have the following:
   a. A one megabyte Lisa with at least one 5 Megabyte profile. Two 5 Megabyte profiles or one built-in 10 Megabyte hard disk is better. We strongly discourage using just one 5 Megabyte profile; a single profile leaves very little space in which to work.
   b. The Lisa Office System, version 3.0.
   c. The Lisa Workshop development environment, version 3.0.

3. *ToolKit 3.0 Release Notes.* This memo describes the ToolKit release you were sent and how to install it on your machine along with the other software you have purchased. Please read it carefully and completely and follow the instructions exactly. This will give your

Lisa the standard ToolKit development configuration and make it easier to find problems if they occur.

Clascal Documentation

1. *An Introduction to Clascal.* This explains the Clascal language and why we are using it to implement the ToolKit. **(read this first)**
2. *Clascal Syntax Diagrams.* This short manual contains all the syntax diagrams for Clascal.

ToolKit Segments (a self-paced ToolKit course)

The ToolKit Segments are a self-paced tutorial explaining most of the concepts you need to know to use the ToolKit. You should read this section after you have mastered the Clascal documentation above. The segments start out with the very simple ToolKit concepts and build towards more complicated subjects as your knowledge of the ToolKit grows. Included with the Segments are listings of actual ToolKit sample programs. These sample programs are also included on the disks which accompany this package so you can try them out and modify them as directed in the segments. Here are the titles to the 11 ToolKit Segments.

0. *Conceptual Foundation of the ToolKit.*
1. *Introduction to the ToolKit.*
2. *What is a Document.*
3. *Creating from the Generic Application.*
4. *BlankStationary: The View of a New Document.*
5. *Intro to the Boxer Application*
6. *Selections and Highlighting in Boxer.*
7. *Moving Boxes.*
8. *Creating a Box. A Second Selection Class.*
9. *Recoloring, Duplicating, and Clear All Commands with Undo.*
10. *Filters.*
11. *Cut + Paste and Mouse Key Events as Commands; Advanced Commands.*

The Lisa Applications ToolKit Reference Manual

This is a complete reference manual describing the entire ToolKit. It is a very useful reference manual. You should look through this after reading the segments above. Entirely read chapters 1 through 5. Chapter six should be scanned and then used later while programming to answer specific questions about a particular Class, Method or Variable.

1. Introduction to the ToolKit
2. UObject
3. UDraw
4. Application Base Classes
5. ToolKit Debugger
6. Reference Information for ToolKit Classes

Debugging ToolKit Applications

Chapter 5 of The Lisa Applications ToolKit Reference Manual and More on Debugging show you how to use the ToolKit debugger as well as how to interpret Clascal code from within LisaBug.

Using Phrase Files

Phrase files are used for all menu, alert, dialog and other textual information in a ToolKit application. By keeping all this information in a phrase file that can be easily edited, a ToolKit application can be converted to the language of another country by a bi-lingual person familiar with the application. A recompile is not required to move a ToolKit application to another language. No programming is needed.

1. *The Phrase File document.* This document explains everyhthing you need to know about using phrase files.
2. *TK/PABC.* PABC is the phrase file that is used in all ToolKit applications. It contains all the standard 'Generic Application' phrases.

ToolKit Flow of Control Diagrams

Flow of Control (the order in which ToolKit methods are called and how the ToolKit calls your Application) is discussed in many places within these ToolKit documents. This wall poster contains flow of control diagrams to give you an overview of the structure and logic of the ToolKit Generic Application and how your Application fits into this architecture.

ToolKit Interfaces

The interfaces contain all the ToolKit global constant, type and variable declarations as well as all the ToolKit methods that you may want to call from your application. You will never have to actually call or even understand many of these methods, so don't get overwhelmed by the number of methods in the interfaces. The Interfaces are very useful however and you can learn a lot by reading through them and studying them. Any serious ToolKit developer needs to become very familar with the interfaces to UObject, UDraw and UABC. These three units make up the core of the ToolKit Generic Application.

1. *UObject* ToolKit unit that implements the standard lowest level methods (procedures and functions) that are used by all Classes in the ToolKit environment. UObject also implements a standard set of "collection" primitives used throughout the ToolKit. You need to understand Collections and Scanners well. They are documented in the interface as well as the reference manual. The ToolKit debugger is also implemented in UObject. Both UObject and the debugger are documented in The Lisa Applications ToolKit Reference Manual.
2. *UDraw* UDraw implements a 32 bit coordinate system within the ToolKit and provides a way to convert graphic items in that coordinate system to the 16-bit QuickDraw world. See The Lisa Applications ToolKit Reference Manual for more documentation on this unit.
3. *UABC* UABC provides the higher level structure for all ToolKit applications. Its classes are the 'Generic Application' upon which all ToolKit 'Specific Applications' are built. See the ToolKit Reference Manual for documentation on this unit; there are also important comments in the interface itself. The Class Hierarchy document is very helpful in finding your way around the ABC structure. You will find that flipping to that document often will save you a lot of time. You may want to make a copy of it to post on the wall and be in constant view.

Building Blocks

Building Blocks are units that add functionality to the ABCs. You need only study the interfaces to the building blocks that are going to be used in the specific application you are writing. If you are using dialogs in your application, study the Dialog Building Block. If you are using text, especially a word processor application, study the Text Building Block. The documentation available for each building block is included in the section for that building block. The sample applications do not demonstrate all the capabilities of their respective building blocks.

1. *UDialog* This building block allows an application to easily put dialog boxes up on the Lisa screen. Dialog boxes are used by all the Lisa applications to get specialized information from the user. The windows that come up when you choose 'Format for Printer...' or 'Print...' from the File/Print menu are examples of dialog boxes. There are many different types of dialog boxes. See the documentation on the Dialog Box Building Block included here. It is quite well documented. There is also a sample application USamDialog, also in this section, for more information.

2. *UText* The text building block provides the basis for ToolKit text editing in any ToolKit applications. This building block provides very extensive text handling as well as cut and paste of text with other ToolKit applications.

Lisa User Interface Guidelines

This document contains useful information about the User Interface Guidelines for all Lisa Applications. It also contains definitions for Lisa terminology you will need to be familar with. You should read this.

Converting from TK7D to the Final ToolKit

This documents most of the changes that happened to the ToolKit and the Clascal language between the TK7D version and the final ToolKit version. Since many people were initially given the TK7D version we hope this will help them convert to the final version.

A Word About Cross-Referencing

You will notice that the program listings in this package are cross-referenced. This means that at the end of each program listing you will find an index of identifiers that appear in that listing. The identifiers which are indexed are those which are declared at the outer-most scope level of the program. Thus, globally defined variables, class names, and the fields and methods of classes will be indexed. Also, any unnested procedure or function names will be indexed. Hence, local variables and nested procedure and function names will be indexed only if they were previously defined at an outer-most scope level. If you find a variable, procedure, or function name which is used in a program but which is not cross-referenced and which is not defined in the scope of the global procedure or function in which it appears, then you will find it defined in one of the files *used* by that program (note: some files may reside in libraries, in which case you will not be able to examine them).

An index entry is of the form:

<id name> <line number> <info. symbol> (<file number>),

where <id name> is the name of the identifier,

<file number> is the nth file included in the cross-referenced listing. Often, a program is so large that it exists in separate files which are linked together in a chain by *$I* compiler directives. Usually, the program's interface is the first file in the chain and the program's implementation is the second file in the chain. If the program is large enough, like UABC is, then its implementation may be spread over many files (UABC, for example, has its implementation spread over the second, third, fourth, and fifth files in its chain). A file number n, then, refers to the nth file in the chain.

<line number> is the mth line number in the file denoted by <file number>.

<info. symbol> is an *, =, or a blank. An asterisk means that the identifier is defined at the specified location. An equals sign means that an assignment is made to the identifier at the specified location. A blank means that the identifier is neither defined nor assigned to at that location.

In the program listings, the *file number* is the left-most number on every line. The *line number* is the number to the immediate right of the file number. Thus, for example, if an index entry were:

MousePress 255*(1)

then you would know that *MousePress* was defined on line 255 of file number 1. This location is labelled 1 255 in the left-margin of the listing.