1. INTRODUCTION

Medley is a *programming system* that consists of a programming *language*, a large number of predefined programs (or *functions*) that you can use directly or as subroutines, and an *environment* that supports you with a variety of specialized programming tools. The language and predefined functions of Lisp are rich, but similar to those of other modern programming languages. The Medley programming environment, on the other hand, is very distinctive. Its main feature is an integrated set of programming tools that know enough about Interlisp and Common Lisp to act as semi-autonomous, intelligent "assistants" to you. This environment provides a completely self-contained world for creating, debugging and maintaining Lisp programs.

This manual describes all three parts of Medley. There are discussions of the language, about the pieces of the system that can be incorporated into your programs, and about the environment. The line between your code and the environment is thin and changing. Most users extend the environment with some special features of their own. Because Medley is so easily extended, the system has grown over time to incorporate many different ideas about effective and useful ways to program. This gradual accumulation over many years has resulted in a rich and diverse system. It is also the reason this manual is so large.

The rest of this manual describes the individual pieces of Medley; this chapter describes system as a whole—including the otherwise-unstated philosophies that tie it all together. It will give you a global view of Medley.

Lisp as a Programming Language

This manual is not an introduction to programming in Lisp. This section highlights a few key points about lisp that will make the rest of the manual clear.

In Lisp, large programs (or functions) are built up by composing the results of smaller ones. Although Medley, like most modern Lisps, lets you program in almost any style you can imagine, the natural style of Lisp is functional and recursive—each function computes its result by calling lower-level "building-block" functions, then passing that result back to its caller (rather than by producing "side-effects" on external data structures, for example).

Lisp is also a list-manipulation language. Like other languages, Lisp can process characters and numbers. But you get more power if you program at a higher level. The primitive data objects of Lisp are "atoms" (symbols or identifiers) and "lists" (sequences of atoms or lists), which you use to represent information and relationships. Each Lisp dialect has a set of operations that act on atoms and lists, and these operations comprise the core of the language.

Invisible in the programs, but essential to the Lisp style of programming, is an automatic memory management system (an "allocator" and a "garbage collector"). New storage is allocated automatically whenever a you create a new data object. And that storage is automatically reclaimed for reuse when no other object refers to it. Automated memory management is essential for rapid, large-scale program development because it frees you from the task of maintaining the details of memory administration, which change constantly during rapid program evolution.

A key property of Lisp is that Lisp function definitions are just pieces of Lisp list data. Each subfunction "call" (or *function application*) is written as a list with the function first, followed by its arguments. Thus, `(PLUS 1 2)` represents the expression `1+2`. A function's definition, then, is just a list of such function applications, to be evaluated in order. This representation of program as data lets you use the same operations on programs that you use on data—making it very easy to write Lisp programs that look at and change *other Lisp programs*. This, in turn, makes it easy to develop programming tools and translators, which was essential to the development of the Medley environment.

The most important benefit of this is that you can extend the Lisp programming language itself. Do you miss some favorite programming idiom? Just define a function that translates the desired expression into simpler Lisp. Now your idiom is *part of the language*. Medley has extensive facilities for making this type of language extension. Using this ability to extend itself, Interlisp has incorporated many of the constructs that have been developed in other modern programming languages (e.g. if-then-else, do loops, etc.).

Medley as an Interactive Environment

Medley programs should not be thought of as simple files of source code. All Medley programming takes place within the Medley environment, which is a completely self-sufficient environment for developing and using Medley programs. Beyond the obvious programming facilities (e.g., program editors, compilers, debuggers, etc.), the envionrment also contains a variety of tools that "keep track" of what happens. For example, the Medley File Manager notices when programs or data have been changed, so the system will know what needs to be saved at the end of a session. The "residential" style, where you stay inside the environment throughout the development, is essential for these tools to operate. Furthermore, this same environment is available to support the final production version, some parts providing run time support and other parts being ignored until the need arises for further debugging or development.

For terminal interaction, Medley provides a top level "Read-Eval-Print" executive, which reads whatever you type in, evaluates it, and prints the result. (This interaction is also recorded, so you can ask to do an action again, or even to undo the effects of a previous action.) Although Executives understand some specialized commands, most of the interaction will consist of simple Lisp expressions. So rather than special commands for operations like manipulating your files, you just type the same expressions that you would use to accomplish them in a Lisp program. This creates a very rich, simple, and uniform set of interactive commands, since any Lisp expression can be typed at an executive and evaluated immediately.

In normal use, you write a program (or rather, "define a function") by typing in an expression that invokes the "function defining" function `(DEFINEQ)`, giving it the name of the function being defined and its new definition. The newly-defined function can be executed immediately, simply by using it in a Lisp expression.

In addition to these basic programming tools, Medley also provides a wide variety of programming support mechanisms:

List structure editor	Since Lisp programs are represented as list structure, Medley provides an editor which allows one to change the list structure of a function's definition directly. See Chapter 16.
Pretty-printer	The pretty printer is a function that prints Lisp function definitions so that their syntactic structure is displayed by the indentation and fonts used. See page Chapter 26.
Debugger	When errors occur, the debugger is called, allowing you to examine and modify the context at the point of the error. Often, this lets you continue execution without starting from the beginning. Within a break, the full power of Interlisp is available to you. Thus, the broken function can be edited, data structures can be inspected and changed, other computations carried out, and so on. All of this occurs in the context of the suspended computation, which remains available to be resumed. See Chapter 14.
DWIM	The "Do What I Mean" package automatically fixes misspellings and errors in typing. See Chapter 20.
Programmer's Assistant	Medley keeps track of your actions during a session and allows each one to be replayed, undone, or altered. See Chapter 13.
Masterscope	Masterscope is a program analysis and management tool which can analyze users' functions and build (and automatically maintain) a data base of the results. This allows you to ask questions like "`WHO CALLS ARCTAN`" or "`WHO USES COEF1 FREELY`" or to request systematic changes like "`EDIT WHERE ANY` [function] `FETCHES ANY FIELD OF` [the data structure] `FOO`". See Chapter 19.
Record/Datatype Package	Medley allows you to define new data structures. This enables one to separate the issues of data access from the details of how the data is actually stored. See Chapter 8.
File Manager	Source code files in Medley are managed by the system, removing the problem of ensuring timely file updates from the user. The file manager can be modified and extended to accomodate new types of data. See Chapter 17.
Performance Analysis	These tools allow statistics on program operation to be collected and analyzed. See Chapter 22.
Multiple Processes	Multiple and independent processes simplify problems which require logically separate pieces of code to operate in parallel. See Chapter 23.

Windows	The ability to have multiple, independent windows on the display allows many different processes or activities to be active on the screen at once. See Chapter 28.
Inspector	The inspector is a display tool for examining complex data structures encountered during debugging. See Chapter 26.

These facilities are tightly integrated, so they know about and use each other, just as they can be used by user programs. For example, Masterscope uses the structural editor to make systematic changes. By combining the program analysis features of Masterscope with the features of the structural editor, large scale system changes can be made with a single command. For example, when the lowest-level interface of the Medley I/O system was changed to a new format, the entire edit was made by a single call to Masterscope of the form `EDIT WHERE ANY CALLS '(BIN BOUT ...)`. [Burton et al., 1980] This caused Masterscope to invoke the editor at each point in the system where any of the functions in the list `'(BIN BOUT ...)` were called. This ensured that no functions used in input or output were overlooked during the modification.

Philosophy

Medley's extensive environmental support has developed over the years to support a particular style of programming called "exploratory programming" [Sheil, 1983]. For many complex programming problems, the task of program creation is *not* simply one of writing a program to fulfill specifications. Instead, it is a matter of exploring the problem (trying out various solutions expressed as partial programs) until one finds a good solution (or sometimes, any solution at all!). Such programs are by nature evolutionary; they are transformed over time from one realization to another in response to a growing understanding of the problem. This point of view has lead to an emphasis on having the tools available to analyze, alter, and test programs easily. One important aspect of this is that the tools be designed to work together in an integrated fashion, so that knowledge about the user's programs, once gained, is available throughout the environment.

The development of programming tools to support exploratory programming is itself an exploration. No one knows all the tools that will eventually be found useful, and not all programmers want all of the tools to behave the same way. In response to this diversity, Interlisp has been shaped, by its implementors and by its users, to be easily extensible in several different ways. First, there are many places in the system where its behavior can be adjusted by the user. One way that this can be done is by changing the value of various "flags" or variables whose values are examined by system code to enable or suppress certain behavior. The other is where the user can provide functions or other behavioral specifications of what is to happen in certain contexts. For example, the format used for each type of list structure when it is printed by the pretty-printer is determined by specifications that are found on the list `PRETTYPRINTMACROS`. Thus, this format can be changed for a given type simply by putting a printing specification for it on that list.

Another way in which users can affect Medley's behavior is by redefining or changing system functions. The "Advise" capability, for instance, lets you modify the operation of virtually any function in the system by wrapping code "around" the selected function. (This same philosophy extends to breaking and tracing, so almost any function in the system can be broken or traced.) Since the entire system is implemented in Lisp, there are few places where the system's behavior depends on anything that you can't modify (such as a low level system implementation language).

While these techniques provide a fair amount of tailorability, there's a price: Medley is complex. There are many flags, parameters, and controls that affect its behavior. Because of this complexity, Interlisp tends to be more comfortable for experts, rather than casual users. Beginning users of Interlisp should depend on the default settings of parameters until they learn what dimensions of flexibility are available. At that point, they can begin to "tune" the system to their preferences.

Appropriately enough, even Medley's underlying philosophy was itself discovered during Medley's development, rather than laid out beforehand. The Medley environment and its interactive style were first analyzed in Sandewall's excellent paper [Sandewall, 1978]. The notion of "exploratory programming" and the genesis of the Interlisp programming tools in terms of the characteristic demands of this style of programming was developed in [Sheil, 1983]. The evolution and structure of the Interlisp programming environment are discussed in greater depth in [Teitelman & Masinter, 1981].

How to Use this Manual

This document is a reference manual, not a primer. We have tried to provide a manual that is complete, and that lets you find particular items as easily as possible. Sometimes, these goals have been achieved at the expense of simplicity. For example, many functions have a number of arguments that are rarely used. In the interest of providing a complete reference, these arguments are fully explained, even though you will normally let them default. There is a lot of information in this manual that is of interest only to experts.

Do not try to read straight through this manual, like a novel. In general, the chapters are organized with overview explanations and the most useful functions at the beginning of the chapter, and implementation details towards the end. If you are interested in becoming acquainted with Medley, we urge you to work through *An Introduction to Medley* before attempting this manual.

A few comments about the notational conventions used in this manual:

Lisp object notation: All Interlisp objects in this manual are printed in the same font: Functions (`AND`, `PLUS`, `DEFINEQ`, `LOAD`); Variables (`MAX.INTEGER`, `FILELST`, `DFNFLG`); and arbitrary Interlisp expressions: `(PLUS 2 3)`, `(PROG ((A 1)) ...)`, etc.

Case is significant: *In Interlisp, upper and lower case is significant.* The variable `FOO` is not the same as the variable `foo` or the variable `Foo`. By convention, most Interlisp system functions and variables are all uppercase, but users are free to use upper and lower case for their own functions and variables as they wish.

One exception to the case-significance rule is provided by the CLISP facility, which lets you type iterative statements and record operations in either all uppercase or all lowercase letters: `(for X` `from 1 to 5 ...)` is the same as `(FOR X FROM 1 TO 5 ...)`. The few situations where this is the case are explicitly mentioned in the manual. Generally, assume that case is significant.

This manual contains a large number of descriptions of functions, variables, commands, etc, which are printed in the following standard format:

(**FOO** *BAR BAZ*) [Function]

This is a description for the function named `FOO`. `FOO` has two arguments, *BAR* and *BAZ*. Some system functions have extra optional arguments that are not documented and should not be used. These extra arguments are indicated by "—".

The descriptor [Function] indicates that this is a function, rather than a [Variable], [Macro], etc. For function definitions only, this can also indicate whether the function takes a fixed or variable number of arguments, and whether the arguments are evaluated or not. [Function] indicates a lambda spread function (fixed number of arguments, evaluated), the most common type.

References

[Burton, et al., 1980]	Burton, R. R., L. M. Masinter, A. Bell, D. G. Bobrow, W. S. Haugeland, R.M. Kaplan and B.A. Sheil, "Interlisp-D: Overview and Status" — in [Sheil & Masinter, 1983].
[Sandewall, 1978]	Sandewall, Erik, "Programming in the Interactive Environmnet: The LISP Experience" — *ACM Computing Surveys*, vol 10, no 1, pp 35-72, (March 1978).
[Sheil, 1983]	Sheil, B.A., "Environments for Exploratory Programming" — *Datamation*, (February, 1983) — also in [Sheil & Masinter, 1983].
[Sheil & Masinter, 1983]	Sheil, B.A. and L. M. Masinter, "Papers on Interlisp-D", Xerox PARC Technical Report CIS-5 (Revised), (January, 1983).
[Teitelman & Masinter, 1981]	Teitelman, W. and L. M. Masinter, "The Interlisp Programming Environment" — *Computer*, vol 14, no 4, pp 25-34, (April 1981) — also in [Sheil & Masinter, 1983].