LISP370 files and the EXF operation.

Cyril N. Alberga

PREFACE

What follows is advice, not dictum. It is based on several years use of the YKTLISP system, mainly system building, but also some application programming. It assumes some familiarity with LISP in general, and LISP370 in particular. YKTLISP is the Yorktown version of LISP370, with numerous changes and amendations. This is one of a series of extant and projected notes on using YKTLISP.

CONTENTS

Introduction 1

Creating LISPLIBs 2
SUPV 2
EXF 2

Definitional Facility 4
Function definition 4
Object definition 4
Action definition 4
Environment of definition 5
Messages and value. 5
Status of output files. 5

Now I have a LISPLIB, what can I do with it? 7

INTRODUCTION

In YKTLISP we encourage a mode of system building which differs from that which is common to most LISP systems. (In fact we encourage two modes of system building, the one described herein, and the use of LISPEDIT, which is much superior.) The kernel of the advice is: "don't FILELISP until you really mean it".

YKTLISP supports files which are analogous to CMS TXTLIB files, (akin to other LISPs' FASL files). These files (traditionally of file-type LISPLIB, will be referred to as LISPLIBs herein) are random access, key addressed files of LISP objects, stored in a form which can be read faster than the forms read by the READ function. The keys are normally strings. Unlike files read by the READ function LISPLIBs may contain representations of compiled object code (binary program images, BPI's). In addition, each item has an associated flag, with a value of 0 to 255. Facilities exist for creating LISPLIBs, for "loading" the entire contents or selected items from them, and for copying, merging and pruning them.

At present all items are given flags of 0 or 1. The loading functions test the flag, and for a value of 0 perform an assignment of the item to the INTERNed key. For a value of 1, the item is passed to the interpreter to be evaluated. The later action allows load-time evaluations, which can be used to establish values in property lists, to create streams, and for any other action requiring values which pre-exist in the LISP system being augmented by the loading process.

CREATING LISPLIBS

While LISPLIBs can be created directly by the user (by means of the R... functions) they are generally created by use of the EXF operator. In order to describe how this is accomplished I must digress a bit. EXF, an earlier version of which is described in the LISP370 manual, interacts with the SUPV function and with a collection of functions including COMPILE, ASSEMBLE, FILEQ etc., which I will refer to as the definitional facility.

SUPV

SUPV is the "usual", system supplied READ-EVAL-PRINT loop. It is invokes with two arguments, an input stream and an output stream which it binds FLUIDly as CURINSTREAM and CUROUTSTREAM. It proceeds to read expressions from the input stream and evaluate them. The expression read and the value computed will be PRETTYPRINTed on the output stream.

EXF

EXF is called with arguments which specify a source for expressions, destinations for various values produced, and various "options" which control the production of values by the functions which make up the definitional facility. It should be remembered that EXF receives it's arguments unevaluated.

(EXF input <output <option-name option-value>* >)

The "input" argument specifies an input stream to be given to SUPV. A value of NIL specifies that the current CURINSTREAM is to be used. This is rarely used, but it allows manual entry of expressions while specifying various options for the definition facility. The other legitimate values for "input" are an identifier or a list of 1 to 3 identifiers. These are used as filename, filetype and filemode for an input file. The filetype defaults to LISP370 or LISP (with LISP370 preferred), while the filemode defaults to *.

The "output" argument specifies the output stream for SUPV. If it is missing it defaults to the current value of CUROUTSTREAM. If it is "=", a file output stream is created, with the same filename as the input stream, a filetype of EXF and a filemode of A1. (Note that (EXF () = ...) is illegal.) Otherwise, "output" must be an identifier or a list of 1 to 3 identifiers. This is interpreted in the same manner as "input", with default filetype of EXF and default filemode of A1. If the extant option list

contains an OUTPUTLENGTH entry, or if one is found in the option name/value part of the call to EXF, it is used to define the output file line length. Otherwise the line length defaults to 80.

If the CUROUTSTREAM at the time of the invocation of EXF is a console stream EXF provides a MESSAGE option with that stream as value. This stream is used by the definitional facility for informative and warning messages, unless suppressed by a QUIET option.

The only one of the option name/value pairs which is treated specially is FILE. It is used to define a LISPLIB file, with the value item interpreted in the same way as the "output" argument, with the filetype defaulting to LISPLIB. In addition, the presence of a FILE option in the absence of an explicit NOLINK option force a (NOLINK . T) option on the optionlist.

All remaining options are added to the option list, OPTIONLIST, which is bound FLUIDly by EXF. The options are as described in the LISP370 manual, with the addition of a QUIET option, which if ¬NIL suppresses most informational output to the console.

DEFINITIONAL FACILITY

The definitional facility consists of a collection of functions which are called solely for their side-effect. There are four groups, those which define functions, those which define objects, those which define actions and those which modify the environment of definition.

FUNCTION DEFINITION

Functions are defined by DEFINE, COMPILE, ASSEMBLE and COMP370. (See COMPILE notes.) These functions accept one or more function definitions ([M]LAMBDA expression in the cases of DEFINE COMPILE and COMP370, Lisp Assembler Programs in the case of ASSEMBLE), process them to varying extents, and make the resulting objects ([M]LAMBDA expressions or BPIs) available by assignment and/or writing them onto a LISPLIB. When a definition is added to a LISPLIB by any of these functions it is flagged with 0, the ASSIGN flag.

OBJECT DEFINITION

Objects are defined by FILEQ and SETANDFILEQ. These functions have the same form as SETQ, that is, their first argument is unevaluated. Their second argument is evaluated, and in the presence of a FILE option it is placed in the LISPLIB with a flag of 0 and a key derived from the first (unevaluated) argument. SETANDFILEQ, in addition, performs the assignment in the current context.

ACTION DEFINITION

Actions are defined by FILEACTQ and EVALANDFILEACTQ. These function have two unevaluated arguments. The first is an arbitrary key, for use in the presence of a FILE option. The second is any s-expression. The S-expression is placed in the LISPLIB (when present) with a flag of 1, to be evaluated at load time. EVALANDFILEACTQ, in addition, evaluates the expression in the current context.

An example of the use of FILEACTQ would be:

```
(FILEACTQ FOO (MAKEPROP "FOO "BAR "123))
```

This would cause the id FOO to be given BAR as a property with the value 123 at load time.

ENVIRONMENT OF DEFINITION

Function which modify the environment of definition (see the note on COMPILE) are ORTEMPDEFINE MATEMPDEFINE ORADDTEMPDEFS MAADDTEMPDEFS ORTEMPSETQ MATEMPSETQ and ADDOPTIONS. The first six of these add definitions to the compilers Operation Recognition environment (OR) or its Macro Application environment (MA). The definitions are added to the head of the respective environments in such a way that they are dropped upon exit from the invocation of EXF in which they are encountered. They do not replace pre-existing values of the same identifiers, but simply mask them, allowing the earlier definitions to become available again after the current file is processed. Thus, private definition for system functions or macros can be made available temporarily, e.g. for "bootstrapping" to a new system. The last (ADDOPTIONS) is described in the PDOM.

eeTEMPDEFINE has the same format as DEFINE. It allows macro or function definitions to be placed within the file that uses them.

eeTEMPSETQ allows arbitrary values to be added to the environments. The format is that of the SETQ.

eeADDTEMPDEFS has the same format as LOADVOL. It augments the environment with definitions and values loaded from a LISPLIB. Only items with flags of 0 are loaded, flag 1 (actions) are ignored. This is done in order to maintain the "push down" nature of these functions. There would be no way to undo the arbitrary actions possible with flag 1 items. Note that we still do not have a "hyper"garbage collector, thus BPIs, once loaded consume space forever (at least until YKTLISP is unloaded). Thus ADDTEMPDEFS must be used with care to avoid BPI space overflow. If you are willing to pay the cost of interpreting (as opposed to the cost of reloading YKTLISP) you should created LISPLIBs for ADDTEMPDEFS with symbolic definitions, using DEFINE, rather than COMPILE.

MESSAGES AND VALUE.

EXF binds (FLUIDly) a set of variables which are used to collect counts of compiler, assembler and macro-expansion errors. Upon receiving control back from SUPV (usually because of reaching EOF, or reading (FIN)), EXF reports the various error counts and then returns their sum as its value. Thus an error free EXF run will return a value of 0.

STATUS OF OUTPUT FILES.

If an EXF file is to be created any previously existing EXF file of the same name is first erased. If a LISPLIB is to be created and a file of the

same name already exists the new LISPLIB is created under the name TEMPLIB (any existing TEMPLIB file being erased first). In this case the pre-existing file is erased and the TEMPLIB renamed only if SUPV returns normally (whether or not errors have occurred). If control return to EXF via an UNWIND the TEMPLIB is left and the pre-existing file is retained.

LISPLIBs can be read and written directly, using the "R" (random-access) functions, RDEFIOSTREAM, RREAD, RWRITE, RTYPE, RSETTYPE, RSHUT, etc. All of these save RTYPE and RSETTYPE are described in the LISP370 manual. RTYPE returns the flag value for an item, while RSETTYPE sets (changes) it.

For housekeeping there are three functions, RDROPITEMS, RCOPYITEMS and RPACKFILE.

(RDROPITEMS file itemlist)

will delete all items whose keys are in the itemlist from the directory of the file, "file". Filetype defaults to LISPLIB, while filemode defaults to *. Remember, only the directory is changed, to actually shorten the file you must use RPACKFILE.

(RCOPYITEMS inputfile outputfile itemlist)

copies all items whose keys are in "itemlist" from the inputfile to the outputfile. Items with identical keys which pre-exist in the outputfile will be lost.

(RPACKFILE file)

copies the file "file", dropping any unreachable items, then erases the original and renames the copy.

In all of these functions (and the ones to follow) a file is named either by a single ID or by a list, (fn <ft <fm>>), e.g.

```
(RPACKFILE "MYJUNK)
(RPACKFILE "(MYJUNK LISPLIB C))
```

with ft defaulting to LISPLIB and fm to *. The elements of the "itemlist"s may be either strings or identifiers.

Finally (and most importantly) one can load all or some of the items in a LISPLIB or a set of related LISPLIBs.

(LOADVOL file)

will read all items from the designated file. Those with flags of 0 will be assigned to the identifier resulting from (INTERN key). Those with flags of 1 will be evaluated for side effects, and discarded. The value is a list of items loaded.

(SUBLOAD file idlist)

will load only those items whose keys are in "idlist". The flag values have the same meanings as in LOADVOL. The value is a list of two lists, the first is a list of items loaded, the second a list of items not found.

(LOADCOND file)

ignores all items with flag values of 1. Of the others only those are loaded which will not replace a pre-existing functional value (LAMBDA expression, BPI or FUNARG). Thus you may have two LISPLIBs which share certain function definitions, and by use of LOADCOND you may load only one copy of the shared functions. The value is a list of those items loaded.

(DEPLOAD (file*) itemlist)

will load all items from itemlist (ignoring flag 1 items), from which ever LISPLIB in the file list it finds them in. It then examines any BPIs that it has loaded and lists all the functions that they call. If any of these functions does not have a functional definition the set of LISPLIBs is searched for one, and if found, it is loaded. This function is checked in the same way. The value is a list of two lists, the first being the names of all items loaded, the second a list of items which could not be found.