F.G.H.
2/14/85

LispCourse #8: Tailoring Parameters, the File Package, Init Files

Tailoring Interlisp to your needs and desires: (FLGS, Parameters, etc.)

Parameters & FLGs

There are literally thousands of atoms in Interlisp whose values are used by the system and/or by various packages to decide how to "behave". By setting the values of these atoms, you can tailor Interlisp to your particular needs and to your particular style of interaction.

For example, the system uses the value of the atom DEFAULTPRINTINGHOST to determine what printer to use to print your output. SETQing DEFAULTPRINTINGHOST to 'Quake will make your output go to Quake, while SETQing it to 'Expresso will direct your output to Expresso. In general, most packages in Interlisp use the value of DEFAULTPRINTINGHOST to determine where to send their printed output. [There is no guarantee of this. But when it comes to printer output, packages are very well behaved.]

DEFAULTPRINTINGHOST functions a ***parameter*** that you set to specify how you want your system to behave. Its value can be the name of any available printer, or it can be a list of the names of several printers of different types (e.g., a Press, FullPress and an InterPress printer [e.g., (Quake LispPrint: Stinger)]).

Parameters that take on values of T (non-NIL) or NIL are generally known as FLGs since in general their names end in "FLG". For example, PROMPT#FLG determines whether or not the number appears before the back-arrow prompt in your top-level tty window. If PROMPT#FLG is T, the number is there (e.g., the prompt will be "12_"). If PROMPT#FLG is NIL, no number will be printed (e.g., the prompt will be "_").

FLGs are no different from any other parameters. They are just parameters used to make a simple Yes-No decision. FLGs are to parameters what predicates are to functions. Note also that not all parameters that function like FLGs have names ending in "FLG". So it goes.

Note that DEFAULTPRINTINGHOST is a system-wide parameter. Its value is used by Interlisp and all of the various packages.

Default Settings

All parameters and FLGs have default settings. You don't have to fool with a parameter unless you don't like the default setting!

For example, PROMPT#FLG is set to T by the system. You don't even have to know about PROMPT#FLG unless you want to get rid of those pesky little numbers, in which case you have to find out about PROMPT#FLG and set it to NIL.

Parameters like DEFAULTPRINTINGHOST really have to be set differently for every group of Interlisp users since they use different printers. BUT as a user you generally don't have to worry about it (unless your printer is down and you want to change where your printout goes) because it will be set to the proper value in your Init file (see below!!!).

Package-specific Parameters and FLGs

Almost all packages have their own set of parameters and FLGs that can be set to slightly (or sometimes not so slightly) modify their behavior.

Example: DEdit has 3 parameters that determine its behavior.

EDITEMBEDTOKEN ž Value is used as the special "embed token" described last class. Default value (used in examples last class) is &.

DEditLinger ž if T, then when you exit DEdit, the DEdit window will remain on the screen to be used for the next DEdit. If NIL, the DEdit window will be closed when you exit and reopened next time you call DEdit. Note this parameter applies only to the top-level call to DEdit and not to recursive calls which always close their windows. Default value is T. [Note: this is a FLG without "FLG" in its name.]

DEDITTYPEINCOMS ž value is a list that defines the control character commands that you can type in to DEdit. Default value is a list that contains definitions for the Ctrl-F and Ctrl-S described last time. Also contains a Ctrl-Z command not described last time. [Note: this is a parameter for real hackers only.]

Second Example: Chat has 5 parameters. (Note: Chat is a package we haven't yet covered, so if you aren't familiar with it, don't worry. Just look at the kind of things the parameters allow you to tailor).

CLOSECHATWINDOWFLG ž If non-NIL, every Chat window is closed on exit. If NIL, the initial setting, then the primary Chat window is not closed. Default value is *????*.

CHAT.FONT ž If non-NIL, the font that Chat windows are created with. If CHAT.FONT is NIL, Chat windows are created with (DEFAULTFONT 'DISPLAY) [To be covered in a later class!]. Default value is *????*.

DEFAULTCHATHOST ž The host to which CHAT connects when it is called with no HOST argument. Default value is *????*.

CHAT.ALLHOSTS ž A list of host names, as uppercase litatoms, that the user desires to Chat to. Chatting to a host not on the list adds it to the list. These names are placed in the menu that the background Chat command prompts with. Default value is *????*.

CHAT.DISPLAYTYPE ž The type of display (a number) that Chat should tell the remote host the user is on. If Datamedia emulation is desired, this variable should be set to the number corresponding to the terminal type Datamedia for the remote host. If the remote host does not respond to the terminal type protocol in Pup Telnet, this variable is irrelevant. Default value is 10.

How to find out about available parameters and FLGs.

Package-specific parameters

Package-specific parameters are usually included in the package documentation.

For example, the DEdit parameters described above are described under the heading "DEdit Parameters" in the DEdit section of the Interlisp Reference Manual (see Section 20.1.4). The Chat parameters are described at the end of the Chat section of the Interlisp Reference Manual (see Section 20.5). They have no

heading, but are preceded by the sentence: "The following variables control aspects of Chat's behavior."

If you want to change the behavior of some particular package, look in the documentation for that package. You will almost certainly find a list of parameters. To find the documentation of a package is another matter. Some are documented in the IRM (Chat, DEdit, TEdit, etc.), the rest are documented in separate documents usually stored in the same directory as their LOAD files. You just have to poke around to find them.

System-wide parameters

Good luck!

Documentation for system-wide parameters is scattered all over the IRM. Moreover, there is no independent listing or description of them in esistence ANYWHERE.

If you want to change some system behavior, your only chance is to poke around the IRM, looking for where that behavior is described. Sometimes along with the description there will be a description of some interesting parameters.

You might expect DEFAULTPRINTINGHOST to be documented in the INPUT/OUTPUT chapter (#6) of the IRM. After all most of the printing functions are described there. But in fact, DEFAULTPRINTINGHOST is documented in the Interlisp-D Sepcifics chapter (#18). This fact was easy to discover because I could look up DEFAULTPRINTINGHOST in the index. But if I wanted to find "the parameter that changes my default printer". Hah!

Learn to do the following:

> Accept things as they are and don't mess with parameters.
>
> Wander around the IRM looking for interesting parameters and FLGs.
>
> Look at the Init files from lots of experienced users. They are a gold mine of parameters being set.
>
> When you see someone whose system behaves differntly from yours, ask how they did that.

Ask your local Interlisp wizard lots of questions.

Dream up and implement a good, permanent solution to the problem of managing and documenting the thousands of system parameters. Parameters are great, but only if you can find out about them.

Some More Interesting System-Wide Parameters and FLGs

INITIALSLST ž a list of the form ((*UserName1 FirstName1 Initials1*)(*UserName2 FirstName2 Initials2*) ... (*UserNameN FirstNameN InitialsN*)). The system makes sure that if UserNamei is logs in, then various parts of the system will use FirstNamei as the users first name and Initialsi as the users initials. For example, my Init file sets this parameter to ((Halasz Frank fgh:)). Thus the system says "Hi, Frank." when is comes up with a clean sysout. It also puts my initials "fgh:" in a comment in the beginning of evey function that I edit. Note the INITIALSLST must be set in your Init file or you must run "(SETINITIALS)" after setting INITIALSLST to a new value so that the system recognizes the changes INITIALSLST.

LOGINHOST/DIR ž the host and directory that is considered to be your home directory by various programs. E.g., my Init file sets LOGINHOST/DIR to be {PHYLUM}<HALASZ>.

DIRECTORIES ž a list of host/directories that the system will look on when it can’t find a file. Used by the LOAD function and certain other functions, but not by TEdit, COPYFILE and many other functions. Ususally you just want to add on to the default value of this list. For example, NoteCards does the following when it comes up: "(SETQ DIRECTOIRES (CONS ’{PHYLUM}<NOTECARDS>CURRENT> DIRECTORIES))". This just makes sure that the NoteCards directories are searched along with all the standard Lisp ones when the system can’t find a file.

WINDOWTITLESHADE ž the shade that the right part (i.e., the part to the right of the title) of every window title bar will be. Value can be any shade (i.e., a number less than 65535) but the values of

BLACKSHADE, WHITESHADE and GRAYSHADE are particularly easy shades to use.

???? and many, many more I can't think of now.

The File Package

Introduction

When you DEFINEQ a function, the function definition is entered into the system's virtual memory.

When you SETQ an atom to a value, the atom-value pair is entered into the system's virtual memory.

When you apply a function or when you evaluate an atom, the system just looks up the function definition or atom in its virtual memory.

All is fine and dandy UNTIL your virtual memory is destroyed; for example, when you copy a clean sysout to your disk and restart Lisp or when your machine bombs leaving an inconsistent vmem.

When your vmem goes away, the effect of your DEFINEQs and SETQs goes away as well. When a new sysout is loaded (i.e., a "new" vmem is started), you have to redefine all the functions and reset all the variables.

You could type all those DEFINEQs and SETQs again. But there is an easier way: the File Package.

The File Package helps you write out on a file all the Lisp function calls necessary to redefine a given set of functions and/or to reset the values for a given set of variables.

Then when you load a new vmem (or for that matter, in another person's vmem), LOADing this file will cause these DEFINEQs and SETQs on the file to be evaluated just as if you had typed them in, thus redefining all the functions and resetting the values of all the variables.

So LOADing is just the process of evaluating in a bunch of DEFINEQs and SETQs from a file rather than from the user's type-in.

MAKEFILE and LOAD, with LISTFILES too.

There are two major functions in the File Package:

MAKEFILE ž makes a LOADable file. The first argument specifies the name of the file to be made. The making of a LOADable file is controlled by the files COMS list described in the next section.

LOAD ž reads in a LOADable file made by MAKEFILE and evaluates all the Lisp function calls thereon. The first argument is the name of the file.

A third, handy function is **LISTFILES** which prints a LOADable file on your DEFAULTPRINTINGHOST with a handy little index included. Note that LISTFILES is a special form (NLambda function).

The COMS list

Introduction

The COMS list contains a set of commands that the File Package uses to determine what functions, variables, and other Lisp objects you want to appear on a file made by MAKEFILE.

Each LOADable file (i.e., file made by MAKEFILE) has its own COMS list.

Terminology: The root part of a file name is the file name without its extension. For example, the root part of EXAMPLES.LISP is EXAMPLES.

The COMS for a file is the value of the atom constructed by adding COMS on to the root of its file name.

For example, if the name of a file is EXAMPLES, then the COMS for that file is the value of the atom EXAMPLESCOMS.

Note this limits your choice of file names a bit. You cannot have two LOADable files with the same root part. E.g., you can't have an

EXAMPLES.OLD and an EXAMPLES.NEW because they both would want to have the value of EXAMPLESCOMS as their COMS list and hence would interfere with each other.

Warning: This is one place where Interlisp is case sensitive. All loadable files must be named in ALL CAPS. The File Package does not handle non-CAPS well at all. Your file name must be EXAMPLES and you COMS must be EXAMPLESCOMS; Examples and ExamplesCOMS will not work!

The structure of a COMS list

A COMS list has the following form:

((*FilePackageCommand1 Arg1.1 Arg1.2 ... Arg1.P*)
(*FilePackageCommand2 Arg2.1 Arg2.2 ... Arg2.Q*)
...
(*FilePackageCommandN ArgN.1 ArgN.2 ... ArgN.R*))

Each clause in this list tells the FilePackage to write something on the file. What is written is determined by the *FilePackageCommandi* that is the CAR of the clause and by the *Args* that are in the CDR of the clause.

There are many possible *FilePackageCommands*.

The most important are:

FNS ž tells the File Package to write function definitions on the file. The FP essentially write a DEFINEQ for every function named in the rest of the list.

Example: (FNS CountList CountAtoms CloseWindowList) tells the FP to write DEFINEQs for the three functions onto the file.

VARS ž tells the file package to write the values of variables on the file. The FP essentially writes SETQs for every variable named in the *Arg* items in the rest of the list.

If a *Arg* is simply an **atom**, then the FP will write a SETQ that sets the atom to the value it had WHEN THE FILE WAS WRITTEN.

So if *Arg* is simply MyVariable and if MyVariable has the value 12 when the file is written, the FP will write (SETQ MyVariable 12) onto the file. When the file is LOADed, MyVariable will be set to 12. BUT if MyVariable had the value 77 when the file was written, the SETQ would be (SETQ MyVariable 77) , with the corresponding change when the file was LOADed.

If a *Arg* is a **list**, then the FP will CONS a SETQ onto this list and write the result down onto the file. For example, if *Arg* is (MyVariable 5). The FP will write (SETQ MyVariable 5) onto the file so that MyVariable will always be set to 5 when the file is LOADed.

For example, (VARS DEFAULTPRINTINGHOST (LOGINHOST/DIR '{PHYLUM}<HALASZ>) (FOUR (PLUS 2 2))) is a FP COMS clause.

ADDVARS ž like VARS except that each *Arg* is a list of two items, the CAR is a variable name and the CDR is a value. ADDVARS write the necessary Lisp code on the file so that when the file is LOADed, the value is added to the list that is already the value of the variable.

Example: (ADDVARS (DIRECTORIES {phylum}<notecards>current>)) .

Then when the file is LOADed, the atom {phylum}<notecards>current> will be added to the list that is already the value of DIRECTORIES. If DIRECTORIES had the value ({PHYLUM}<HALASZ> {ERIS}<LISP>) before the LOAD, then after the LOAD it would have the value ({phylum}<notecards>current> {PHYLUM}<HALASZ> {ERIS}<LISP>).

FILES ž tells the file package to write load commands for some other files on this file. The FP essentially writes LOADs for every files named in the rest of the list. When the file is LOADed, it will cause these other files to be LOADed as well.

For example, (FILES NOTECARDS1 NOTECARDS2 NOTECARDS3 NOTECARDSREST) might be a clause in the COMS file for NOTECARDS.LSP. Then whenever NOTECARDS.LSP was LOADed, the files NOTECARDS1, NOTECARDS2, NOTECARDS3, and NOTECARDSREST would also be loaded.

P ž tells the FP to print each *Arg* item in the rest of the list out on the file. Then when the file is LOADed, these items will be evaluated.

Example: (P (LAFITE 'ON)(PRINT "Hello, Tiger")(CLOSEW LOGOW)) tells the FP to write (LAFITE 'ON), (PRINT "Hello, Tiger"), and (CLOSEW LOGOW) on the file. Then when this file is LOADed, these function calls will be executed causing LAFITE to be turned on, "Hello, Tiger" to be printed in the tty window, and the Interlisp-D logo window to be closed.

***** ž tells the FP that this is a comment. * can be used to put comments in your COMS lists so that they are more understandable.

Note that there can be as many FNS clauses as you like in a COMS list. They can be in any order as well. It usually helps to group your functions into FNS clauses in a way that makes some structural sense to you. Ditto for VARS, P, ADDVARS, *.

What does MAKEFILE do?

MAKEFILE looks at the COMS list for the file it is making and simply follows the instructions there.

Actually, the first thing it does is equivalent to (VARS *File*COMS). That is it writes out a SETQ so that the COMS for *File* is reset when it is LOADed. So the first thing you always see in a LOADable file is the SETQ for the file's COMS list.

Then, MAKEFILE follows the FP clauses in the COMS list in order. If the COMS says FNS, MAKEFILE writes out DEFINEQs. If the COMS says VARS, the FP writes out SETQs. And so on.

When its done, it returns the name of the file it just made.

How does LOAD work?

LOAD just reads in the file and evaluates each item (DEFINEQ, SETQ, etc.) in the file, just like "normal" Lisp reads the user's type-in and evaluates it.

How does the user work with the FP?

1. Define a few functions ž say, CarOfListItems, CountList and CloseWindowList.
2. SETQ a few variables ž say, AtomList, WhoCaresList and FranksAge
3. Create a COMS for this file, which we'll call EXAMPLE.LSP.

```
(SETQ EXAMPLECOMS
      '((FNS CarOfListItems CountList CloseWindowList)
        (VARS AtomList (WhoCaresList (LIST 'Who 'Cares))
              FranksAge)))
```

4. Evaluate "(MAKEFILE 'EXAMPLE.LSP)", creating the file EXAMPLE.LSP.
5. Define a new function, say ExampleFunction.
6. Evaluate "(DC EXAMPLE)" which will call DEdit on EXAMPLECOMS.

   You can then edit the EXAMPLECOMS list. In particular, add a new clause (FNS ExampleFunction) or add ExampleFunction to the already existing FNS clause.

7. Redo Step 4 to make an updated version of EXAMPLE.LSP.
8. Get a new sysout, i.e., wipe out your vmem.
9. Try to evaluate (CountList '(A B C)). You will get a "undefined function" error because in your clean vmem CountList has not yet been defined.
10. Evaluate "(LOAD 'EXAMPLE.LSP)".
11. Try to evaluate (CountList '(A B C)), resulting in "3". CountList was defined during the LOAD.
12. Change CountList to return 2 times the list length.
13. Evaluate "(MAKEFILE 'EXAMPLE.LSP)" to update EXAMPLE.LSP with this change. No need to remake the EXAMPLECOMS because it was properly set during the LOAD.
14. Define a new function, say LastExample.
15. Evaluate "(DC EXAMPLE)" and edit the COMS to add LastExample to some FNS clause.
16. Evaluate "(MAKEFILE 'EXAMPLE.LSP)" to update EXAMPLE.LSP with this change.

17. You get the picture.

Example LOADable File

A LISTFILES of the EXAMPLE.LSP file (created by "(LISTFILES EXAMPLE.LSP)") is in the Appendix.

The first page is the handy index created by LISTFILES for your convenience in handling large files. The MAKEFILE output really starts on the second page.

IMPORTANT NOTE: There are no SETQs. Thats because I've been lying a bit. The FP uses the functions RPAQ and RPAQQ instead of SETQ and SETQQ. Just translate in your head, the RPAQs and RPAQQs into SETQs and SETQQs. For all practical purposes they are the same.

Note that the first variable set in the file is in fact the EXAMPLECOMS.

The FP has ways to make life really easy

FILES?

The function (FILES?) will list in your tty window all functions, variables and other Lisp objects that you have defined but not saved using a MAKEFILE. It will then ask you if you want to say where these things go. If you say Yes, it will ask for each function and variable to what file it belongs. You give it a file name, and it will add it to the appropriate clause in the COMS for the file name.

You can even, specify a new file name and it will create the appropriate COMS list from scratch.

If you give NIL as the file name, it will not add the function or variable to any COMS and it will never ask you about it again.

If you simply type a <return>, it will not add it to any COMS, but will ask again the next time you call FILES?.

MAKEFILES

Does a FILES? to capture all the orphans.

Then for each file that has to be updated for any reason, does a MAKEFILE.

Note: Some users exist on FILES? and MAKEFILES alone and never touch a COMS directly. Other (like me) prefer to keep track of our own changes, use DEdit on the COMS, and then call MAKEFILE ourselves. The choice is yours to make.

Final Note on the FP

I have described about 20% of the FP. But its probably all you need to know for a while.

The FP is documented in glorious detail in Chapter 11 of the IRM.

Section 11.7 is probalby the most interesting because it describes the FilePackageCommands you can put in your COMS lists.

Sections 11.1 and 11.2 cover LOAD and MAKEFILE et al.

At Last, Init Files

An Init file is just another LOADable file. Nothing mysterious about it all. In fact, as LOADable files go its quite dull!

An Init file is a special LOADable file for the following reason:

When Lisp starts up from a clean sysout, the first thing it does is evaluate the function "(GREET)".

GREET immediately goes out and finds a file called INIT.LISP on your local disk (if it can't find it, it asks you to specify where it is).

GREET LOADs INIT.LISP.

Then GREET tries to find another Init file, one that is somehow associated with your login. It looks in a number of places like on your directory on your file server.

The first Init file it finds, it LOADs.

So, Init files are just LOADable files that Lisp always LOADs the first time it comes up froma clean sysout.

There are two Init files:

1) A **site-specific Init file** that is common to all users at a given site (e.g., ISL, KSA, etc)

This init file sets things like DEFAULTPRINTINGHOST, DIRECTORIES, FONTDIRECTORIES, CH.NET.HOSTS, etc. All those parameters that have to do with what printer to use, what file server to use, what your ethernet looks like, where to find the local copies of the Lisp files, AND where to look for the user specific init files.

At PARC, the site-specific init files are located on {eris}<lisp>harmony>basics>init.cis (for ISL) and init.ksa for KSA. The appropriate file should be copied to your local disk and renamed to be INIT.LISP.

2) A **user-specific Init file** that is setup by each user as he or she pleases

This init file is generally used to set all those parameters and FLGs to make the system behave "correctly", to make the screen look pretty, to load the files for all the packages that are frequently used, etc.

My Init file has four basic clauses:

A **VARS** clause that sets parameters like INITIALSLST, WINDOWTITLESHADE, DeditLinger, CHAT.FONT, CLOSECHATWINDOWFLG, etc. to my liking.

An **ADDVARS** clause something like: (ADDVARS (DIRECTORIES {PHYLUM}<HALASZ>LISP>)(DIRECTORIES {PHYLUM}<NOTECARDS>RELEASE1.1>))

A **FILES** clause that loads CROCK, ARCHIVETOOL, SKETCH, BITMAPFNS, FILEBROWSER, and other such optional packages from either {eris}<lispusers> or from {eris}<lisp>harmony>library>.

A **P** clause something like: (P (CROCK (CREATEREGION 816 687 128 115))(CLOSEW LOGOW)(LAFITE 'ON)) which causes CROCK to start up, the Logo window to close and Lafite to start up when my Init file is LOADed.

At PARC, your Init file should be named with one of the following names:

{ERIS}<user>LISP>INIT.DCOM, {ERIS}<user>LISP>INIT, {ERIS}<user>INIT.DCOM, {ERIS}<user>INIT.LISP,{PHYLUM}<user>LISP>INIT.DCOM, {PHYLUM}<user>LISP>INIT, {PHYLUM}<user>INIT.DCOM, {PHYLUM}<user>INIT.LISP, {IVY}<user>LISP>INIT.DCOM, {IVY}<user>LISP>INIT, {IVY}<user>INIT.DCOM, {IVY}<user>INIT.LISP, where user should be replaced by your login name.

Commonly asked questions:

How do I change my init file?

Use "(DC INIT)" to DEdit the INITCOMS, then do a (MAKEFILE '{PHYLUM}<user>LISP>INIT) or (MAKEFILE '{IVY}<user>INIT.LISP) or whatever.

How do I start an Init file?

You have two choices:

Create an INITCOMS from scratch using (SETQ INITCOMS '((FNS ...))), then edit it using DC, then do a MAKEFILE.

Copy someone elses INIT file to your directory, load Lisp, edit the INITCOMS using DC to change all the "bad" parameters settings, then do a MAKEFILE to update the INIT file with your changes.

What do I put in my Init file?

Whatever you like. Generally, you set parameters for the system and specific packages you commonly use. You also load your commonly used packages.

Don't keep your own functions in your init file. Put your functions in a file called MYUTILITIES.LISP or some such, and then put a (FILES MYUTILITIES.LISP) clause in your INITCOMS so this file gets LOADed when your Init file gets LOADed.

Final Comments:

1. You don't need Init files. The system runs fine without them. Though, it can't find printers, etc. because DEFAULTPRINTINGHOST, etc, haven't been set. The system will run perfectly without your personal Init file. You might have to load some packages by hand every time the system gets reloaded, but ...

2. If you want to take over an already loaded system from someone else, you can call (GREET) yourself. This will "ungreet" the previous person and then redo the GREET sequence with your Init file.

References

The IRM, Chapter 11

{ERIS}<LISP>HARMONY>DOC>INITFILES.TEDIT

The Init file documentation from the Sysdoc group.

Exercises

Define a few fucntions like CountAtoms and then save them on a file. Get a new sysout and try LOADing the file. Etc.

Make yourself a nice, interesting Init file.