****** The IMTEDIT IM-to-TEDIT translation program. ******

author: Michael Sannella
file: IMTEDIT.DCOM
loads file: IMTRAN.DCOM
related files: IMTOOLS.DCOM

IM format is the text formatting language that the Interlisp Reference Manual is represented in. It is somewhat like TEX source code, in that there are keywords, and brackets are used to delimit text. However, IM format was specifically designed for representing the Interlisp Manual, so the "text objects" used are semantically meaningful objects within the manual, such as "function definition", "lisp code", "subsection". IM format is described in detail below.

IM format files are easy to edit using Tedit, but they don't look very pretty. To produce the manual, use the function IM.TEDIT, which translates IM format files to formatted Tedit text streams. These text streams can be proofread and edited by the user, or automatically printed.

A useful feature of IM.TEDIT is that is very forgiving about errors in IM format, even misplaced bracket errors! This is not to say that the output will be pretty, but at least the translation program will not bomb out on you.

***** Formatting a File with IM.TEDIT *****

To translate an IM format file into a formatted Tedit text stream, use the following function:

(IM.TEDIT *INFILE.NAME OUTFILE.FLG***)** [Function]

This function takes an IM format file, and produces a formatted Tedit text stream. *INFILE.NAME* is the name of an IM format file. *OUTFILE.FLG* determines what happens to the translated textstream. If *OUTFILE.FLG* = T, the Tedit textstream is returned by IM.TEDIT. A Tedit window showing the translated document can be created by typing **(TEDIT (IM.TEDIT** *xxx* **T))**. If *OUTFILE.FLG* = NIL, the document is immediately sent to the printer. If *OUTFILE.FLG* = anything else, it is taken as a file name for the Interpress file which is created <but not printed>.

As IM.TEDIT runs, it prints out warning messages. These messages are also saved in the file <infile>.IMERR.

Important note: It is necessary to understand that IM.TEDIT produces a totally separate document from the IM format original. Any edits to this document will NOT be reflected in the original. In general, all of the editing should be done to the IM format files, to insure that there are no inconsistancies.

Note: If IM.TEDIT is called with *OUTFILE.FLG* = T to produce a textstream, and **IM.INDEX.FILE.FLG** (below) is set to T to add index information to the textstream, the textstream cannot be stored or printed. **IM.INDEX.FILE.FLG** should only be set to T if IM.TEDIT is called with *OUTFILE.FLG* not T, so it automatically prints the document to a printer or interpress file.

***** Variables Affecting IM.TEDIT *****

The operation of IM.TEDIT affected by a number of variables:

IM.NOTE.FLG [Variable]
If T, notes will be printed out, otherwise they will be suppressed. Initially NIL. (Note: If this is T, the translation programs will print out a message to remind you that notes will be printed.)

IM.DRAFT.FLG [Variable]
If T, the output will have "--DRAFT--" and the date printed on the top and bottom of every page. Initially NIL.

IM.CHECK.DEFS [Variable]
If T, checks whether variables and functions are bound/defined in the current Interlisp environment, and prints a warning if not. For functions, will also check arg list consistancy. Initially NIL.

IM.EVEN.FLG [Variable]
If T, IM.TEDIT will add an extra page at the end of the file saying "[This page intentionally left blank]". This can be used if you need a document with an even number of pages (for double-sided copying). Initially NIL.

The following FLGs are only of interest when generating an index:

IM.INDEX.FILE.FLG [Variable]
If T, index information will be added to the formatted Tedit text stream, and the file <infile>.IMPTR will be generated containing index information when the formatted Tedit textstream is printed. Initially NIL.

IM.REF.FLG [Variable]
If T, the translation program will try to resolve cross-references by looking at various hash tables. Initially NIL.

IM.SEND.IMPLICIT [Variable]
If T, send "implicit references" for functions and variables (if not in index hash array). Initially NIL. {fn...} or {var...} objects generate "implicit references" if they are not contained in the index hash tables. This can be used to find variables and functions that are not formally defined, but only mentioned.

***** Producing an Index and Table of Contents *****

The file IMTOOLS.DCOM contains functions for gathering index information, generating an index and a table of contents, and using index information to resolve cross-references.

Currently, these tools are rather primitive. Eventually, it is hoped that they will be improved to make this process easier.

To produce an index and TOC:

(1) Evaluate (SETQ IM.INDEX.FILE.FLG T), which tells IM.TEDIT to create index pointer files <XX>.IMPTR.

(2) Run (IM.TEDIT <file> <ipfile>) on all files in the document

(3) Evaluate (INIT.INDEX.VARS) to clear the index.

(4) For each of the files in the document, load the index information by evaluating (GRAB.IMPTR xxx.IMPTR).

(5) Generate an index by evaluating (MAKE.IM.INDEX OUTFILE.FLG), where the argument OUTFILE.FLG is interpreted the same as in IM.TEDIT.

(6) Generate a table of contents by evaluating (MAKE.IM.TOC OUTFILE.FLG), where the argument OUTFILE.FLG is interpreted the same as in IM.TEDIT.

To generate formatted files with cross-references resolved, do the following:

(1) Evaluate (SETQ IM.INDEX.FILE.FLG T), which tells IM.TEDIT to create index pointer files <XX>.IMPTR.

(2) Run (IM.TEDIT <file> '{NULL}FOO.IP) on all files in the document

(3) Evaluate (INIT.INDEX.VARS) to clear the index.

(4) For each of the files in the document, load the index information by evaluating (GRAB.IMPTR xxx.IMPTR).

(5) Evaluate (SETQ IM.REF.FLG T), which tells IM.TEDIT to resolve cross references using the loaded index information.

(6) Format all of the files using (IM.TEDIT <file> <ipfile>).

Note that creating a formatted file with cross-references requires formatting the file twice.

The process of formatting a large set of documents such as the IRM can automated using the following function:

(DO.MANUAL *CHAPNAMES MAKE.INDEX.FLG GET.REFS NO.IP.FLG***)** [Function]
Formats the IRM manual chapters specified by CHAPNAMES, producing Interpress files, "imptr" files (explained below), and error files. Uses the global variable **IM.MANUAL.DIRECTORY** (initially {erinyes}<lispmanual>) to indicate where the IRM files are taken from, and where the processed files should go.

Note: **DO.MANUAL** initially puts all files on {DSK}, and then copies them to the value of **IM.MANUAL.DIRECTORY**. Therefore, you should have at least 3000-4000 free pages on DSK.

"IMPTR" files are files with the extension "**IMPTR**" (such as ChapLitatoms.IMPTR), which contain index info, including the page number where each index appears. These are generated whenever a chapter is formatted. These must be read in to generate an index, or format a chapter resolving cross-references.

CHAPNAMES indicates which chapters should be processed. If *CHAPNAMES* = a list of chapter file names such as (ChapLitatoms ChapStack), only these chapters are formatted. If *CHAPNAMES*=**T**, all chapters in the IRM are formatted. If *CHAPNAMES*=**NIL**, all chapters that have been modified since the corresponding Interpress files were made are formatted. The global variable **IM.MANUAL.CHAPTERS** specifies all of the chapters, and all of the sub-files in each chapter, and the chapter numbers.

If *MAKE.INDEX.FLG* is non-**NIL**, after processing the chapters, the index, title pages, and many tables of contents are generated. Note that if this is done, all of the IMPTR files for all of the chapters are loaded again.

If *GET.REFS* is non-**NIL**, all of the IMPTR files are loaded before any of the chapters are processed, and all cross-references in the chapters are resolved.

If *NO.IP.FLG* is non-**NIL**, the chapters are formatted to the interpress file **{NULL}FOO.IP**, and thus not kept. This can be used when you simply want to generate all of the IMPTR files, but do not want the interpress files.

Ideally, all you should need to do to re-format the IRM is:

(DO.MANUAL T NIL NIL T) --- to generate the IMPTR files

(DO.MANUAL T T T) -- to generate the Interpress files, and the index.

If **DO.MANUAL** doesn't do exactly what you want, the code is fairly self-explainatory.

***** Modifying the dimensions of the formatted pages *****

Note on modifying the formatting tools for different paper sizes: All of the lengths that IMTEDIT uses (left margin size, page height, etc.) are stored in global variables, set in the filecoms of IMTEDIT. IMTEDITCOMS also includes comments documenting the meaning of each of these global variables. Modify these variables until you get what you want.

***** Known problems with IMTEDIT *****

Currently, IM.TEDIT produces a good-looking document. However, there are some features that I was not able to provide, because Tedit did not provide the formatting capability. These are as follows:

(1) Footnotes. Tedit does not supply footnotes. Currently, I "fake" footnotes by positioning them in-line after the paragraph wherein they were created. They also will not appear within definitions, tables, or lisp code.

(2) Tables. Tedit doesn't do table formatting. Currently, IM.TEDIT simply prints out the items in the table without formatting.

****** IM Format Description *****

This is a complete description of the syntax and vocabulary of "IM format". The general syntax is not likely to change, but the vocabulary will probably be extended as the Interlisp Manual is edited, and we discover new text objects that we need.

IM Format Syntax

****

An IM-format file consists of a linear string of visible, editable characters (and Tedit image objects, which are treated like characters). No funny control characters are allowed. "Text" is defined to be a linear string of characters, organized into paragraphs (delimited by blank lines), interspersed with any number of "Text Objects". Text Objects are used to specify the meaning of various pieces of text, which may be processed and formatted in different ways.

Text Objects can be divided into two types: those that take a single unnamed argument, and those that take a number of labeled arguments. All of the 'arguments' to Text Objects can be arbitrary text, organized in paragraphs, and including sub-text-objects nested to any level.

Text Objects within a file and arguments within a text object are specified using the characters "{" and "}". These characters are ALWAYS interpreted as Text Object delimiters or Text Object argument delimiters---there are special text objects for indicating that you really want a left or right bracket character as part of your text.

The format of Text Objects is:

{<TOname> <TOarg>} - single argument TO
{<TOname> {<TOargname1> <TOarg1>} {<TOargname2> <TOarg2>} .... } - multi-argument TO

Between a TOname or TOargname and its TOarg, and between named TOargs, there can be any combination of spaces, tabs, and CRs. These can be used to make your file look better.

In order to help with the problem of matching brackets in large TOs (such as SubSecs, which can extend over many pages), I have introduced a new piece of syntax: Begin and End:

"{Begin <TOname> <tag>}" is treated exactly like "{<TOname> "
"{End <TOname> <tag>}" is treated exactly like "}"
except that additional checks are made when the file is processed that the "Begin"s and "End"s match up. The (optional) "<tag>"s can be used as an additional check, to distinguish between different TOs of the same type. If you use a Begin TO, you should use a matching End TO.

Note that Begin and End TOs can be used with TOs that have labeled args, for example:

{Begin SubSec <tag>}
{Title ----}
{Text
.
.
}
{End SubSec <tag>}

In order to allow the IM format translation program to recover from errors (like missing brackets), information has been included in the program about what TOs "should" appear in other TOs. In general, "complex" TOs such as LabeledLists can only appear in other complex TOs. "Simple" TOs such as Lisp can appear within anything. These rules are defined on a per-argument basis for TOs with multiple labeled arguments --- for example, a LabeledList can appear within the "Text" argument of a FnDef, but not within the "Name" argument. This feature should not cause any problems, as long as TOs are used "reasonably."

Other rules followed by the translation program: FnDef's, VarDef's, Def's, etc can only appear at the "top-level", or inside a SubSec or Chapter. (this rule is very useful, because it helps trap many bracket errors before they propogate too far) Footnotes may not be nested.

New feature <which may get changed>: If a TO name is unrecognized, it will be printed out surrounded by brackets. For example, {FOO} will actually print as "{FOO}". This allows simple expressions to be bracketed without actually using the "{bracket ...}" TO.

Text Objects currently defined:
****

(note: the order of arguments in multi-argument TOs IS significant.)

(note: No distinction is made between upper and lower case in the TO names and the argument names, or in Begin/End tags)

(note: Many TO names and argument names have synonyms. These are indicated below.)

********** Plain Text Text objects ******

<paragraph>
All plain text is organized into paragraphs, delimited by blank lines.

{Chapter {number <number>} {title <title>} {text <text>} }
Specifies the number, title, and text of a chapter. If the number is not specified, the IM format translator will ask you to supply a number.

{subsec {title <title>} {text <text>} }
This can be used to generate sections, subsections, etc. to any depth. Heirarchical numbering is done automatically.

{Comment <text>}
Used to insert comments (which won't appear in the final formatted output).

{Note <text>}
Inserts comments that may be printed in the final formatted output, depending on the value of the variable IM.NOTE.FLG. Should be used for comments such as {Note I should write something about X here}

{foot <text>}
Generates a footnote. Footnotes may not occur within other footnotes. Currently, Tedit does not support footnotes, so these are just printed after the paragraph in which they appear.
Synonyms: footnote -> foot

{sub <text>}
Subscripts <text>.

{super <text>}
Superscripts <text>.

{it <text>}
Used to italisize pieces of text.
Synonyms: italics, emphasize -> it

{rm <text>}
Used to print text in the default, "roman" font.

{lisp <text>}
The text object for normal single-line references to lisp code. This is used for writing things like: "Obviously, {lisp (CONS 'A 'B)} evaluates to {lisp (A . B)}."

{lispcode <text>}
The text object for multiline lisp code, which do not appear in the middle of text, and have to be formatted differently. Spaces, tabs, and carriage returns are significant within <text>.

********** TOs for Interlisp manual objects ******

{FnDef {Name <name>} {Args <args>} {Type <keywords>} {Text <text>} }
This is used to define all lisp system functions. It needs to know the name of the function, and the args, and the text of the function description. (If the function has 0 args, the {Args --} argument may be omitted.) {Type ...} is an optional argument used to specify the argument type of the function. If the keywords NLambda or NoSpread (case doesn't matter) are included in <keywords>, the function is specified to have the corresponding argument type.
Synonyms: FnName -> Name
FnArgs -> Args
FnType -> Type

{vardef {name <name>} {text <text>} }
Used to define system variables.

{propdef {name <name>} {text <text>} }

Used to define property names.

{MacDef {Name <name>} {Args <args>} {Type <keywords>} {Text <text>} }
Like FnDef, but for macros.

{arg <name>}
Used to talk about an abstract argument to a function, such as "x" or "y" or "number". It is used primarily within function definitions.

{fn <name>}
Used to talk about the name of a function. i.e.: "...it calls {fn CONS} to do something."

{var <name>}
Used to talk about a system variable.

{prop <name>}
Used to talk about a property name.

{Mac <name>}
Like Fn, but for macros.

{EditCom <name>}
Used to talk about an edit command.

{BreakCom <name>}
Used to talk about a break command.

{PACom <name>}
Used to talk about a programmer’s assistant command.

{FileCom <name>}
Used to talk about a file package command.

********** Indexing Text objects ******

The following TOs (for indexing, defining, and referencing) all deal with objects with specific "object-types." The Interlisp Manual contains a very large number of names (CONS, NIL, FOO, etc). It is not enough just to list the name in the index --- it is also important to indicate WHAT the name is (a function, a variable, an error message, etc.) An "object-type" is essentially the description that would be printed in the index to describe a particular name. In IM format, such a description is given by a list of words, within parenthesis. [note upper/lower case are NOT distinguished in "object-types"] For example:

(Function)
(Error Message)
(Transor Command)
(Compiler Question)

Some commonly-used object-types may be abbreviated with single words:

FN	->	(Function)
Var	->	(Variable)
Prop	->	(Property Name)
BreakCom	->	(Break Command)
EditCom	->	(Editor Command)
PACom	->	(Prog. Asst. Command)
FileCom	->	(File Package Command)
Error	->	(Error Message)
Litatom	--,	
Atom	--->	(Litatom)

[note: In {PageRef ...} and {SectionRef ...}, the word TAG can be used to refer to tags, as specified below. The word FIGURE is a synonym for TAG; this can be used to refer to a figure tag. The word TERM can be used in {index ...} to index an English term.]

{index <text>}
Creates an index reference. Some of the text objects (such as function definitions) will automatically create an index reference, but it is useful to be able to create one explicitly. This should have the format:

{index <keywords> <object> <object type>}

<keywords> (optional) can be any combination of the words
BEGIN
END
PRIMARY
<object> can be any number of words.
<object type> should be an object-type as specified above.
If this is omitted (<text> does not end with ")" and the last word in <text> is not one of the special object-type words), then this is the index of a term.
This can also be specified by using the word TERM.

Examples: {index *PRIMARY* BLOBBY (Transor Command)}
{index SELF-DESTRUCT SEQUENCE INITIATED Error)
{index *BEGIN* *PRIMARY* file names)

{indexX {name <name>} {type <type>} {info <info words>} {text <text>}
{subname <name>} {subtype <type>} {subtext <text>}
{subsubname <name>} {subsubtype <type>} {subsubtext <text>} }
Used for creating special index references whose printname is different from the "index name", or who have sub-index entries, or should not have page numbers in the index. <name> is the index name, used for referencing the object, and alphabetizing the index entry. <type> is the object type. <info words> (optional) can contain one or more of the keywords *BEGIN*, *END*, *PRIMARY*. <text> is the text printed in the index for this index entry.
Example: {indexX {name +} {type (Infix Operator)} {text {lisp {arg X}+{arg Y}}}} could be used to index "+" as an infix operator so that it would appear in the index alphabetized near other "+"s, but printing as "A+B".

The arguments {subname <name>} {subtype <type>} {subtext <text>} {subsubname <name>} {subsubtype <type>} {subsubtext <text>} are all optional. These are used to specify 1st and 2nd level subentries in the index. The subentry can have their own type and text definition, although these are mostly terms.

If <info terms> includes *NOPAGE*, this indicates that the index entry should not have a page number associated with it. This can be used with subentries to create "See also..." notes. Within a list of subentries, the *NOPAGE* entries will always be listed last.

Example: The following set of indexx commands:

{indexX {name FOO} {type (Big Command)} {text {lisp /FOO/}}}
{indexX {name FOO} {type (Big Command)} {subname ZZZ}}
{indexX {name FOO} {type (Big Command)} {subname BAZ}}
{indexX {index *NOPAGE*} {name FOO} {type (Big Command)} {subname See also BAR}}
{indexX {name FOO} {type (Big Command)} {subname BAZ} {subsubname QWERTY} {subsubtype VAR}}

Would produce the index entries:

/FOO/ (Big Command) x.xx
    BAZ x.xx
        QWERTY (Variable) x.xx
    ZZZ x.xx
    See also BAR

{Def {type <type>} {name <name>} {printname <pname>} {args <args>} {parens} {noparens} {text <text>} }
May be used to specify the definition of anything. <type> should be an object-type, as specified above. <name> should be the name of the object, ideally a single word. If {printname <pname>} (optional) is given, it will be used for printing the object in the top line of the definition and in the index. If {printname <pname>} is not given, the printed name is specified by {name <name>}, {args <args>}, {parens}, and {noparens}. <args> should be the "arguments" of this object. [{args <args>} is optional.] Normally, {Def will put parenthesis around the object if and only if {Args <args>} is given. {parens} and {noparens} (if given) can be used to change this default behavior. Examples:

```
{Def {Type (Blobby Command)} {Name DOBLOB} {Args A B}
        {Text ....}}
{Def {Type (CLISP Character)}  {Name +}  {PrintName {lisp {arg X}+{arg Y}}}
        {Text ....}}
```

{Tag <tag>}
This may be used to associate <tag>, a single word (upper/lower case doesn't matter), with a particular place in the text. This can be referenced by {SectionRef and {PageRef as described below.

{SectionRef <object-type> <object>}
{PageRef <object-type> <object>}
These TOs are used to provide a cross-reference facility. {SectionRef will print "section " and the section that the "main" occurence of the object with the given type occurs. <"main"

shall remain undefined for now> {PageRef prints "page " and the appropriate page number. If <object-type> is "Tag", these TOs can refer to tags generated with {Tag <tag>}. Examples:
{SectionRef Fn CONS} could print "section 3.1.1"
{PageRef (File Package Type) Expressions} could print "page 14.34"
{PageRef Tag CompilingCLISP} could print "page 20.42"

Note: {PageRef...} and {SectionRef...} can only be used to reference 1st level index references. They cannot be used to reference subentries generated with {indexx...}

{Term <text>}
Prints <text> exactly as given, and also puts it into the index. Equivalent to "<text>{index <text> Term}".

{Figure {tag <tag>} {text <text>} {caption <text>} }
Prints a simple "figure", with figure text and caption as specified by the "text" and "caption" arguments. "{tag <tag>}" is an optional argument which, if specified, generates a tag at the beginning of the figure. This tag can be used to refer to the page or section that this figure appears on (using {PageRef Figure <tag>} or {SectionRef Figure <tag>}), or to refer to the figure number (using {FigureRef <tag>})

{FigureRef <tag>}
Prints "figure" and the figure number of the figure with the specified tag.

********** Complex Text objects ******

{numberedlist {item <text>} {item <text>} {item <text>} ...}
Used for making numbered lists of items.
Synonyms: Text -> Item

{unnumberedlist {item <text>} {item <text>} {item <text>} ...}
Like {numberedlist ...}, except that it uses bullets to mark each item in the list.
Synonyms: UnlabeledList -> Unnumberedlist
Text -> Item

{labeledlist {name <text>} {item <text>} {name <text>} {item <text>} ...}
Used for making a table associating a set of labels with descriptions of their meanings.
The argument {LName <text>} can be used to add a label left-justified on the page. Either an LName or a Name, or both, can appear as labels for an item. If an LName is supposed to be on the same line as a Name, it should appear first: {Labeledlist {LName xx} {Name yy} {item zz} ...}
Synonyms: Label -> Name
Text -> Item

{Table {Column} ... {Column} {First <text>} {Next <text>} {Next <text>} ... {First <text>} ... }
Used for formatting multi-column tables. The number of {Column}'s before the first {First or {Next indicates how many columns the table should have. If no {Column}'s are given,

the default is three columns. There are other formatting arguments, but they will probably change. <<<Note: This is not currently supported in Tedit. Currently, this TO only prints out the values of each {First} or {Next} argument, without formatting. Do not use tables>>>

********** Other Text objects ******

{Include <filename>}
Used to include the text of one file within another. Files may be nested arbitrarily deep.

{Lbracket}
{Rbracket}
{bracket <text>}
These are used to insert the curly-bracket symbol into the text. {Lbracket} and {Rbracket} insert left and right brackets, and {bracket <text>} puts brackets around the given piece of text.

The following TOs are used to specify certain non-typeable characters.

{CRsymbol} or {cr}	-	carriage return symbol (currently a superscripted 'CR')
{pi}	-	Greek letter pi
{plusminus}	-	Plus or Minus ("+" + "-")
{GE}	-	greater than or equal (">" + "=")
{LE}	-	less than or equal ("<" + "=")
{NE}	-	not equal ("=" + "|")
{endash}	-	short dash
{emdash}	-	long dash
{ellipsis}	-	ellipsis (...)
{bullet}	-	bullet
{anonarg}	-	long dash (should be used in the argument list of a function definition to specify the "unspecified arguments" specified in the current manual with a dash)