0.1 TTYIN - A DISPLAY TYPEIN EDITOR

TTYIN is an Interlisp function for reading input from the terminal. It features altmode completion, spelling correction, help facility, and fancy editing, and can also serve as a glori ed free text input function. This document is divided into two major sections: how to use TTYIN from the user's point of view, and from the programmer's.

TTYIN exists in implementations for Interlisp- 10 and Interlisp- D. The two are substantially compatible, but the capabilities of the two systems di er (Interlisp- D has a more powerful display and allows greater access to the system primitives needed to control it e ectively; it also has a mouse, greatly reducing the need for keyboard- oriented editing commands). Descriptions of both are included in this document for completeness, but Interlisp- D users may nd large sections irrelevant.

0.1.1 Entering Input With TTYIN

There are two major ways of using TTYIN: (1) set `LISPXREADFN` to `TTYIN`, so the LISPX executive uses it to obtain input, and (2) call `TTYIN` from within a program to gather text input. Mostly the same rules apply to both; places where it makes a di erence are mentioned below.

The following characters may be used to edit your input, independent of what kind of terminal you are on. The more TTYIN knows about your terminal, of course, the nicer some of these will behave. Some functions are performed by one of several characters; any character that you happen to have assigned as an interrupt character will, of couse, not be read by TTYIN. There is a (somewhat inelegant) way of changing which characters perform which functions, described under `TTYINREADMACROS` later on.

^A, BS, DELETE
: Deletes a character. At the start of the second or subsequent lines of your input, deletes the last character of the previous line.

^W
: Deletes a ''word''. Generally this means back to the last space or parenthesis.

^Q (^U for Tops20 users)
: Deletes the current line, or if the current line is blank, deletes the previous line.

^R
: Refreshes the current line. Two in a row refreshes the whole bu er (when doing multi- line input).

ESCAPE
: Tries to complete the current word from the spelling list provided to `TTYIN`, if any. In the case of ambiguity, completes as far as is uniquely determined, or rings the bell. For `LISPX` input, the spelling list may be `USERWORDS` (see discussion of `TTYINCOMPLETEFLG`, page X.XX).

  Interlisp- 10 only: If no spelling list was provided, but the word begins with a ''<'',tries directory name completion (or lename completion if there is already a matching ''>'' in the current word).

?
: If typed in the middle of a word will supply alternative completions from the `SPLST` argument to `TTYIN` (if any). `?ACTIVATEFLG` (page X.XX) must be true to enable this feature.

^F
: Sumex, Tops20 only: Invokes `GTJFN` for lename completion on the current ''word''.

^Y Escapes to a Lisp userexec, from which you may return by the command OK. However, when in READ mode and the buffer is non-empty, ^Y is treated as Lisp's unquote macro instead, so you have to use edit-^Y (below) to invoke the userexec.

<middle-blank> in Interlisp-D, LF in Interlisp-10
Retrieves characters from the previous non-empty buffer when it is able to; e.g., when typed at the beginning of the line this command restores the previous line you typed at TTYIN; when typed in the middle of a line fills in the remaining text from the old line; when typed following ^Q or ^W restores what those commands erased.

; If typed as the first character of the line means the line is a comment; it is ignored, and TTYIN loops back for more input.

^X Goes to the end of your input (or end of expression if there is an excess right parenthesis) and returns if parentheses are balanced, beeps if not. Currently implemented in Interlisp-D only.

During most kinds of input, TTYIN is in ''auto fill'' mode: if a space is typed near the right margin, a carriage return is simulated to start a new line. In fact, on cursor-addressable displays, lines are always broken, if possible, so that no word straddles the end of the line. The ''pseudo-carriage return'' ending the line is still read as a space, however; i.e., the program keeps track of whether a line ends in a carriage return or is merely broken at some convenient point. You won't get carriage returns in your strings unless you explicitly type them.

0.1.2 Mouse Commands [Interlisp-D Only]

The mouse buttons are interpreted as follows during TTYIN input:

LEFT Moves the caret to where the cursor is pointing. As you hold down LEFT, the caret moves around with the cursor; after you let up, any typein will be inserted at the new position.

MIDDLE Like LEFT, but moves only to word boundaries.

RIGHT Deletes text from the caret to the cursor, either forward or backward. While you hold down RIGHT, the text to be deleted is complemented; when you let up, the text actually goes away. If you let up outside the scope of the text, nothing is killed (this is how to ''cancel'' the command). This is roughly the same as CTRL-RIGHT with no initial selection (below).

If you hold down CTRL and/or SHIFT while pressing the mouse buttons, you instead get secondary selection, move selection or delete selection. You make a selection by bugging LEFT (to select a character) or MIDDLE (to select a word), and optionally extend the selection either left or right using RIGHT. While you are doing this, the caret does not move, but your selected text is highlighted in a manner indicating what is about to happen. When you have made your selection (all mouse buttons up now), lift up on CTRL and/or SHIFT and the action you have selected will occur, which is:

SHIFT The selected text as typein at the caret. The text is highlighted with a broken underline during selection.

CTRL Delete the selected text. The text is complemented during selection.

CTRL-SHIFT
Combines the above: delete the selected text and insert it at the caret. This is how you move

text about.

You can cancel a selection in progress by pressing LEFT or MIDDLE as if to select, and moving outside the range of the text.

The most recent text deleted by mouse command can be inserted at the caret by typing <middle-blank> (the same key that retrieves the previous buffer when issued at the end of a line).

0.1.3 Display Editing Commands

On edit-key terminals (Datamedia): In Interlisp-10, TTYIN reads from the terminal in binary mode, allowing many more editing commands via the edit key, in the style of TVEDIT commands. Note that due to Tenex's unfortunate way of handling typeahead, it is not possible to type ahead edit commands before TTYIN has started (i.e., before its prompt appears), because the edit bit will be thrown away. Also, since ESCAPE has numerous other meanings in Lisp and even in TTYIN (for completion), ESCAPE is not used as a substitute for the edit key.

In Interlisp-D: Users will probably have little use for most of these commands, as cursor positioning can often be done more conveniently, and certainly more obviously, with the mouse. Nevertheless, some commands, such as the case changing commands, can be useful. The <bottom-blank> key can be used as an edit (meta) key in Chorus and subsequent releases if you perform (TTYINMETA T). This calls (METASHIFT T) to enable the meta key, redefines the middle and top blank keys, and informs TTYIN that you want to use them. Alternatively, you can use the EDITPREFIXCHAR (by default on <top-blank>) as described in the next paragraph.

On edit-keyless display terminals (Heath): If you want to type any of these commands, you need to prefix them with the ''edit prefix'' character. Set the variable EDITPREFIXCHAR to the character code of the desired prefix char. Type the edit prefix twice to give an ''edit-ESCAPE'' command. Some users of the TENEX TVEDIT program like to make ESCAPE (33Q) be the edit prefix, but this makes it somewhat awkward to ever use escape completion.

On edit-keyless hardcopy terminals: You probably want to ignore this section, since you won't be able to see what's going on when you issure edit commands; there is no attempt made to echo anything reasonable.

In the descriptions below, ''current word'' means the word the cursor is under, or if under a space, the previous word. Currently parentheses are treated as spaces, which is usually what you want, but can occasionally cause confusion in the word deletion commands. The notation [CHAR] means edit-CHAR, if you have an edit key, or <editprefixchar> CHAR if you don't; $ = escape. Most commands can be preceded by numbers or escape (means infinity), only the first of which requires the edit key (or the edit prefix). Some commands also accept negative arguments, but some only look at the magnitude of the arg. Most of these commands are taken from the display editors TVEDIT and/or E, and are confined to work within one line of text unless otherwise noted.

Cursor Movement Commands:

[delete], [bs], [<]
: Back up one (or n) characters.

[space], [>]
: Move forward one (or n) characters.

Display Editing Commands

[^] Moves up one (or n) lines.

[lf] Moves down one (or n) lines.

[(] Move back one (or n) words.

[)] Move ahead one (or n) words.

[tab] Moves to end of line; with an argument moves to nth end of line; [$tab] goes to end of bu er.

[^L] Moves to start of line (or nth previous, or start of bu er).

[{] and [}]
Go to start and end of bu er, respectively (like [$^L] and [$tab]).

[[ ] (edit-left-bracket)
Moves to beginning of the current list, where cursor is currently under an element of that list or its closing paren. (See also the auto-parenthesis-matching feature below under ‘‘Flags’’.)

[ ]] (edit-right-bracket)
Moves to end of current list.

[Sx] Skips ahead to next (or nth) occurrence of character x, or rings the bell.

[Bx] Backward search, i.e., short for [-S] or [-nS].

Bu er Modi cation Commands:

[Zx] Zaps characters from cursor to next (or nth) occurrence of x. There is no unzap command yet.

[A] or [R]
Repeat the last S, B or Z command, regardless of any intervening input (note this di ers from Tvedit’s A command).

[K] Kills the character under the cursor, or n chars starting at the cursor.

[I] Begin inserting. Exit insert with any edit command. Characters you type will be inserted, rather than overwriting the existing text. If EMACSFLG (page X.XX) is true (default in Interlisp-D), you are always in insert mode, and this command is a noop.

Inserting <cr> behaves slightly di erent from in tvedit. The sequence [I<cr>] behaves as in TVEDIT; it inserts a blank line ahead of the cursor. <cr> typed any other time while in insert mode actually inserts a <cr>, behaving somewhat like TVEDIT’s [B]. [$I] is the same as [I<cr>].

[cr] When the bu er is empty is the same as <lf>, i.e. restores bu er’s previous contents. Otherwise is just like a <cr> (except that it also terminates an insert). Thus, [<cr><cr>] will repeat the previous input (as will <lf><cr> without the edit key).

[O] Does ‘‘Open line’’, inserting a crlf after the cursor, i.e., it breaks the line but leaves the cursor where it is.

[T] Transposes the characters before and after the cursor. When typed at the end of a line, transposes the previous two characters. Refuses to handle funny cases, such as tabs.

[G] Grabs the contents of the previous line from the cursor position onward. [nG] grabs the nth previous line.

[L] Lowercases current word, or n words on line. [$L] lowercases the rest of the line, or if given at the end of line lowercases the entire line.

[U] Uppercases analogously.

[C] Capitalize. If you give it an argument, only the rst word is capitalized; the rest are just lowercased.

[^Q] Deletes the current line. [$^Q] deletes from the current cursor position to the end of the bu er. No other arguments are handled.

[^W] Deletes the current word, or the previous word if sitting on a space.

[D<del>] and [D<cr>]
Are the same as [^W] and [^Q], for approximate compatibility with TVEDIT.

[J] ''Justify'' this line. This will break it if it is too long, or move words up from the next line if too short. Will not join to an empty line, or one starting with a tab (both of which are interpreted as paragraph breaks). Any new line breaks it introduces are considered spaces, not carriage returns. [nJ] justi es n lines.

The linelength is de ned as `TTYJUSTLENGTH`, ignoring any prompt characters at the margin. If `TTYJUSTLENGTH` is negative, it is interpreted as relative to the right margin. `TTYJUSTLENGTH` is initially 8 in Interlisp- D, 72 in Interlisp- 10.

[$F] ''Finishes'' the input, regardless of where the cursor is. Speci cally, it goes to the end of the input and enters a <cr>, ^Z or '']'',depending on whether normal, `REPEAT` or `READ` input is happening. Note that a '']''won't necessarily end a `READ`, but it seems likely to in most cases where you would be inclined to use this command, and makes for more predictable behavior.

Miscellaneous Commands:

[P] Interlisp- D: Prettyprint bu er. Clears the bu er and reprints it using prettyprint. If there are not enough right parentheses, it will supply more; if there are too many, any excess remains unprettyprinted at the end of the bu er. May refuse to do anything if there is an unclosed string or other error trying to read the bu er.

[N] Refresh line. Same as ^R. [$N] refreshes the whole bu er; [nN] refreshes n lines. Cursor movement in TTYIN depends on TTYIN being the only source of output to the screen; if you do a ^T, or a system message appears, or line noise occurs, you may need to refresh the line for best results. In Interlisp- 10, if for some reason your terminal falls out of binary mode (e.g. can happen when returning to a Lisp running in a lower fork), Edit-<anything> is unreadable, so you'd have to type ^R instead.

[^Y] Gets userexec. Thus, this is like regular ^Y, except when doing a READ (when ^Y is a read macro and hence does not invoke this function).

[$^Y] Gets a userexec, but rst unreads the contents of the bu er from the cursor onward. Thus if you typed at TTYIN something destined for the Lisp executive, you can do [^L$^Y] and give it to Lisp.

[_] Adds the current word to the spelling list USERWORDS. With zero arg, removes word. See TTYINCOMPLETEFLG (page X.XX).

Note to Datamedia, Heath users: In addition to simple cursor movement commands and insert/delete, TTYIN uses the display's cursor-addressing capability to optimize cursor movements longer than a few characters, e.g. [tab] to go to the end of the line. In order to be able to address the cursor, TTYIN has to know where it is to begin with. Lisp keeps track of the current print position within the line, but does not keep track of the line on the screen (in fact, it knows precious little about displays, much like Tenex). Thus, TTYIN establishes where it is by forcing the cursor to appear on the last line of the screen. Ordinarily this is the case anyway (except possibly on startup), but if the cursor happens to be only halfway down the screen at the time, there is a possibly unsettling leap of the cursor when TTYIN starts.

0.1.4 Using TTYIN for Lisp Input

When TTYIN is loaded, or a sysout containing TTYIN is started up, the function SETREADFN is called. If the terminal is a display, it sets LISPXREADFN to be TTYINREAD; if the terminal is non-display, SETREADFN will set the variable back to READ. (SETREADFN 'READ) will also set it back to READ.

There are two principal differences between TTYINREAD and READ: (1) parenthesis balancing. The input does not activate on an exactly balancing right paren/bracket unless the input started with a paren/bracket, e.g., ''USE (FOO) FOR (FIE)'' will all be on one line, terminated by <cr>; and (2) read macros.

In Interlisp-10, TTYIN does not use a read table (TTYIN behaves as though using the default initial Lisp terminal input readtable), so read macros and redefinition of syntax characters are not supported; however, '' ' '' (QUOTE) and ''^Y'' (EVAL) are built in, and a simple implementation of ? and ?= is supplied. Also, the TTYINREADMACROS facility described below can supply some of the functionality of immediate read macros in the editor.

In Interlisp-D, read macros are (mostly) supported. Immediate read macros take effect only if typed at the end of the input (it's not clear what their semantics should be elsewhere).

0.1.5 Useful Macros

There are two useful edit macros that allow you to use TTYIN as a character editor: (1) ED loads the current expression into the ttyin buffer to be edited (this is good for editing comments and strings). Input is terminated in the usual way (by typing a balancing right parenthesis at the end of the input, typing <cr> at the end of an already balanced expression, or ^X anywhere inside the balanced expression). Typing ^E or clearing the buffer aborts ED. (2) EE is like ED but prettyprints the expression into the buffer, and uses its own window. The variable TTYINEDITPROMPT controls what prompt, if any, EE uses; see prompt argument description in next section (the initial setting is no prompt). EE is not yet implemented in Interlisp-10.

The macro BUF loads the current expression into the buffer, preceded by E, to be used as input however desired; as a trivial example, to evaluate the current expression, BUF followed by a <cr> to activate the buffer will perform roughly what the edit macro EVAL does. Of course, you can edit the E to something else to make it an edit command.

BUF is also defined at the executive level as a programmer's assistant command that loads the buffer with

the VALUEOF the indicated event, to be edited as desired.

TV is a programmer's assistant command like EV [EDITV] that performs an ED on the value of the variable.

And nally, if the event is considered ''short'' enough, the programmer's assistant command FIX will load the bu er with the event's input, rather than calling the editor. If you really wanted the Interlisp editor for your x, you could either say FIX EVENT - TTY:, or type ^U (or whatever on tops20) once you got TTYIN's version to force you into the editor.

0.1.6 Programming With TTYIN

(TTYIN PROMPT SPLST HELP OPTIONS ECHOTOFILE TABS UNREADBUF RDTBL) [Function]
TTYIN prints PROMPT , then waits for input. The value returned in the normal case is a list of all atoms on the line, with comma and parens returned as individual atoms; OPTIONS may be used to get a di erent kind of value back.

PROMPT is an atom or string (anything else is converted to a string). If NIL, the value of DEFAULTPROMPT, initially "** ", will be used. If PROMPT is T, no prompt will be given. PROMPT may also be a dotted pair (PROMPT$_1$. PROMPT$_2$), giving the prompt for the rst and subsequent (or over ow) lines, each prompt being a string/atom or NIL to denote absence of prompt. Note that rebinding DEFAULTPROMPT gives a convenient way to a ect all the ''ordinary'' prompts in some program module.

SPLST is a spelling list, i.e., a list of atoms or dotted pairs (SYNONYM . ROOT). If supplied, it is used to check and correct user responses, and to provide completion if the user types ESCAPE. If SPLST is one of the Lisp system spelling lists (e.g., USERWORDS or SPELLINGS3), words that are escape-completed get moved to the front, just as if a FIXSPELL had found them. Autocompletion is also performed when user types a break character (cr, space, paren, etc), unless one of the ''no xspell'' options below is selected; i.e., if the word just typed would uniquely complete by ESCAPE, TTYIN behaves as though ESCAPE had been typed.

HELP , if non-NIL, determines what happens when the user types ? or HELP. If HELP = T, program prints back SPLST in suitable form. If HELP is any other atom, or a string containing no spaces, it performs (DISPLAYHELP HELP). Anything else is printed as is. If HELP is NIL, ? and HELP are treated as any other atoms the user types. [DISPLAYHELP is a user-supplied function, initially a noop; systems with a suitable HASH package, for example, have de ned it to display a piece of text from a hash le associated with the key HELP.]

OPTIONS is an atom or list of atoms chosen from among the following:

NOFIXSPELL — Uses SPLST for HELP and Escape completion, but does not attempt any FIXSPELLing. Mainly useful if SPLST is incomplete and the caller wants to handle corrections in a more exible way than a straight FIXSPELL.

MUSTAPPROVE — Does spelling correction, but requires con rmation.

CRCOMPLETE — Requires con rmation on spelling correction, but also does autocompletion on <cr> (i.e. if what user has typed so far uniquely identi es a member of SPLST , completes it). This allows you to have the bene ts of autocompletion and still allow new words to be typed.

DIRECTORY — (only if SPLST = NIL) Interprets Escape to mean directory name completion [Interlisp-10 only].

USER — Like DIRECTORY, but does username completion. This is identical to DIRECTORY under Tenex [Interlisp-10 only].

FILE — (only if SPLST = NIL) Interprets Escape to mean filename completion, i.e. does a GTJFN [Sumex and Tops20 only].

FIX — If response is not on, or does not correct to, SPLST, interacts with user until an acceptable response is entered. A blank line (returning NIL) is always accepted. Note that if you are willing to accept responses that are not on SPLST, you probably should specify one of the options NOXFISPELL, MUSTAPPROVE or CRCOMPLETE, lest the user's new response get FIXSPELLed away without their approval.

STRING — Line is read as a string, rather than list of atoms. Good for free text.

NORAISE — Does not convert lower case letters to upper case.

NOVALUE — For use principally with the ECHOTOFILE arg (below). Does not compute a value, but returns T if user typed anything, NIL if just a blank line.

REPEAT — For multi-line input. Repeatedly prompts until user types ^Z (as in Tenex sndmsg). Returns one long list; with STRING option returns a single string of everything typed, with carriage returns (EOL) included in the string.

TEXT — Implies REPEAT, NORAISE, and NOVALUE. Additionally, input may be terminated with ^V, in which case the global flag CTRLVFLG will be set true (it is set to NIL on any other termination). This flag may be utilized in any way the caller desires.

COMMAND — Only the first word on the line is treated as belonging to SPLST, the remainder of the line being arbitrary text; i.e., ''command format''. If other options are supplied, COMMAND still applies to the first word typed. Basically, it always returns (CMD . REST-OF-INPUT), where REST-OF-INPUT is whatever the other options dictate for the remainder. E.g. COMMAND NOVALUE returns (CMD) or (CMD . T), depending on whether there was further input; COMMAND STRING returns (CMD . "REST-OF-INPUT"). When used with REPEAT, COMMAND is only in effect for the first line typed; furthermore, if the first line consists solely of a command, the REPEAT is ignored, i.e., the entire input is taken to be just the command.

READ — Parens, brackets, and quotes are treated a la READ, rather than being returned as individual atoms. Control characters may be input via the ^Vx notation. Input is terminated roughly along the lines of READ conventions: a balancing or over-balancing right paren/bracket will activate the input, or <cr> when no parenthesis remains unbalanced. READ overrides all other options (except NORAISE).

LISPXREAD — Like READ, but implies that TTYIN should behave even more like READ, i.e., do NORAISE, not be errorset-protected, etc.

NOPROMPT — Interlisp-D only: The prompt argument is treated as usual, except that TTYIN assumes that the prompt for the first line has already been printed by the caller; the prompt for the first line is thus used only when redisplaying the line.

ECHOTOFILE if specified, user's input is copied to this file, i.e., TTYIN can be used as a simple text-to-file routine if NOVALUE is used. If ECHOTOFILE is a list, copies to all files in the list. PROMPT is not included on the file.

TABS is a special addition for tabular input. It is a list of tabstops (numbers). When user types a tab, TTYIN automatically spaces over to the next tabstop (thus the first tabstop is actually the second ''column'' of input). Also treats specially the characters * and '; they echo normally, and then automatically tab over.

UNREADBUF allows the caller to ''preload'' the TTYIN buffer with a line of input. UNREADBUF is a list, the elements of which are unread into the buffer (i.e., ''the outer parentheses are stripped off'') to be edited further as desired; a simple <cr> (or ^Z for REPEAT input) will thus cause the buffer's contents to be returned unchanged. If doing READ input, the ''PRIN2 names'' of the input list are used, i.e., quotes and %'s will appear as needed; otherwise the buffer will look as though UNREADBUF had been PRIN1'ed. UNREADBUF is treated somewhat like READBUF, so that if it contains a pseudo-carriage return (the value of HISTSTR0), the input line terminates there.

Input can also be unread from a file, using the HISTSTR1 format: UNREADBUF = (<value of HISTSTR1> (FILE START . END)), where START and END are file byte pointers. This makes TTYIN a miniature text file editor.

RDTBL [Interlisp-D only] is the read table to use for READing the input when one of the READ options is given. A lot of character interpretations are hardwired into TTYIN, so currently the only effect this has is in the actual READ, and in deciding whether a character typed at the end of the input is an immediate read macro, for purposes of termination.

If the global variable TYPEAHEADFLG is T, or option LISPXREAD is given, TTYIN permits type-ahead; otherwise it clears the buffer before prompting the user.

0.1.7 EE Interface

The following may be useful as a way of outsiders to call TTYIN as an editor. These functions are currently only in Interlisp-D.

(TTYINEDIT EXPRS WINDOW PRINTFN) [Function]

This is the body of EE. Switches the tty to WINDOW, clears it, prettyprints EXPRS, a list of expressions, into it, and leaves you in TTYIN to edit it as Lisp input. Returns a new list of expressions.

If PRINTFN is non-NIL, it is a function of two arguments, EXPRS and FILE, which is called instead of PRETTYPRINT to print the expressions to the window (actually a scratch file). Note that EXPRS is a list, so normally the outer parentheses should not be printed. PRINTFN =T is shorthand for ''unpretty''; use PRIN2 instead of PRETTYPRINT.

TTYINAUTOCLOSEFLG [Variable]

If TTYINAUTOCLOSEFLG is true, TTYINEDIT closes the window on exit.

TTYINEDITWINDOW [Variable]

If the WINDOW arg to TTYINEDIT is NIL, it uses the value of TTYINEDITWINDOW, creating it if it does not yet exist.

TTYINPRINTFN [Variable]

The default value for PRINTFN in EE's call to TTYINEDIT.

(SET.TTYINEDIT.WINDOW WINDOW) [Function]

Called under a RESETLST. Switches the tty to WINDOW (defaulted as in TTYINEDIT) and clears it. The window's position is left so that TTYIN will be happy with it if you now call TTYIN yourself. Specifically, this means positioning an integral number of lines from the bottom of the window, the way the top-level tty window normally is.

(TTYIN.SCRATCHFILE) [Function]

Returns, possibly creating, the scratch file that TTYIN uses for prettyprinting its input. The file pointer is set to zero. Since TTYIN does use this file, beware of multiple simultaneous use of the file.

0.1.8 ?= Handler

In Interlisp, the ?= read macro displays the arguments to the function currently ''in progress'' in the typein. Since TTYIN wants you to be able to continue editing the buffer after a ?=, it processes this macro specially on its own, printing the arguments below your typein and then putting the cursor back where it was when ?= was typed. For users who want special treatment of ?=, the following hook exists:

TTYIN?=FN [Variable]

The value of this variable, if non-NIL, is a user function of one argument that is called when ?= is typed. The argument is the function that ?= thinks it is inside of. The user function should return one of the following:

NIL Normal ?= processing is performed.

T Nothing is done. Presumably the user function has done something privately, perhaps diddled some other window, or called TTYIN.PRINTARGS (below).

a list (ARGS . STUFF)

Treats STUFF as the argument list of the function in question, and performs the normal ?= processing using it.

anything else

The value is printed in lieu of what ?= normally prints.

At the time that ?= is typed, nothing has been ''read'' yet, so you don't have the normal context you might expect inside a conventional readmacro. If the user function wants to examine the typed-in arguments being passed to the fn, however, it can perform (TTYIN.READ?=ARGS), which bundles up everything between the function and the typing of ?= into a list, which it returns (thus it parallels an arglist; NIL if ?= was typed immediately after the function name).

(TTYIN.PRINTARGS FN ARGS ACTUALS ARGTYPE) [Function]

Does the function/argument printing for ?=. ARGS is an argument list, ACTUALS is a list of actual parameters (from the typein) to match up with args. ARGTYPE is a value of the function ARGTYPE; it defaults to (ARGTYPE FN).

0.1.9 Read Macros

When doing READ input in Interlisp-10, no Lisp-style read macros are available (but the ' and ^Y macros are built in). Principally because of the usefulness of the editor read macros (set by SETTERMCHARS), and the desire for a way of changing the meanings of the display editing commands, the following exists as a hack:

TTYINREADMACROS [Variable]

Value is a set of shorthand inputs useable during READ input. It is an alist of entries (CHAR CODE . SYNONYM). If the user types the indicated character (edit bit is denoted by the 200Q bit in charcode), TTYIN behaves as though the synonym character had been typed.

Special cases: 0 - the character is ignored; 200Q - pure Edit bit; means to read another char and turn on its edit bit; 400Q - macro quote: read another char and use its original meaning. For example, if you have macros ((33Q . 200Q) (30Q . 33Q)), then Escape (33Q) will behave as an edit prefix, and ^X (30Q) will behave like Escape. Note: currently, synonyms for edit commands are not well-supported, working only when the command is typed with no argument.

Slightly more powerful macros also can be supplied; they are recognized when a character is typed on an empty line, i.e., as the first thing after the prompt. In this case, the TTYINREADMACROS entry is of the form (CHAR CODE T . RESPONSE) or (CHAR CODE CONDITION . RESPONSE), where CONDITION is a list that evaluates true. If RESPONSE is a list, it is EVALed; otherwise it is left unevaluated. The result of this evaluation (or RESPONSE itself) is treated as follows:

NIL The macro is ignored and the character reads normally, i.e., as though TTYINREADMACROS had never existed.

An integer

A character code, treated as above. Special case: -1 is treated like 0, but says that the display may have been altered in the evaluation of the macro, so TTYIN should reset itself appropriately.

Anything else

This TTYIN input is terminated (with a crlf) and returns the value of ''response'' (turned into a list if necessary). This is the principal use of this facility. The macro character thus stands for the (possibly computed) reponse, terminated if necessary with a crlf. The original character is not echoed.

Interrupt characters, of course, cannot be read macros, as TTYIN never sees them, but any other characters, even non-control chars, are allowed. The ability to return NIL allows you to have conditional macros that only apply in specified situations (e.g., the macro might check the prompt (LISPXID) or other contextual variables). To use this specifically to do immediate editor read macros, do the following for each edit command and character you want to invoke it with:

```
(ADDTOVAR TTYINREADMACROS (CHAR CODE 'CHARMACRO? EDITCOM )))
```

For example, (ADDTOVAR TTYINREADMACROS (12Q CHARMACRO? NXP)) will make linefeed do the

NXP command. Note that this will only activate linefeed at the beginning of a line, not anywhere in the line. There will probably be a user function to do this in the next release.

Note that putting (12Q T . NXP) on TTYINREADMACROS would also have the effect of returning "NXP" from the READ call so that the editor would do an NXP. However, TTYIN would also return NXP outside the editor (probably resulting in a u.b.a. error, or convincing DWIM to enter the editor), and also the clearing of the output buffer (performed by CHARMACRO?) would not happen.

0.1.10 Assorted Flags

These flags control aspects of TTYIN's behavior. Some have already been mentioned. Their initial values are all NIL. In Interlisp-D, the flags are all initially T.

TYPEAHEADFLG [Variable]
: If true, TTYIN always permits typeahead; otherwise it clears the buffer for any but LISPXREAD input.

?ACTIVATEFLG [Variable]
: If true, enables the feature whereby ? lists alternative completions from the current spelling list.

EMACSFLG [Variable]
: Affects display editing. When true, TTYIN tries to behave a little more like EMACS (in very simple ways) than TVEDIT. Specifically, it has the following effects currently: (1) all non-edit characters self-insert (i.e. behave as if you're always in Insert mode); (2) [D] is the EMACS delete to end of word command.

SHOWPARENFLG [Variable]
: If true, then whenever you are typing Lisp input and type a right parenthesis/bracket, TTYIN will briefly move the cursor to the matching parenthesis/bracket, assuming it is still on the screen. The cursor stays there for about 1 second, or until you type another character (i.e., if you type fast you'll never notice it). This feature was inspired by a similar EMACS feature, and turned out to be pretty easy to implement.

TTYINBSFLG [Variable]
: Causes TTYIN to always physically backspace, even if you're running on a non-display (not a DM or Heath), rather than print \deletedtext\ (this assumes your hardcopy terminal or glass tty is capable of backspacing). If TTYINBSFLG is LF, then in addition to backspacing, TTYIN x's out the deleted characters as it backs up, and when you stop deleting, it outputs a linefeed to drop to a new, clean line before resuming. To save paper, this linefeed operation is not done when only a single character is deleted, on the grounds that you can probably figure out what you typed anyway.

TTYINRESPONSES [Variable]
: An alist of special responses that will be handled by routines designated by the programmer. See ''Special Responses'', below.

TTYINERRORSETFLG [Variable]
: [Interlisp-D only] If true, non-LISPXREAD inputs are errorset-protected (^E traps

back to the prompt), otherwise errors propagate upwards. Initially NIL.

TTYINMAILFLG [Variable]

[Tenex only] When true, performs mail checking, etc. before most inputs (except EVALQT inputs, where it is assumed this has already been done, or inputs indented by more than a few spaces). The MAILWATCH package must be loaded for this.

TTYINCOMPLETEFLG [Variable]

If true, enables Escape completion from USERWORDS during READ inputs. Details below.

USERWORDS (page X.XX) contains words you mentioned recently: functions you have de ned or edited, variables you have set or evaluated at the executive level, etc. This happens to be a very convenient list for context-free escape completion; if you have recently edited a function, chances are good you may want to edit it again (typing ''EF xx$'') or type a call to it. If there is no completion for the current word from USERWORDS, the escape echoes as ''$'', i.e. nothing special happens; if there is more than one possible completion, you get beeped. If typed when not inside a word, Escape completes to the value of LASTWORD, i.e., the last thing you typed that the p.a. ''noticed'' (setting TTYINCOMPLETEFLG to 0 disables this latter feature), except that Escape at the beginning of the line is left alone (it is a p.a. command).

If you really wanted to enter an escape, you can, of course, just quote it with a ^V, like you can other control chars.

You may explicitly add words to USERWORDS yourself that wouldn't get there otherwise. To make this convenient online the edit command [_] means ''add the current atom to USERWORDS'' (you might think of the command as ''pointing out this atom''). For example, you might be entering a function de nition and want to ''point to'' one or more of its arguments or prog variables. Giving an argument of zero to this command will instead remove the indicated atom from USERWORDS.

Note that this feature loses some of its value if the spelling list is too long, for then the completion takes too long computationally and, more important, there are too many alternative completions for you to get by with typing a few characters followed by escape. Lisp's maintenance of the spelling list USERWORDS keeps the ''temporary'' section (which is where everything goes initially unless you say otherwise) limited to #USERWORDS atoms, initially 100. Words fall o the end if they haven't been used (they are ''used'' if FIXSPELL corrects to one, or you use <escape> to complete one).

0.1.11 Special Responses

There is a facility for handling ''special responses'' during any non-READ TTYIN input. This action is independent of the particular call to TTYIN, and exists to allow you to e ectively ''advise'' TTYIN to intercept certain commands. After the command is processed, control returns to the original TTYIN call. The facility is implemented via the list TTYINRESPONSES.

TTYINRESPONSES [Variable]

TTYINRESPONSES is a list of elements, each of the form:

(COMMANDS RESPONSE-FORM OPTION)

COMMANDS is a single atom or list of commands to be recognized; RESPONSE-FORM is EVALed (if a list), or APPLYed (if an atom) to the command and the rest

of the line. Within this form one can reference the free variables COMMAND (the command the user typed) and LINE (the rest of the line). If OPTION is the atom LINE, this means to pass the rest of line as a list; if it is STRING, this means to pass it as a string; otherwise, the command is only valid if there is nothing else on the line. If RESPONSE- FORM returns the atom IGNORE, it is not treated as a special response (i.e. the input is returned normally as the result of TTYIN).

In MYCIN, the COMMENT command is handled this way; any time the user types COMMENT as the rst word of input, TTYIN passes the rest of the line to a mycin-de ned function which prompts for the text of the comment (recursively using TTYIN with the TEXT option). When control returns, TTYIN goes back and prompts for the original input again. The TTYINRESPONSES entry for this is (COMMENT (GRIPE LINE) LIST); GRIPE is a MYCIN function of one argument (the one-line comment, or NIL for extended comments).

Suggested use: global commands or options can be added to the toplevel value of TTYINRESPONSES. For more specialized commands, rebind TTYINRESPONSES to (APPEND NEWENTRIES TTYINRESPONSES) inside any module where you want to do this sort of special processing.

Special responses are not checked for during READ-style input.

0.1.12 Display Types

[This is not relevant in Interlisp- D]

TTYIN determines the type of display by calling DISPLAYTERMP, which is initially de ned to test the value of the GTTYP jsys. It returns either NIL (for printing terminals) or a small number giving TTYIN's internal code for the terminal type. The types TTYIN currently knows about:

0 = glass tty (capable of deleting chars by backspacing, but little else);

1 = Datamedia;

2 = Heath.

Only the Datamedia has full editing power. DISPLAYTERMP has built into it the correct terminal types for Sumex and Stanford campus 20's: Datamedia = 11 on tenex, 5 on tops20; Heath = 18 on Tenex, 25 on tops20. You can override those values by setting the variable DISPLAYTYPES to be an alist associating the GTTYP value with one of these internal codes. For example, Sumex displays correspond to DISPLAYTYPES = ((11 . 1) (18 . 2)) [although this is actually compiled into DISPLAYTERMP for speed]. Any display terminal other than Datamedia and Heath can probably safely be assigned to ''0'' for glass tty.

To add new terminal types, you have to choose a number for it, add new code to TTYIN for it and recompile. The TTYIN code speci es what the capabilities of the terminal are, and how to do the primitive operations: up, down, left, right, address cursor, erase screen, erase to end of line, insert character, etc.

For terminals lacking an Edit key (currently only Datamedias have it), set the variable EDITPREFIXCHAR to the ascii code of an Edit ''pre x'' (i.e. anything typed preceded by the pre x is considered to have the edit bit on). If your EDITPREFIXCHAR is 33Q (Escape), you can type a real Escape by typing 3 of them (2 won't do, since that means ''Edit-Escape'', a legitimate argument to another command). You could also de ne an Escape synonym with TTYINREADMACROS if you wanted (but currently it doesn't work in

lename completion). Setting `EDITPREFIXCHAR` for a terminal that is not equipped to handle the full range of editing functions (only the Heath and Datamedia are currently so equipped) is not guaranteed to work, i.e. the display will not always be up to date; but if you can keep track of what you're doing, together with an occasional ^R to help out, go right ahead.

Display Types