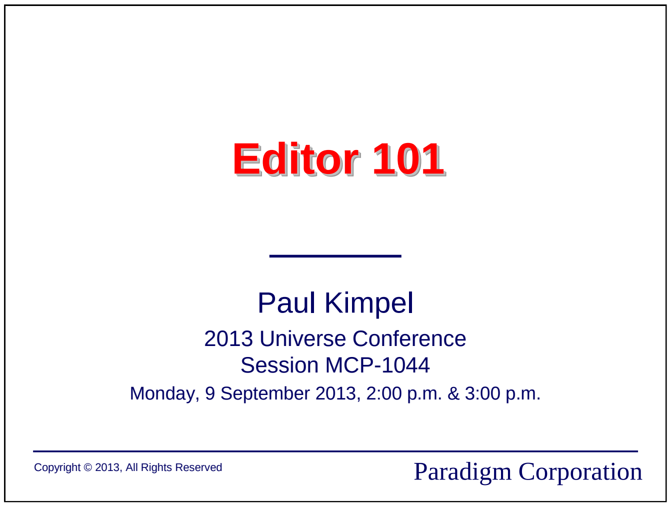

Editor 101

Paul Kimpel
2013 Universe Conference
Session MCP-1044
Monday, 9 September 2013, 2:00 p.m. & 3:00 p.m.

Copyright © 2013, All Rights Reserved

Paradigm Corporation

Editor 101

2013 Universe Conference
Chicago, Illinois

Session MCP-1044

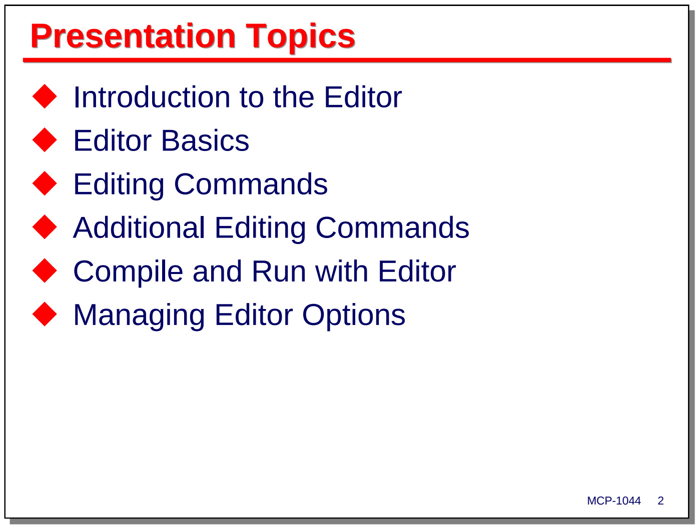
Monday, 9 September 2013, 2:00 p.m. & 3:00 p.m.

Paul Kimpel

Paradigm Corporation
San Diego, California

http://www.digm.com

e-mail: paul.kimpel@digm.com



Presentation Topics

- Introduction to the Editor
- Editor Basics
- Editing Commands
- Additional Editing Commands
- Compile and Run with Editor
- Managing Editor Options

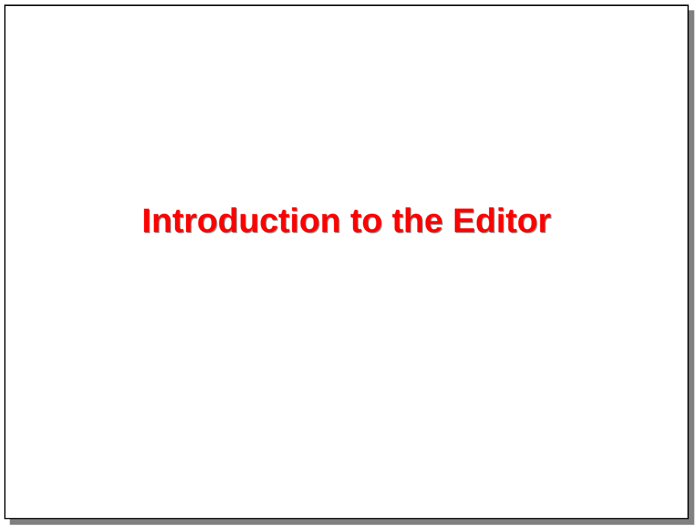
MCP-1044 2

The Editor (or just ED) is a text editor and development environment designed for traditional Burroughs-type block-mode terminals. In this presentation, I will discuss basic use of the Editor for maintaining MCP source files and compiling and running the resulting programs.

- I will begin with an introduction to the Editor, how it came about, how it differs from CANDE, the terminal or terminal emulator requirements for use with Editor, and some of its peculiarities.
- Next I will talk about basic use of the Editor, including how you run it from CANDE and MARC, the general layout of the editing screen, selecting lines of text for editing, inserting new lines into the file, and accessing the command line.
- The bulk of the presentation will be given over to the commands most often used for editing program text. These two sections will also discuss how to use the help pages, scrolling, panning, and navigating through a source file, quitting the Editor, and how recovery works.
- The next section will discuss how to compile and run programs directly from the Editor. This section will also discuss how to use the Editor to disassemble segments of object code in the context of its source code – an extremely useful facility when debugging programs and reading programdumps.
- Finally I will briefly discuss the options available for configuring and operating the Editor.

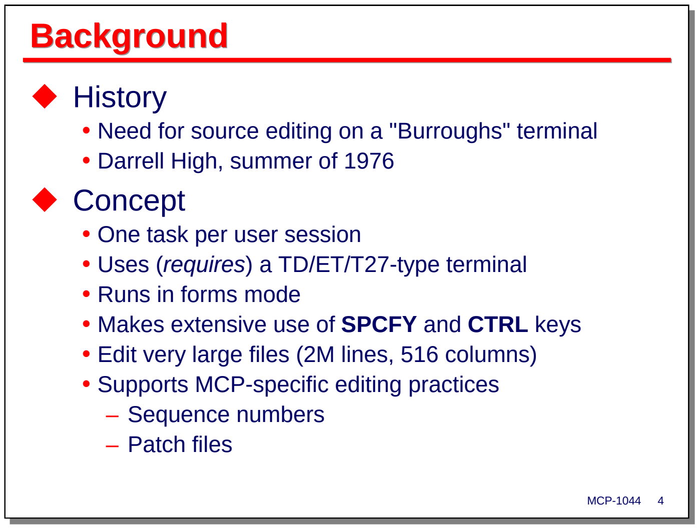
There is a lot more to the Editor than we can cover in this presentation. Some of its additional features are mentioned on the last slide. Also, there is a talk about using Editor with TADS.View to debug programs in the MCP 4049 presentation at this conference.

Note that the Editor is not included in the standard MCP Integrated Operating Environment (IOE). For user- and performance-based licenses, it is an extra-charge option, although it is not very expensive as MCP software products go. It is bundled, however, with metered systems and with the MCP SDK.



Introduction to the Editor

Let us being with a brief background for Editor and the environment in which it runs.



Background

- History
  - Need for source editing on a "Burroughs" terminal
  - Darrell High, summer of 1976
- Concept
  - One task per user session
  - Uses (*requires*) a TD/ET/T27-type terminal
  - Runs in forms mode
  - Makes extensive use of **SPCFY** and **CTRL** keys
  - Edit very large files (2M lines, 516 columns)
  - Supports MCP-specific editing practices
    - Sequence numbers
    - Patch files

MCP-1044 4

The story I've heard is that one day in the mid-1970s a bunch of suits from Burroughs HQ in Detroit visited the plant in Mission Viejo, California. As they were touring through the software engineering offices they noticed lots of terminals sitting on the programmers' desks. There was just one problem – these were not Burroughs terminals.

The terminals were instead Lear-Siegler ADM-3s, which those who are old enough will remember, were an early and very popular "glass teletype." They were asynchronous, character-at-a-time terminals, with an 80x24 screen that scrolled automatically. They were relatively inexpensive, reliable, and worked well with CANDE, which by that time had been in use for a few years. But they weren't Burroughs terminals.

Burroughs at that time had some nice terminals, the TD820 and TD830, but they were designed for commercial transaction processing, collecting data via forms, and the efficient use of slow, telephone-based, multi-drop telecommunication circuits. They were block-mode devices, and while you could use them with CANDE (this was long before CANDE page mode was invented), they weren't the easiest thing to deal with.

Apparently the suits went back to Detroit and issued a fatwa – Burroughs developers should be using Burroughs terminals – which caused some consternation and no doubt a lot of discontent among the developers, who liked their ADM-3s.

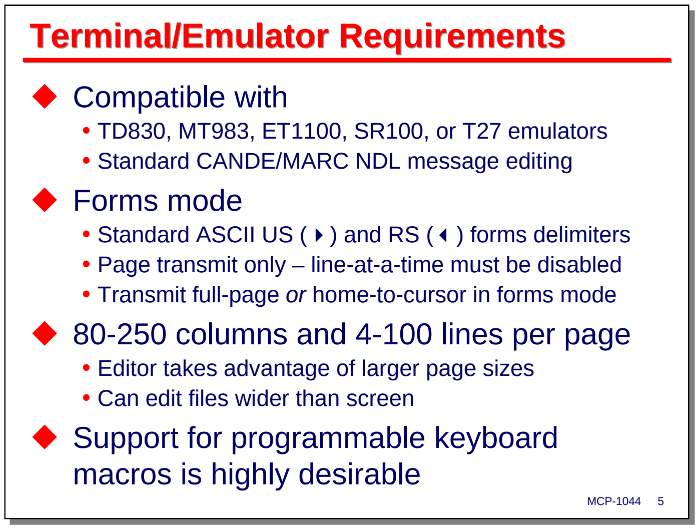
Into the breach stepped Darrell High, the original author of CANDE, in the summer of 1976. Darrell did an amazing thing – he took the features of the TD-series terminals, and around them designed an editor that was in many respects better than CANDE. Dave Bauerle and Bill Arnett, among others, also made major contributions to extend the feature set. It soon became, and remains, the primary editor for the MCP Engineering groups.

The Editor is an ordinary program, written in DCAlgol, and uses a standard remote file. Unlike CANDE, each user runs their own copy. It is specifically designed for use with the Burroughs TD/ET/T27 family of terminals, and will not work with any other type of terminal. It runs almost entirely in forms mode, and makes extensive use of the **CTRL**-key sequences and **SPCFY** key that are unique to those terminals.

In addition, the Editor supports some MCP-specific source file and editing practices. Like CANDE, it understands the MCP source file formats and sequence numbers. Unlike CANDE, however, the Editor is very smart about sequence numbers. It allows you to mostly ignore sequence numbers while you are editing and defer assigning sequence numbers until you save the file.

Another major advance with the Editor is that it understands patch files. More importantly, Editor allows you to edit a file without thinking about patch records, and will extract just the changed lines and save those for you.

Many Editor features have since found their way into the newer MCP development environments, PWB and the IDE for Eclipse. They are more or less direct clones of the techniques that Editor pioneered. There are some advantages to the GUI context of these newer editors, but the Editor is still the most capable of them.



Terminal/Emulator Requirements

- Compatible with
  - TD830, MT983, ET1100, SR100, or T27 emulators
  - Standard CANDE/MARC NDL message editing
- Forms mode
  - Standard ASCII US (▸) and RS (◂) forms delimiters
  - Page transmit only – line-at-a-time must be disabled
  - Transmit full-page *or* home-to-cursor in forms mode
- 80-250 columns and 4-100 lines per page
  - Editor takes advantage of larger page sizes
  - Can edit files wider than screen
- Support for programmable keyboard macros is highly desirable

MCP-1044 5

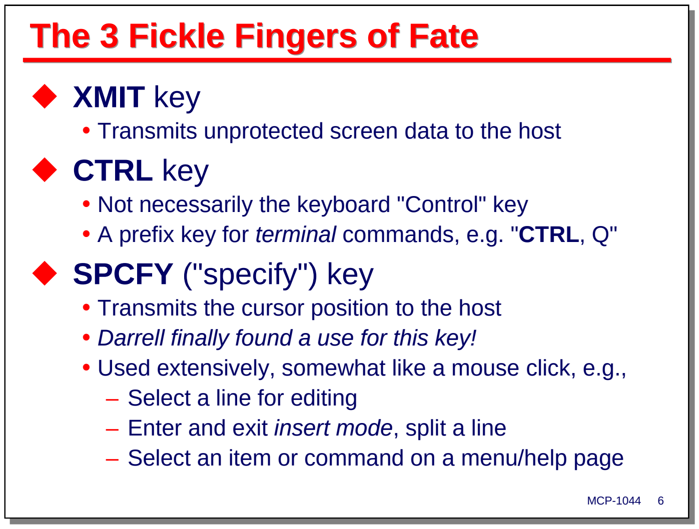
As previously mentioned, the Editor works only with Burroughs-type terminals and terminal emulators. Basically, this is anything that is compatible with the TD, MT, ET, SR, or T27 terminal families. The Editor also requires standard CANDE/MARC NDL message editing, which is the default, and what almost everyone uses, anyway.

The Editor uses forms mode extensively. It requires that the forms delimiters in the terminal or emulator be configured as the ASCII US (right-arrow) and RS (left-arrow) characters. Devices configured to use square brackets or curly braces will not work.

The Editor supports page-transmit mode only. Terminals configured to do line transmits will not work, although terminals with a line-transmit capability can still be used as long as it is not the default mode of transmission. The terminal can be configured for either full-page or home-to-cursor transmit in forms mode – the Editor supports both, although the behavior is slightly different.

In addition to supporting large files, the Editor can support large screen sizes – up to 250 columns wide and 100 lines per page. Most modern LCD displays will easily support a 50-line by 132-column Editor window. For files that are wider than the editing window, Editor can pan left and right to view vertical slices of the file.

Most modern terminals and emulators support a programmable keyboard. This can be of great use in the Editor. As we will see, the Editor supports a strange keyboard-shortcut feature of the Burroughs terminals, and those shortcuts are often best programmed into function keys.



The 3 Fickle Fingers of Fate

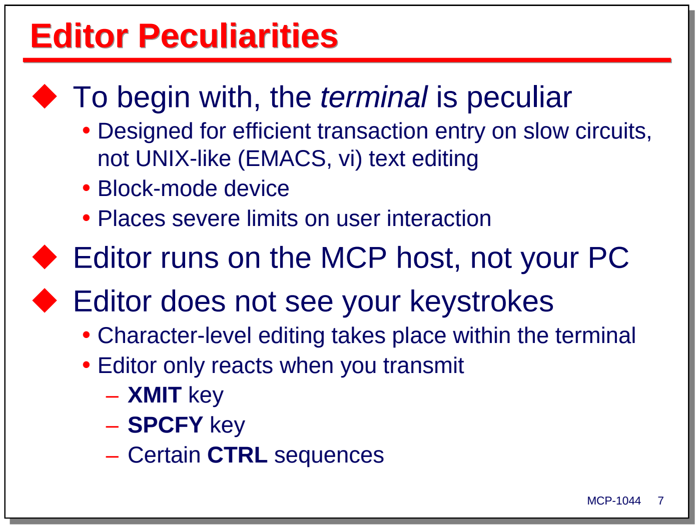
- **XMIT** key
  - Transmits unprotected screen data to the host
- **CTRL** key
  - Not necessarily the keyboard "Control" key
  - A prefix key for *terminal* commands, e.g. "**CTRL**, Q"
- **SPCFY** ("specify") key
  - Transmits the cursor position to the host
  - *Darrell finally found a use for this key!*
  - Used extensively, somewhat like a mouse click, e.g.,
    - Select a line for editing
    - Enter and exit *insert mode*, split a line
    - Select an item or command on a menu/help page

MCP-1044 6

There are three terminal control keys that are used extensively by the Editor. You will need to know where these are in your terminal or emulator.

- **XMIT** key. This one is easy – it's the standard transmit-page key, and is usually the large "**+**" key on the numeric keypad. If you are using a laptop that does not have a numeric keypad, you may want to program this function into one of the function keys. I like to use F12 for this.
- **CTRL** key. This is not necessarily the "Control" key below the shift keys on your keyboard, although some emulators (particularly Attachmate) implement it that way. Some emulators use ESC for **CTRL**. This is a prefix key used for terminal commands. You press *and release* the **CTRL** key to signal command mode, then press other keys to complete the command. For example, **CTRL,W** will place the terminal in forms mode, and **CTRL,V** exchanges the line containing the cursor with the line above it.
- **SPCFY** key. Pressing this key sends a four-character message to the host without disturbing the contents of the screen. The message (ESC, **"**, col, row) "specifies" the current cursor location on the page, hence the name. When the TD-series terminals came out, we all thought this was a really cool feature, but no one could figure out quite what to do with it. Well, Darrell High finally found a use for this key. It is used much like a mouse click to:
  - Select a line for editing
  - Insert a new line above or below an existing one
  - Select an item from a list of choices
  - Select an item or a command on a menu or help page

The **SPCFY** key is mode-sensitive – what it does depends on the state you are in at the time. We'll discuss the various modes shortly, when discussing Editor Basics.



Editor Peculiarities

- To begin with, the *terminal* is peculiar
  - Designed for efficient transaction entry on slow circuits, not UNIX-like (EMACS, vi) text editing
  - Block-mode device
  - Places severe limits on user interaction
- Editor runs on the MCP host, not your PC
- Editor does not see your keystrokes
  - Character-level editing takes place within the terminal
  - Editor only reacts when you transmit
    - **XMIT** key
    - **SPCFY** key
    - Certain **CTRL** sequences

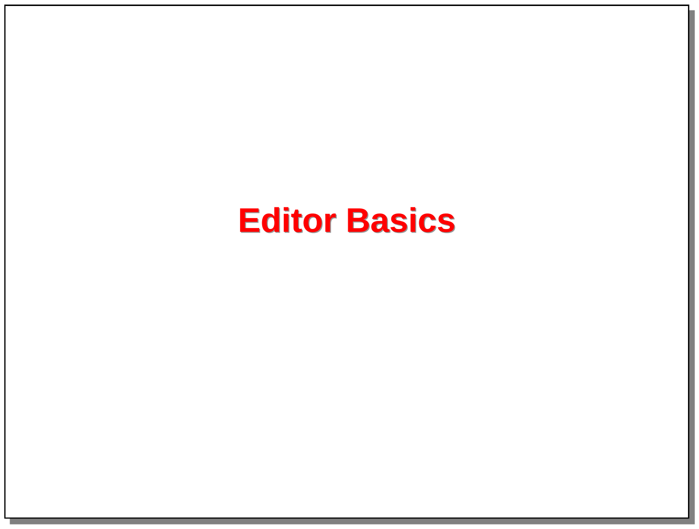
MCP-1044 7

As I said earlier, the Editor is something of an acquired taste. In fact, compared to other editors, it's downright peculiar. The reason for that is the terminal it's designed for is peculiar.

The terminal is a block-mode device designed for transaction entry on slow, multi-drop datacom circuits. It just doesn't support the sort of keystroke-interactive style of editor that has grown out of the UNIX EMACS- and vi-style editors, and which has become to dominant style on PCs. This places some severe limits on the way users can interact with both the terminal and the Editor running behind it.

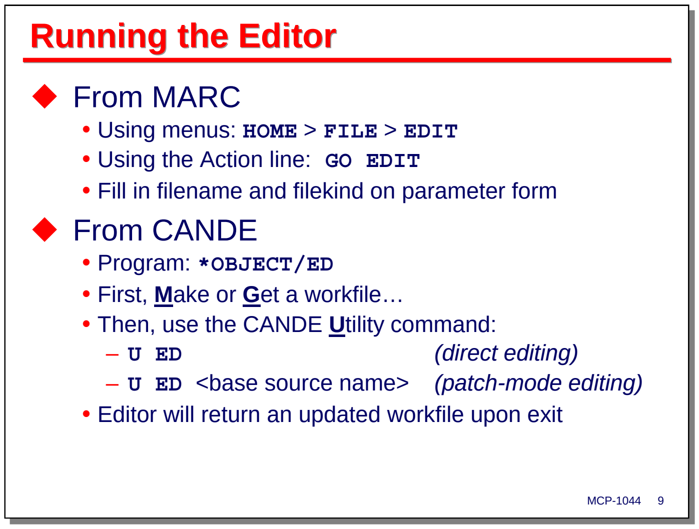
Another thing to recognize is that the Editor is running on the MCP host, not your workstation. The Editor does not see your individual keystrokes. You can do character-level editing within a line locally on the terminal, but the Editor can only react when it receives a message from the terminal, and that happens only when you press **XMIT**, **SPCFY**, and certain **CTRL** command sequences.

When Detroit issues a fatwa, you have to make do with what you have. It's amazing the Editor turned out as well as it did.



Editor Basics

With that background, let us now discuss the basics involved in using the Editor.



Running the Editor

- From MARC
  - Using menus: **HOME** > **FILE** > **EDIT**
  - Using the Action line: **GO EDIT**
  - Fill in filename and filekind on parameter form
- From CANDE
  - Program: ***OBJECT/ED**
  - First, Make or Get a workfile…
  - Then, use the CANDE Utility command:
    - **U ED** *(direct editing)*
    - **U ED** <base source name> *(patch-mode editing)*
  - Editor will return an updated workfile upon exit

MCP-1044 9

Most people typically run the Editor from CANDE, but it can also be run from MARC. From the MARC home screen, select the **FILE** and then **EDIT** menu items. Alternatively, you can **GO** directly to the **EDIT** page. This will display a form with fields for the name of the file to edit, whether it is a new file, the name of a file to edit against (for patch mode editing), and the **FILEKIND** of the file. Transmitting from this form will initiate the Editor.

When running the Editor from CANDE, you use the **Utility** command. This command was specifically designed for the Editor – it helps if the guy designing the Editor also wrote CANDE. The **Utility** command allows CANDE to pass a workfile to an external program, the external program to modify the workfile, and the external program to pass the updated workfile back to CANDE. The **Utility** command can be used with programs other than the Editor – the interface is documented in the *CANDE Operations Manual*.

Most systems have the Editor codefile under two names, ***OBJECT/ED** and ***SYSTEM/EDITOR**. The former means you can run it under CANDE as just "**ED**".

The Editor runs in two modes: direct editing and patch-mode editing. Direct editing works a lot like normal CANDE editing – you get a workfile, make some changes, and save the updated work file. To use the Editor in this mode:

- Get or make a file in CANDE in the normal way
- Enter the command: **U ED**

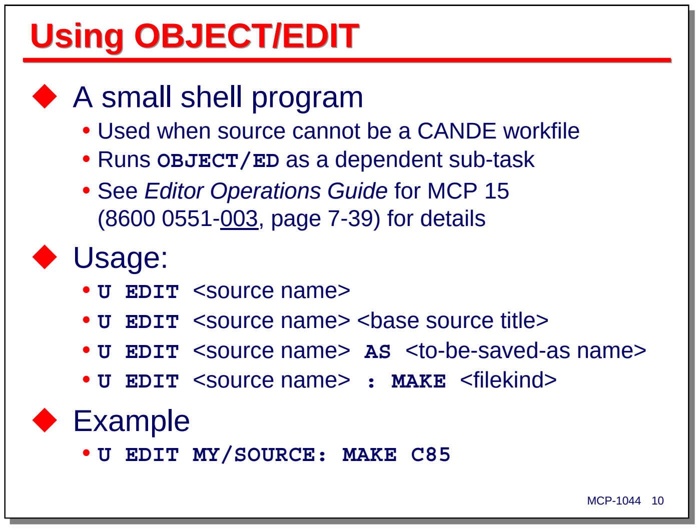
Patch-mode editing is a little different. In this mode, the workfile is not the whole source file, it contains only changed lines in the source. The patched lines are identified by the sequence numbers of the lines in the full source file that they effect. All of the MCP compilers understand how to merge a patch file with a base source during compilation, and there is a utility for merging multiple patch files, ***SYSTEM/PATCH**.

Thus, when working in patch mode, there are two files, the base source file, and the patch file, which is the CANDE workfile. To initiate the Editor in this mode:

- Get or make a file for the patches
- Enter the command: **U ED** <base source file name>

The Editor can start with an empty workfile and place the changes from your editing session in it. The Editor can also start with an existing patch file and create an updated patch file at the end of your editing session. When working in patch mode, you view a unified source. You don't need to worry about which lines are patched and which aren't (although the Editor indicates this on the screen). You just edit lines. The Editor keeps track of the changes and accurately generates the appropriate set of patch records when your session ends.

Patch files are an important and useful topic in MCP software development, but we don't have time to talk more about them here. That is a subject for another presentation.



Using OBJECT/EDIT

- A small shell program
  - Used when source cannot be a CANDE workfile
  - Runs **`OBJECT/ED`** as a dependent sub-task
  - See *Editor Operations Guide* for MCP 15 (8600 0551-003, page 7-39) for details
- Usage:
  - **`U EDIT`** <source name>
  - **`U EDIT`** <source name> <base source title>
  - **`U EDIT`** <source name> **`AS`** <to-be-saved-as name>
  - **`U EDIT`** <source name> **`: MAKE`** <filekind>
- Example
  - **`U EDIT MY/SOURCE: MAKE C85`**

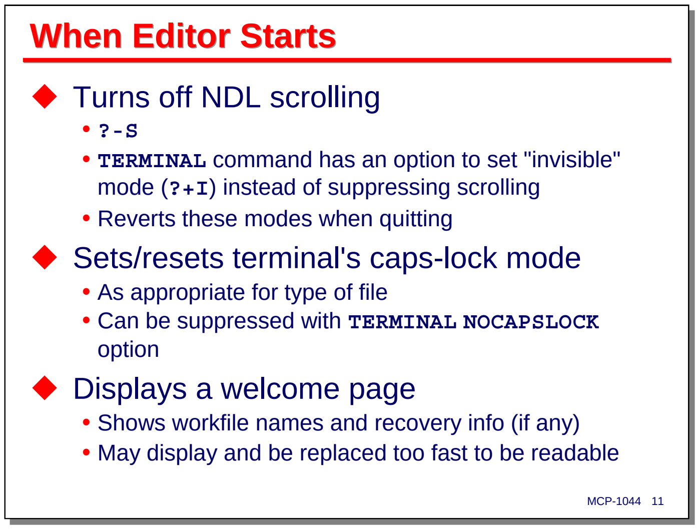
MCP-1044 10

There is one more way to run the Editor, using a small shell program named **`*OBJECT/EDIT`**. You use this when you want to edit a file that cannot be the subject of a Get or Make in CANDE. Examples of this include files with records longer than 255 characters and byte-stream files.

`OBJECT/EDIT` supports both direct- and patch-mode editing:

- When run with a single file name, the Editor does direct editing
- When run with two files names, the Editor does patch-mode editing. The first name is the patch file and the second is the base source file.
- When run with the **`AS`** clause, the Editor does not modify the first file name. It does direct editing, but places the result in the second file name at the end of the session.
- Finally, when run with the **`:MAKE`** clause, the file name need not exist beforehand. Editor will create a new file of the specified **`FILEKIND`**.

`OBJECT/ED` initiates the Editor as a dependent sub-task, so you will probably notice multiple BOT and EOT messages on your terminal.



When Editor Starts

- Turns off NDL scrolling
  - **`?-S`**
  - **`TERMINAL`** command has an option to set "invisible" mode (**`?+I`**) instead of suppressing scrolling
  - Reverts these modes when quitting
- Sets/resets terminal's caps-lock mode
  - As appropriate for type of file
  - Can be suppressed with **`TERMINAL NOCAPSLOCK`** option
- Displays a welcome page
  - Shows workfile names and recovery info (if any)
  - May display and be replaced too fast to be readable

MCP-1044 11

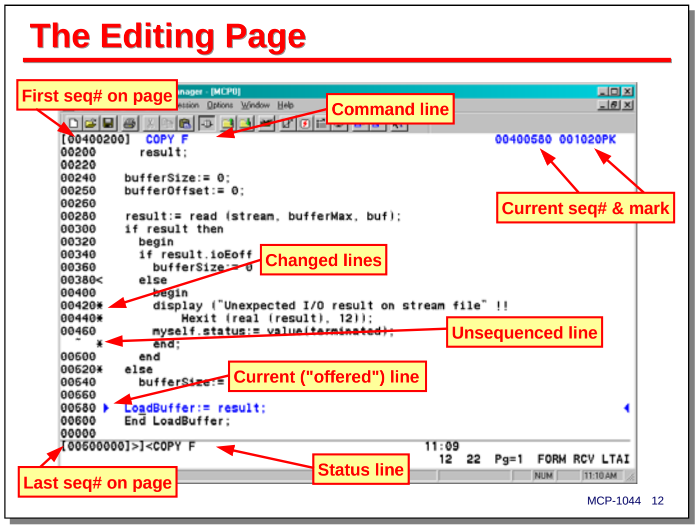
Some users like to run their terminals using the NDL scrolling (**`?+S`**) mode. This interferes with the way that the Editor paints the screen, so one of the first things it does when starting is test whether the terminal is in scrolling mode. If it is, the Editor turns scrolling off, and will reinstate it at the end of the editing session.

As an alternative, you can configure the Editor using its **`TERMINAL`** command to use "invisible" (**`?+I`**) mode during an editing session. Editor will also revert this mode at the end of the editing session.

Depending on the type of file you are editing, the Editor may alter the terminal's internal caps-lock setting. For example, if you are editing Algol or COBOL source, the Editor will set the internal caps lock by default, overriding the caps-lock key on the keyboard. You can disable this behavior by setting the **`NOCAPSLOCK`** option with the **`TERMINAL`** command.

The final thing that the Editor does when starting up is to display a welcome page. This shows the name of file being edited, the base source name (if running in patch mode), any other files that are associated with the session, and whether the session is being recovered from an abnormal termination. Normally this welcome page displays and is replaced by the editing page too quickly to read it. If you are running in patch mode, however, the patch file is merged with the base source while the welcome page is being displayed, so for a large patch file, this page may be viewable for a longer time.

Once the Editor finishes its initialization, it displays the editing page. If this is a normal startup, the first page of the source file will be displayed; if a prior session is being recovered, the last page of source you were editing will be displayed instead.

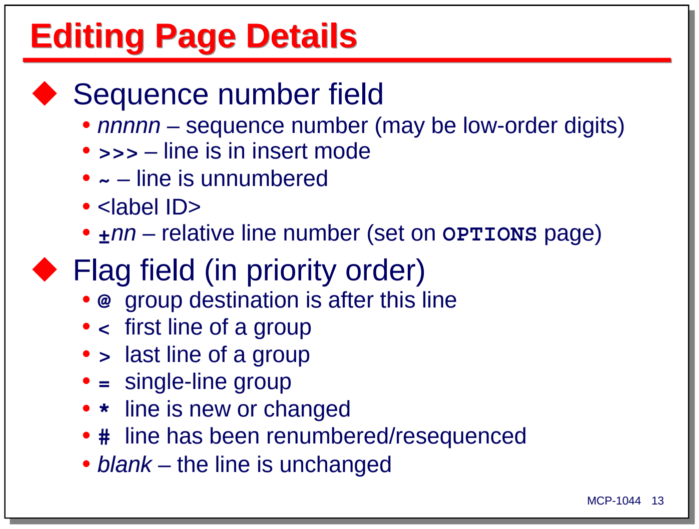


The Editing Page

```
[00400200]  COPY F                                      00400580 001020PK
00200      result;
00220
00240    bufferSize:= 0;
00250    bufferOffset:= 0;
00260
00280    result:= read (stream, bufferMax, buf);
00300    if result then
00320      begin
00340      if result.ioEoff
00360        bufferSize:= 0
00380<     else
00400        begin
00420*       display ("Unexpected I/O result on stream file" !!
00440*           Hexit (real (result), 12));
00460        myself.status:= value(terminated);
   ~   *     end;
00600      end
00620*   else
00640      bufferSize:=
00660
00680 ▸  LoadBuffer:= result;                                            ◂
00600    End LoadBuffer;
00000
[00600000]>]<COPY F                          11:09
                                              12   22  Pg=1  FORM RCV LTAI
```

This slide shows the page format the Editor uses while in editing mode. The layout of this page varies somewhat, depending on the width of the screen and the width of the sequence and text fields of the source file. This example shows the layout for an Algol program in an 80-column screen.

- Along the left margin is the sequence number field, or the low-order digits of the sequence number if there is not room for the entire field. This often contains something other than a sequence number, as discussed on the next slide. Note that a line with just a tilde (**~**) in it indicates an unsequenced line. This can result from inserting new lines into the file, or by copying or moving lines to this location.
- The top line contains, from left to right:
  - The full sequence number of the first line on the page
  - The contents of the "command line"
  - The full sequence number of the currently-offered (selected) line
  - The contents of the patchmark field of the current line, if any
- The last (status) line of the page contains the full sequence number of the last line on the page, the current "command character" (a right bracket in this case), a field of command status that varies with the current state of the editing session, and the time of day.
- The column to the right of the sequence number field (termed the "flag") shows a character that indicates the status of that line. We will discuss the flag characters on the next slide. On this example, the "**<**" indicates the first line of a <group> of lines and the "**`*`**" indicates a new or changed line.
- Finally, note that the entire page is in forms mode, but only the text of one line is an "unprotected" field. This is called the "offered line," as it is the line the Editor is currently offering to the user for modification. You can use the local editing features of your terminal to modify this line (overtyping text, character insert/delete, etc.). By default, the Editor offers only one line for editing at a time, but you can cause multiple lines to be offered using the **`OPTIONS`** command, as discussed at the end of the presentation.



Editing Page Details

- Sequence number field
  - *nnnnn* – sequence number (may be low-order digits)
  - **>>>** – line is in insert mode
  - **~** – line is unnumbered
  - <label ID>
  - **±***nn* – relative line number (set on **OPTIONS** page)
- Flag field (in priority order)
  - **@** group destination is after this line
  - **<** first line of a group
  - **>** last line of a group
  - **=** single-line group
  - ***** line is new or changed
  - **#** line has been renumbered/resequenced
  - *blank* – the line is unchanged

MCP-1044 13

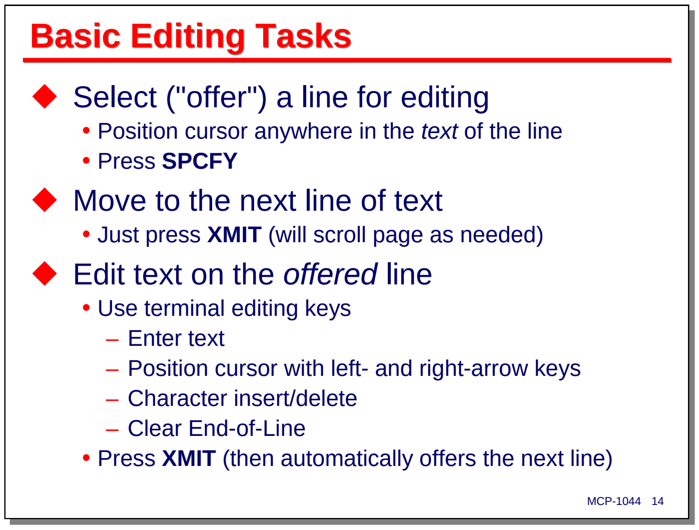
Two fields on the editing page deserve more attention.

The sequence number field normally holds a sequence number, but it can show other things as well:

- A line with "**>>>**" in the sequence field indicates this is an offered line in insert mode.
- A line with " **~** " in the sequence field indicates this is an unnumbered line.
- The sequence field can also display a <label ID>. Labels are like bookmarks, and will be discussed shortly.
- A positive or negative signed number indicates a *relative* line number. If a line is currently offered, the numbers are relative to the offered line, otherwise they are relative to the top of the page. You can display relative line numbers instead of sequence numbers using the **OPTIONS RELATIVE** command.

The flag field to the right of the sequence number field may show the following codes. If more than one code could apply to that line at the same time, only the first code in the order discussed below is shown:

- **@** indicates the current destination (as for a copy or move) is immediately after this line.
- **<** indicates this is the first line of a <group> of lines. We will discuss <group>s shortly.
- **>** indicates this is the last line of a <group>.
- **=** indicates this is the only line of a <group>.
- ***** indicates that this line has been changed or inserted into the file.
- **#** indicates this line has been renumbered or resequenced.
- *blank* indicates the line has not been changed.



Basic Editing Tasks

- Select ("offer") a line for editing
  - Position cursor anywhere in the *text* of the line
  - Press **SPCFY**
- Move to the next line of text
  - Just press **XMIT** (will scroll page as needed)
- Edit text on the *offered* line
  - Use terminal editing keys
    - Enter text
    - Position cursor with left- and right-arrow keys
    - Character insert/delete
    - Clear End-of-Line
  - Press **XMIT** (then automatically offers the next line)

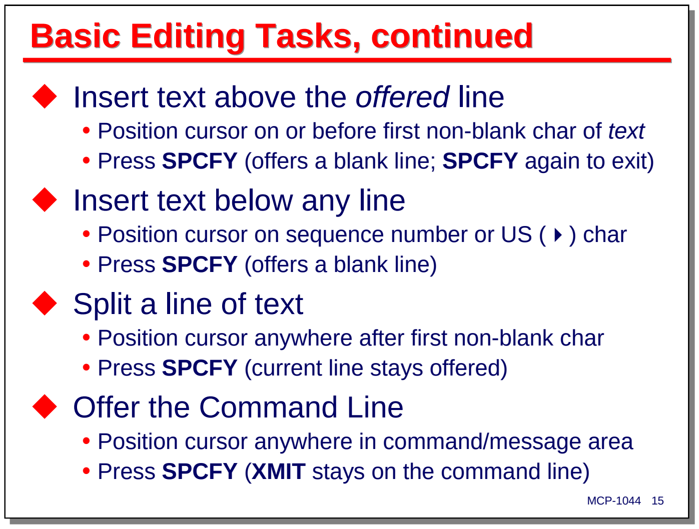
MCP-1044 14

The next two slides discuss the most basic editing tasks using the Editor. This is *very* different from most other editors, and is the part that is can be most confusing when first learning the Editor, as it is very modal. It makes sense, though, and after a little practice you will be doing it without even thinking.

- To select (or "offer") a line for editing, simply position the cursor anywhere on that line and press **SPCFY**. Editor will display the text of that line as an unprotected field.
- To move to the next line in the file, press **XMIT**. If you have not made any changes to the original line, it will not be marked as changed.
- To edit a line, make it the offered line, and modify it using the local terminal editing keys. You can overtype, insert, or delete characters any way you want on that line. You can also use the terminals clear-to-end-of-line function. When finished, press **XMIT**. Editor will record your changes and automatically offer the next line of the file.
- Note that editing a line will work somewhat differently, depending on whether your terminal is configured for full-page or home-to-cursor transmission in forms mode.
  - With full-page transmission, the entire line is transmitted to the Editor, and that text completely replaces the former contents of the line.
  - With home-to-cursor transmission, only the text up to (but not including) the position of the cursor is transmitted (unless the cursor is at the beginning of the line). Editor remembers the original contents of the line and overlays the text received from the terminal onto it. This is usually what you want, unless you have inserted or deleted characters on the line, in which case the result will probably not be what you intended. For this reason, I like to use full-page transmit in forms mode.

You can continue stepping through lines and modifying them (or not) all the way down the page. You can use **SPCFY** to reposition the offered line at any time. When you transmit on the next-to-last line on the page, the Editor will automatically start scrolling to bring a new lines in at the bottom of the page.

We will discuss how to insert new lines on the next slide. The way to delete lines will come up in a few more slides.



Basic Editing Tasks, continued

- Insert text above the *offered* line
  - Position cursor on or before first non-blank char of *text*
  - Press **SPCFY** (offers a blank line; **SPCFY** again to exit)
- Insert text below any line
  - Position cursor on sequence number or US (▸) char
  - Press **SPCFY** (offers a blank line)
- Split a line of text
  - Position cursor anywhere after first non-blank char
  - Press **SPCFY** (current line stays offered)
- Offer the Command Line
  - Position cursor anywhere in command/message area
  - Press **SPCFY** (**XMIT** stays on the command line)

MCP-1044 15

The prior slide discussed how to step through and modify existing lines in the file. There are two ways to insert a new line into the file:

- On the currently-offered line, position the cursor *on or before the first non-blank character* of the line and press **SPCFY**. Editor will insert and offer a blank line *above* the currently-offered line. It will also enter *insert mode* and stay there until you do something to exit insert mode.
- On any line of the page (not necessarily the offered line), position the cursor on the sequence number field, or on the US (right-arrow opening delimiter) character, and press **SPCFY**. The Editor will insert and offer a blank line *below* the one where you pressed **SPCFY** and enter insert mode.

In either case, the Editor will normally auto-indent and place the cursor under the first non-blank character of the prior line. Enter the text of the new line and press **XMIT**. The Editor will add that line to the file and insert another blank line below it. You can continue entering text for new lines and pressing **XMIT** as many times as necessary. Once you reach the next-to-last line on the page, Editor will automatically start scrolling.

To exit insert mode, do one of three things:

- On the currently-offered insert line, press **SPCFY**. The Editor will ignore any text on that insert line, exit insert mode, and offer the next line of the file.
- Position the cursor to any other line of the page and press **SPCFY**. The Editor will exit insert mode and offer the line you selected.
- Enter a command in the insert line that navigates to a different line, or off the current page. We will discuss how to enter commands next.

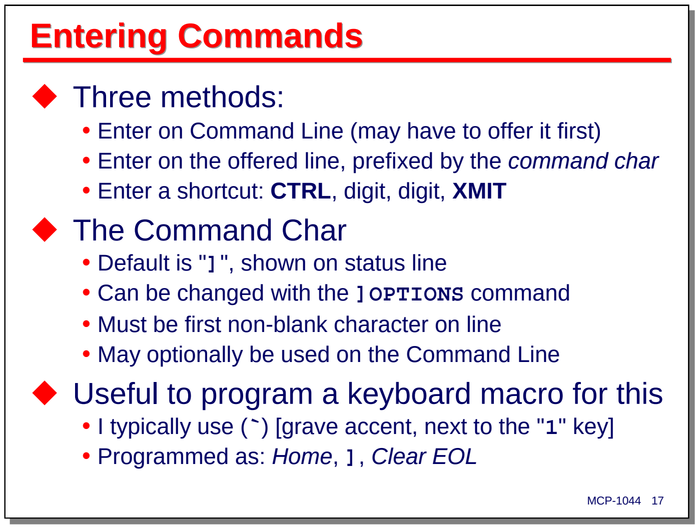
To split a line of text, make that line the offered line, position the cursor to the point where you want to split, and press **SPCFY**. Any text on or after that position will be inserted as a new line below this one. The original line will remain as the offered line. Note that you cannot split before the first non-blank character – pressing **SPCFY** there puts the Editor into insert mode.

Finally, you can offer the command line at the top of the page by positioning the cursor anywhere after the sequence number on the top line and pressing **SPCFY**. Only Editor commands can be entered on the command line, not source text, although it is more common to enter commands on the currently-offered line, as we will discuss next.

Editing Commands

The basic editing tasks just discussed allow us to alter existing lines in a file and insert new lines, but we are pretty much stuck with navigating within the current page or scrolling, one line at a time, forward in the file. If that is all there was, it would be somewhat, ah, limiting.

To do more, we need to use Editor commands, of which there are many. The bulk of the rest of the presentation will be take up with the main commands used for editing.



Entering Commands

- Three methods:
  - Enter on Command Line (may have to offer it first)
  - Enter on the offered line, prefixed by the *command char*
  - Enter a shortcut: **CTRL**, digit, digit, **XMIT**
- The Command Char
  - Default is "`]`", shown on status line
  - Can be changed with the `]OPTIONS` command
  - Must be first non-blank character on line
  - May optionally be used on the Command Line
- Useful to program a keyboard macro for this
  - I typically use (`` ` ``) [grave accent, next to the "`1`" key]
  - Programmed as: *Home*, `]`, *Clear EOL*

MCP-1044 17

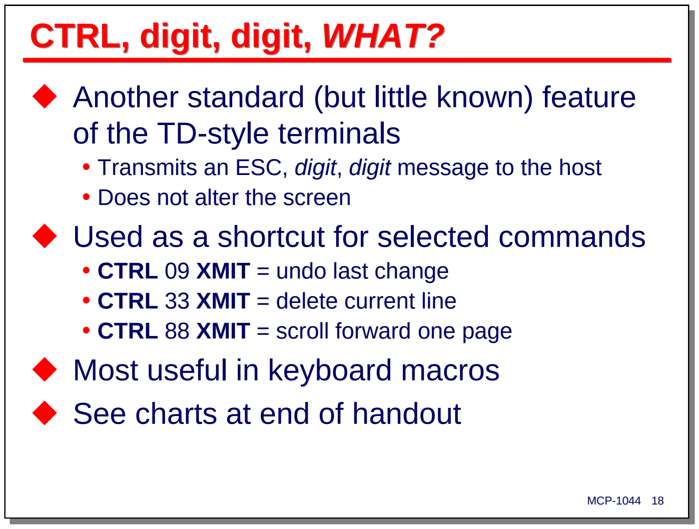
There are three ways you an enter a command in the Editor. Many commands can be entered in any of the three ways. They are all equivalent, and you should choose the method that is most convenient at the time.

- On the command line at the top of the page. In order to do this, the command line must be offered (i.e., it must be an unprotected field. You can offer the command line the same way you offer a line of text, by positioning the cursor there and pressing **SPCFY**. Also, whenever you navigate to a completely new page (as when scrolling forward or backward a full page, as we will discuss shortly), the Editor will automatically offer the command line).
- On the currently-offered line, whether in insert mode or normal edit mode. In order to distinguish a command from ordinary text, however, the command must be prefixed by the "command character," and that character must be the first non-blank character on the line. The default command character is "`]`".
- By entering a terminal shortcut command. You do this by pressing, in succession, the keys **CTRL**, *digit*, *digit*, **XMIT**, where "*digit*" is a decimal numeral, 0-9. More on this weirdness in a minute…

The command character defaults to "`]`", but you can change this using the **`OPTIONS`** command. When you enter a command on a currently-offered text line, it must be the first non-blank character on the line. When you enter a command on the command line, the command character is accepted, but it is optional.

Because entering commands on the offered line is a common thing to do, and because the Editor parses the entire text of the offered line as the command, you usually do not want any dangling text from the source line hanging around, especially if your terminal is set for transmit-full-page-in-forms-mode. For this reason, it is a good idea to clear the rest of the line after entering your command.

An even better idea is to program a keyboard macro to set up a line for command entry. I typically use the grave accent (`` ` ``) key next to the "`1`" key for this, although some emulators do not allow you to program this key. I program the key to perform the following: *Home*, `]`, *Clear-end-of-line*. That leaves the cursor after the right-bracket and the rest of the line clear for entry of the command and a subsequence press of the **XMIT** key.



CTRL, digit, digit, *WHAT?*

- Another standard (but little known) feature of the TD-style terminals
  - Transmits an ESC, *digit*, *digit* message to the host
  - Does not alter the screen
- Used as a shortcut for selected commands
  - **CTRL** 09 **XMIT** = undo last change
  - **CTRL** 33 **XMIT** = delete current line
  - **CTRL** 88 **XMIT** = scroll forward one page
- Most useful in keyboard macros
- See charts at end of handout

MCP-1044 18

What in the world is this **CTRL**, *digit*, *digit* business? It is standard feature of the Burroughs TD- and later-style terminals, and another nice feature that Darrell High finally found a good use for. What it does is transmit that two-digit code as a short message to the MCP host without altering the contents of the screen. In this way it is similar to the **SPCFY** key. The message the host receives is simply the three-character sequence ESC, *digit*, *digit*.

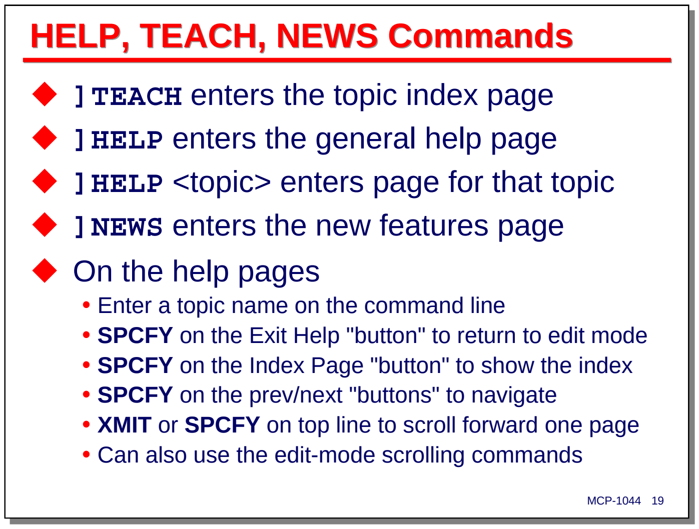
The Editor uses these two-digit codes as shortcuts for many of its commands. The slide shows a few examples. There are 100 possible two-digit codes, of which the Editor uses all but 11. Accompanying this presentation is a one-page chart showing the code assignments to Editor commands. In addition, I will show the relevant shortcut codes on the slides for the commands to which they apply.

This may seem like a wacky way to drive an Editor, but it's no worse than the cryptic commands used with any other dumb-terminal editor – such as EMACS or vi – it's just different. There are only a handful of commands that you use on a regular basis, and I've found that you quickly learn these and remember them.

Where these shortcut commands are really useful, however, are in keyboard macros. Because they are very short, and sending them does not disturb what's on the screen, they are generally a better choice than the command-character equivalents for programming into function keys for frequently used commands.

There is another chart accompanying this presentation that shows the keyboards macros that I use with the Editor. For example, I program the delete-one-line command, **CTRL** 33 **XMIT**, into the Control-F3 key.

These keyboard macros work well for me, and they may be a good starting point for you to develop your own set.



HELP, TEACH, NEWS Commands

- **]TEACH** enters the topic index page
- **]HELP** enters the general help page
- **]HELP** <topic> enters page for that topic
- **]NEWS** enters the new features page
- On the help pages
  - Enter a topic name on the command line
  - **SPCFY** on the Exit Help "button" to return to edit mode
  - **SPCFY** on the Index Page "button" to show the index
  - **SPCFY** on the prev/next "buttons" to navigate
  - **XMIT** or **SPCFY** on top line to scroll forward one page
  - Can also use the edit-mode scrolling commands

MCP-1044 19

Now that we have talked about how to enter a command, it is time to do some of them. The first commands to discuss are those for the help system.

The Editor has a very good help system, with a good index. It is organized a series of topics, or pages, with hyperlinks between topic pages. There are four commands that enter the help system:

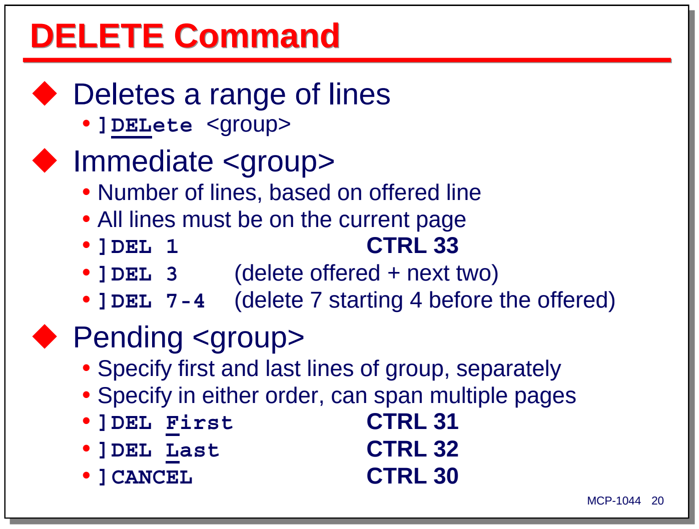
- **]TEACH** enters the help system at the topic index page. This is an alphabetical list of commands and other major help topics. You can **SPCFY** on one of the list items to go to that page.
- **]HELP** enters at the **TEACH** topic. **]HELP HELP** enters at the help-on-help topic.
- **]HELP** <topic> or **]TEACH** <topic> enters the page for that topic. For example, **]HELP COPY** displays the page for the **COPY** command.
- **]NEWS** displays a topic page with information about recent updates to the Editor.

Once you enter the help system, Editor stays in it until you explicitly exit back to editing mode. While in help mode, you can:

- Enter the name of a topic on the command line to display the page for that topic.
- Position and cursor and press **SPCFY** on the Exit Help "button" in the upper-left corner of the page to exit help.
- Position the cursor and press **SPCFY** on the Index Page "button" in the left margin to display the topic index page.
- Position the cursor and press **SPCFY** on the "prev" and "next" "buttons" to navigate through links and topics.

You can scroll through the pages of a help topic, and sequentially from topic to topic. Simply pressing **XMIT**, or pressing **SPCFY** with the cursor on the top line of the page will scroll forward one page. You can **SPCFY** on any word to attempt to retrieve a help topic of that name.

You can also use the regular Editor scrolling commands to move through help pages. Entering a positive or negative number in the command line will scroll that number of lines forward or backward. You can also use the **CTRL** 7x **XMIT** and **CTRL** 8x **XMIT** shortcut commands in help mode that we will be discussing shortly for scrolling lines and pages of text on the editing page.



DELETE Command

- Deletes a range of lines
  - `]DELete` <group>
- Immediate <group>
  - Number of lines, based on offered line
  - All lines must be on the current page
  - `]DEL 1` **CTRL 33**
  - `]DEL 3` (delete offered + next two)
  - `]DEL 7-4` (delete 7 starting 4 before the offered)
- Pending <group>
  - Specify first and last lines of group, separately
  - Specify in either order, can span multiple pages
  - `]DEL First` **CTRL 31**
  - `]DEL Last` **CTRL 32**
  - `]CANCEL` **CTRL 30**

MCP-1044 20

The first real editing command we will discuss is **`DELETE`**. This command, as you might expect, deletes a range of lines from the file. That is straightforward, but this command also introduces the idea of <group> and of a "pending" command.

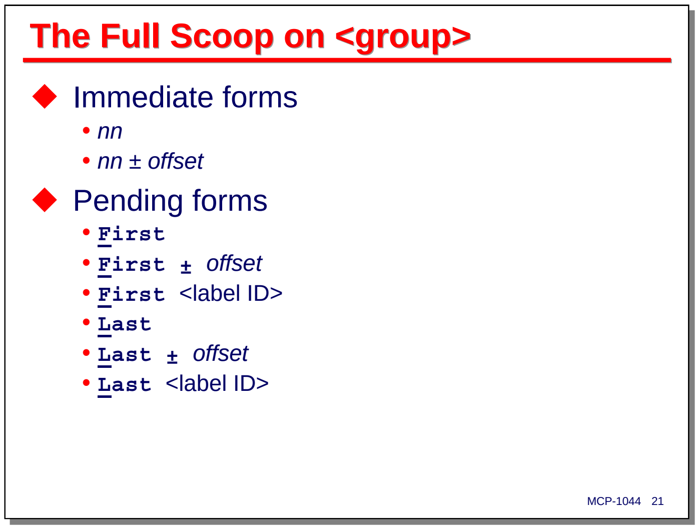
Many commands, like **`DELETE`**, operate on a contiguous range of lines. In order to use the command, you need to specify what that range of lines is. In CANDE, you specify the range by sequence numbers. Editor does it a different way – two different ways, in fact:

- For small numbers of lines, all on the current page, you can specify the range of lines as an "immediate" group. It is termed immediate because you specify the range all at once. You can specify an immediate group as:
  - A number of lines starting with the currently-offered line, e.g., **`]DEL 3`**. That will delete the currently-offered line plus the next two lines.
  - A number of lines relative to the currently-offered line, e.g., **`]DEL 3+7`**. That will delete three lines starting with the one seven lines below the currently-offered line. In both cases, the entire group must be visible on the current page. You cannot specify an immediate group that is off the page or only partly on the page.
- For any number of lines, large or small, you can use a "pending" group, which is a mark-and-bound technique.
  - You make the first line of the range the offered line and enter **`]DEL FIRST`**. You can also enter just **`]DEL F`**, or the shortcut **CTRL** 31 **XMIT**.
  - Then you position to the ending line of the range and enter **`]DEL LAST`**, or **`]DEL L`**, or **CTRL** 32 **XMIT**.
  - You can mark-and-bound in either order, first/last or last/first. Once the range is fully-specified, Editor automatically executes the command. This is why this method is termed "pending" – the command does not execute until both ends of the range are specified.

Once you have a command that is in a pending status, you cannot enter any other command that uses a <group> until that first command is either completed or canceled. The one exception to that is that you may enter a **`DELETE`** command with an immediate group.

To cancel a pending command, enter the **`]CANCEL`** command. Alternatively, there are several shortcut commands you can use, specific to the command that is pending. The scheme will be clear if you look at the chart of shortcut command codes and how they are mapped by tens to families of commands.

- **CTRL** 20 **XMIT** cancels **`COPY`** and **`MOVE`** commands
- **CTRL** 30 **XMIT** cancels **`DELETE`** and **`RENEW`** commands
- **CTRL** 40 **XMIT** cancels **`NUMBER`** and **`RESEQUENCE`** commands
- **CTRL** 50 **XMIT** cancels **`PARAGRAPH`**, **`SHIFT`**, **`CENTER`**, **`CHANGE`**, and **`ALIGN`** commands
- **CTRL** 60 **XMIT** cancels **`REPLACE`** commands.



The Full Scoop on <group>

- Immediate forms
  - *nn*
  - *nn ± offset*
- Pending forms
  - **F**irst
  - **F**irst ± *offset*
  - **F**irst <label ID>
  - **L**ast
  - **L**ast ± *offset*
  - **L**ast <label ID>

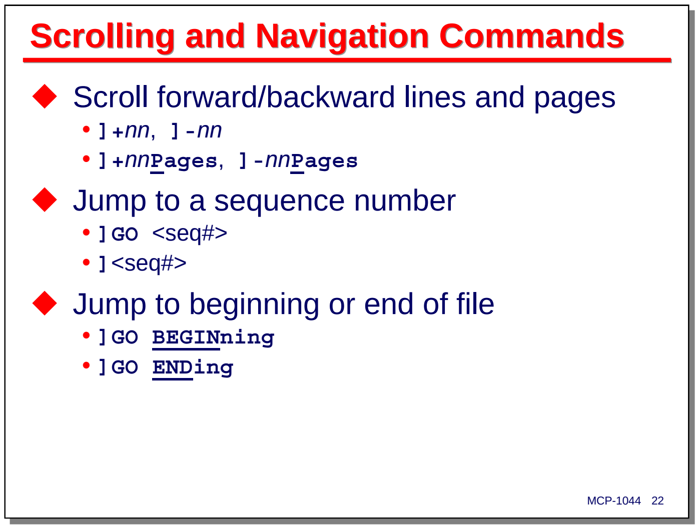
MCP-1044 21

The <group> construct is used in many Editor commands, so it it worthwhile at this point to take a closer look at it.

- *nn* – a simple integer specifies a group of that many lines, starting with the currently-offered line. All lines in the group must be visible on the current page.
- *nn* ± *offset* – an integer plus-or-minus another integer specifies a number of lines (the first integer) starting with the line offset by a number of lines (the second integer) from the currently-offered line. All lines in the group must be visible on the current page.
- **First** – specifies that the first line of the group is the currently-offered line.
- **First ±** *offset* – specifies that the first line of the group is the one offset by the specified number of lines above or below currently-offered line. The target line must be visible on the page.
- **First** <label ID> – specifies that the first line of the group is the one with the specified label. The target line does *not* need to be on the current page.
- **Last** – specifies that the last line of the group is the currently-offered line.
- **Last ±** *offset* – specifies that the last line of the group is the one offset by the specified number of lines above or below currently-offered line. The target line must be visible on the page.
- **Last** <label ID> – specifies that the last line of the group is the one with the specified label. The target line does *not* need to be on the current page.

A <label ID> is a bookmark – a name assigned to a line of the file. The discussion of labels and how they are used is coming up next.

When a command has a pending group, the command and portion of the group that has been specified will be displayed in the status line at the bottom of the page, e.g., **COPY F**, **DELETE L**, etc.



Scrolling and Navigation Commands

- Scroll forward/backward lines and pages
  - `]`+*nn*, `]`-*nn*
  - `]`+*nn*`Pages`, `]`-*nn*`Pages`
- Jump to a sequence number
  - `]GO` <seq#>
  - `]`<seq#>
- Jump to beginning or end of file
  - `]GO BEGINning`
  - `]GO ENDing`

MCP-1044 22

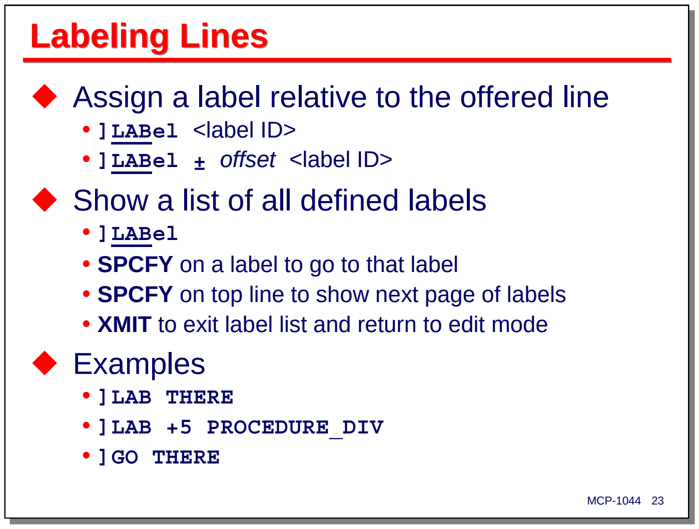
The next commands to discuss are those for scrolling the page and navigating through the source file.

First, you can scroll forward or backward a specified number of lines by simply entering a positive or negative number. This is actually a form the of **`]GO`** command, e.g., **`]GO+45`**, but the **`GO`** keyword is optional in this case and usually omitted. You can also scroll a number of pages by adding **`PAGES`** or **`P`** after the positive or negative number. No space between the number and **`P`** is necessary.

A common thing to do in editing is to jump to a particular sequence number in the source file. In CANDE you might say **`PAGE 310050`**. In Editor, you simply give the sequence number as a command, e.g., **`]310050`**. This is another example of the **`GO`** command where **`GO`** is optional and usually omitted. Note that the difference between scrolling a number of lines and jumping to a sequence number is the presence or absence of a plus or minus sign.

You can jump to the first or last line in the file by entering **`]GO BEGINNING`** or **`]GO ENDING`**. In this case the **`GO`** keyword is required.

Many of the navigation commands have shortcut commands. These are summarized on a slide coming up shortly.



Labeling Lines

- Assign a label relative to the offered line
  - `]LABel` <label ID>
  - `]LABel ±` *offset* <label ID>
- Show a list of all defined labels
  - `]LABel`
  - **SPCFY** on a label to go to that label
  - **SPCFY** on top line to show next page of labels
  - **XMIT** to exit label list and return to edit mode
- Examples
  - `]LAB THERE`
  - `]LAB +5 PROCEDURE_DIV`
  - `]GO THERE`

MCP-1044 23

You can assign a name to a line in a program. You can assign multiple names to the same line. If you assign a name that is already assigned to another line, it is automatically unassigned from its original line and then assigned to the new line that you specified.

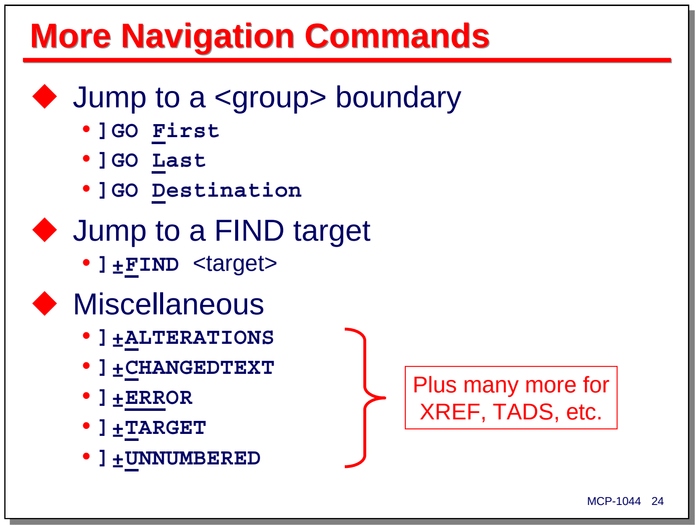
A label must begin with a letter, but can also contain decimal digits, the underscore character, and if the file being edited is **TEXTDATA** or one of the COBOL kinds, the hyphen (**-**). Labels can be up to 15 characters long.

You assign a label to a line using the **]LABEL** command, e.g., **]LAB THERE**. That will assign that <label ID> to the currently-offered line. You can also precede the <label ID> by a positive or negative offset, which will assign the label that many lines below or above the currently-offered line.

Once you have a label defined, you can jump to the line with that label entering a **GO** command with that <label ID>, e.g., **]GO THERE**.

The **]LABEL** command by itself will display a list of all labels in the file, along with the text of the line to which each label is assigned. You can do the following things on this list:

- **SPCFY** on the top line to scroll forward through the list (alas, there is no way to scroll backward).
- **SPCFY** on one of the labels to go to the line with that label.
- Transmitting any other message returns you to the current editing page.



More Navigation Commands

- Jump to a <group> boundary
  - `]GO First`
  - `]GO Last`
  - `]GO Destination`
- Jump to a FIND target
  - `]±FIND` <target>
- Miscellaneous
  - `]±ALTERATIONS`
  - `]±CHANGEDTEXT`
  - `]±ERROR`
  - `]±TARGET`
  - `]±UNNUMBERED`

Plus many more for XREF, TADS, etc.

MCP-1044 24

You can use the **GO** command to jump to a <group> boundary, **First**, **Last**, or **Destination**. The Destination boundary is used with the **COPY** and **MOVE** commands.

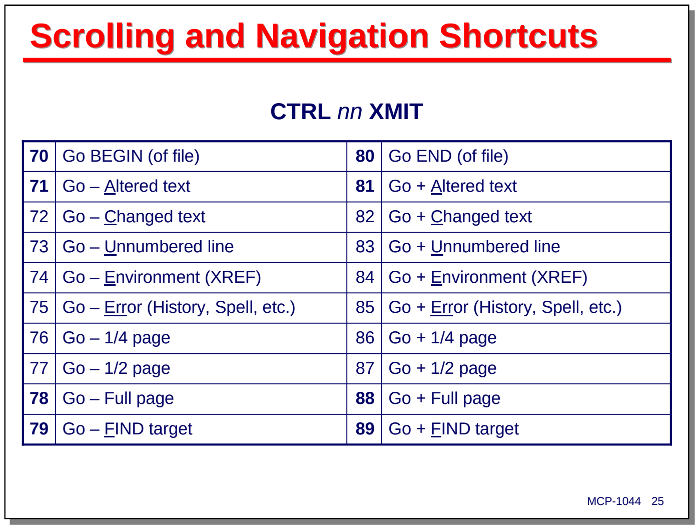
We have not yet discussed the **FIND** command, but one of the things it does is establish a <target> to search for.

- You can use **]GO+FIND** or **]GO-FIND** to move forward or backward among the occurrences of the <target> in the file.
- You can also use this form of **GO** command to simultaneously establish a <target> and go to the next or previous occurrence of it in the file, e.g., **]+FIND W-LAST-KEY**.

A common use of the **GO** command is to move forward or backward in the file to the next line that meets a certain criterion. The most common of these are:

- **]+ALTERATIONS** or **]-ALTERATIONS** – go to the next/prior line that has been changed in any way. I use these a lot, and have the shortcut versions programmed into function keys.
- **]+CHANGEDTEXT** or **]-CHANGEDTEXT** – go to the next/prior line that has been changed, except those changes for **DELETE**, **MOVE**, **RENUMBER**, or **RESEQUENCE**.
- **]+ERROR** or **]-ERROR** – go to the next/prior line that has an error. This is typically used to examine syntax errors after a compile.
- **]+TARGET** or **]-TARGET** – go to the next/prior line containing the current **FIND** target. I also use these a lot and have them programmed into function keys.
- **]+UNNUMBERED** or **]-UNNUMBERED** – go to the next/prior line that is on a boundary between a line that has a sequence number assigned and one that does not.

There many more options for the **GO** command that are used with features of the Editor that are outside the scope of this presentation, such as XREF, TADS, and the Dump Analyzer.



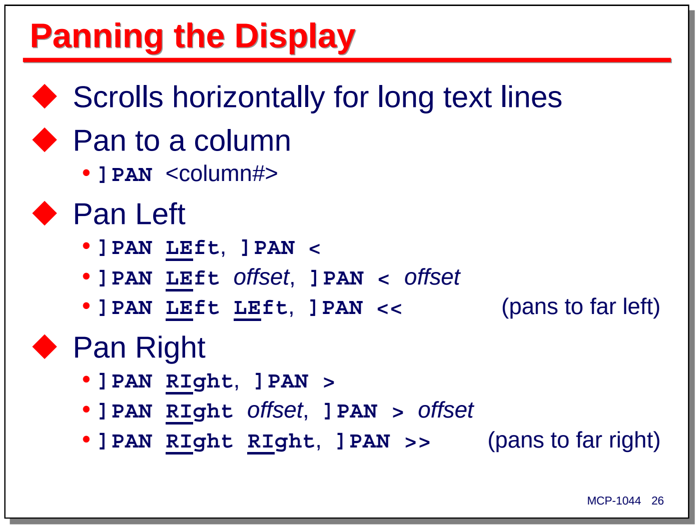
Scrolling and Navigation Shortcuts

CTRL *nn* XMIT

70	Go BEGIN (of file)	**80**	Go END (of file)
71	Go – Altered text	**81**	Go + Altered text
72	Go – Changed text	82	Go + Changed text
73	Go – Unnumbered line	83	Go + Unnumbered line
74	Go – Environment (XREF)	84	Go + Environment (XREF)
75	Go – Error (History, Spell, etc.)	85	Go + Error (History, Spell, etc.)
76	Go – 1/4 page	86	Go + 1/4 page
77	Go – 1/2 page	87	Go + 1/2 page
78	Go – Full page	**88**	Go + Full page
79	Go – FIND target	**89**	Go + FIND target

MCP-1044 25

Because navigation through the source file is such a common activity while editing, many of the variants of the **GO** command have command shortcut codes. The table on this slide summarizes the 7x and 8x-series codes. Many of these are good candidates for programming into keyboard macros.



Panning the Display

- Scrolls horizontally for long text lines
- Pan to a column
  - `]PAN` <column#>
- Pan Left
  - `]PAN LEft`, `]PAN <`
  - `]PAN LEft` *offset*, `]PAN <` *offset*
  - `]PAN LEft LEft`, `]PAN <<` (pans to far left)
- Pan Right
  - `]PAN RIght`, `]PAN >`
  - `]PAN RIght` *offset*, `]PAN >` *offset*
  - `]PAN RIght RIght`, `]PAN >>` (pans to far right)

MCP-1044 26

Thus far, we have been talking about scrolling vertically through the file. The Editor can also scroll horizontally for files that are wider than the number of columns available on the screen. This is referred to *panning* the display.

The **PAN** command allows you to pan several ways:

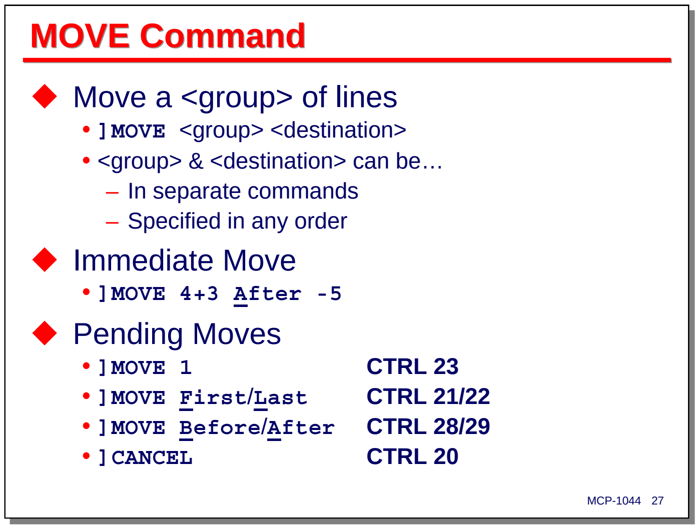
- **]PAN** nn – adjusts the display so that the specified column is the one at the left margin.
- **]PAN LEFT** – pans the display one screen-width to the left.
- **]PAN LEFT** nn – pans left by the specified number of columns.
- **]PAN LEFT LEFT** pans as far left as possible.

The keyword **RIGHT** can be substituted for **LEFT** in each of these forms to pan in the opposite direction. In addition the following substitutions can be made:

- **<** for **LEFT**
- **<<** for **LEFT LEFT**
- **>** for **RIGHT**
- **>>** for **RIGHT RIGHT**

When **>**, **>>**, **<**, and **<<** are used, the **PAN** command keyword may be omitted, e.g., **]>** is equivalent to **]PAN RIGHT**.

Note that panning is only effective if the text field of the file being edited will not completely fit in its area of the screen. You cannot pan farther left than the left margin, nor farther right than the right margin of the text field.



MOVE Command

- Move a <group> of lines
  - `]MOVE` <group> <destination>
  - <group> & <destination> can be…
    - In separate commands
    - Specified in any order
- Immediate Move
  - `]MOVE 4+3 After -5`
- Pending Moves

`]MOVE 1`	CTRL 23
`]MOVE First/Last`	CTRL 21/22
`]MOVE Before/After`	CTRL 28/29
`]CANCEL`	CTRL 20

MCP-1044 27

The **`MOVE`** command will move a range of lines from one place in the source file to another. This is another command that uses the <group> construct. It also introduces a variation on the <group> we have not seen before – the <destination>.

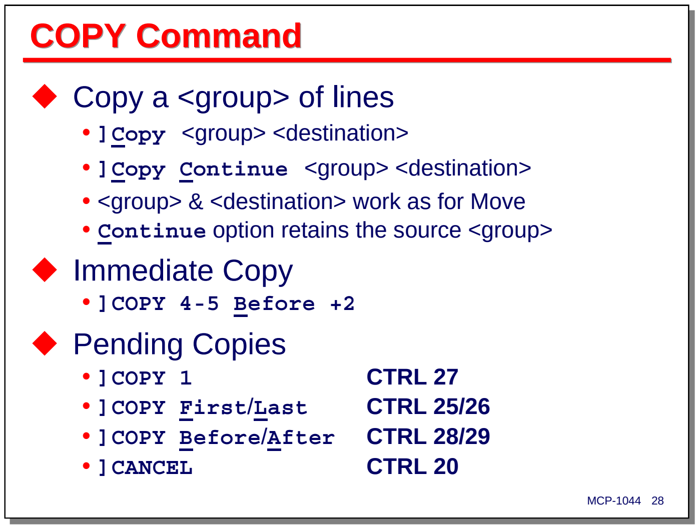
The <group> construct works for **`MOVE`** the same way it does for **`DELETE`**, in that it defines the range of lines to be moved. You may specify this range using either the immediate or pending methods.

The <destination> construct specifies where the records are to be moved. It designates a point in the file before or after a specified line.

- **`Before`** designates the moved records to be inserted before the currently-offered line.
- **`Before`** followed by a positive or negative number designates the moved records to be inserted before the line that number of lines below or above the currently-offered line. The target line must be visible on the current page.
- Similarly **`After`** designates the destination to be after the currently-offered line or some number of lines relative to the currently-offered line.

The first, last, and destination lines can be specified in any order. The move will not take place until all three have been specified. The moved lines will be inserted at the destination without sequence numbers.

As with all commands that can have a pending group, you must complete the **`MOVE`** command or cancel it before you can enter another command that uses a pending group.



COPY Command

- Copy a <group> of lines
  - `]Copy` <group> <destination>
  - `]Copy Continue` <group> <destination>
  - <group> & <destination> work as for Move
  - `Continue` option retains the source <group>
- Immediate Copy
  - `]COPY 4-5 Before +2`
- Pending Copies

`]COPY 1`	CTRL 27
`]COPY First/Last`	CTRL 25/26
`]COPY Before/After`	CTRL 28/29
`]CANCEL`	CTRL 20

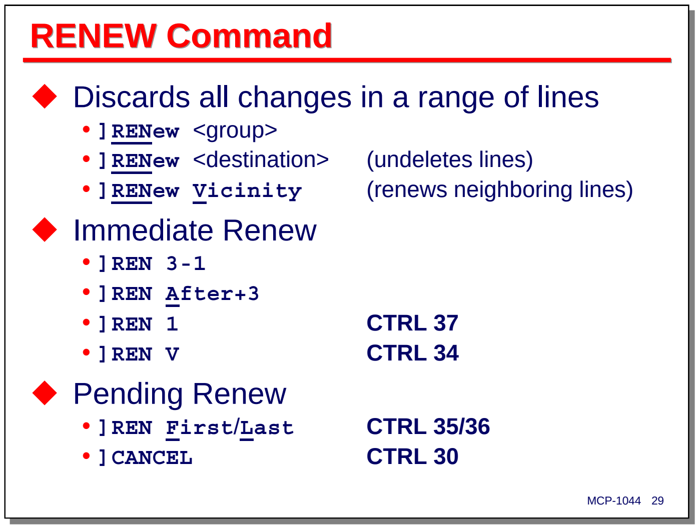
MCP-1044 28

The **`COPY`** command will copy a range of lines from one place in the source file to another. This command is similar to the **`MOVE`** command, except that it does not delete the lines at their original location.

The **`COPY`** command has an additional feature: **`Continue`**. When this keyword is specified after the command, the Editor retains the setting for the <group> of source lines. This allows you to copy the same set of lines to multiple destinations, one after the other, without having to re-specify the <group> each time. To turn off **`Continue`** mode, either **`]CANCEL`** the copy command or use **`]CONTINUE OFF`** (**CTRL** 24 **XMIT**).

The <group> construct and <destination> constructs work the same was as they do for **`MOVE`**. The first, last, and destination lines can be specified in any order. The copy will not take place until all three have been specified. The copied lines will be inserted at the destination without sequence numbers.

As with all commands that can have a pending group, you must complete the **`COPY`** command or cancel it before you can enter another command that uses a pending group.



RENEW Command

- Discards all changes in a range of lines
  - `]RENew <group>`
  - `]RENew <destination>` (undeletes lines)
  - `]RENew Vicinity` (renews neighboring lines)
- Immediate Renew
  - `]REN 3-1`
  - `]REN After+3`
  - `]REN 1` **CTRL 37**
  - `]REN V` **CTRL 34**
- Pending Renew
  - `]REN First/Last` **CTRL 35/36**
  - `]CANCEL` **CTRL 30**

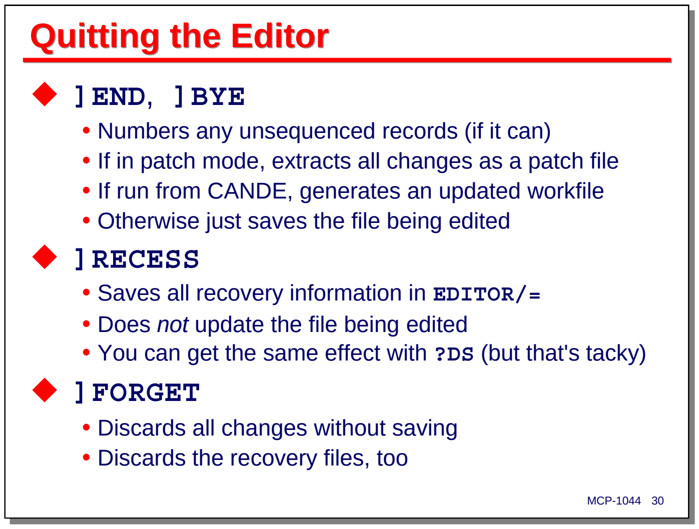
MCP-1044 29

The **`RENEW`** command is somewhat like the **`DELETE`** command, but instead of deleting a range of lines, this command deletes *changes to a range of lines*. In other words, it reverts the lines back to the state they were in at the beginning of the editing session. If you are running in patch mode, it deletes all patches in the specified range, whether they have been added in the present session or earlier.

Renumbered and resequenced lines are considered to be changed by the **`RENEW`** command and will be reverted to their prior sequence numbers.

`RENEW` can be used in both immediate and pending forms. It has the following variations:

- **`]RENEW`** <group> – reverts all changes in the specified range of lines. The first and last lines of the group must have sequence numbers, but other lines within they group may be unsequenced.
- **`]RENEW`** <destination> – renews lines above or below the specified destination line. This is used to un-delete lines that were formerly adjacent to the destination line. For example, **`]REN A-3`** will revert any deleted lines that were formerly between three lines above the offered line and two lines above the offered line. **`]REN B+3`** would revert any lines between two and three lines after the offered line.
- **`]RENEW Vicinity`** – reverts all changes surrounding the currently-offered line. The start of the range is the first changed line before the offered line. The end of the range is the last line after the offered line.



Quitting the Editor

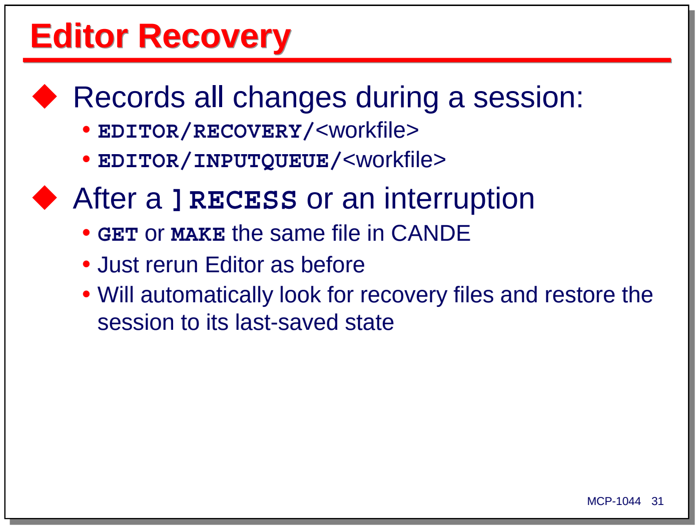
- **]END**, **]BYE**
  - Numbers any unsequenced records (if it can)
  - If in patch mode, extracts all changes as a patch file
  - If run from CANDE, generates an updated workfile
  - Otherwise just saves the file being edited
- **]RECESS**
  - Saves all recovery information in **EDITOR/=**
  - Does *not* update the file being edited
  - You can get the same effect with **?DS** (but that's tacky)
- **]FORGET**
  - Discards all changes without saving
  - Discards the recovery files, too

MCP-1044 30

With this initial set of commands under our belt, it's time to talk about how you end an editing session and get out of the Editor. There are four commands that do this:

- **]END** and **]BYE** are equivalent. They end the editing session normally. If you are working in direct-edit mode, the Editor will attempt to assign sequence numbers to any unsequenced lines, do the equivalent of a CANDE **UPDATE** command, write a new workfile, and exit, passing that workfile back to the originating environment. If you are working in patch mode, the Editor will extract just the changes to the base source file and write those to the work file. Note that in the case of CANDE, these commands do not save the workfile. The updated workfile remains as the CANDE workfile, and you will still need to do a **SAVE** in CANDE.
- **]RECESS** saves the current state of the editing session in the Editor recovery files and exits. It does not create an updated workfile. You can run the Editor with the same workfile later to recover the session, as discussed on the next slide. You can get nearly the same effect by DS-ing the Editor, as the recovery files are updated as each command completes, but that is not a recommended technique for ending an editing session.
- **]FORGET** discards all changes made during the editing session and exits. No Editor recovery files are saved, and the original workfile is unaffected. Except for updating some file last-accessed timestamps in the disk directory, the result is as if the editing session never took place.[1]

[1] Assuming you did not otherwise create or modify other files during the editing session, as is possible with the **PRINT**, **SAVE AS**, and a few other commands.



Editor Recovery

- Records all changes during a session:
  - `EDITOR/RECOVERY/`<workfile>
  - `EDITOR/INPUTQUEUE/`<workfile>
- After a `]RECESS` or an interruption
  - `GET` or `MAKE` the same file in CANDE
  - Just rerun Editor as before
  - Will automatically look for recovery files and restore the session to its last-saved state

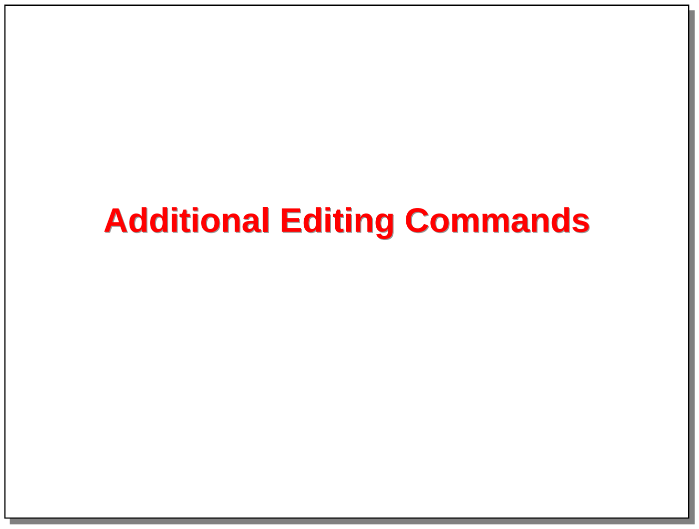
MCP-1044 31

The Editor has an excellent recovery mechanism. As you enter lines of text and commands, the Editor records your changes in two recovery files, named as shown on the slide. If the Editor terminates abnormally (or worse, the whole system crashes), these files will allow you to recover your editing session, generally to the last command that was successfully completed.

The recovery files are saved after both an abnormal termination and a **`]RECESS`** command. Recovery of the editing session for both cases is done in the same way.

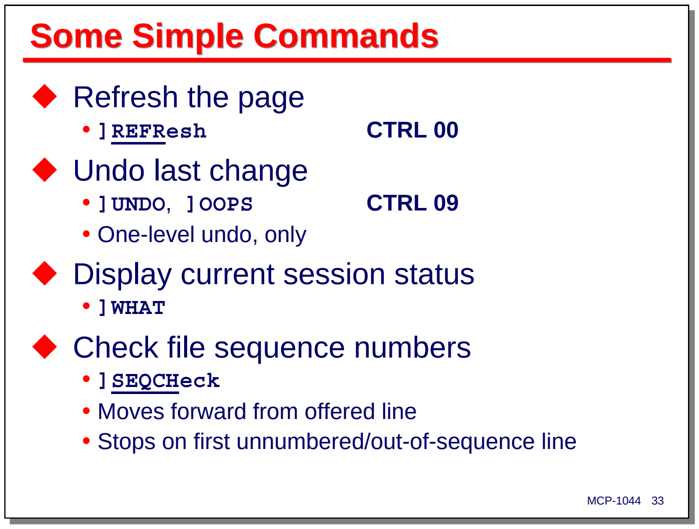
- First, **`Get`** or **`Make`** the same workfile in CANDE. If necessary, recover the file in CANDE first. The workfile *must* have the same name as for the original editing session, since the recovery files are located using that name.
- Next, just rerun the Editor – **`U ED`**. If you are running in patch mode, you do not need to specify the base source file. The name of that file is in the recovery files and Editor will pick it up from there.
- The Editor will then locate the recovery files for your workfile, recover your editing session, and display the last page of text you were working on. Essentially what the Editor stores in the recovery files is a patch to the original workfile, whether you were working in patch mode or not. If you have a very large number of changes to a large source file, it may take a little while to process all of that data and completely restore the session to its former state.

Note that the **`]RECESS`** command quits the Editor gracefully, saving the recovery files. **`]FORGET`** quits the Editor and discards the recovery files. Recovery after a **`]FORGET`** command is not possible.



Additional Editing Commands

There is lots more to the Editor, so let us continue with some additional editing commands.



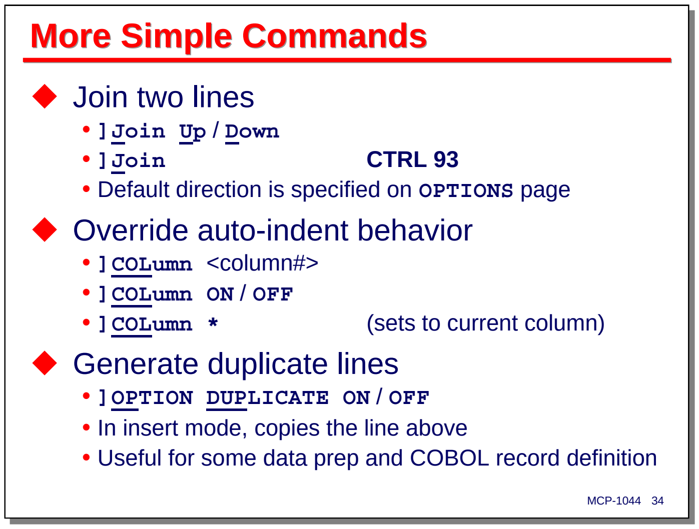
Some Simple Commands

- Refresh the page
  -]REFResh   CTRL 00
- Undo last change
  -]UNDO,]OOPS   CTRL 09
  - One-level undo, only
- Display current session status
  -]WHAT
- Check file sequence numbers
  -]SEQCHeck
  - Moves forward from offered line
  - Stops on first unnumbered/out-of-sequence line

MCP-1044 33

To begin this section, we will discuss some simple Editor commands.

- **]REFRESH** – Sometimes the editing page can become corrupted, usually by messages from other programs that are sending messages to your terminal. You can redisplay the last screen that Editor sent you using the **]REFRESH** command (**CTRL** 00 **XMIT**).
- **]UNDO** – Editor supports one level of undo. To activate this, use **]UNDO**, **]OOPS**, or **CTRL** 09 **XMIT**.
- **]WHAT** – Editor will display the current status of the editing session in response to the **]WHAT** command. This display is very similar to the welcome page that usually displays too quickly to read what it says when the Editor starts up.
- **]SEQCHECK** – One of the nice features of the Editor is that it is a lot less dependent on sequence numbers than CANDE. In particular, CANDE usually cannot edit a file that has no sequence numbers, or which has numbers out of sequence. The Editor **]SEQCHECK** command will scan the workfile forward from the currently-offered line, checking for valid sequence numbers. It will stop at the first sequence error or unsequenced record it finds. You can resequence the offending records at at that point and reenter **]SEQCHECK** to continue checking the file for more errors.



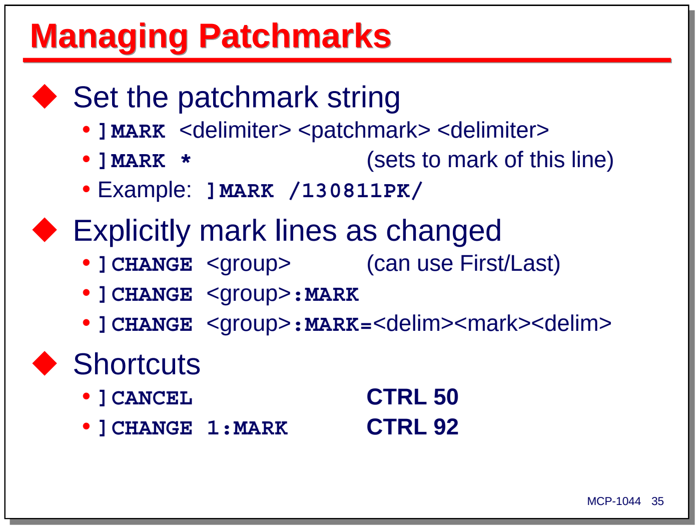
More Simple Commands

- Join two lines
  - `]Join Up / Down`
  - `]Join`   **CTRL 93**
  - Default direction is specified on `OPTIONS` page
- Override auto-indent behavior
  - `]COLumn <column#>`
  - `]COLumn ON / OFF`
  - `]COLumn *`   (sets to current column)
- Generate duplicate lines
  - `]OPTION DUPLICATE ON / OFF`
  - In insert mode, copies the line above
  - Useful for some data prep and COBOL record definition

MCP-1044 34

Here are some more simple Editor commands:

- **`]JOIN`** – In talking about Editor basics, we discussed how to split a line with the **SPCFY** key. You can go the other way and join two adjacent lines with the **`]JOIN`** command.
  - The default join direction is down (the offered line is joined with the line below it), but you can change this with the **`]OPTIONS`** command.
  - You can also specify the join direction in the command, e.g., **`]JOIN UP`**.
  - If the full text of the two lines will not fit in the text field, Editor does not do the join, and positions the cursor at the first character of the portion that won't fit.
- **`]COLUMN`** – Normally when you move to a new line while editing, the Editor will automatically position the cursor at the first non-blank character on the line. When inserting lines into the file, the Editor will position the cursor below the first non-blank character on the prior line. You can override this default behavior with the **`]COLUMN`** command.
  - **`]COL 25`** will turn on fixed-column mode and position the cursor to the 25th character of the record.
  - **`]COL OFF`** will return to auto-indent mode.
  - **`]COL *`** will turn on fixed-column mode and set the column to the one the cursor was in when the currently-offered line became the offered line.
- **`]OPTION DUPLICATE`** – Remember how with a card punch you used to be able to duplicate portions of the previous card you punched? No? Anyone…? Anyway, the Editor has a similar feature if you need to enter a number of lines that are mostly the same but differ in only a few columns. This is often useful when entering data for tables or record descriptions in the COBOL Data Division. The **`DUPLICATE`** option changes the way that new lines are offered in insert mode. Normally in insert mode, the offered line is initially blank. When **`DUPLICATE`** is on, new lines are initially a duplicate of the line above, i.e., the one you just entered. The setting of this option only has effect in insert mode, but the state of the option persists after you exit insert mode. To reset it, use **`]OP DUP OFF`**. It is not unusual for **`]COLUMN`** and **`]OPTION DUPLICATE`** to be used together.



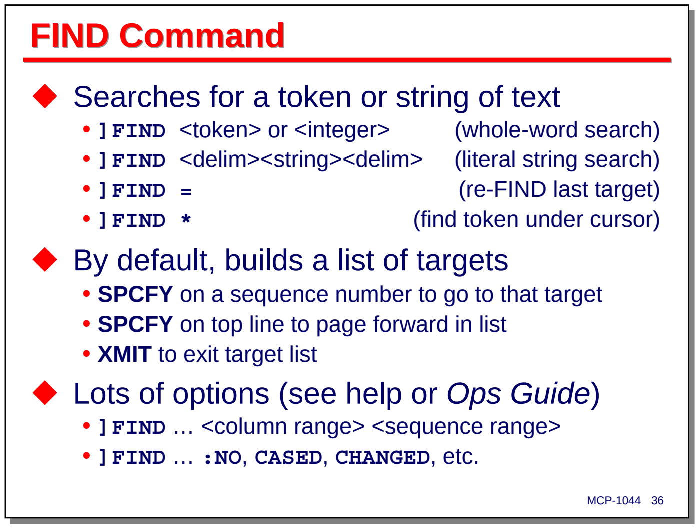
Managing Patchmarks

- Set the patchmark string
  - `]MARK` <delimiter> <patchmark> <delimiter>
  - `]MARK *` (sets to mark of this line)
  - Example: `]MARK /130811PK/`
- Explicitly mark lines as changed
  - `]CHANGE` <group> (can use First/Last)
  - `]CHANGE` <group>`:MARK`
  - `]CHANGE` <group>`:MARK=`<delim><mark><delim>
- Shortcuts
  - `]CANCEL` **CTRL 50**
  - `]CHANGE 1:MARK` **CTRL 92**

MCP-1044 35

Many MCP source file record formats support extra columns that can be used for a patchmark – eight to 10 columns typically used for version or "blame" information. Editor has a couple of commands that are useful for managing patchmarks.

- **]MARK** – You can specify a string to be used as a patchmark during your session using this command. The string is delimited the same way that **FIND** targets are delimited in CANDE – the delimiter can be any special character, but the two delimiters must be the same character. **]MARK *** sets the session patchmark to the patchmark field of the currently-offered line. Editor will automatically place that patchmark string on every line that you modify. Patchmarks in lines affected by the **CENTER**, **CHANGE**, **INSERT**, **MERGE**, **MOVE**, **NUMBER**, **RENEW**, **RESEQUENCE**, and **SHIFT** commands are not, by default, changed when a session patchmark is in effect. Some of these commands do, however, have a **MARK** option that will set a patchmark on the records they affect.
- **]CHANGE** – Occasionally it is useful (especially when editing in patch mode) to mark a range of lines as changed without actually changing them. One common case is setting a patchmark on an existing group of lines. The **]CHANGE** command will mark all lines in the specified <group> (which can be immediate or pending) as changed. If its **:MARK** option is specified, the command will also modify the patchmark field, either to the current session patchmark, or to a specified patchmark string.



FIND Command

- Searches for a token or string of text
  -]FIND <token> or <integer> (whole-word search)
  -]FIND <delim><string><delim> (literal string search)
  -]FIND = (re-FIND last target)
  -]FIND * (find token under cursor)
- By default, builds a list of targets
  - **SPCFY** on a sequence number to go to that target
  - **SPCFY** on top line to page forward in list
  - **XMIT** to exit target list
- Lots of options (see help or *Ops Guide*)
  -]FIND … <column range> <sequence range>
  -]FIND … :NO, CASED, CHANGED, etc.

MCP-1044 36

Every editor needs a **FIND** command. The Editor has one that is based on CANDE's.

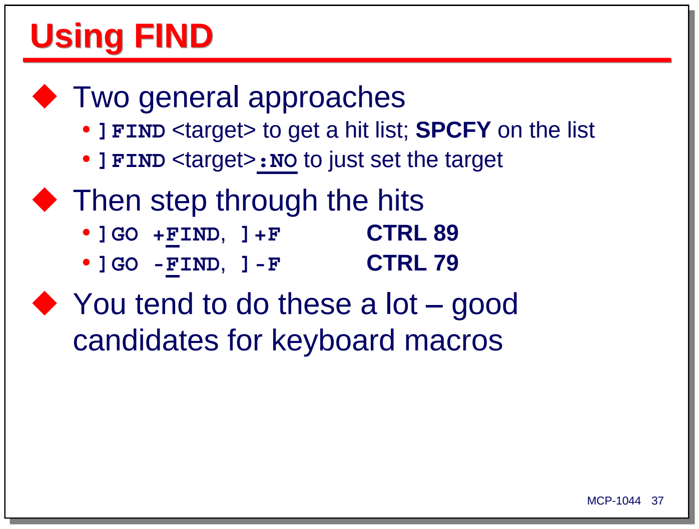
Like CANDE, **FIND** has two ways of matching its target, token and literal, but they are specified slightly differently than in CANDE.

- If the target is a simple token, identifier, or an integer, Editor will search using token mode. What constitutes a token or identifier depends on the file kind of the workfile, and the Editor will parse lines from the workfile accordingly. For example, when editing COBOL source, the hyphen (**-**) is considered part of an identifier.
- If the target is bracketed by a delimiter character, Editor will search using literal mode. The target will be matched even if it exists as part of a larger identifier or token.
- **]FIND =** simply reuses the last target that was specified.
- **]FIND *** initiates a token-mode search using the word currently under the cursor.

By default, Editor searches the file forward from the currently-offered line and builds a list of all lines that match the target.

- It replaces the editing page with this list, which shows the sequence number and text field of each of the lines.
- If the list occupies more than one page, you can continue the list by pressing **SPCFY** on the top line. You can only scroll this list forward. There is no way to scroll the list backward.
- Pressing **XMIT** will terminate the list and return you to the original editing page, but will leave the current target specification in place.
- You can navigate to one of those lines by placing the cursor anywhere on the sequence number and pressing **SPCFY**.
- Once back on the editing page, you can move forward and backward from match to match by using **]+FIND** (**CTRL** 89 **XMIT**) and **]-FIND** (**CTRL** 79 **XMIT**).

]FIND has a large number of options, including the ability to restrict the search to a specified range of columns and/or sequence numbers. You can also do case sensitive or insensitive searches, search only changed lines, etc. See the help pages for details.



Using FIND

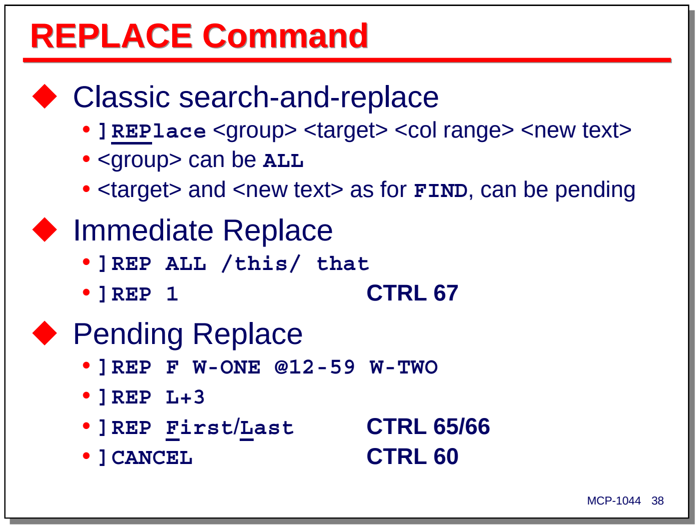
- Two general approaches
  - `]FIND` <target> to get a hit list; **SPCFY** on the list
  - `]FIND` <target>`:NO` to just set the target
- Then step through the hits
  - `]GO +FIND, ]+F` **CTRL 89**
  - `]GO -FIND, ]-F` **CTRL 79**
- You tend to do these a lot – good candidates for keyboard macros

MCP-1044 37

There are two general approaches to using **`FIND`**.

- Specify a target and get the list of matching lines, then **SPCFY** on a sequence number in that list to go to one of the matches.
- Specify a target with the **`:NO`** command option, then move to the first match using **`]GO +FIND`**, or **`]+FIND`**, **`]+F`**, or the command shortcut, **CTRL** 89 **XMIT**.

I use the second approach, with **`:NO`**, quite a lot. For example, **`]FIND W-TXN-COUNT:NO`**. This sets the find target but does not search the file. Once the target is set you can do **`]+FIND`** and **`]-FIND`** (I have their shortcuts programmed as keyboard macros) to do the actual searching.



REPLACE Command

- Classic search-and-replace
  - **]REPlace** <group> <target> <col range> <new text>
  - <group> can be **ALL**
  - <target> and <new text> as for **FIND**, can be pending
- Immediate Replace
  - **]REP ALL /this/ that**
  - **]REP 1** **CTRL 67**
- Pending Replace
  - **]REP F W-ONE @12-59 W-TWO**
  - **]REP L+3**
  - **]REP First/Last** **CTRL 65/66**
  - **]CANCEL** **CTRL 60**

MCP-1044 38

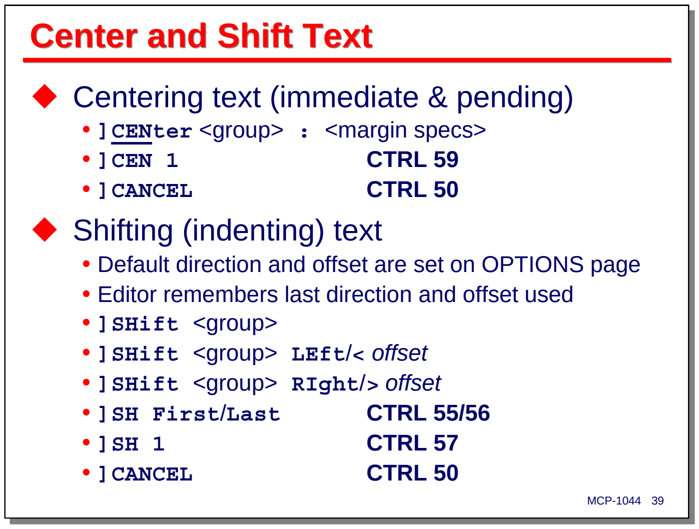
If you have **FIND**, you probably also have **REPLACE**, and Editor does. Unlike **]FIND**, **]REPLACE** requires you to specify a <group> of lines over which the replacement will take place. You also need to specify a source target to search for and a replacement string to be substituted for the matching targets. Targets can be matched using either the token or literal method, as for **]FIND**.

The <group> can be specified using either the immediate or pending methods. This command supports the special group **ALL**, which will search and replace over the entire workfile, regardless of where the currently-offered line is.

For the pending method, you can specify the target and substitution strings at either the start or end of the group (or both, with the second specification overriding the first). You can also specify the target/substitution strings separately from the <group>. Editor remembers the target and substitution strings until they are changed, allowing you to do an incremental search and replace. The **REPLACE** target is the same as the **FIND** target, so you can use **]FIND** to locate a line to change and then one of the immediate forms of **]REPLACE** to do the actual substitution.

If the substitution of a target match would result in overflow of the text field on a line, Editor stops the **]REPLACE** command and scrolls to that line, placing the cursor on the first character that would overflow. You can manually handle the overflow (say, by splitting the line) and then restart the search-and-replace operation at that line by doing **]REP F** or **CTRL** 65 **XMIT**. This is a much nicer way to handle line overflows that CANDE has.

You can restrict the columns used in a **]REPLACE** command by specifying the column range between the source and substitution strings. There are also options to do case sensitive/insensitive searches, search only changed lines, and to replace only the first match on a line. See the help page for details.



Center and Shift Text

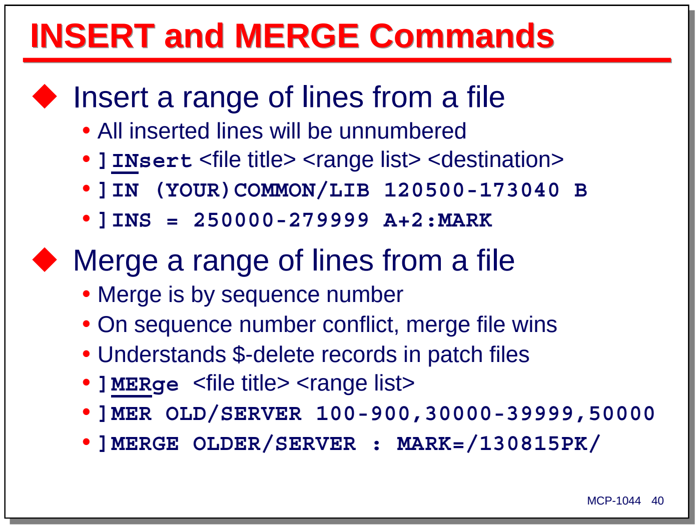
- Centering text (immediate & pending)
  - `]CENter <group> : <margin specs>`
  - `]CEN 1` CTRL 59
  - `]CANCEL` CTRL 50
- Shifting (indenting) text
  - Default direction and offset are set on OPTIONS page
  - Editor remembers last direction and offset used
  - `]SHift <group>`
  - `]SHift <group> LEft/<` *offset*
  - `]SHift <group> RIght/>` *offset*
  - `]SH First/Last` CTRL 55/56
  - `]SH 1` CTRL 57
  - `]CANCEL` CTRL 50

MCP-1044 39

Proper indentation and alignment of text is important to reliable and maintainable programming. The Editor has a couple of commands that assist with this.

- **]CENTER** – You can center a line or group of lines using this command. By default, the text is centered between the left and right margins of the text field (e.g., columns 7-72 for COBOL). As an option, you can include specifications of the left and right margins, first line indent, etc., as an option. See the help page for details.
- **]SHIFT** – You can shift groups of lines left or right within the text field using this command. This is especially useful when adjusting lines due to, say, the addition of an **IF** statement in the code.
  - You specify the direction of shift (**<** and **>** can be used in place of **LEFT** and **RIGHT**, respectively) and the number of columns to shift. The direction and number of columns can be specified separately from the <group>. A default direction and shift can be specified with the **]OPTIONS** command (normally it's ***RIGHT* 2**).
  - If the shift would cause non-blank text to overflow the text field, the **]SHIFT** command stops at the first line where this would occur, with the cursor positioned on the character that would overflow. You can manually fix the overflow condition and then restart the command with **]SH F**. This is similar to the overflow behavior of **]REPLACE**.

In addition to centering text, there is a **]PARAGRAPH** command that will reflow text within a group of lines. There is also an **]ALIGN** command that is similar to **]SHIFT**. Both of these are beyond the scope of this presentation, so see the help pages for details.



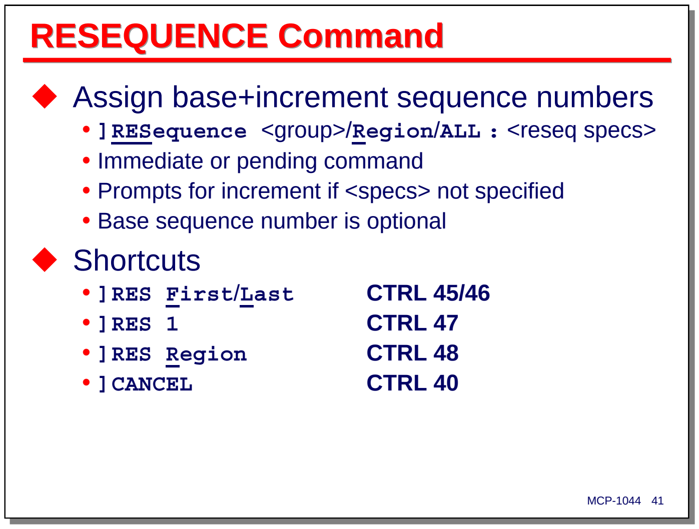
INSERT and MERGE Commands

- Insert a range of lines from a file
  - All inserted lines will be unnumbered
  - `]INsert` <file title> <range list> <destination>
  - `]IN (YOUR)COMMON/LIB 120500-173040 B`
  - `]INS = 250000-279999 A+2:MARK`
- Merge a range of lines from a file
  - Merge is by sequence number
  - On sequence number conflict, merge file wins
  - Understands $-delete records in patch files
  - `]MERge` <file title> <range list>
  - `]MER OLD/SERVER 100-900,30000-39999,50000`
  - `]MERGE OLDER/SERVER : MARK=/130815PK/`

MCP-1044 40

The next two commands allow you to include text from an external file into your workfile. These are similar to the corresponding commands in CANDE.

- **`]INSERT`** – This command will allow you to insert a range of lines (or a list of ranges) from a file at a specified point in your workfile.
  - – The <destination> is specified the same way as for the **`]COPY`** and **`]MOVE`** commands.
  - – The Editor remembers the name of the last external file you used in the session, so specifying "**`=`**" instead of a file name will obtain the lines from that file.
  - – The records will be inserted into your workfile without sequence numbers, but with their original patchmark, unless you override that with the **`:MARK`** option.
- **`]MERGE`** – This command will merge records from an external file into your workfile, positioning them by their sequence numbers.
  - – If lines are merged from the external file with the same sequence numbers as in the workfile, the external records replace the corresponding ones in the workfile. This is the behavior of the **`RMERGE`** command in CANDE.
  - – The Editor **`]MERGE`** command understands the compiler convention for $-records, and will apply those conventions when merging. In particular, a line with just a "**`$`**" in the first column will delete the corresponding line from the work file. **`$SET`**/**`POP VOIDT`** and **`$SET`**/**`POP DELETE`** records will cause the corresponding ranges of records to be deleted from the work file. This allows the Editor to merge patch files the same way that compilers and the **`SYSTEM/PATCH`** utility do.
  - – Patchmarks from the external file will be preserved, unless they are overridden by the **`:MARK`** option on the command.
  - – An "**`=`**" can be used instead of a file name, as with the **`]INSERT`** command.



RESEQUENCE Command

- Assign base+increment sequence numbers
  - `]RESequence` <group>/`Region`/`ALL :` <reseq specs>
  - Immediate or pending command
  - Prompts for increment if <specs> not specified
  - Base sequence number is optional
- Shortcuts

`]RES First/Last`	CTRL 45/46
`]RES 1`	CTRL 47
`]RES Region`	CTRL 48
`]CANCEL`	CTRL 40

MCP-1044 41

Editor has two approaches to applying sequence numbers to lines in the workfile. The first these is similar to the CANDE **`RESEQ`** command.

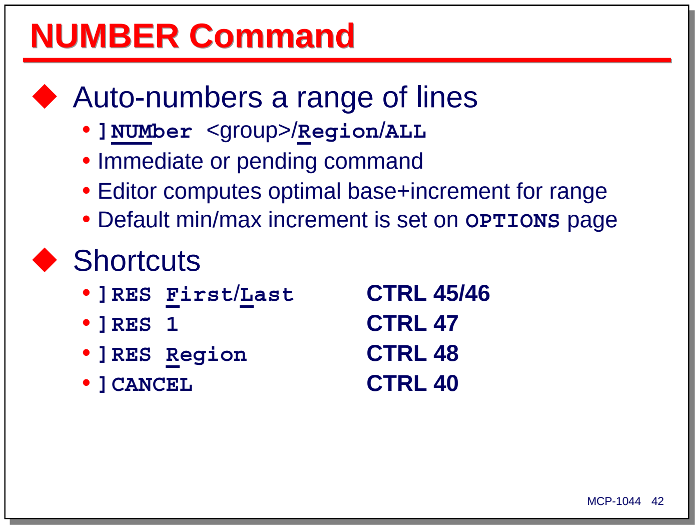
The Editor **`]RESEQUENCE`** command applies sequence numbers with a specified increment, and optionally, a specified base, to a <group> of lines in the workfile. In addition to a standard immediate or pending <group>, you can also specify:

- **`REGION`** – the range of unsequenced lined that contains the currently-offered line
- **`ALL`** – the entire workfile.

You can optionally specify the resequence specifications after a colon (**`:`**) in the typical base+increment fashion. If you omit the base (starting) sequence number, Editor will compute it by taking the sequence number from the line before the start of the <group> and adding the increment value to that. If you omit the colon and resequence specifications, Editor will prompt you for them with form fields at the top of the page. The base field may be left blank, but the increment field must have a number.

If the resequence specifications would cause an out-of-sequence condition in the file, the command is aborted. You can use a smaller base or increment, or extend the range of records to be resequenced. Alternatively, you can cancel the command.

Resequenced records are indicated by a "**`#`**" in the flag column next to the sequence number. Resequencing does not alter the patchmark field in the affected records.



NUMBER Command

- Auto-numbers a range of lines
  - `]NUMber <group>/Region/ALL`
  - Immediate or pending command
  - Editor computes optimal base+increment for range
  - Default min/max increment is set on `OPTIONS` page
- Shortcuts

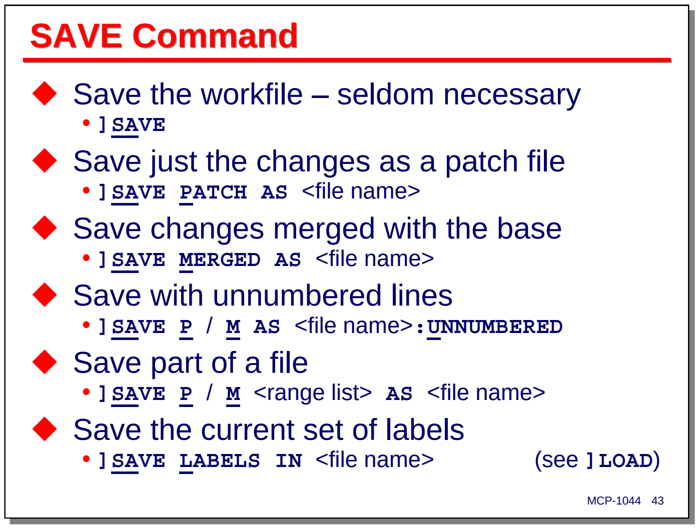
• `]RES First/Last`	**CTRL 45/46**
• `]RES 1`	**CTRL 47**
• `]RES Region`	**CTRL 48**
• `]CANCEL`	**CTRL 40**

MCP-1044 42

The **`]NUMBER`** command is similar to the **`]RESEQUENCE`** command, but in this case all you do is specify the <group> of lines to be renumbered. Editor chooses the base and increment. The **`REGION`** and **`ALL`** options work as for **`]RESEQUENCE`**.

Editor examines the sequence numbers in the lines immediately before and after the <group> to be renumbered. Based on those sequence numbers and the number of lines in the <group>, Editor determines a suitable increment. If possible, this increment will be a power of ten, five times a power of ten, or two times a power of ten. It then resequences the records in the <group> using this increment.

The **`]NUMBER`** command is particularly useful for assigning sequence numbers to a range of unsequenced lines that have been inserted into the workfile.



SAVE Command

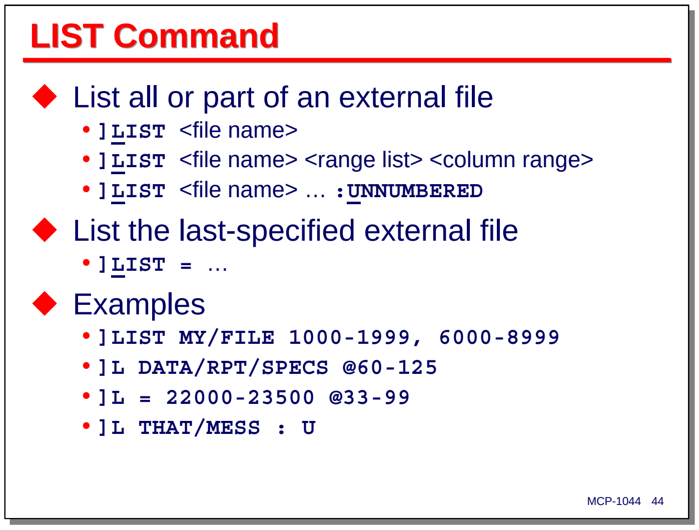
- Save the workfile – seldom necessary
  - `]SAVE`
- Save just the changes as a patch file
  - `]SAVE PATCH AS` <file name>
- Save changes merged with the base
  - `]SAVE MERGED AS` <file name>
- Save with unnumbered lines
  - `]SAVE P / M AS` <file name>`:UNNUMBERED`
- Save part of a file
  - `]SAVE P / M` <range list> `AS` <file name>
- Save the current set of labels
  - `]SAVE LABELS IN` <file name> (see `]LOAD`)

MCP-1044 43

Normally you do not save the workfile from within Editor. Instead, you exit the editing session using **`]END`** or **`]BYE`** and then save the workfile in CANDE. The **`]SAVE`** command in Editor can save the workfile, but it is usually used for other purposes:

- **`]SAVE`** – used by itself, this command will save the workfile and continue the editor session. There is seldom a need to do this, as the workfile is fully recoverable in the event of an interrupted session.
- **`]SAVE PATCH AS`** <file name> – this will save all of the changes you have made during your session as a patch file. That file will contain only the changes, recorded in a format compatible with the compilers and **`SYSTEM/PATCH`**.
- **`]SAVE MERGED AS`** <file name> – this will save a full copy of your workfile under a different name. This variant is useful when working in patch mode to create a complete, updated source file with the patches merged against the base source and included in the new file.
- Both the **`PATCH`** and **`MERGED`** variants can be used with the **`:UNNUMBERED`** option to save a file with blanks in the sequence number field. This may be useful to prepare a source file for transfer to another system that does not use sequence numbers.
- **`]SAVE LABELS IN`** <file name> – this will save your current set of line labels in a file. That file can be reloaded and used in a later editing using a form of the **`]LOAD`** command, discussed later.

The **`]SAVE`** command also supports options to save a range of records, assign a new patchmark string to the saved records, and save the file as a different **`FILESTRUCTURE`** (e.g., as a byte stream file). See the help page for details.



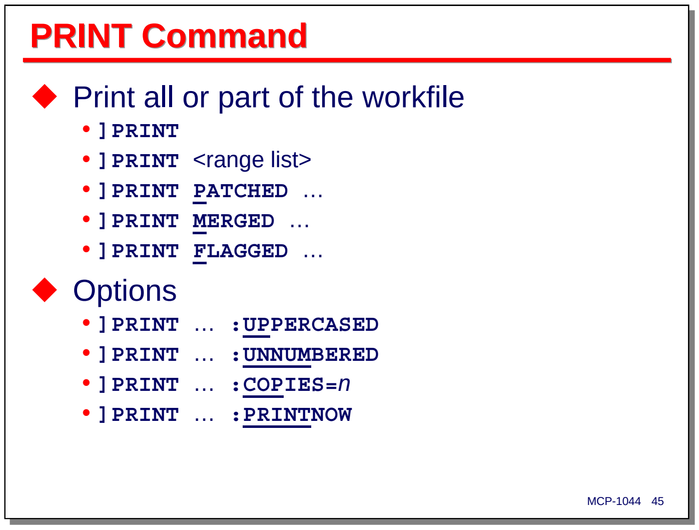
LIST Command

- List all or part of an external file
  - `]LIST <file name>`
  - `]LIST <file name> <range list> <column range>`
  - `]LIST <file name> … :UNNUMBERED`
- List the last-specified external file
  - `]LIST = …`
- Examples
  - `]LIST MY/FILE 1000-1999, 6000-8999`
  - `]L DATA/RPT/SPECS @60-125`
  - `]L = 22000-23500 @33-99`
  - `]L THAT/MESS : U`

MCP-1044 44

The **]LIST** command will simply list an external file, or a range of records from an external file, within your editing session.

- Editor will replace the editing page with the first page of records from the external file.
- Press **SPCFY** on the top line to scroll to the next page of the file being listed. You can only scroll forward. There is no way to scroll backward.
- Press **XMIT** to terminate the list and return to the editing page.
- Specifying an "**=**" instead of a file name will list the last external file used in any command, including **]INSERT**, **]MERGE**, **]PRINT**, etc.
- The **]LIST** command has additional options to list a range of columns (as shown on the slide), suppress sequence numbers, fold long lines, and a few other things. See the help page for details.



PRINT Command

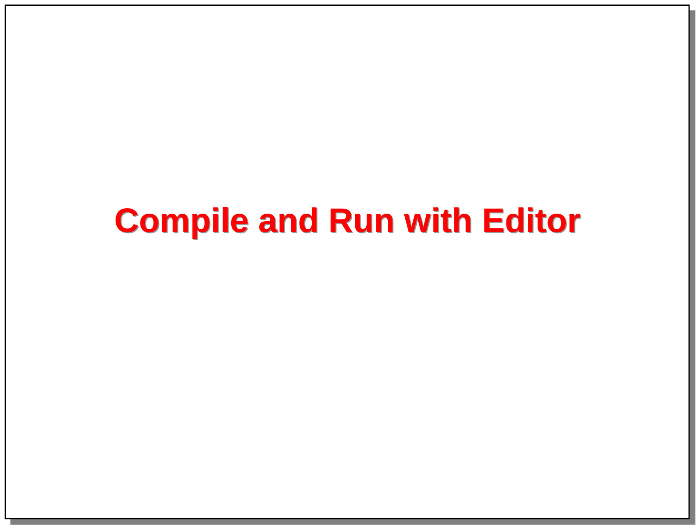
- Print all or part of the workfile
  - `]PRINT`
  - `]PRINT` <range list>
  - `]PRINT PATCHED …`
  - `]PRINT MERGED …`
  - `]PRINT FLAGGED …`
- Options
  - `]PRINT … :UPPERCASED`
  - `]PRINT … :UNNUMBERED`
  - `]PRINT … :COPIES=n`
  - `]PRINT … :PRINTNOW`

MCP-1044 45

The **`]PRINT`** command will print all or a portion of your workfile. It has a number of options:

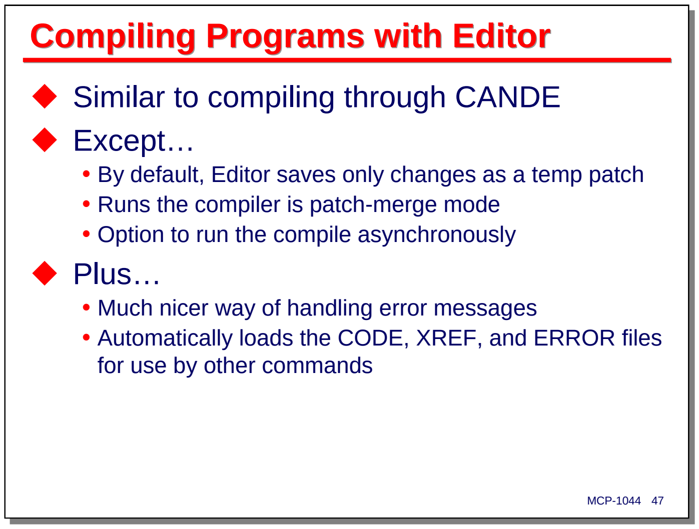
- **`]PRINT`** –by itself prints just the workfile. If you are working in patch mode, it prints only the patch records.
- **`]PRINT`** <range list> – prints the portion of the workfile specified by the sequence number range list.
- **`]PRINT PATCHED`** – prints the changes in the workfile, comparing them to the base source. The format of this output is similar to the **`$.COMPARE`** option in **`SYSTEM/PATCH`**.
- **`]PRINT MERGED`** – prints the merged result of the workfile with its base source. Changed lines are flagged with a "**`*`**" in their left column.
- **`]PRINT FLAGGED`** – prints the workfile with any deleted or modified lines in their original form.

There are options with this command to print in all upper case (seldom needed anymore since the advent of laser printers), print without sequence numbers, and to print multiple copies. The **`:PRINTNOW`** option will cause Editor to generate the print file with **`PRINTDISPOSITION=CLOSE`**, so that it will print immediately, without having to quit the Editor and split your CANDE session.



Compile and Run with Editor

In this next section, we will discuss how to use the Editor to compile and run program. We will also discuss some of Editor's advanced features for dealing with codefiles, which is one of the things Editor does that sets it apart from other MCP-based development tools.



Compiling Programs with Editor

- Similar to compiling through CANDE
- Except…
  - By default, Editor saves only changes as a temp patch
  - Runs the compiler is patch-merge mode
  - Option to run the compile asynchronously
- Plus…
  - Much nicer way of handling error messages
  - Automatically loads the CODE, XREF, and ERROR files for use by other commands

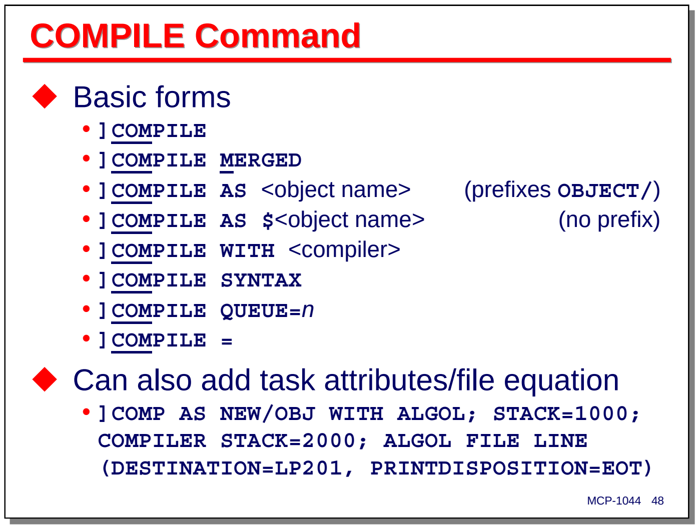
MCP-1044 47

You can compile programs through the Editor in much the same way you can through CANDE. The big difference is that, by default, Editor does not update your workfile to create a complete source file to pass to the compiler. Instead, it creates a patch file from your changes in the workfile, adds a **`$MERGE`** record at the front of the patch file, and file-equates this temporary patch file to the compiler's **`CARD`** file. The base source is file-equated to the compiler's **`SOURCE`** or **`TAPE`** file, as appropriate.

Editor initiates the compiler as a dependent task. Normally that would mean that you need to wait until the compile is finished before you could resume using the Editor. If you include the **`CLASS`** or **`QUEUE`** option on the compile, however, the Editor initiates the compile as an asynchronous job, and you can immediately continue to edit while the compile is running.

One of the nice things about the way Editor does compiling is the way it handles syntax errors reported by the compiler. These are written to the compiler's **`ERROR`** file, which the Editor captures after compilation. It will then show you the errors in the context of the line on which the error was reported. We will discuss this a little more in a few slides.

In addition to the **`ERROR`** file, the compiler may also generate a **`CODE`** file and a set of XREF files. The Editor also captures these after the compilation completes and automatically loads them into your session. See the **`]LOAD`** command later for more information on how these can be used.



COMPILE Command

- Basic forms
  - `]COMPILE`
  - `]COMPILE MERGED`
  - `]COMPILE AS` <object name> (prefixes `OBJECT/`)
  - `]COMPILE AS $`<object name> (no prefix)
  - `]COMPILE WITH` <compiler>
  - `]COMPILE SYNTAX`
  - `]COMPILE QUEUE=`*n*
  - `]COMPILE =`
- Can also add task attributes/file equation
  - `]COMP AS NEW/OBJ WITH ALGOL; STACK=1000; COMPILER STACK=2000; ALGOL FILE LINE (DESTINATION=LP201, PRINTDISPOSITION=EOT)`

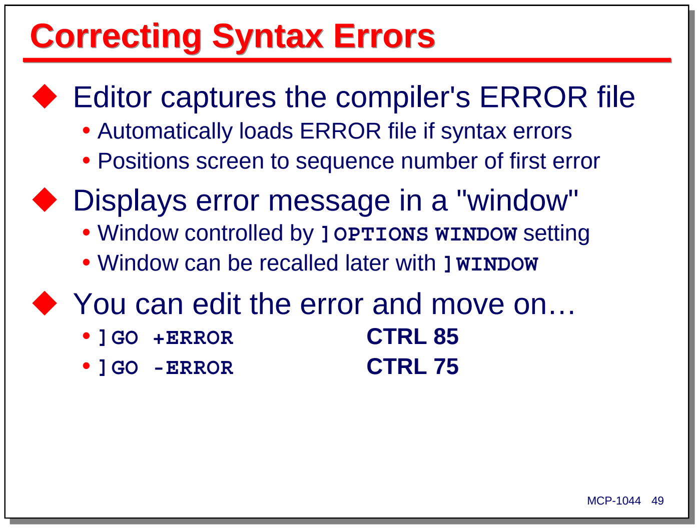
MCP-1044 48

The Editor's **]COMPILE** command has a number of options.

- **]COMPILE** – by itself, this will simply compile the current workfile as described on the prior slide
- **]COMPILE MERGED** – instead of passing separate patch and base source files to the compiler, this form of the command creates a fully-merged source file first, and then passes that file to the compiler's **CARD** file.
- **]COMPILE AS** <object name> – similar to the **AS** option in CANDE, this form generates a codefile with the specified file name. The name you specify will be prefixed with **OBJECT/**, as with CANDE.
- **]COMPILE AS $**<object name> – this is the same as the **AS** option above, but suppresses the **OBJECT/** prefix on the codefile name.
- **]COMPILE WITH** <compiler> – this form is similar to the **WITH** option in CANDE and WFL. It specifies the compiler to be used.
- **]COMPILE SYNTAX** – as you might expect, this option instructs the compiler to check the syntax of the source code, but not to create an object file, even if there are no errors.
- **]COMPILE QUEUE=**n *or* **CLASS=**n – this form initiates the compiler as an asynchronous job, allowing you to continue editing while the compile is taking place.
- **]COMPILE =** – this form indicates that the Editor should use the same **AS**, **WITH**, **CLASS**, and **QUEUE** options as were used for the last **]COMPILE** command.

All of these options can be mixed in any combination in a single **]COMPILE** command.

In addition, the options above can be followed by a semicolon (**;**) and a list of task attribute modifiers for both the codefile being generated and the compiler itself. As with CANDE and WFL, attribute modifiers prefixed by a compiler name or the keyword **COMPILER** will apply to the compiler; all others will apply to the program being compiled.



Correcting Syntax Errors

- Editor captures the compiler's ERROR file
  - Automatically loads ERROR file if syntax errors
  - Positions screen to sequence number of first error
- Displays error message in a "window"
  - Window controlled by **]OPTIONS WINDOW** setting
  - Window can be recalled later with **]WINDOW**
- You can edit the error and move on…
  - **]GO +ERROR** **CTRL 85**
  - **]GO -ERROR** **CTRL 75**

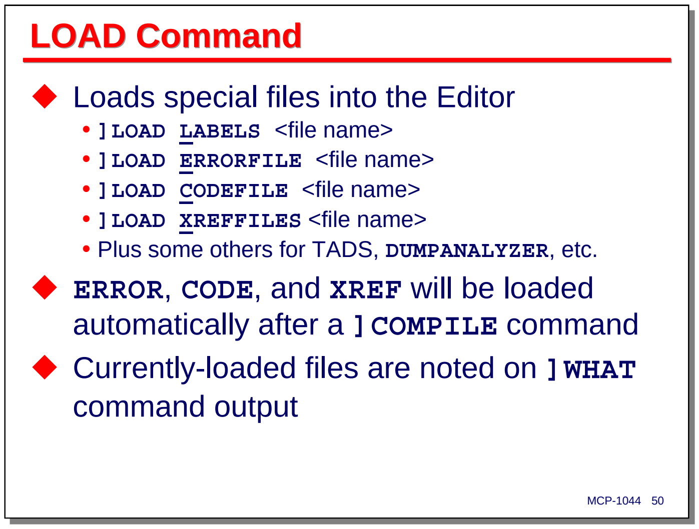
MCP-1044 49

As mentioned earlier, the Editor will capture the compiler's **ERROR** file at the end of a compilation that resulted in syntax errors. The Editor loads the contents of this file into your editing session, and positions the editing page to the first sequence number that has an error message in the **ERROR** file. It then displays the error text below that line in a pseudo "window." The presence of this pseudo-window is controlled by the **]OPTIONS WINDOW** command.

The window disappears the next time you transmit anything to the Editor. You can recall the window with the **]WINDOW** command.

Once an **ERROR** file is loaded into your editing session, you can use it to navigate directly from error to error using **]+ERROR** (**CTRL** 85 **XMIT**) and **]-ERROR** (**CTRL** 75 **XMIT**).

Note that the Editor does not understand the use of **COPY** or **INCLUDE** files within a program, so if syntax errors occur within those, Editor will try to position to a line in the workfile using a sequence number from the **COPY** or **INCLUDE** file. This can produce some seemingly-bizarre results, so be aware that source include files can generate confusing behavior.



LOAD Command

- Loads special files into the Editor
  - **]LOAD LABELS** <file name>
  - **]LOAD ERRORFILE** <file name>
  - **]LOAD CODEFILE** <file name>
  - **]LOAD XREFFILES** <file name>
  - Plus some others for TADS, **DUMPANALYZER**, etc.
- **ERROR**, **CODE**, and **XREF** will be loaded automatically after a **]COMPILE** command
- Currently-loaded files are noted on **]WHAT** command output

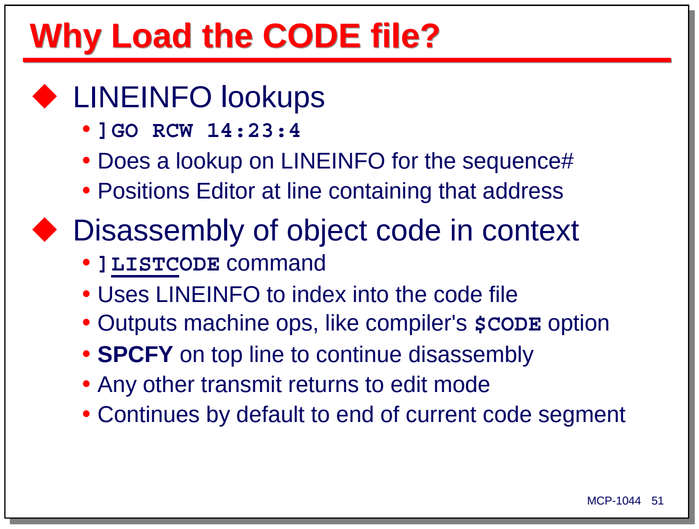
MCP-1044 50

Now that we have talked about the **]COMPILE** command, we can talk about the **]LOAD** command. This command "loads" (actually, "associates" would be a better term) a variety of external files with your editing session.

- **]LOAD LABELS** – will load a file of line labels that was previously saved with the **]SAVE LABELS** command.
- **]LOAD ERRORFILE** – will load a compiler's **ERROR** file.
- **]LOAD CODEFILE** – will load a codefile. The codefile should be one generated from the source file you are editing, and it should have been compiled with the **$LINEINFO** option set. Compiling with **$TADS** sets **$LINEINFO** implicitly.
- **]LOAD XREFFILES** loads the cross-reference files generated by the **$XREFFILES** option. Compilers generate two such files, **XREFFILES/**<codefile name>**/DECS** and **XREFFILES/**<codefile name>**/REFS**. You can name either one in the command. Editor will determine the name of the other one and load it automatically.
- There are a number of other files that can be associated with your editing session, relating to TADS, the Dump Analyzer interface, the data dictionary facility, and the Editor's macro facility. These are all important, but beyond the scope of this presentation.

As mentioned with the **]COMPILE** command, the Editor will automatically load the **ERROR**, **CODE**, and XREF files, as appropriate, after a compilation. If you are not compiling with the Editor, however, or are working with a file that was compiled in some other session, you can manually load these files at any time. This is particularly useful for codefiles and XREF files. We will discuss what you can do with a loaded codefile starting on the next slide. Loading XREF files into the Editor is a powerful capability, but is beyond the scope of this presentation.

Note that all currently-loaded files are mentioned on the status page generated by the **]WHAT** command.



Why Load the CODE file?

- LINEINFO lookups
  - `]GO RCW 14:23:4`
  - Does a lookup on LINEINFO for the sequence#
  - Positions Editor at line containing that address
- Disassembly of object code in context
  - `]LISTCODE` command
  - Uses LINEINFO to index into the code file
  - Outputs machine ops, like compiler's `$CODE` option
  - **SPCFY** on top line to continue disassembly
  - Any other transmit returns to edit mode
  - Continues by default to end of current code segment

MCP-1044 51

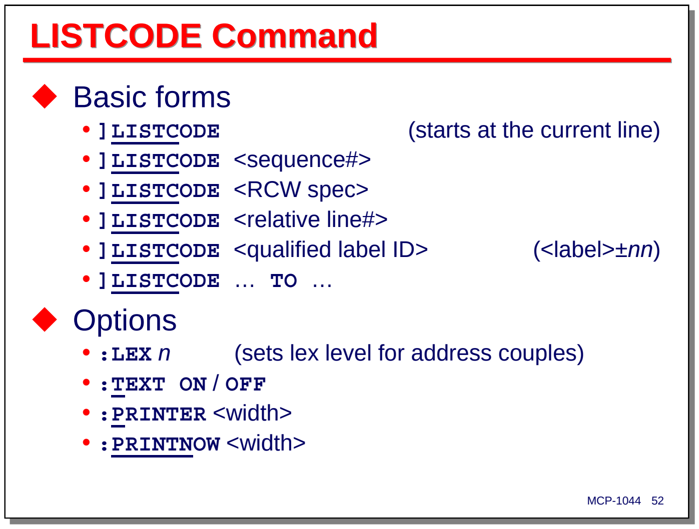
Why load (associate) a code with into your editing session? There are at least two really useful things you can do with it, even without considering the features of TADS or the Dump Analyzer interface.

The first of these is that you can navigate to a line based on its code segment address. The `]GO RCW` command will take a three-part code address (*segment-number* **:** *word* **:** *syllable*), look up the corresponding sequence number in the codefile's **LINEINFO** tables, and navigate to that line in the file. This is particularly useful when reading programdumps where the line number is not shown. One prime example is decoding RCWs in the stack for COBOL **PERFORM**s.

The really, really cool thing you can do with a codefile in Editor, though, is disassemble portions of the object code using the `]LISTCODE` command. Not only will Editor translate the instruction syllables to their mnemonic form, it will merge them with the source, producing a display that looks similar to the output of a compiler's **$CODE** option.

The output of `]LISTCODE` works somewhat like the `]LIST` command. You can scroll forward by pressing **SPCFY** on the top line of the page. Any other transmission terminates the command. There is unfortunately no way to scroll backward, but if you have a very long sequence to disassemble, you can output it to a printer file and study it off-line.

The disassembly will continue until you stop it, or until the Editor reaches the end of the current code segment. To disassemble across code segments, you must enter separate `]LISTCODE` commands for each segment.



LISTCODE Command

- Basic forms
  - **]LISTCODE** (starts at the current line)
  - **]LISTCODE** <sequence#>
  - **]LISTCODE** <RCW spec>
  - **]LISTCODE** <relative line#>
  - **]LISTCODE** <qualified label ID> (<label>±*nn*)
  - **]LISTCODE** … **TO** …
- Options
  - **:LEX** *n* (sets lex level for address couples)
  - **:TEXT ON** / **OFF**
  - **:PRINTER** <width>
  - **:PRINTNOW** <width>

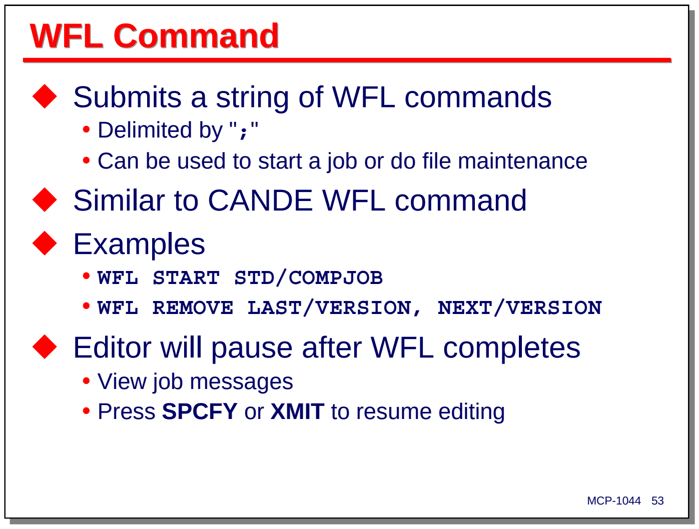
MCP-1044 52

The **]LISTCODE** command has several forms:

- **]LISTCODE** – by itself, this command will start disassembling at the currently-offered line of the workfile.
- **]LISTCODE** <sequence number> – will start disassembling at the start of the specified line.
- **]LISTCODE** <RCW spec> – works like a **]GO RCW** command – it will look up the source line based on the code segment address, and start disassembling at that line.
- **]LISTCODE** <relative line number> – will start disassembling at the line the specified number of lines above or below the currently-offered line.
- **]LISTCODE** <qualified label ID> – will start disassembling at the line specified by the <label ID>, optionally plus or minus some number of lines from that label.
- **]LISTCODE** … **TO** … – allows you to specify a range over which the code will be disassembled. Any of the forms above can be used for both the starting and ending positions of the range. Without the **TO** clause, disassembly continues page by page until you either terminate it or the Editor reaches the end of the code segment.

There are several options for the **]LISTCODE** command, which are delimited from it by a colon (**:**).

- **:LEX** *n* – specifies the lexicographical (nesting) level of the code. Some instructions are interpreted slightly differently at higher lex levels. By default, the disassembly will work properly for levels 2 and 3, which is where most programs have the majority of their code. You may need to specify this option to disassemble code at lex levels 4 and above.
- **:TEXT OFF** – will disassemble code without showing the source lines.
- **:PRINTER** <width> – will send the output of the disassembly to a printer file instead of the screen. The optional <width> specifies the line width to be used for the printer file.
- **:PRINTNOW** <width> – works the same as **:PRINTER**, but the Editor configures the printer file with **DISPOSITION=CLOSE** so that the output will be scheduled for printing immediately.



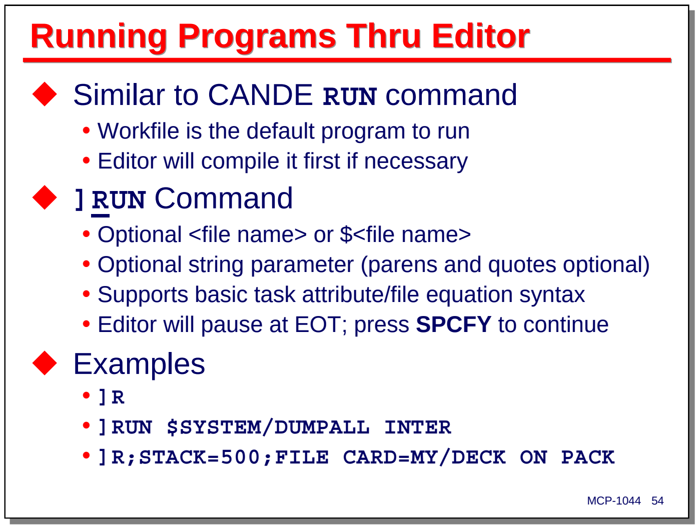
WFL Command

- Submits a string of WFL commands
  - Delimited by "`;`"
  - Can be used to start a job or do file maintenance
- Similar to CANDE WFL command
- Examples
  - `WFL START STD/COMPJOB`
  - `WFL REMOVE LAST/VERSION, NEXT/VERSION`
- Editor will pause after WFL completes
  - View job messages
  - Press **SPCFY** or **XMIT** to resume editing

MCP-1044 53

The Editor has a **`]WFL`** command that works similarly to CANDE's **`WFL`** command. It allows you to enter a string of WFL statements, delimited by semicolons. You can use this command, for example, to initiate Library/Maintenance or to start an external job.

Most uses of the **`]WFL`** command will generate messages that come back to your terminal. The Editor will pause while the WFL statements execute. You must press **SPCFY** or **XMIT** to return to the editing page.



Running Programs Thru Editor

- Similar to CANDE **RUN** command
  - Workfile is the default program to run
  - Editor will compile it first if necessary
- **]RUN** Command
  - Optional <file name> or $<file name>
  - Optional string parameter (parens and quotes optional)
  - Supports basic task attribute/file equation syntax
  - Editor will pause at EOT; press **SPCFY** to continue
- Examples
  - `]R`
  - `]RUN $SYSTEM/DUMPALL INTER`
  - `]R;STACK=500;FILE CARD=MY/DECK ON PACK`

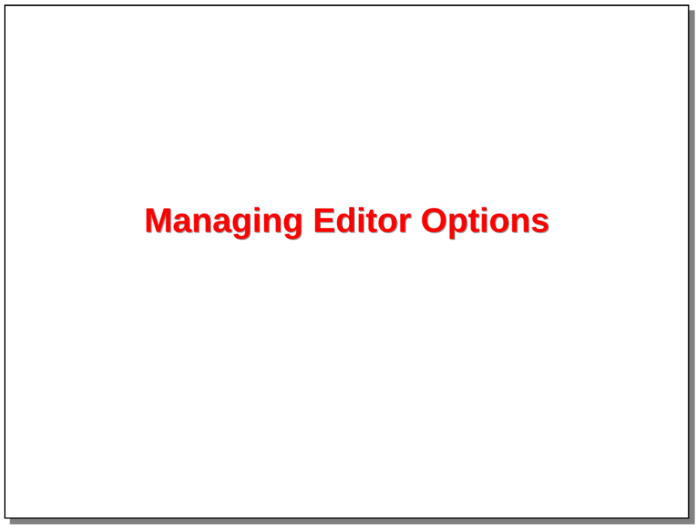
MCP-1044 54

If you can compile using the Editor, can you run programs, too? Yes, using the **]RUN** command, which works in a fashion very similar to CANDE.

By default, the Editor will run the current workfile, compiling it first if necessary. The same conventions are used as for CANDE in naming the codefile – by default, Editor prefixes whatever name you specify with **OBJECT/**, unless you prefix the name with a "**$**".

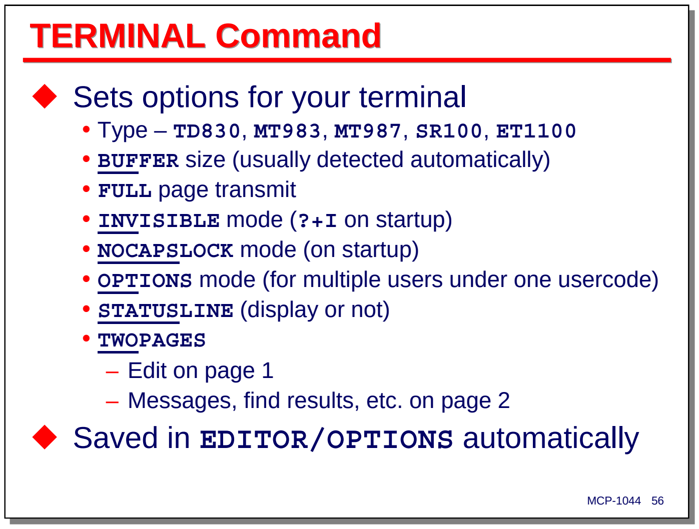
Editor will support passing a single string parameter to the program, but this is optional. The parentheses and quotes around the parameter are also optional.

You can append standard task attribute modifiers to the **]RUN** command, as you would for CANDE.



Managing Editor Options

The Editor has a large number of options that control how it operates. These options are maintained using two commands, **`]TERMINAL`** and **`]OPTIONS`**, which we will discuss next.



TERMINAL Command

- Sets options for your terminal
  - Type – **TD830**, **MT983**, **MT987**, **SR100**, **ET1100**
  - **BUFFER** size (usually detected automatically)
  - **FULL** page transmit
  - **INVISIBLE** mode (**?+I** on startup)
  - **NOCAPSLOCK** mode (on startup)
  - **OPTIONS** mode (for multiple users under one usercode)
  - **STATUSLINE** (display or not)
  - **TWOPAGES**
    - Edit on page 1
    - Messages, find results, etc. on page 2
- Saved in **EDITOR/OPTIONS** automatically

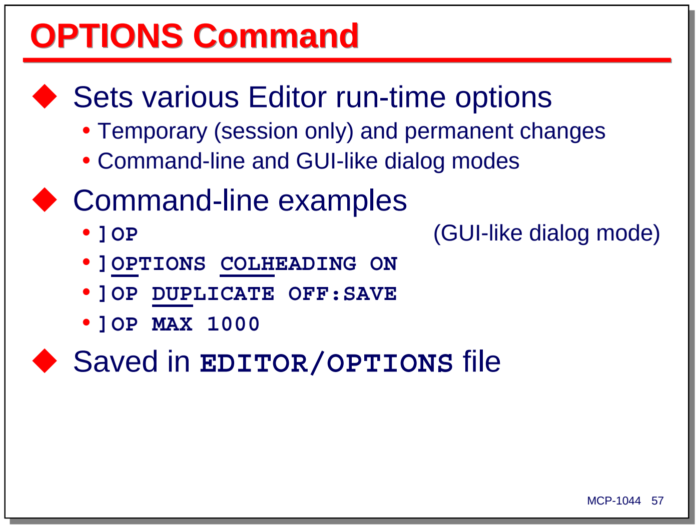
MCP-1044 56

There were many models of Burroughs-style terminals, not to mention the variety of terminal emulators that are currently available. Each of these has varying sets of features. In addition, there are some personal preferences on terminal behavior that the Editor will accommodate. These are controlled using the **]TERMINAL** command, which has the following options:

- Type – you can specify that the terminal is a **TD830**, **MT983**, **MT987**, **SR100**, or **ET1100**. With most modern emulators, either TD830 or ET1100 seems to work fine. I tend to use ET1100.
- **BUFFER** – You can specify the maximum buffer (message) size the Editor will send to the terminal. Larger is generally better (if the terminal can handle it).
- **FULL** – Setting this option will cause the Editor to configure the terminal to send the full form during transmit. If this option is already configured in the terminal, it has no effect.
- **INVISIBLE** – Setting this option will cause the Editor to use "invisible" mode (**?+I**) on startup, instead of simply suppressing scrolling.
- **NOCAPSLOCK** – Setting this option will suppress the Editor's default behavior of setting the internal caps lock when editing source files for languages that are normally written in all upper case.
- **OPTIONS** – stores certain values for the **]OPTIONS** command with the terminal configuration command. This is useful when you have multiple users at different terminals using the same usercode.
- **STATUSLINE** – controls whether the Editor will use the terminal's status line (the default) or the last line of the text area on the screen for status information. The primary use of this is with terminals and emulators that do not support the status line feature.
- **TWOPAGES** – controls how messages are displayed on the terminal. When reset (the default), messages are displayed on the editing page. When set, editing occurs on page 1 and messages are automatically displayed on page 2.

You can use the **]TERMINAL** command in two ways. You can use it as a standard command to change a single option, e.g., **]TERM STATUSLINE ON**. Alternatively, you can just enter the **]TERMINAL** command by itself. The Editor will display a page showing the current terminal options. You can toggle the Boolean options simply by positioning the cursor to them and pressing **SPCFY**. Pressing **SPCFY** on non-Boolean options displays an input box or a pick list from which you can set or choose the desired value. Simply press **XMIT** to return to the editing page

The terminal options are saved automatically in the **EDITOR/OPTIONS** file under your usercode each time they are modified.



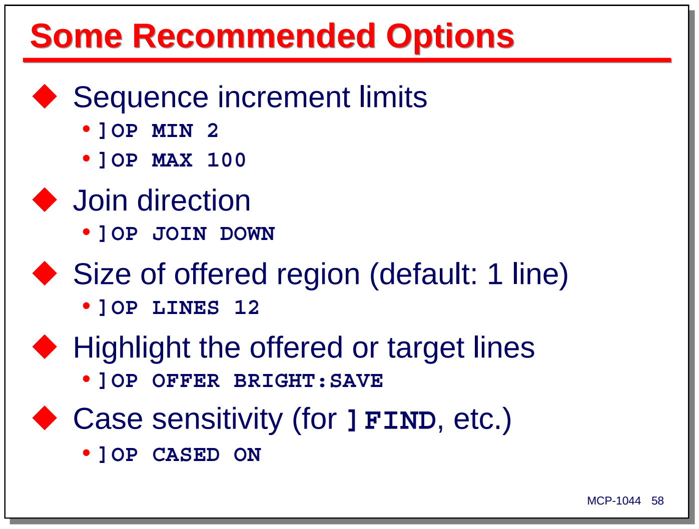
OPTIONS Command

- Sets various Editor run-time options
  - Temporary (session only) and permanent changes
  - Command-line and GUI-like dialog modes
- Command-line examples
  - `]OP` (GUI-like dialog mode)
  - `]OPTIONS COLHEADING ON`
  - `]OP DUPLICATE OFF:SAVE`
  - `]OP MAX 1000`
- Saved in **`EDITOR/OPTIONS`** file

MCP-1044 57

The **`]OPTIONS`** command displays and maintains operating parameters for the Editor program. These are also saved in the **`EDITOR/OPTIONS`** file under your usercode. With these options, however, you can choose whether to change them temporarily for your editing session, or to make them permanent by saving them in that file.

Like the **`]TERMINAL`** command, you can use **`]OPTIONS`** in an imperative manner to change a single option, e.g., **`]OP DUP ON`**. You can also transmit just the **`]OPTIONS`** command by itself. The Editor will respond with a page showing the status of all of the options. You can position the cursor over an option and use the **SPCFY** key to change it. Boolean options will simply toggle each time they are selected. For other types of options, the Editor will display a small form or pick list where you can enter or select the value of the option.



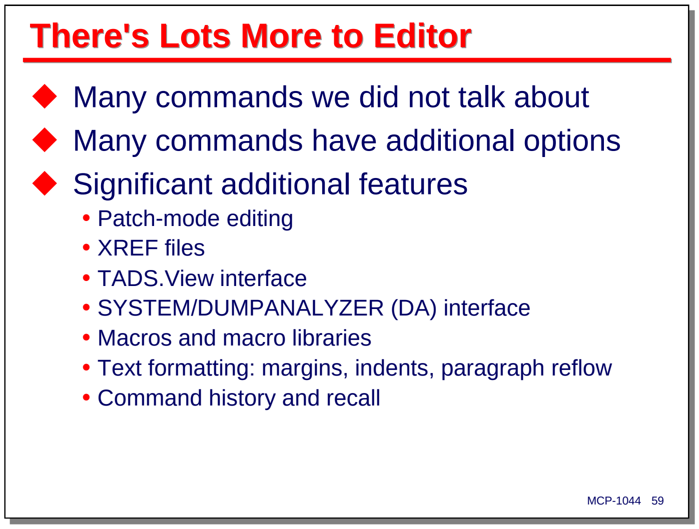
Some Recommended Options

- Sequence increment limits
  - `]OP MIN 2`
  - `]OP MAX 100`
- Join direction
  - `]OP JOIN DOWN`
- Size of offered region (default: 1 line)
  - `]OP LINES 12`
- Highlight the offered or target lines
  - `]OP OFFER BRIGHT:SAVE`
- Case sensitivity (for **]FIND**, etc.)
  - `]OP CASED ON`

MCP-1044 58

There are too many options to attempt to discuss in this presentation, but here are a few that I find useful.

- **]OP MIN** *nn*, **]OP MAX** *nn* – sets the minimum and maximum sequence number increments the **]NUMBER** command will use. When **]NUMBER** is computing the increment from the gap between sequence numbers and the number of lines to be numbered, its calculation will be constrained to be between these limits.
- **]OP JOIN DOWN** – sets the default direction for a **]JOIN** command. If you prefer to join to the prior line, specify **]OP JOIN UP**.
- **]OP LINES** *nn* – changes the number of lines in the offered area. By default, the Editor only offers one line at a time. You can change that with this option. Offering multiple lines is often useful when you have a large amount of text to enter in one region of the file.
- **]OP OFFER BRIGHT:SAVE** – configures the Editor to display the offered line using the terminals bright-video mode. In most modern emulators, you can chose a color for bright video, and this allows the offered line to stand out from the test of the text on the screen.
- **]OP CASED ON** – controls whether **]FIND** and **]REPLACE** will do case-sensitive or case-insensitive searches. **CASED ON** implies case-sensitive matching. **CASED OFF** will do case-insensitive matching



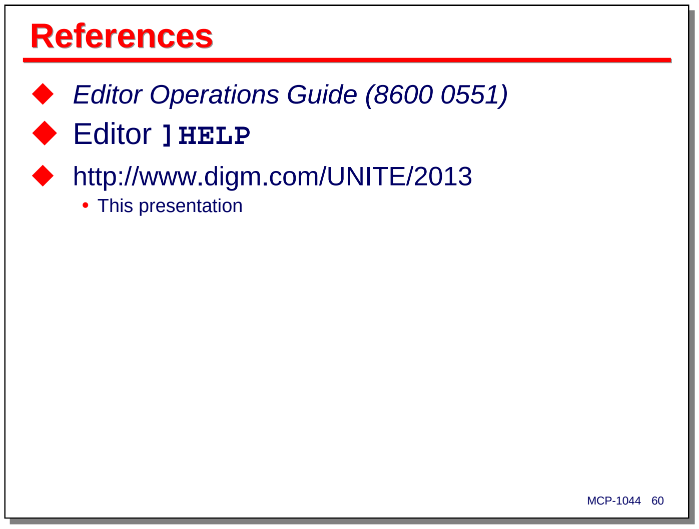
There's Lots More to Editor

- Many commands we did not talk about
- Many commands have additional options
- Significant additional features
  - Patch-mode editing
  - XREF files
  - TADS.View interface
  - SYSTEM/DUMPANALYZER (DA) interface
  - Macros and macro libraries
  - Text formatting: margins, indents, paragraph reflow
  - Command history and recall

MCP-1044 59

There's lots more to the Editor that we haven't (and don't have time to) talk about. Many of the commands have additional features I have omitted for simplicity or due to time constraints. There are also some significant features that I have mentioned, but not discussed in any detail, as shown on the slide.

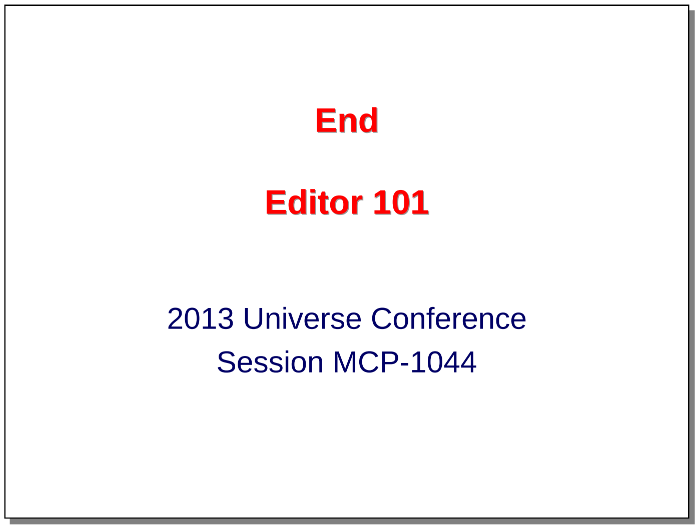
This presentation should give you what you need to get started using the Editor. The best way to learn the Editor is to use it. When you have mastered this material, you can learn more by looking at the on-line help file or the *Editor Operations Guide*.



References

- *Editor Operations Guide (8600 0551)*
- Editor `]HELP`
- http://www.digm.com/UNITE/2013
  - This presentation

MCP-1044 60



End

Editor 101

2013 Universe Conference
Session MCP-1044