free42 Programming Tools

Mitch Richling

2021-03-19

Author: Mitch Richling
Updated: 2021-05-18 23:30:23

Contents

1 Metadata

The home for this HTML file is: `https://richmit.github.io/hp42/hp42s-meta.html`
A PDF version of this file may be found here: `https://richmit.github.io/hp42/hp42s-meta.pdf`
Files related to this document may be found on github: `https://github.com/richmit/hp42`
Directory contents:

`src`	-	The org-mode file that generated this HTML document
`src_42s`	-	Ready to convert source listings for 42s code in this document
`docs`	-	This html document and associated PDF
`bin`	-	Importable RAW program files

2 Introduction

This org-mode document contains various tools (mostly for Emacs) that help me write programs for the 42s:

- Elisp function to generate 42s menu programs
- A tool to insert free42 utf-8 characters given a list of character numbers
- Elisp function to generate custom menus (menus of built-in functions and global labels)
- yasnippet templates to help type code
- ElISP Code to translate my code listings into code for free42 & MD42
- 42s notes
  - free42 utf-8 characters with character numbers

3 Generating 42s menu code

3.1 Generic menu generator

This code will read an org-mode table describing a menu, and generate 42s code to implement the menu. The only limitation on menu depth and size are the number of available two digit labels.

Line	Menu alpha strings	Menu targets	Some Data	Some More Data
1	foo:bar:foobar		marry	red
2	foo:bar:LBL 77	LBL 87	had a	blue
3	foo:bar:LBL 78	fooboo	little	yellow
4	foo:bar:LBL 78		lamb	green

One column of the table is used to define the menu alpha strings. The table above provides an example. The string `foo:bar:foobar` (table line 1) defines a top level button named `foo` that leads to a menu containing another menu named `bar` which contains an action key named `foobar`. The final component, `foobar` in this example, is used for the alpha string for the menu key. It is used as-is with one exception. That exception occurs when the final component looks like `"LBL NN"` where `NN` is a two digit number (table line 2). In this case the local label will be `XEQ`'ed just before the call to `KEY`. This allows the key's alpha string to be dynamically generated by a subroutine at run time. If this subroutine returns `RETNO` then the menu key's `KEY` command will be skipped. In this way one can dynamically decide if a menu key should be active or not. If the key string is `""` (empty) or `"□□□□"`, then a blank key will be placed in the menu.

Now that we know how the menu key strings are constructed, what about the `GTO/XEQ` target for the generated `KEY` commands? The following rules are applied with the first one matching being used:

`gen-target-label`	`target-column`	Menu Key	Menu Target	The `XEQ/GTO` Target
`nil`	`nil`	`"LBL NN"`	N/A	Auto-generated
`nil`	`nil`	N/A	N/A	Menu key string
`nil`	non-`nil`	`"LBL NN"`	`""` (empty)	Auto-generated
`nil`	non-`nil`	N/A	`""` (empty)	Menu key string
`nil`	non-`nil`	N/A	`"LBL NN"`	`NN`
`nil`	non-`nil`	N/A	N/A	Menu target
non-`nil`	N/A	N/A	N/A	See next table

When `gen-target-label` is non-`nil`, the return value of the function determines the target. If the return is `nil`, then the label is auto-generated. Otherwise the returned string is used. Note the returned string must contain embedded quotes if it is a global label target.

In general this might be summarized as follows: When `gen-target-label` is non-`nil`, the target is determined by the `gen-target-label` function. When `gen-target-label` is `nil`, then the menu target column is used unless it is empty, and then the menu key is used.

When a local label is generated, the subroutine for that label will also be generated. The content of that subroutine can be provided by calling the a user provided function via the `gen-target-code` argument. This allows the entire program to be generated from the contents of the table my constants & units programs are good examples. Note that if all the lables are generated, then the resulting program is ended with an `END` instruction.

Note the Unicode point 166 (the "¦" character) is automatically converted to the pipe character ("|"). This lets you include the 42s pipe symbol in org-mode tables.

`MJR-generate-42-menu-code` arguments:

`top-lab` The global label to use for the generated program

`numeric-lbl-start` Beginning for a range of local numeric labels that will be used for the program

`tbl` The org-mode table with the menu description

`menu-alpha-column` The column with the menu alpha strings

`menu-exit-behavior` What to do when `[EXIT]` is pressed.

- `'exit`: Exit the application
- `'up`: Return to parent menu or exit if no parent

`after-leaf-action` What do do when a action menu is used (a leaf node in the menu)

- `'stay`: Keep the menu active
- `'exit`: Exit the menu

`include-end` Include final `END` statement

- `'yes`: Create final `END` statement
- `'exit`: Do not create final `END` statement
- `'auto`: Create final `END` statement if all menu target labels were generated

`target-column` nil means no target column.

`gen-target-label` A function that generates the label for the `KEY` commands `GTO=/=XEQ`. Return `nil` for autogen.

- Arguments: autoish-target, list of row data
- autoish-target is essentially the target that would have been used if `gen-target-label` were `nil`. Local labels look like `"LBL NN"` and global ones look like `"FOO"`.

`gen-target-code` A function that generates the code for the action. It gets a list that contains the table row for the menu item.

- Arguments: autoish-target, menu target label, list of row data

```
(defun MJR-generate-42-menu-code (top-lab
                                  numeric-lbl-start
                                  tbl
                                  menu-alpha-column
                                  target-column
                                  after-leaf-action
                                  menu-exit-behavior
                                  include-end
                                  gen-target-label
                                  gen-target-code)
  (let* ((no-local 't)
         (min-free-lab (+ numeric-lbl-start 2))
         (m-code "")
         (x-code ""))
    (cl-labels ((add-m-code (a) (setq m-code (concat m-code a "\n")))
                (add-x-code (a) (setq x-code (concat x-code a "\n")))
                (prc-mnu (menu)
                         (if (or (not (listp menu))
                                 (null (cdr menu)))
                             menu
                             (mapcar #'prc-mnu
                                     (append (list (car menu))
                                             (reverse
```

```
                              (cl-reduce
                               (lambda (result cur-elt)
                                 (let ((last-elt (cl-first result)))
                                   (if (and (listp last-elt)
                                            (cdr last-elt)
                                            (cdr cur-elt)
                                            (string-equal (cl-first last-elt) (cl-first cur-elt)))
                                       (progn (nconc (cl-first result) (list (cdr cur-elt)))
                                              result)
                                       (if (cdr cur-elt)
                                           (append (list (list (cl-first cur-elt) (cdr cur-elt)))
                                                   result)
                                           (append (list (cl-first cur-elt))  result)))))
                               (cdr menu)
                               :initial-value ()))))))
(gen-mnu (parent-lbl lbl menu)
         (let* ((num-menu-keys (1- (length menu)))
                (num-menu-page (ceiling (/ num-menu-keys 6.0)))
                (page-labs     (cl-loop repeat num-menu-page
                                        for i = lbl then min-free-lab
                                        collect i
                                        when (not (= i lbl))
                                        do (cl-incf min-free-lab)))
                (rec-key-labs     nil)
                (rec-pag-labs     nil))
           (if (= parent-lbl numeric-lbl-start)
               (add-m-code (format "LBL \"%s\"" (cl-first menu))))
           (cl-loop for mkey-elt in (cdr menu)
                    for mkey-num from 0
                    for page-num = (truncate (/ mkey-num 6))
                    for page-key = (mod mkey-num 6)
                    for mkey-str = (if (vectorp mkey-elt) (aref mkey-elt 0) (cl-first mkey-elt))
                    for is-leaf  = (vectorp mkey-elt)
                    for auto-trg = (and is-leaf
                                        (or (if target-column
                                                (let ((tmp (nth target-column (aref mkey-elt 1))))
                                                  (if (not (string-empty-p tmp))
                                                      tmp)))
                                            (if (not (string-match-p "^LBL [0-9][0-9]$" mkey-str))
                                                mkey-str)
                                            ""))
                    for mkey-trg = (and is-leaf
                                        (if gen-target-label
                                            (funcall gen-target-label auto-trg (aref mkey-elt 1))
                                            (and (not (string-empty-p auto-trg))
                                                 (if (string-match-p "^LBL [0-9][0-9]$" auto-trg)
                                                     (substring auto-trg 4)
                                                     (format "\"%s\"" auto-trg)))))
                    ;;do (print (format "mkey-str: %s    auto-trg: %s    mkey-trg: %s" mkey-str auto-trg mkey-trg))
                    when (= page-key 0)
                    do (progn (add-m-code (format "LBL %02d            @@@@ Page %d of menu %s"
                                                  (nth page-num page-labs)
                                                  (1+ page-num)
                                                  (cl-first menu)))
                              (add-m-code "CLMENU"))
                    when (not (or (string-empty-p mkey-str) (string-equal mkey-str "□□□□")))
                    do (progn (if (string-match-p "^LBL [0-9][0-9]$" mkey-str)
```

```
                                    (progn (add-m-code (format "XEQ %s" (substring mkey-str 4)))
                                           (setq no-local nil))
                                    (add-m-code (format "\"%s\"" mkey-str)))
                                (if (or (not is-leaf) (not mkey-trg))
                                    (add-m-code (format "KEY %d %s %02d"
                                                        (1+ page-key)
                                                        (if is-leaf "XEQ" "GTO")
                                                        min-free-lab))
                                    (progn
                                      (add-m-code (format "KEY %d XEQ %s" (1+ page-key) mkey-trg))
                                      (setq no-local nil)))
                                (if (and (not mkey-trg) is-leaf)
                                    (progn (add-x-code (format
                                                        "LBL %02d    @@@@ Action for menu key %s"
                                                        min-free-lab
                                                        mkey-str))
                                           (if gen-target-code
                                               (add-x-code (funcall gen-target-code
                                                                    auto-trg
                                                                    mkey-trg
                                                                    (aref mkey-elt 1)))
                                               (add-x-code (format "@@@@ TODO: Code for %s!"
                                                                   mkey-str)))
                                           (add-x-code "RTN")))
                                (push min-free-lab rec-key-labs)
                                (if (not mkey-trg)
                                    (cl-incf min-free-lab))
                                (push (nth page-num page-labs) rec-pag-labs))
                            when (or (= page-key 5) (= mkey-num (1- num-menu-keys)))
                            do (progn (if (< 1 num-menu-page)
                                          (progn (add-m-code (format "KEY 7 GTO %02d"
                                                                     (nth (mod (1- page-num)
                                                                               num-menu-page)
                                                                          page-labs)))
                                                 (add-m-code (format "KEY 8 GTO %02d"
                                                                     (nth (mod (1+ page-num)
                                                                               num-menu-page)
                                                                          page-labs)))))
                                      (if (string-equal menu-exit-behavior "up")
                                          (add-m-code (format "KEY 9 GTO %02d" parent-lbl))
                                          (add-m-code (format "KEY 9 GTO %02d" 0)))
                                      (add-m-code "MENU")
                                      (add-m-code "STOP")
                                      (if (string-equal after-leaf-action "stay")
                                          (add-m-code (format "GTO %02d" (nth page-num page-labs)))
                                          (add-m-code (format "GTO %02d" 0)))))
                    (cl-loop for mkey-elt in (cdr menu)
                             for m-lab in (reverse rec-key-labs)
                             for p-lab in (reverse rec-pag-labs)
                             when (listp mkey-elt)
                             do (gen-mnu p-lab m-lab mkey-elt)))))
(gen-mnu numeric-lbl-start
         (1+ numeric-lbl-start)
         (prc-mnu (append (list top-lab) (cl-loop for row in tbl
                                                  for row-strs = (mapcar (lambda (x) (replace-regexp-in-string "¦" "|" (format "%s" x)  't 't))
                                                                         row)
                                                  for n from 0
```

```
                                                for menu-parts = (split-string
                                                                  (nth menu-alpha-column row-strs)
                                                                  ":")
                                                do (setf (car (last menu-parts))
                                                         (vector (car (last menu-parts)) row-strs))
                                                collect menu-parts))))
    (add-m-code (format "LBL %02d @@@@ Application Exit" numeric-lbl-start))
    (add-m-code "EXITALL")
    (add-m-code "RTN")
    (if (< 100 min-free-lab)
        (error "ERROR: Too many local labels: %d" min-free-lab)))
  (princ (format "%s (ref:%s)\n" (make-string 80 ?@)  top-lab))
  (princ (format "@@@@ DSC: Auto-generated menu program\n"))
  (princ m-code)
  (princ x-code)
  (princ (format "@@@@ Free labels start at: %d\n" min-free-lab))
  (if (or (string-equal include-end "yes") (and (string-equal include-end "auto") no-local))
      (princ "END"))))
```

3.2 For CUSTOM-type Menus

These functions are useful for CUSTOM-type menus – that is menus that call other programs or built in functions. I use this as a way to add hierarchy to the built in CUSTOM menu. Example:

Menu	Prog
LN	
log	LOG
MYPROG	

In the first line "`LN`" is the menu name and function called. In the second line "`log`" is the menu name, and "`LOG`" is the function called. In the third line "`MYPROG`" is the name of a program – the code below figures out if a thing is a built in function or a program and uses `XEQ` for programs. Note that I may have missed a built in function, so you may have to add one to the list. ;)

If the menu is of the for "LBL NN", then it will be `XEQ`'ed to get the menu label. If the prog is of the form "LBL NN", then it will be `XEQ`'ed directly. If any menu or prog is a label, then an `END` will not be generated at the end of the listing – this allows one to put the local subroutines later in the org-mode file and the whole thing will then be tangled together into one program.

```
(defun MJR-is-42-builtin (astring) (cl-position astring
                                    '("%" "%CH" "+" "+/-" "-" "1/X" "10↑X" "ABS" "ACOS" "ACOSH" "ADV" "AGRAPH" "AIP" "ALENG" "ALL" "ALLΣ" "AND" "AOFF" "AON"
                                      "ARCL" "AROT" "ASHF" "ASIN" "ASINH" "ASSIGN" "ASTO" "ATAN" "ATANH" "ATOX" "AVIEW" "BASE+" "BASE+/-" "BASE-" "BASE×"
                                      "BASE÷" "BEEP" "BEST" "BINM" "BIT?"  "BIT?"  "CF" "CLA" "CLD" "CLKEYS" "CLLCD" "CLMENU" "CLP" "CLRG" "CLST" "CLV"
                                      "CLX" "CLΣ" "COMB" "COMPLEX" "CORR" "COS" "COSH" "CPX?"  "CPXRES" "CROSS" "CUSTOM" "DECM" "DEG" "DELAY" "DELR"
                                      "DET" "DIM" "DIM?"  "DOT" "DSE" "EDIT" "EDITN" "END" "ENG" "ENTER" "EXITALL" "EXPF" "E↑X" "E↑X-1" "FC?"  "FC?C"
                                      "FCSTX" "FCSTY" "FIX" "FNRM" "FP" "FS?"  "FS?C" "FUNC" "GAMMA" "GETKEY" "GETM" "GRAD" "GROW" "GTO" "HEXM" "HMS+"
                                      "HMS-" "I+" "I-" "INDEX" "INPUT" "INSR" "INTEG" "INVRT" "IP" "ISG" "J+" "J-" "KEY" "KEYASN" "L4STK" "LASTX" "LBL"
                                      "LCLBL" "LINF" "LINΣ" "LN" "LN1+X" "LOG" "LOGF" "LSTO" "MAN" "MAT?"  "MEAN" "MENU" "MOD" "MVAR" "N!"  "NEWMAT"
                                      "NOP" "NORM" "NOT" "OCTM" "OFF" "OLD" "ON" "OR" "PERM" "PGMINT" "PGMSLV" "PIXEL" "POLAR" "POSA" "PRA" "PRLCD"
                                      "PROFF" "PROMPT" "PRON" "PRSTK" "PRUSR" "PRV" "PRX" "PRΣ" "PSE" "PUTM" "PWRF" "R<>R" "RAD" "RAN" "RCL" "RCL+"
                                      "RCL-" "RCLEL" "RCLIJ" "RCL×" "RCL÷" "RDX," "RDX." "REAL?"  "REALRES" "RECT" "RND" "RNRM" "ROTXY" "RSUM" "RTN"
                                      "RTNERR" "RTNYES" "RTNNO" "R↑" "R↓" "SCI" "SDEV" "SEED" "SF" "SIGN" "SIN" "SINH" "SIZE" "SLOPE" "SOLVE" "SQRT"
                                      "STO" "STO+" "STO-" "STOEL" "STOIJ" "STOP" "STO×" "STO÷" "STR?"  "SUM" "TAN" "TANH" "TONE" "TRACE" "TRANS" "UVEC"
                                      "VARMENU" "VIEW" "WMEAN" "WRAP" "WSIZE?"  "X<0?"  "X<>" "X<>Y" "X<Y?"  "X=0?"  "X=Y?"  "X>0?"  "X>Y?"  "XEQ" "XTOA"
                                      "X↑2" "X≠0?"  "X≠Y?"  "X≤0?"  "X≤Y?"  "X≥0?"  "X≥Y?"  "X≥Y?"  "YINT" "Y↑X" "[FIND]" "[MAX]" "[MIN]" "×" "÷" "Σ+"
                                      "Σ-" "ΣREG" "ΣREG?"  "←" "↑" "→" "→DEC" "→DEG" "→HMS" "→HR" "→OCT" "→POL" "→RAD" "→REC" "↓" "DDAYS" "DOW" "CLK12"
                                      "CLK24" "DMY" "MDY" "YMD" "DATE" "TIME" "DATE+" "PI" "WSIZE" "BSIGNED" "BWRAP" "XOR")
                                    :test #'string-equal))

(defun MJR-custom-gen-lab (atrg row) (if (not (MJR-is-42-builtin atrg))
                                         (if (string-match-p "^LBL [0-9][0-9]$" atrg)
                                             (substring auto-trg 4)
                                             (message "\"%s\"" atrg))))

(defun MJR-custom-gen-sub (atrg target row) (message "%s" atrg))
```

3.3 Always generate local lables

This function is handy when you want to always generate local labels.

```
(defun MJR-local-only-gen-lab (atrg row) nil)
```

4 Emacs Helper Stuff

4.1 Emacs function to insert charcters given a list of character codes

```
(defun MJR-ins42char (charn)
  "Insert free42 character at point."
  (interactive "sCharacter Number(s): ")
  (if charn
      (cl-loop for c in (split-string charn)
               do (insert (nth (eval (car (read-from-string c))) '("÷" "×" "√" "∫" "□" "Σ" "▶" "π" "ž" "≤" "[LF]" "≥" "≠" "↵" "↓" "→"
                                                                    ;; CHAR 30 & CHAR 4 are the same unicode.  How to put 30 in a string?
                                                                    "←" "μ" "£" "°" "Å" "Ñ" "Ä" "⌿" "⊑" "Æ" "..." "[ESC]" "Ö" "Ü" "□" "•"
                                                                    " " "!" "\"" "#" "$" "%" "&" "'" "(" ")" "*" "+" "," "-" "." "/"
                                                                    "0" "1" "2" "3" "4" "5" "6" "7" "8" "9" ":" ";" "<" "=" ">" "?"
                                                                    "@" "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O"
                                                                    "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z" "[" "\\" "]" "↑" "_"
                                                                    "`" "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
                                                                    "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z" "{" "|" "}" "~" "⊢"
                                                                    ":" "Y"))))))
```

4.2 Emacs Mode for 42s Code

This isn't really a proper mode for 42s code. Just a quick hack with `define-generic-mode` to get some syntax highlighting – which doesn't fully work as some of the characters in keywords are recognized as punctuation. Still it makes listings a little better. Someday I may take the time to write a real mode, but this works for now.

```
(define-generic-mode 'hp42s-mode
                     '("@@@@" "@@##" "@NM@")
                     '("%" "%CH" "+" "+/-" "-" "1/X" "10↑X" "ABS" "ACOS" "ACOSH" "ADV" "AGRAPH" "AIP" "ALENG" "ALL" "ALLΣ" "AND" "AOFF" "AON" "ARCL" "AROT" "ASHF"
                       "ASIN" "ASINH" "ASSIGN" "ASTO" "ATAN" "ATANH" "ATOX" "AVIEW" "BASE+" "BASE+/-" "BASE-" "BASE×" "BASE÷" "BEEP" "BEST" "BINM" "BIT" "BIT"
                       "CF" "CLA" "CLD" "CLKEYS" "CLLCD" "CLMENU" "CLP" "CLRG" "CLST" "CLV" "CLX" "CLΣ" "COMB" "COMPLEX" "CORR" "COS" "COSH" "CPX" "CPXRES"
                       "CROSS" "CUSTOM" "DECM" "DEG" "DELAY" "DELR" "DET" "DIM" "DIM" "DOT" "DSE" "EDIT" "EDITN" "END" "ENG" "ENTER" "EXITALL" "EXPF" "E↑X"
                       "E↑X-1" "FC" "FC?C" "FCSTX" "FCSTY" "FIX" "FNRM" "FP" "FS" "FS?C" "FUNC" "GAMMA" "GETKEY" "GETM" "GRAD" "GROW" "GTO" "HEXM" "HMS+"
                       "HMS-" "I+" "I-" "INDEX" "INPUT" "INSR" "INTEG" "INVRT" "IP" "ISG" "J+" "J-" "KEY" "KEYASN" "L4STK" "LASTX" "LBL" "LCLBL" "LINF" "LINΣ"
                       "LN" "LN1+X" "LOG" "LOGF" "LSTO" "MAN" "MAT" "MEAN" "MENU" "MOD" "MVAR" "N!"  "NEWMAT" "NOP" "NORM" "NOT" "OCTM" "OFF" "OLD" "ON" "OR"
                       "PERM" "PGMINT" "PGMSLV" "PIXEL" "POLAR" "POSA" "PRA" "PRLCD" "PROFF" "PROMPT" "PRON" "PRSTK" "PRUSR" "PRV" "PRX" "PRΣ" "PSE" "PUTM"
                       "PWRF" "R<>R" "RAD" "RAN" "RCL" "RCL+" "RCL-" "RCLEL" "RCLIJ" "RCL×" "RCL÷" "RDX," "RDX."  "REAL" "REALRES" "RECT" "RND" "RNRM" "ROTXY"
                       "RSUM" "RTN" "RTNERR" "R↑" "R↓" "SCI" "SDEV" "SEED" "SF" "SIGN" "SIN" "SINH" "SIZE" "SLOPE" "SOLVE" "SQRT" "STO" "STO+" "STO-" "STOEL"
                       "STOIJ" "STOP" "STO×" "STO÷" "STR" "SUM" "TAN" "TANH" "TONE" "TRACE" "TRANS" "UVEC" "VARMENU" "VIEW" "WMEAN" "WRAP" "WSIZE" "X<0" "X<>"
                       "X<>Y" "X<Y" "X=0" "X=0" "X=Y" "X>0" "X>Y" "XEQ" "XTOA" "X↑2" "X≠0" "X≠Y" "X≤0" "X≤Y" "X≥0" "X≥Y" "X≥Y" "YINT" "Y↑X" "[FIND]" "[MAX]"
                       "[MIN]" "×" "÷" "Σ+" "Σ-" "ΣREG" "ΣREG" "←" "↑" "→" "→DEC" "→DEG" "→HMS" "→HR" "→OCT" "→POL" "→RAD" "→REC" "↓" "DDAYS" "DOW" "CLK12"
                       "CLK24" "DMY" "MDY" "YMD" "DATE" "TIME" "ADATE" "ATIME" "DATE+" "XEQ IND ST" "XEQ IND" "GTO IND" "GTO IND ST" "STO IND ST" "STO+ IND ST"
                       "STO- IND ST" "STOEL IND ST" "STOIJ IND ST" "STOP IND ST" "STO× IND ST" "STO÷ IND ST" "STO ST" "STO+ ST" "STO- ST" "STOEL ST" "STOIJ ST"
                       "STOP ST" "STO× ST" "STO÷ ST" "RCL IND ST" "RCL+ IND ST" "RCL- IND ST" "RCLEL IND ST" "RCLIJ IND ST" "RCL× IND ST" "RCL÷ IND ST" "RCL
                       ST" "RCL+ ST" "RCL- ST" "RCLEL ST" "RCLIJ ST" "RCL× ST" "RCL÷ ST" "RTNNO" "RTNYES" "PI")
                     '(("@@## REQ:.*" . font-lock-preprocessor-face)) ;; Not sure why this is broken.
                     '(".42s\\'")
                     nil
                     "Major mode for editing HP-42s programs")
```

4.3 yas templates

```
echo ''
```

```
for f in ~/core/yasnippet/hp42s-mode/*; do cat $f; echo ''; done


# -*- mode: snippet -*-
# name: if-then-end
# key: if
# --
...?   @@@@ IF-BEGIN ($1/${1:$(format "%02d" (+ 1 (string-to-number yas-text)))})
GTO ${1:1$(format "%02d" (string-to-number yas-text))}
GTO ${1:$(format "%02d" (+ 1 (string-to-number yas-text)))}
LBL $1 @@@@ IF-THEN ($1/${1:$(format "%02d" (+ 1 (string-to-number yas-text)))})
@@@@ True Code ($1/${1:$(format "%02d" (+ 1 (string-to-number yas-text)))})
LBL ${1:$(format "%02d" (+ 1 (string-to-number yas-text)))} @@@@ IF-END ($1/${1:$(format "%02d" (+ 1 (string-to-number yas-text)))})

# -*- mode: snippet -*-
# name: if-then-else-end
# key: ife
# --
...?   @@@@ IF-BEGIN ($1/${1:$(format "%02d" (+ 1 (string-to-number yas-text)))}/${1:$(format "%02d" (+ 2 (string-to-number yas-text)))})
GTO ${1:1$(format "%02d" (string-to-number yas-text))}
GTO ${1:$(format "%02d" (+ 1 (string-to-number yas-text)))}
LBL $1 @@@@ IF-THEN ($1/${1:$(format "%02d" (+ 1 (string-to-number yas-text)))}/${1:$(format "%02d" (+ 2 (string-to-number yas-text)))})
@@@@ True Code ($1/${1:$(format "%02d" (+ 1 (string-to-number yas-text)))}/${1:$(format "%02d" (+ 2 (string-to-number yas-text)))})
GTO ${1:$(format "%02d" (+ 2 (string-to-number yas-text)))}
LBL ${1:$(format "%02d" (+ 1 (string-to-number yas-text)))} @@@@ IF-ELSE ($1/${1:$(format "%02d" (+ 1 (string-to-number yas-text)))}/${1:$(format "%02d" (+ 2 (string-to-number yas
@@@@ False Code ($1/${1:$(format "%02d" (+ 1 (string-to-number yas-text)))}/${1:$(format "%02d" (+ 2 (string-to-number yas-text)))})
LBL ${1:$(format "%02d" (+ 2 (string-to-number yas-text)))} @@@@ IF-END ($1/${1:$(format "%02d" (+ 1 (string-to-number yas-text)))}/${1:$(format "%02d" (+ 2 (string-to-number yas-

# -*- mode: snippet -*-
# name: if-not-then-end
# key: ifn
# --
...?   @@@@ IF-NOT-BEGIN ($1)
GTO ${1:1$(format "%02d" (string-to-number yas-text))}
@@@@ False Code ($1)
LBL $1 @@@@ IF-NOT-END ($1)

# -*- mode: snippet -*-
# name: if-not-then-end
# key: ifn
# --
...?   @@@@ IF-NOT-BEGIN ($1)
GTO ${1:1$(format "%02d" (string-to-number yas-text))}
@@@@ False Code ($1)
LBL $1 @@@@ IF-NOT-END ($1)

# -*- mode: snippet -*-
# name: if-then-end-return
# key: ifr
# --
...?      @@@@ IF-BEGIN ($1)
GTO ${1:1$(format "%02d" (string-to-number yas-text))}    @@@@ IF-FALSE-BEGIN ($1)
@@@@ False Code ($1)
RTN       @@@@ IF-FALSE-END ($1)
LBL $1    @@@@ IF-TRUE-BEGIN ($1)
@@@@ True Code ($1)
RTN       @@@@ IF-TRUE-END ($1)
```

4.4 Export & Tangle from dired

This will export to HTML and tangle all marked files in a dired buffer.

```
(defun MJR-dired-org-export (execute-blocks tangle-blocks export-html export-pdf)
  "Evaluate all code blocks, export to HTML, and tangle current file or all marked files in dired-mode"
  (interactive (if (null current-prefix-arg)
                   (list (y-or-n-p "Evaluate all blocks?")
                         (y-or-n-p "Tangle all blocks?")
                         (y-or-n-p "Export to HTML?")
                         (y-or-n-p "Export to PDF via LaTeX?"))
                   (list 't 't 't)))
  (if (not (equal major-mode 'dired-mode))
      (message "MJR-dired-org-to-html-and-code: ERROR: Not in dired mode!!")
      (let ((marked-files (dired-get-marked-files)))
        (if (null marked-files)
            (message "MJR-dired-org-to-html-and-code: ERROR: No marked files!!")
            (mapc (lambda (f)
                    (with-current-buffer (find-file-noselect f)
                      (if execute-blocks (MJR-org-babel-execute-buffer))
                      (if tangle-blocks (org-babel-tangle))
                      (if export-html (org-html-export-to-html))
                      (if export-pdf (org-latex-export-to-pdf))))
                  marked-files)))
            (message "dired-get-marked-files: Complete!")))
```

Note the previous block is set `:eval never` because I keep this function in my `.emacs` file, and thus don't need to evaluate it to get it into my personal environment.

5 Prepare Code For Conversion

I include comments like `@@##` that I use like compiler directives in C. They provide a way to remove lines of code that are incompatible or unnecessary for the target version of free42. It's a bit of a hack, but it lets me maintain one program for diff-rent versions of free42. When I "tangle" an `org-mode` buffer, the code is automatically filtered for the selected version of free42 (set via `MJR-target-free42-version`).

DM42 Version	Target Free42
DMCP 3.21 / DM42 v3.18	3.0.2
DMCP-3.20 / DM42-3.17	2.5.20
DMCP-3.20 / DM42-3.16	2.5.20
DMCP 3.18 / DM42 v3.15	2.5.16

This function cleans up tangled (C-c C-v t) code so that it can be cleanly pasted into free42.

```
(defvar MJR-target-free42-version nil)
(setq MJR-target-free42-version "3.0.2")

(defun MJR-process-tangled-42s-code ()
  "Prepare tangled hp42s code for upload"
  (interactive)
  (let ((zap-loc-too-new 0)
        (zap-loc-too-old 0))
  ;; Remove code requireing a newer version of free42 than our target
  (goto-char (point-min))
  (while (re-search-forward "^.*@@## REQ:free42>=\\([0-9.]+\\) *$" nil t)
    (if (and MJR-target-free42-version (version< MJR-target-free42-version (match-string 1)))
        (progn (cl-incf zap-loc-too-new)
```

```
            (replace-match ""))))
  ;; Remove code only needed for an older version of free42 than our target
  (goto-char (point-min))
  (while (re-search-forward "^.*@@## REQ:free42<\\([0-9.]+\\) *$" nil t)
    (if (and MJR-target-free42-version (version<= MJR-target-free42-version (match-string 1)))
        (progn (cl-incf zap-loc-too-old)
               (replace-match ""))))
  ;; Zap comments
  (goto-char (point-min))
  (while (re-search-forward "\\(@@@@\\|@@##\\).*$" nil t)
    (replace-match ""))
  ;; Zap trailing whitespace
  (goto-char (point-min))
  (while (re-search-forward "  *$" nil t)
    (replace-match ""))
  ;; Zap blank lines
  (goto-char (point-min))
  (flush-lines "^[[:space:]]*$")
  ;; Add lines after END commands
  (goto-char (point-min))
  (while (re-search-forward "^END$" nil t)
    (replace-match "END\n"))
  ;; Save the buffer
  (if (buffer-file-name)
      (save-buffer))
  ;; Report our status
  (if MJR-target-free42-version
      (message "MJR-process-tangled-42s-code (%s): Tangle complete. Zapped %d loc too old, and %d loc too new." (buffer-name) zap-loc-too-old zap-loc-too-new)
      (message "MJR-process-tangled-42s-code (%s): Tangle complete. MJR-target-free42-version not set." (buffer-name) zap-loc-too-old zap-loc-too-new))
  ))
```

The above function must be added to the tangle-hook like this:

```
(add-hook 'org-babel-post-tangle-hook 'MJR-process-tangled-42s-code)
```

6 Prepare Marked Region for Code Conversion

```
(defun MJR-copy-42s-code-in-region ()
  "Copy region and prepare it for paste into free42"
  (interactive)
  (let ((pmin (min (point) (mark)))
        (pmax (max (point) (mark))))
    (if (< pmin pmax)
        (progn (kill-ring-save (min (point) (mark)) (max (point) (mark)))
               (with-temp-buffer  (yank)
                                  (MJR-process-tangled-42s-code)
                                  (kill-ring-save (point-min) (point-max)))))))
```

7 Tangle & Load Tangled File

```
(defun MJR-tangle-and-load ()
  "Tangles current file and attempts to load the tangled file.
The tangled file is determiend by looking backward for a :tangle argument."
  (interactive)
  (save-excursion (org-babel-tangle)
                  (if (re-search-backward ":tangle[[:blank:]]+\\([^[:blank:]]+.hp42s\\)$" nil t)
                      (let ((file-name (match-string 1)))
```

```
(if (or (string-equal "yes" file-name) (string-equal "no" file-name))
    (message "No explicit tangle file specified")
    (find-file file-name))))))
```

8 free42 Notes

8.1 Character Set

```
 0 ÷
 1 ×
 2 √
 3 ∫
 4 □
 5 Σ
 6 ▶
 7 π
 8 ¿
 9 ⩽
10 [LF]
11 ⩾
12 ≠
13 ↵
14 ↓
15 →
16 ←
17 μ
18 £
19 °
20 Å
21 Ñ
22 Ä
23 ∠
24 ∈
25 Æ
26 ...
27 [ESC]
28 Ö
29 Ü
30 □
31 •
32 [SPACE]
33 !
34 "
35 #
36 $
37 %
38 &
39 ’
40 (
41 )
42 *
43 +
44 ,
45 -
46 .
47 /
```

```
 48 0
 49 1
 50 2
 51 3
 52 4
 53 5
 54 6
 55 7
 56 8
 57 9
 58 :
 59 ;
 60 <
 61 =
 62 >
 63 ?
 64 @
 65 A
 66 B
 67 C
 68 D
 69 E
 70 F
 71 G
 72 H
 73 I
 74 J
 75 K
 76 L
 77 M
 78 N
 79 O
 80 P
 81 Q
 82 R
 83 S
 84 T
 85 U
 86 V
 87 W
 88 X
 89 Y
 90 Z
 91 [
 92 \
 93 ]
 94 ↑
 95 _
 96 ‘
 97 a
 98 b
 99 c
100 d
101 e
102 f
103 g
104 h
```

```
105 i
106 j
107 k
108 l
109 m
110 n
111 o
112 p
113 q
114 r
115 s
116 t
117 u
118 v
119 w
120 x
121 y
122 z
123 {
124 |
125 }
126 ~
127 ⊢
128 :
129 Y
```

8.2 Date format

Flag 67	Flag 31	Mode
Set	N/A	Y.MD
Clear	Clear	M.DY
Clear	Set	D.MY

8.3 Stats registers

Register	Contents
ΣREG? + 0	Σx
ΣREG? + 1	Σx^2
ΣREG? + 2	Σy
ΣREG? + 3	Σy^2
ΣREG? + 4	Σxy
ΣREG? + 5	n
ΣREG? + 6	Σlnx
ΣREG? + 7	Σ(lnx)^2
ΣREG? + 8	Σlny
ΣREG? + 9	Σ(lny)^2
ΣREG? + 10	Σlnxlny
ΣREG? + 11	Σxlny
ΣREG? + 12	Σylnx

9 DM 42 Notes

9.1 Display

GrMod is a read write variable.

- 0 = standard HP-42S resolution 131x16

- 2 = DM42 full resolution 200x120
- 3 = DM42 full resolution 400x240

ResX & ResY are read only variables that have the display resolution

10 EOF