# MP\_Keypad\_Layouts.csv

Column 1 is internal name of keypad, All other columns are the 5 x 8 grid location x=B through F, y=8 through 1

Each row defines the one mapping of all key locations to a name close to the method name but including everything printed on the keys. The program then maps this name to the real function name because there may be two keys in different sports that have different text but do the same thing.

## Special values

blank is a dead key with no text. None is like blank but the keypad graphic outline will be gone too.

## To make a new keypad

1. Add a new row of keypad data to the spreadsheet and save.
2. Add new keypad name to end of keypadList in the selectKeypad method and update comments with last position number
3. Add tuple to end of loopList in the selectKeypad method. pos=index in keypadList. (pos,pos) if not sport reversible, (guest pos, home pos) if reversible.
4. Add any update to adjust the position variable in the next section.

# MP\_Keypad\_Button\_Names.csv

FUNCTION column corresponds to the Game class method name.

BUTTON\_NAME column is the text displayed on the keypad button graphic. % means new line.

If a game method is to be called by clicking a button graphic it will need to be added here to have text.

# gameDefaultValues.csv, teamDefaultValues.csv, playerDefaultValues.csv

Each column has the variable name in row 1 and the value in row 2.

To add a new variable just add a new pair at the end.

## Special values

255 is handled special in the code, usually as a blank initial value.

# MenuMap.csv

For each row a dictionary of all key-value pairs are added to a dictionary as a value with its key being the concatenation of column 1 and 2.

self.Menu\_LCD\_Text[func+menuNum]=row

Each row represents one menu and all of the data in the row defines how it will behave and decodes variables data types and position.

Correctly adding a row will create a new menu without any further code if the reference to the button already exists.

## To make a new menu

1. First have a key mapped to an empty game method.
2. Then add the method name to the self.funcDict and point it to a new menu method of that name. This is the PRE-MENU area for this method name. It’s mainly used to control if this \*\*funcString\*\* has a menu or only uses it in certain cases. The method should be
   1. def < method name>(self, game):
   2. self.menuOn(game)
   3. return game
3. If this is a standard menu then you can add the row data to MenuMap.csv and it will work.
   1. Standard menu
      1. Has one variable to display and set
      2. Each menus data is independent of each other
      3. No choice leads to another set of menus
4. Fill in row
   1. function=funcString
   2. menuNumber=order each menu of the same name is called
   3. row\_1=text of row 1 of the 16 x 2 lcd
   4. row\_2=text of row 2 of the 16 x 2 lcd
   5. varName=name of the variable to be displayed and adjusted
   6. varClock=name of clock if it is a clock variable
   7. team=name of team if it is team data
   8. gameSettingsFlag=if variable is in gameSettings put a 1 here
   9. places=number of digits in variable
   10. col=position of variables most left digit counting from left to right of the row
   11. row=1 is top, 2 is bottom
   12. blockNumList=decimal separated list of buttons that do nothing in a menu
   13. startingMenuNumber=duh, canceling here exits the menu
   14. endingMenuNumber=duh, entering here exits the function
5. If it is not standard then delve into each of the major sub-methods of self.UpdateMenu. These methods are all structured the about same and in this order, most of the time. Start and self are like this:
   1. Do any basic preparation needed
   2. Handle special cases to quit or leave the rest of the method
   3. Handle changes to the next parameters of the menu about to be chosen
   4. Load the menu data from the chosen row into a temp dictionary
   5. Handle any special cases that need to happen only after the data is loaded
   6. Combine final variable values on to the menu and update the delivery location with the changes
   7. Start or stop the menu timer

# AddressMap.csv

Each row contains the data to load the MP\_Data\_Handler.Encode method to make one two byte ‘word’ in the mp protocol. Only rows with the current sport and sport type will be added to the dictionary.

self.fullAddressMapDict[ADDRESS\_WORD\_NUMBER][ALT][remaining key-value pairs]

The final dictionary only has: GROUP, BANK, WORD, I\_BIT, H\_BIT, HIGH\_NIBBLE, LOW\_NIBBLE, BLANK\_TYPE, SEGMENT\_DATA

I\_BIT, H\_BIT, HIGH\_NIBBLE, LOW\_NIBBLE = Must be the exact name of the variable or has to have the code defined prefixes to decode with. Take special care.

### Blanking Types

These are handled in the MP\_Data\_Handler class, although the Address\_Mapping class may change the type in some cases before it hits the Encode method.

quantumDimmingTunnel = Send 0xBC for dimming tunnel in word 1.

Always section

AlwaysHighLow = Blank both BCD digits and send them no data

AlwaysHigh = Blank the BCD high nibble, send it no data, and don’t effect low nibble

AlwaysLow = Blank the BCD low nibble, send it no data, and don’t effect high nibble

Numeric section

L = Blank a one digit number in the low nibble position if it is zero. (Used a lot)

H = Blank a one digit number in the high nibble position if it is zero. (Used a lot)

HL = Blank a two digit number in the high nibble position if it is zero. (Used a lot)

LH = Blank a two digit number in the low nibble position if it is zero.

IbL = Blank low nibble position if it is zero and I bit is False.

IbH = Blank high nibble position if it is zero and I bit is False.

HbL = Blank low nibble position if it is zero and H bit is False.

HbH = Blank high nibble position if it is zero and H bit is False.

These give completeness and allow the logical 3 digit number when assigning.

IbHL = IbH (Used a lot)

IbLH = IbL

HbHL = HbH

HbLH = HbL

Isolated section

IsolateHL = Blank high nibble position if it is zero and blank low nibble position if it is zero. (Used a lot)

IsolateIbL\_H = Blank low nibble position if it is zero and I bit is False.

Also, blank high nibble position if it is zero.

IsolateIbH\_L = Blank high nibble position if it is zero and I bit is False.

Also, blank low nibble position if it is zero.

IsolateHbL\_H = Blank low nibble position if it is zero and H bit is False.

Also, blank high nibble position if it is zero.

IsolateHbH\_L = Blank high nibble position if it is zero and H bit is False.

Also, blank low nibble position if it is zero.

IsolateIbLHbH = Blank low nibble position if it is zero and I bit is False.

Also, blank high nibble position if it is zero and H bit is False.

IsolateIbHHbL = Blank high nibble position if it is zero and I bit is False.

Also, blank low nibble position if it is zero and H bit is False.

SEGMENT\_DATA = Must be the name of a decoding method in the Address\_Mapping class or a string of “abcdefg” (any of these can be deleted from the string to make the segment 0 instead of 1).

## Alternates

Rows with alts other than 1 are for functions that can change on the fly. Common alts use the same number for ease of coding and don’t have to be consecutively added.

To make a new MP sport copy and paste an old one, change names, delete alternate rows, and begin correcting all values.

LX\_Jumper\_Definition.csv

Each row contains the info for a unique LX driver.

X stands for adding a jumper.

To add a new LX add and fill in a new row.

# ETN\_Jumper\_Definition.csv

Each row contains the info for a unique ETN driver.

X stands for adding a jumper.

height and width is the ETN panel’s LED matrix.

rows is how many rows of ETN panels are in one mask.

To add a new LX add and fill in a new row.

# Models\_CX\_Jumpers\_Defaults.csv

Each row contains the info for a unique CX driver jumper configuration.

sport column is the default value selected when the model is clicked in the model menu.

To add a new CX jumper configuration add and fill in a new row.

# LED\_Positions.csv

pcbSize and pcbType together designate a unique pcb graphic object.

RefDes is the ID for each LED of the pcb. This ID and the X, Y columns are exported directly from the PADS file for the pcb with the origin the top left LED.

The 4 bounding columns are measured from the PADS file for the pcb. X, Y correspond to the offset from the origin LED to the top left corner of the pcb with a reverse polarity Y-axis. Width and Height are the dimensions of the pcb from the X, Y point. All values are copied in rows of the same pcb.

Each row is the info for one LED. The segment column has an ID (usually A-H). All rows with a common ID will light together as one segment. ETNs use this column differently. Blank is the right side and “half” is the side that is removed if it is cut in half.

When a new pcb is designed update this spreadsheet.

# Mask\_Parts.csv

maskType column describes the mask by size and function generally

Each row represents the info for a pcb on the mask.

pcbSize is the approximant height of the pcb.

positionLtoR is the ID of the pcb from right to left or from top to bottom.

The maskWidth, maskHeight, and maskRadius columns are measured from the CAD file for the mask. All values are copied in rows of the same mask.

X, Y correspond to the coordinates of each pcb from the origin at the top left corner of the mask with a reverse polarity Y-axis.

# Chassis\_Parts.csv

Column 1 is the maskType a chassis is behind. For each row with the same maskType each row is the position info for a component on the chassis.

partType is either LX (LX Driver) or PS (power supply).

positionLtoR is to name the position of each partType. LX counts from left to right but PS counts from left to right.

X and Y columns are the coordinates from the top left corner of the chassis with a reverse polarity Y-axis.

# Digits\_Per\_Model.csv

Only the rows with the same name as the model being simulated are loaded.

Each row represents a single pcb in the scoreboard.

pcbValue is the function the pcb provides and the ID.

addressWord is only used by the UnMap method to decode MP data.

LXDriver and LXHeader are the driver and header the pcb is connected to.

pcbSize and pcbType together designate a unique pcb graphic object.

maskType column describes the mask by size and function generally.

maskID is the function named mask the pcb is in.

dataOrder is the same for all rows with the same LXDriver and is the number displayed on the top right corner of the LXDriver graphic to show the order of the data from low (start) to high (end).

positionRtoL selects the position of the pcb in the mask corresponding to the same column in the Mask\_Parts.csv file.

lxPosition selects the position of the LX graphic on the chassis. For a unique value in LXDriver column this number must always match or the graphic will not appear. If the the row is an ETN panel this column will cause the ETN driver to be highlighted when the power supply is clicked that matches the same value as the psChassis column.

psWires controls the color of highlighting of the ETN panels when is clicked.

psChassis is the ID for all power supply graphics. They will be generated from left to right and top to bottom on each chassis of the whole board.

psPosition is the position of each power supply on each chassis

psLabel is the label on the power supply graphic. (In linescore sometimes this differs from the ID number)

chassisMask controls the LXDriver graphic to be on this chassis.

# Masks\_Per\_Model.csv

Only the rows with the same name as the model being simulated are loaded.

Each row represents a single mask in the scoreboard.

boardWidth and boardHeight are the dimensions of the whole face of a single cabinet and are the same on each row of the same cabinet.

X and Y columns are the coordinates from the top left corner of the face with a reverse polarity Y-axis.

qtyOfCabinets is the total number of cabinets.

positionTopToBot is the ID of the cabinet numbered from top to bottom.

# AddressMapVisualUnMapReference.xlsx

Should stay the same as the AddressMap.csv file in each cell.

Used to highlight the cells used by the UnMap function so only one value is used if value appears multiple times.

yellow = read and saved, orange = alternate used after event occurs

#### Method for adding a new sport

Go through and highlight everything needed.

Use this to add a new entry for the addressWordList for this sport.

Go through and add any special cases.