MatrixKeypad

Simple to use Arduino library to interface matrix keypads.

Data Types

`MatrixKeypad_t`

Structure that holds the physical parameters of the keypad, the pin mapping, the key mapping and the state variables.

Fields

- **`uint8_t rown`** Number of rows. Must be greater than zero.
- **`uint8_t coln`** Number of columns. Must be greater than zero.
- **`uint8_t *rowPins`** Pin mapping for the rows. These pins are set as output. Is a unidimentional matrix with length = *"rown"*.
- **`uint8_t *colPins`** Pin mapping for the columns. These pins are set as inputs. Is a unidimentional matrix with length = *"coln"*.
- **`char *keyMap`** _ Key mapping for the keypad. Its a bidimentional matrix with *"rown"* rows and *"coln"* columns. When a keypress is detect at row R and column C, the returned key is the one at *keyMap[R][C]*. The key mapping is directly related to the pin mappings. Dont use '\0' as a mapped key.
- **`char lastKey`** Holds the last key detected. Used to avoid the same keypress to be read multiple times.
- **`char buffer`** Holds the last key accepted. Is cleared after the keypress is requested. Its overwritten if a new key is pressed before the old one is requested.

Methods

`MatrixKeypad_create`

Creates a keypad object that represents the physical keypad and the pin mappings

A matrix keypad will have some pins or wires that you have to connect to digital inputs of the arduino. You will need to sort out which pins are connected to the rows, which are connected to the colluns and their ordering. Then define a matrix and initialize it with the ordered row pin number. Define another matrix for the columns. To create the key mapping, define a bidimentional array and initialize with the character to be returned when the key on it's place is pressed. Note that the key mapping ordering is directly related to the pin mapping ordering. The library don't make a copy of mappings to use less storage. The library references the mappings defined in the main sketch, so they can't be reporpused or edited.

As an example, consider a 4x3 keypad:

```
 uint8_t rown = 4; //4 rows
uint8_t coln = 3; //3 columns
uint8_t rowPins[rown] = {10, 9, 8, 7}; //frist row is connect to pin 10, second to 9...
uint8_t colPins[coln] = {6, 5, 4}; //frist column is connect to pin 6, second to 5...
char keymap[rown][coln] =
  {{'1','2','3'}, //key of the frist row frist column is '1', frist row second column column is '2'
   {'4','5','6'}, //key of the second row frist column is '4', second row second column column is '5'
   {'7','8','9'},
   {'*','0','#'}};
MatrixKeypad_t *keypad = MatrixKeypad_create((char*)keymap, rowPins, colPins, rown, coln); //keypad is the variable that you will ne
```

Definition

```
MatrixKeypad_t *MatrixKeypad_create (char *keymap, uint8_t *rowPins, uint8_t *colPins, uint8_t rown, uint8_t coln);
```

Parameters

- **`keymap`** Key mapping for the keypad. Its a bidimentional matrix with *"rown"* rows and *"coln"* columns. You can define a variable as `char keymap[rown][coln]` and cast it as `(char*)keymap`.
- **`rowPins`** Pin mapping for the rows. Is a unidimentional matrix with length "*rown*".
- **`colPins`** Pin mapping for the columns. Is a unidimentional matrix with length *"coln"*.
- **`rown`** Number of rows. Must be greater than zero.
- **`coln`** Number of columns. Must be greater than zero.

Returns

A pointer to the structure representing the keypad.

Since

1.0.0

`MatrixKeypad_scan`

Scans the keypad to check if a key is currently pressed. The time interval between scans will affect the responsiveness of the keypad. This function must be called inside the *"loop()"* function to scan the keypad periodically if you are using the **NON-BLOCKING** reading mode. A interval too long will make the keybord miss press events. However, a interval too short will consume unnecessary cpu time. A interval between 20ms and 100ms. You can put a lower limit on the scan interval saving the time of the last scan. Example to limit to at least 100ms using the variable lastScan (long):

```
if((millis() - lastScan) >= 100) {
   MatrixKeypad_scan(keypad);
   lastScan = millis();
}
```

Definition

```
void MatrixKeypad_scan (MatrixKeypad_t *keypad);
```

Parameters

- `keypad` The keypad object returned by *MatrixKeypad_create*.

Since

1.0.0

`MatrixKeypad_hasKey`

Checks if a keypress was detected.

Definition

```
uint8_t MatrixKeypad_hasKey (MatrixKeypad_t *keypad);
```

Parameters

- `keypad` The keypad object returned by *MatrixKeypad_create*.

Returns

1 if a key was pressed or 0 if none was pressed.

Since

1.0.0

`MatrixKeypad_getKey`

Returns the last key pressed. This function is **NON-BLOCKING**. It won't scan the keyboard for new events or wait for a event. You must use the *MatrixKeypad_scan* function to scan the keypad periodically.

Definition

```
char MatrixKeypad_getKey (MatrixKeypad_t *keypad);
```

Parameters

- `keypad` The keypad object returned by *MatrixKeypad_create*.

Returns

The pressed key character from the key mapping or '\0' (null character) if none key was pressed.

Since

1.0.0

`MatrixKeypad_waitForKey`

Waits until a key is pressed and returns it. If there is a unread event in the buffer, that event is returned instead. This function is **BLOCKING**. The program will freeze until a key press is detected.

Definition

```
char MatrixKeypad_waitForKey (MatrixKeypad_t *keypad);
```

Parameters

- **`keypad`** The keypad object returned by *MatrixKeypad_create*.

Returns

The pressed key character from the key mapping.

Since

1.0.0

`MatrixKeypad_waitForKeyTimeout`

Waits until a key is pressed and returns it. If there is a unread event in the buffer, that event is returned instead. This function is **BLOCKING**. The program will freeze until a key press is detected or it timeouts.

Definition

```
char MatrixKeypad_waitForKeyTimeout (MatrixKeypad_t *keypad, uint16_t timeout);
```

Parameters

- **`keypad`** The keypad object returned by *MatrixKeypad_create*.
- **`timeout`** Maximum time in milliseconds to wait for a event.

Returns

The pressed key character from the key mapping or '\0' if a timeout occurs.

Since

1.1.0

`MatrixKeypad_flush`

Cleans the unread keys buffer. You can use this function to flush the queued keypresses that weren't read by *MatrixKeypad_getKey*.

Definition

```
void MatrixKeypad_flush (MatrixKeypad_t *keypad);
```

Since

1.0.0

Source Code Version

1.1.0