FRAME Command User Manual

Overview

The "FRAME" command provides a character-cell based frame buffer with box-drawing and panel management for text-mode user interfaces. It works on both VGA displays and serial terminals (e.g. TeraTerm).
The frame buffer is a grid of character cells sized to fit the current display resolution using the active font. Each cell stores an ASCII character, a 4-bit foreground colour, and a 4-bit background colour. Changes are composed in memory and rendered to the screen with "FRAME WRITE", allowing flicker-free updates.
Panels are rectangular regions within the frame buffer that act as independent text output areas. Each panel maintains its own cursor position, enabling structured layouts such as status bars, log windows, and multi-pane dashboards.

Terminal Setup

For serial terminal use (e.g. TeraTerm), the terminal must be configured to use a CP437-compatible font so that box-drawing characters (codes 179-218) display correctly. Text colours are sent as standard ANSI 16-colour escape sequences ("\033[3Xm" / "\033[9Xm") which are universally supported by terminal emulators.

TeraTerm configuration:

- Setup -> Font -> select a CP437-capable font (e.g. **Terminal**)
- Setup -> Terminal -> Coding -> Receive: **English/US** or **CP437**

Command Reference

FRAME CREATE

```
FRAME CREATE
```

Creates the frame buffer. The buffer dimensions are calculated from the display resolution and the current font size:

- Width = "HRes / font_width" characters
- Height = "VRes / font_height" characters

The frame buffer is initially filled with spaces. On a serial terminal, the terminal window is resized to match and the screen is cleared.
Only one frame buffer can exist at a time. Calling "FRAME CREATE" when a frame already exists produces an error. The frame is automatically destroyed by "CloseAllFiles" (e.g. when a program ends or "RUN" is issued).

Example:

```
FRAME CREATE
```

FRAME CLOSE

```
FRAME CLOSE
```

Destroys the frame buffer and releases all associated memory, including all panel definitions. Produces an error if no frame exists.

Example:

```
FRAME CREATE
' ... use the frame ...
FRAME CLOSE
```

FRAME Command User Manual

FRAME BOX

```
FRAME BOX x, y, w, h [, cols, rows [, fc [, DOUBLE]]]
```

Draws a box-drawing grid in the frame buffer and automatically registers the interior cells of each grid subdivision as a numbered panel.

Parameter	Description
`x, y`	Top-left corner position in character coordinates (0-based)
`w, h`	Width and height of the outer box in characters
`cols`	Number of columns in the grid (default: 1)
`rows`	Number of rows in the grid (default: 1)
`fc`	Foreground colour for the box-drawing characters (default: current ...
`DOUBLE`	Use double-line box-drawing characters instead of single-line

The grid interior is divided as evenly as possible. When the space does not divide evenly, the remainder is distributed across the first columns/rows (each getting one extra character of width/height).
Panels are numbered sequentially starting from the next available panel ID, in row-major order (left to right, top to bottom). For example, a 3x2 grid creates 6 panels:

```
+------+------+------+
|  P1  |  P2  |  P3  |
+------+------+------+
|  P4  |  P5  |  P6  |
+------+------+------+
```

Multiple "FRAME BOX" calls accumulate panels. A second "FRAME BOX" creating 2 panels after the first created 4 would number them as panels 5 and 6.
The box must be large enough to hold the grid structure:
- Width must be at least "2 x cols" characters
- Height must be at least "2 x rows" characters

Examples:

```
' Simple single box (1 panel)
FRAME BOX 0, 0, 40, 12

' 2x2 quad layout (4 panels)
FRAME BOX 0, 0, 60, 20, 2, 2

' Double-line 3x3 grid in white
FRAME BOX 5, 2, 60, 24, 3, 3, RGB(WHITE), DOUBLE
```

FRAME TITLE

```
FRAME TITLE panel_id, text$ [, colour]
```

Centres a title string in the top border of a panel or overlay. The title is rendered between bracket characters that connect naturally with the border line.
The border style (single or double) is auto-detected from the panel's corner character.

Parameter	Description
`panel_id`	Panel number
`text$`	Title text
`colour`	Title colour (default: panel's default foreground)

Examples:

```
' Title on a single-line box:  -+ My Panel +-
FRAME BOX 0, 0, 40, 10
```

```
  FRAME TITLE 1, "My Panel"

  ' Title on a double-line overlay:  =+ Dialog +=
  FRAME OVERLAY 10, 30, 8, RGB(WHITE), DOUBLE
  FRAME TITLE 10, "Dialog", RGB(YELLOW)
```

FRAME HLINE

```
  FRAME HLINE panel_id, row [, colour [, DOUBLE]]
```

Draws a horizontal divider line across the panel at the specified row, with correct T-junction characters connecting to the left and right borders. The junction style is automatically matched to the panel's border style.

Parameter	Description
`panel_id`	Panel number
`row`	Row within the panel (0-based) where the divider is drawn
`colour`	Divider colour (default: current foreground)
`DOUBLE`	Use double-line horizontal line style

Note: The divider overwrites one row of panel content. Position panel cursor accordingly.

Example:

```
  FRAME BOX 0, 0, 40, 12
  FRAME PRINT 1, "Header text"
  FRAME HLINE 1, 1                ' Single-line divider at row 1
  FRAME CURSOR 1, 0, 2
  FRAME PRINT 1, "Body text below the divider"
  FRAME WRITE
```

FRAME COLOUR

```
  FRAME COLOUR panel_id, fg [, bg]
```

Sets the default foreground and optional background colour for a panel. These defaults are used by "FRAME PRINT" when no explicit colour is specified, and by "FRAME CLS" when clearing the panel.

Parameter	Description
`panel_id`	Panel number
`fg`	Default foreground colour
`bg`	Default background colour (default: black)

Changing colours does not repaint existing content. Use "FRAME CLS" after "FRAME COLOUR" to fill the panel with the new background colour.

Example:

```
  ' Create a panel with blue background and white text
  FRAME BOX 0, 0, 40, 10
  FRAME COLOUR 1, RGB(WHITE), RGB(BLUE)
  FRAME CLS 1                     ' Fill with blue background
  FRAME PRINT 1, "White on blue!"

  ' Highlight bar (change bg, print, change back)
  FRAME COLOUR 1, RGB(BLACK), RGB(YELLOW)
  FRAME PRINT 1, CHR$(10) + " Selected Item  "
  FRAME COLOUR 1, RGB(WHITE), RGB(BLUE)
  FRAME PRINT 1, CHR$(10) + " Normal Item"
  FRAME WRITE
```

```
  FRAME PRINT panel_id, text$ [, fc [, WRAP]]
```

Writes text into the specified panel, starting at the panel's current cursor position.

Parameter	Description
`panel_id`	Panel number (1-based, as assigned by `FRAME BOX` or `FRAME PANEL`)
`text$`	String to write
`fc`	Foreground colour (default: panel's default foreground, set by `FRA...
`WRAP`	If specified, text wraps to the next line at the panel edge. If omi...

The panel cursor advances with each character written and persists between calls. This means successive "FRAME PRINT" calls continue where the previous one left off.
Each character is written with the specified foreground colour and the panel's default background colour. Use "FRAME COLOUR" to change the background colour before printing.
Newline characters ("CHR$(10)") move the cursor to the beginning of the next line within the panel. Carriage return ("CHR$(13)") moves to the beginning of the current line.
Text that reaches the bottom of the panel is discarded unless "WRAP" is specified.
When "WRAP" is enabled and the cursor moves past the last line, the panel contents automatically scroll up by one line and writing continues on the newly cleared bottom line. This makes panels behave like a scrolling terminal window.

Examples:

```
  ' Simple text output
  FRAME PRINT 1, "Hello World"

  ' Coloured text
  FRAME PRINT 1, "Warning!", RGB(RED)

  ' Wrapping text
  FRAME PRINT 1, "This long message will wrap within the panel.", , WRAP

  ' Cursor persistence - outputs "AB" on the same line
  FRAME PRINT 1, "A"
  FRAME PRINT 1, "B"

  ' Newline
  FRAME PRINT 1, "Line 1" + CHR$(10) + "Line 2"
```

FRAME INPUT

```
  FRAME INPUT panel_id, variable [, prompt$ [, fc]]
```

Panel-aware line input. Reads a line of text from the console into the specified panel, echoing each character as it is typed. The result is stored in "variable" when Enter is pressed.

Parameter	Description
`panel_id`	Panel number (1-based). Can be a main-frame panel or a visible over...
`variable`	The variable to store the input in. Can be string, integer, or floa...
`prompt$`	Optional prompt text displayed in the panel before the input cursor...
`fc`	Optional foreground colour (RGB121). Defaults to the panel's defaul...

The input uses the panel's default background colour (set via "FRAME COLOUR").

Behaviour:

- The frame is rendered to the screen before each keystroke, providing live visual feedback
- On a serial terminal the cursor is positioned at the input location via VT100 sequences
- When "FRAME CURSOR ON" is active, the cursor blinks at the input position on both the LCD display (colour inversion) and serial terminal (VT100 cursor show/hide) at ~500 ms

intervals
- Backspace deletes the last character typed
- Enter terminates input and stores the result
- Non-printable characters and escape sequences are ignored
- If the cursor reaches the right edge of the panel, it wraps to the next line
- If the cursor reaches the bottom of the panel, contents scroll up automatically
- After Enter, the panel cursor advances to the start of the next line
- Works with overlay panels -- the cursor is correctly positioned on-screen
- Ctrl-C aborts the input (standard MMBasic break behaviour)

Variable types:
- String variables store the raw text
- Integer variables convert the text via "strtoll" (base 10)
- Float variables convert the text via "atof"

Examples:
```
' Simple string input
DIM name$ LENGTH 30
FRAME INPUT 1, name$, "Name: "

' Input with colour
FRAME INPUT 1, name$, "Enter: ", RGB(CYAN)

' Numeric input
DIM age%
FRAME INPUT 1, age%, "Age: "

' Input in an overlay
FRAME OVERLAY 10, 40, 5, RGB(WHITE)
FRAME SHOW 10, 10, 8
FRAME INPUT 10, response$, "OK? "
FRAME HIDE 10
```

FRAME CLS

```
FRAME CLS [panel_id]
```

Clears the frame buffer or a specific panel.
- **Without "panel_id":** Clears the entire frame buffer (all cells set to zero/space). All panel cursor positions are reset to (0,0), but panel definitions are preserved.
- **With "panel_id":** Clears only the specified panel's interior and resets its cursor to (0,0). If the panel has a background colour set via "FRAME COLOUR", the cleared area is filled with that background colour. The box-drawing borders and other panels are unaffected.

Examples:
```
' Clear just panel 3
FRAME CLS 3

' Clear entire frame (panels still defined)
FRAME CLS
```

FRAME CLEAR

```
FRAME CLEAR
```

Clears the entire frame buffer **and destroys all panel definitions**. The frame buffer itself remains allocated. After "FRAME CLEAR", new panels must be created with

"FRAME BOX" or "FRAME PANEL" before using "FRAME PRINT".
This is more aggressive than "FRAME CLS" which preserves panel definitions.

Example:

```
  FRAME CLEAR
  ' All panels gone - create new ones
  FRAME BOX 0, 0, 40, 12
```

FRAME WRITE

```
  FRAME WRITE
```

Renders the frame buffer to the screen. Only cells that have changed since the last
"FRAME WRITE" are updated, making this efficient for incremental updates.
On VGA displays, characters are drawn using "DisplayPutC". On serial terminals,
characters are sent as raw bytes with VT100 cursor positioning and colour escape
sequences.
"FRAME WRITE" should be called after composing all changes for a given update cycle. This
approach avoids flicker by batching all modifications before rendering.

Example:

```
  FRAME BOX 0, 0, 40, 10
  FRAME PRINT 1, "Hello"
  FRAME WRITE              ' Now the box and text appear on screen
```

FRAME CURSOR

```
  FRAME CURSOR panel_id, x, y
  FRAME CURSOR ON
  FRAME CURSOR OFF
```

Positioning form: Sets the logical cursor position for a panel. The cursor tracks
where the next "FRAME PRINT" will write text. Each panel maintains its own cursor,
initialised to (0, 0) when the panel is created.

Parameter	Description
`panel_id`	Panel number (1-based)
`x`	Column within the panel interior (0-based)
`y`	Row within the panel interior (0-based)

Query the current cursor with "FRAME(PX id)" and "FRAME(PY id)".
Visibility form: "FRAME CURSOR ON" enables the visible cursor; "FRAME CURSOR OFF"
disables it.
When the cursor is enabled:
  - On the **LCD/VGA display**, the cursor is drawn by inverting the foreground and
background colours of the cell at the cursor position. If the cell's foreground
and background would be the same after inversion (making the cursor invisible),
a high-contrast fallback is used (white-on-black or black-on-white).
  - On a **serial terminal**, the VT100 cursor visibility escape sequences
("\033[?25h" / "\033[?25l") are used.
  - The cursor **blinks** at approximately 500 ms intervals during "FRAME(INKEY)" polling
loops and "FRAME INPUT" wait states. Outside these wait states the cursor is shown
statically after each "FRAME WRITE".
The cursor is drawn at the logical cursor position of the most recently written panel
(tracked by "FRAME PRINT" and "FRAME WRITE"). For overlay panels, the cursor position
is offset to the overlay's on-screen location.

Examples:

```
' Enable cursor, position it, then write
FRAME CURSOR ON
FRAME CURSOR 1, 5, 0
FRAME PRINT 1, "Here"
FRAME WRITE

' Wait for a key with blinking cursor
k$ = FRAME(INKEY): DO WHILE k$ = "": k$ = FRAME(INKEY): LOOP

' Disable cursor
FRAME CURSOR OFF
FRAME WRITE
```

FRAME PANEL

```
FRAME PANEL id, x, y, w, h
```

Manually defines or redefines a panel at arbitrary frame buffer coordinates. This is useful for creating panels without a grid box, or for mapping a panel onto a custom hand-drawn border.

Parameter	Description
`id`	Panel number to define (1-based). If the ID exceeds the current cou...
`x, y`	Top-left corner of the panel interior (character coordinates)
`w, h`	Width and height of the panel interior in characters

The panel's cursor is reset to (0,0). If a panel with the given ID already exists, it is redefined.

Example:

```
' Draw a custom border
FRAME 0, 0, "+----+", RGB(YELLOW)
FRAME 0, 1, "|    |", RGB(YELLOW)
FRAME 0, 2, "|    |", RGB(YELLOW)
FRAME 0, 3, "+----+", RGB(YELLOW)

' Define the interior as panel 1
FRAME PANEL 1, 1, 1, 4, 2

' Now use it
FRAME PRINT 1, "Hi"
FRAME WRITE
```

FRAME x, y, text$ [, fc [, bc]]

```
FRAME x, y, text$ [, fc [, bc]]
```

Legacy direct text output. Writes a string at the specified character position without using the panel system.

Parameter	Description
`x, y`	Starting character position (0-based)
`text$`	String to write
`fc`	Foreground colour (default: current foreground)
`bc`	Background colour (default: black)

Text that reaches the right edge wraps to the beginning of the next line. Text that reaches the bottom of the frame is discarded.

Example:

```
  FRAME 0, 0, "Title Bar", RGB(WHITE)
  FRAME 10, 5, "Status: OK", RGB(GREEN)
  FRAME WRITE
```

Overlays

Overlays are independent frame buffers that can be displayed on top of the main frame at any position. They are useful for pop-up dialogs, menus, tooltips, and other temporary UI elements.
Each overlay acts as a single panel with its own unique panel ID. All standard panel-based commands ("FRAME PRINT", "FRAME CLS", "FRAME CURSOR") work with overlay panels exactly as they do with main frame panels.
When multiple overlays overlap, the most recently shown overlay appears on top.

FRAME OVERLAY

```
  FRAME OVERLAY panel_id, width, height [, colour [, DOUBLE]]
```

Creates an overlay with a private frame buffer and a border.

Parameter	Description
`panel_id`	Unique panel ID for this overlay (must not conflict with existing p...
`width`	Total width of the overlay including border (minimum 3)
`height`	Total height of the overlay including border (minimum 3)
`colour`	Optional border colour (default: current foreground colour)
`DOUBLE`	Optional keyword to use double-line border style

The overlay is created with a single-line border (or double-line if "DOUBLE" is specified).
The panel interior is the area inside the border, i.e. (width-2) x (height-2) characters.
The overlay is initially hidden. Use "FRAME SHOW" to display it.

FRAME SHOW

```
  FRAME SHOW panel_id, x, y
```

Shows an overlay at position (x, y) on the main frame. The overlay is composited on top of the main frame content during "FRAME WRITE".
Calling "FRAME SHOW" on an already visible overlay moves it to the new position and brings it to the top of the z-order.

Parameter	Description
`panel_id`	Panel ID of the overlay
`x, y`	Position on the main frame for the overlay's top-left corner

FRAME HIDE

```
  FRAME HIDE panel_id
```

Hides an overlay. The main frame content underneath is automatically restored on the next "FRAME WRITE". The overlay's content is preserved and can be shown again later.

FRAME DESTROY

```
  FRAME DESTROY panel_id
```

Destroys an overlay and frees its memory. The panel ID becomes inactive. Only works on overlay panels (not main-frame panels created by "FRAME BOX" or "FRAME PANEL").
Use this to free memory when a dialog is no longer needed, rather than keeping it

hidden indefinitely.

Example:

```
FRAME CREATE

' Create a main frame layout
FRAME BOX 0, 0, 78, 24, 2, 1
FRAME PRINT 1, "Main Panel 1"
FRAME PRINT 2, "Main Panel 2"

' Create a pop-up overlay (panel 10) with white border
FRAME OVERLAY 10, 30, 8, RGB(WHITE)
FRAME PRINT 10, "=== Pop-Up Dialog ==="
FRAME PRINT 10, CHR$(10) + "Press any key to close"

' Show it centred
FRAME SHOW 10, 24, 8
FRAME WRITE

k$ = FRAME(INKEY) : DO WHILE k$ = "" : k$ = FRAME(INKEY) : LOOP

' Hide the pop-up, main content restored
FRAME HIDE 10
FRAME WRITE
```

Multiple overlays can be stacked:

```
FRAME OVERLAY 20, 20, 5, RGB(YELLOW)
FRAME PRINT 20, "Overlay A"
FRAME OVERLAY 21, 20, 5, RGB(CYAN), DOUBLE
FRAME PRINT 21, "Overlay B"

FRAME SHOW 20, 5, 5
FRAME SHOW 21, 10, 7    ' Shown last, so on top where they overlap
FRAME WRITE
```

Virtual Buffers

A virtual buffer (vbuf) allows a panel to hold content that is larger than its visible area. Content is written into the virtual buffer using "FRAME PRINT" at the full virtual dimensions. A viewport defined by scroll offsets determines which portion of the virtual buffer is displayed in the panel.
This is ideal for scrollable content such as directory listings, log windows, help text, or any data set that exceeds the visible panel size.

FRAME VBUF

```
FRAME VBUF panel_id, vwidth, vheight
```

Allocates a virtual buffer of "vwidth" x "vheight" cells for the specified panel. The virtual dimensions must be at least as large as the panel interior (they can be larger in either or both directions). The buffer is filled with spaces using the panel's default colours, and the panel cursor is reset to (0, 0).
If the panel already has a virtual buffer, it is freed and replaced.

- "panel_id" -- The panel to attach the virtual buffer to.
- "vwidth" -- Width of the virtual buffer in characters (1-10000).
- "vheight" -- Height of the virtual buffer in characters (1-10000).

Example:

```
FRAME CREATE
FRAME BOX 1, 1, 40, 12          ' Panel 1 interior is 38x10
```

```
  FRAME VBUF 1, 80, 100           ' Virtual buffer is 80x100
```

After creating a vbuf, all "FRAME PRINT" and "FRAME CLS" operations on the panel
write into the virtual buffer at its full dimensions, not the visible panel area.

FRAME SCROLL

```
  FRAME SCROLL panel_id, sx, sy
```

Sets the viewport scroll offset for a panel that has a virtual buffer.
The viewport top-left corner maps to position ("sx", "sy") in the virtual buffer.
- "panel_id" -- The panel to scroll.
- "sx" -- Horizontal scroll offset (0 to vwidth ? panel_width).
- "sy" -- Vertical scroll offset (0 to vheight ? panel_height).

When FRAME WRITE is called, the visible portion of the virtual buffer (from the
scroll offset for the panel's width and height) is copied into the panel's
display area before rendering.

Example:

```
  ' Scroll to show lines 20-29 of an 80x100 virtual buffer in a 38x10 panel
  FRAME SCROLL 1, 0, 20
  FRAME WRITE
```

Virtual Buffer Lifecycle

- "FRAME CLS panel_id" clears the entire virtual buffer (not just the viewport).
- "FRAME CLOSE", "FRAME CLEAR", and "FRAME DESTROY" automatically free any vbufs.
- The virtual buffer persists across "FRAME SCROLL" calls.
- The panel cursor ("cx", "cy") operates in virtual buffer coordinates.

Querying Virtual Buffer State

Function	Returns
`FRAME(VW id)`	Virtual buffer width (0 if no vbuf)
`FRAME(VH id)`	Virtual buffer height (0 if no vbuf)
`FRAME(SX id)`	Current horizontal scroll offset (0 if no vbuf)
`FRAME(SY id)`	Current vertical scroll offset (0 if no vbuf)

Interactive Scrolling Example

```
  FRAME CREATE
  FRAME BOX 1, 1, 40, 12
  FRAME VBUF 1, 80, 50

  ' Fill with content
  FOR i% = 0 TO 49
    FRAME PRINT 1, "Line " + STR$(i%) + ": data..." + CHR$(10)
  NEXT i%

  ' Scroll loop with arrow keys
  DIM sx% = 0, sy% = 0
  DIM pw% = FRAME(PW 1), ph% = FRAME(PH 1)
  DO
    k$ = FRAME(INKEY)
    IF k$ <> "" THEN
      IF ASC(k$) = 128 AND sy% > 0 THEN sy% = sy% - 1        ' Up
      IF ASC(k$) = 129 AND sy% < 50-ph% THEN sy% = sy% + 1   ' Down
      IF ASC(k$) = 130 AND sx% > 0 THEN sx% = sx% - 1         ' Left
      IF ASC(k$) = 131 AND sx% < 80-pw% THEN sx% = sx% + 1    ' Right
      IF k$ = "q" THEN EXIT DO
```

```
    FRAME SCROLL 1, sx%, sy%
    FRAME WRITE
  ENDIF
LOOP
FRAME CLOSE
```

FRAME() Function Reference

The "FRAME()" function queries the state of the frame buffer, panels, and overlays.
All queries return integers unless stated otherwise.

Frame Dimensions

```
FRAME(WIDTH)    ' Returns the frame width in characters
FRAME(HEIGHT)   ' Returns the frame height in characters
```

Returns 0 if no frame has been created.

Panel and Overlay Counts

```
FRAME(PANELS)     ' Returns the number of active panels
FRAME(OVERLAYS)   ' Returns the number of overlays (visible or hidden)
```

Panel Dimensions

```
FRAME(PW panel_id)   ' Returns the interior width of a panel
FRAME(PH panel_id)   ' Returns the interior height of a panel
```

The interior dimensions exclude the box-drawing border. For example, a box drawn at 0,0 to 40,24 with a 2x1 grid will have two panels each with an interior width of 19 and height of 23.

Panel Cursor Position

```
FRAME(PX panel_id)   ' Returns the panel's cursor X position (0-based)
FRAME(PY panel_id)   ' Returns the panel's cursor Y position (0-based)
```

These return the current cursor position within the panel, relative to the panel's top-left interior corner.

Panel Colours

```
FRAME(FC panel_id)   ' Returns the panel's default foreground colour (RGB121 index)
FRAME(BC panel_id)   ' Returns the panel's default background colour (RGB121 index)
```

Panel Status

```
FRAME(ACTIVE panel_id)   ' Returns 1 if the panel is active, 0 if not
FRAME(VISIBLE panel_id)  ' Returns 1 if the overlay is visible, 0 if hidden
```

"ACTIVE" works for any panel (main-frame or overlay). "VISIBLE" only works for overlay panels and will generate an error if used on a main-frame panel.

Virtual Buffer Queries

```
FRAME(VW panel_id)   ' Returns virtual buffer width (0 if no vbuf)
FRAME(VH panel_id)   ' Returns virtual buffer height (0 if no vbuf)
FRAME(SX panel_id)   ' Returns horizontal scroll offset (0 if no vbuf)
FRAME(SY panel_id)   ' Returns vertical scroll offset (0 if no vbuf)
```

These queries return 0 when the panel does not have a virtual buffer attached.

Reading Cells

```
  FRAME(CELL x, y)              ' Read a cell from the main frame buffer
  FRAME(PCELL panel_id, x, y)   ' Read a cell from a panel's buffer
```

Returns the raw 16-bit cell value. To extract the components:

```
  cell% = FRAME(CELL x, y)
  ascii% = cell% AND &HFF                  ' Character code (bits 0-7)
  fg%    = (cell% >> 8) AND &HF            ' Foreground colour (bits 8-11)
  bg%    = (cell% >> 12) AND &HF           ' Background colour (bits 12-15)
```

"PCELL" coordinates are relative to the panel's interior (0-based).

Key Input

```
  FRAME(INKEY)   ' Non-blocking key read (returns string, like INKEY$)
```

Returns a single-character string if a key has been pressed, or an empty string if
no key is available. This is the frame-aware equivalent of "INKEY$" and should be
used in place of "INKEY$" when a frame is active.
When "FRAME CURSOR ON" is active, each call to "FRAME(INKEY)" checks the blink timer
and toggles the cursor on/off at ~500 ms intervals, providing a blinking cursor on
both the LCD display and serial terminal while polling for input.

Example -- wait for a keypress with blinking cursor:

```
  FRAME CURSOR ON
  FRAME WRITE
  k$ = FRAME(INKEY): DO WHILE k$ = "": k$ = FRAME(INKEY): LOOP
```

A frame must be created before calling "FRAME(INKEY)".

Examples

```
  FRAME CREATE
  FRAME BOX 0, 0, 78, 24, 2, 1

  ' Query the layout
  PRINT "Frame: " + STR$(FRAME(WIDTH)) + "x" + STR$(FRAME(HEIGHT))
  PRINT "Active panels: " + STR$(FRAME(PANELS))
  PRINT "Panel 1 size: " + STR$(FRAME(PW 1)) + "x" + STR$(FRAME(PH 1))

  ' Write some text
  FRAME PRINT 1, "Hello"
  PRINT "Cursor at: " + STR$(FRAME(PX 1)) + ", " + STR$(FRAME(PY 1))

  ' Read back a cell
  cell% = FRAME(PCELL 1, 0, 0)
  PRINT "First char: " + CHR$(cell% AND &HFF)

  FRAME CLOSE
```

Colour Handling

Colours are specified as standard MMBasic RGB values (e.g. "RGB(RED)", "RGB(WHITE)",
"RGB(128,255,0)"). Internally, colours are converted to 4-bit RGB121 encoding for
storage in the frame buffer (1 bit red, 2 bits green, 1 bit blue = 16 colours).
The 16 available colours correspond to the standard VGA palette. On serial terminals,
colours are mapped to the nearest ANSI 16-colour SGR codes:

RGB121 Index	Colour	ANSI SGR Code
:----------:	----------	:----------:

0	Black	30
1	Blue	34
2	Dark Green	32
3	Dark Cyan	36
4	Green	32
5	Sky Blue	94
6	Bright Green	92
7	Cyan	96
8	Red	31
9	Magenta	35
10	Brown/Orange	33
11	Light Magenta	95
12	Orange	93
13	Pink	95
14	Yellow	93
15	White	97

Frame Buffer Architecture

Each cell in the frame buffer is stored as a 16-bit value:

Bits	Content
0-7	ASCII character code (0-255)
8-11	Foreground colour (4-bit RGB121)
12-15	Background colour (4-bit RGB121)

A cell value of zero represents an empty/space cell.
The frame buffer uses a shadow buffer ("outframe") to track what has been rendered.
"FRAME WRITE" compares the two buffers and only updates cells that have changed,
providing efficient differential rendering.

Box-Drawing Characters

The "FRAME BOX" command uses CP437 box-drawing characters:

Single Line

Character	Code	Description
+	218	Top-left corner
+	191	Top-right corner
+	192	Bottom-left corner
+	217	Bottom-right corner
-	196	Horizontal line
|	179	Vertical line
+	197	Cross
+	194	Top tee
+	193	Bottom tee
+	195	Left tee
+	180	Right tee
+	198	Single vert / double horiz right
+	181	Single vert / double horiz left
+	199	Double vert / single horiz right

+	182	Double vert / single horiz left

Double Line

Character	Code	Description
+	201	Top-left corner
+	187	Top-right corner
+	200	Bottom-left corner
+	188	Bottom-right corner
=	205	Horizontal line
|	186	Vertical line
+	206	Cross
+	203	Top tee
+	202	Bottom tee
+	204	Left tee
+	185	Right tee

Asymmetric Layouts

"FRAME BOX" creates uniform grids -- every row has the same number of columns. For asymmetric layouts such as a full-width panel above two half-width panels, combine "FRAME BOX", manual box-drawing characters, and "FRAME PANEL".

The technique is:

1. Use "FRAME BOX" to draw the outer border with a horizontal divider (1x2 grid), giving a full-width top panel and a full-width bottom panel.
2. Manually draw a vertical divider in the bottom section only, using legacy text output ("FRAME x, y, text$") with the appropriate box-drawing characters.
3. Redefine the bottom panel with "FRAME PANEL" to cover just the left half, and add a new panel for the right half.

This avoids wasting a column by not placing an unnecessary vertical divider through the top panel.

Example -- full-width header with two columns below:

```
FRAME CREATE

' Draw outer box with horizontal divider (1x2 = top + bottom)
FRAME BOX 0, 0, 80, 25, 1, 2

' Panel 1 = full-width top section (already created by BOX)
' Panel 2 = full-width bottom section (will be redefined below)

' Add vertical divider in bottom section only
' The horizontal divider is at row 12, bottom border at row 24
' Midpoint column for an 80-wide box = column 40

' Top T-junction where vertical meets horizontal divider
FRAME 40, 12, CHR$(194), RGB(WHITE)

' Vertical lines through the bottom section interior
DIM i%
FOR i% = 13 TO 23
  FRAME 40, i%, CHR$(179), RGB(WHITE)
NEXT i%

' Bottom T-junction where vertical meets bottom border
```

```
FRAME 40, 24, CHR$(193), RGB(WHITE)

' Redefine panel 2 as left half, add panel 3 as right half
FRAME PANEL 2, 1, 13, 39, 11
FRAME PANEL 3, 41, 13, 38, 11

' Use the panels
FRAME PRINT 1, "Full-width header panel"
FRAME PRINT 2, "Left column"
FRAME PRINT 3, "Right column"
FRAME WRITE
```

The same approach works for any asymmetric arrangement. For double-line borders, use the corresponding double-line characters (CHR$(186) for vertical, CHR$(203) for top tee, CHR$(202) for bottom tee).

Lifecycle and Cleanup

The frame buffer is automatically cleaned up by "CloseAllFiles", which is called when:
- A program ends normally
- "RUN" is issued
- An error occurs and control returns to the command prompt

This means you do not need to explicitly call "FRAME CLOSE" at the end of a program, though you may call it if you need to release the memory during program execution.

Differences: CLS vs CLEAR vs CLOSE

Command	Frame buffer	Panel definitions	Memory
`FRAME CLS`	Cleared	Preserved	Kept
`FRAME CLS n`	Panel n cleared	Preserved	Kept
`FRAME CLEAR`	Cleared	**Destroyed**	Kept
`FRAME CLOSE`	**Freed**	**Destroyed**	**Freed**
`FRAME DESTROY n`	n/a	Overlay **Destroyed*...	Overlay **Freed**

Complete Example

```
' Dashboard with status and log panels
FRAME CREATE

' Create a 2x1 layout: status on left, log on right
FRAME BOX 0, 0, 78, 24, 2, 1

' Write status info in panel 1
FRAME PRINT 1, "=== STATUS ===", RGB(YELLOW)
FRAME PRINT 1, CHR$(10)
FRAME PRINT 1, "Temperature: 23.5C", RGB(GREEN), WRAP
FRAME PRINT 1, CHR$(10)
FRAME PRINT 1, "Humidity:    45%", RGB(GREEN), WRAP

' Write log entries in panel 2
FRAME PRINT 2, "=== LOG ===", RGB(YELLOW)
FRAME PRINT 2, CHR$(10)
FRAME PRINT 2, "10:00 System start", RGB(WHITE), WRAP
FRAME PRINT 2, CHR$(10)
FRAME PRINT 2, "10:01 Sensors OK", RGB(WHITE), WRAP
FRAME PRINT 2, CHR$(10)
```

```
FRAME PRINT 2, "10:02 Reading data", RGB(CYAN), WRAP

FRAME WRITE

' Wait then update a value
PAUSE 2000
FRAME CLS 1
FRAME PRINT 1, "=== STATUS ===", RGB(YELLOW)
FRAME PRINT 1, CHR$(10)
FRAME PRINT 1, "Temperature: 24.1C", RGB(RED), WRAP
FRAME PRINT 1, CHR$(10)
FRAME PRINT 1, "Humidity:    43%", RGB(GREEN), WRAP
FRAME WRITE

PAUSE 3000
FRAME CLOSE
```

Error Messages

Error	Cause
`Frame already exists`	`FRAME CREATE` called when a frame buffer is already allocate...
`Frame not created`	Any command other than `CREATE` used before `FRAME CREATE`
`Frame does not exist`	`FRAME CLOSE` called when no frame exists
`Panel not active`	`FRAME PRINT` or `FRAME CLS` used with an inactive panel ID
`Not an overlay panel`	`FRAME DESTROY`, `FRAME SHOW`, or `FRAME HIDE` used on a non-...
`Panel ID already in us...	`FRAME OVERLAY` used with a panel ID that is already active
`Panel has no virtual b...	`FRAME SCROLL` used on a panel without a vbuf
`Box too narrow`	Box width is less than `2 x cols`
`Box too short`	Box height is less than `2 x rows`
`Syntax error`	Missing required parameters