METAL REFERENCE

This is **METAL Reference** , a brief tutorial in programming in METAL.
Enjoy,
Marin Saric

Remember, if you have any comments, questions or bug reports write to
`sarimar@iit.edu`

Galactic Dreams Software webpage:
`<http://www.iit.edu/~sarimar/GDS>`

If it looks like you already know what the chapter is talking about, keep scrolling down.

==========

***C* Contents**

`( a list of all the chapters )`

==========

==========

1 BASIC stuff

(a quick start in BASIC language for those who don't know it)

==========

METAL is an extended variant of **BASIC** .
BASIC stands for **B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode.
What this means is that BASIC is supposed to be a simple language that any one can learn and use to program pretty much anything.

If you never programmed in BASIC before, the best thing to do is to get a good book or a tutorial (there is plenty on Internet) on any of the BASIC variants (QBasic, GWBasic, Chipmunk Basic, etc.) . In case you are adventurous, however, here is a quick start in BASIC:

```
print "Hello my man!"
input "What is your name?"; name$
print "Nice to meet you, ";name$
```

This is a BASIC **program** . Programs are like cooking recipes or laundry-lists. You do them in order. BASIC programs are also executed in order.
In METAL, you run programs with command-R, or by choosing Run from the project menu. Here is what this program does when it's run.
(The things I type in the computer are underlined .)

```
Hello my man!
What's your name? Marin
Nice to meet you, Marin
```

There. Not so bad, huh?

print and **input** are **commands** .
name$ is a **variable**. In fact it is a **string variable**, because it contains characters. That's why it has a **$** at the end.

This makes sense. 'Print' and 'input' tell the computer what to do. Since print is tedious to write all the time (and many programs output a lot of things to the screen), you can also write '**?**' :

```
? "hohohoooo..."
```

outputs

```
hohohoooo...
```

Sometimes you want the computer to decide what to do. Let's say you are making an adult game. You want to keep the kids from running your game.

```
input "How old are you?" ; age
if age<18 then ? "You are too young to play this game!!" : end
? "Welcome."
```

Let's try it out..

```
How old are you? 14
You are too young to play this game!!!!
```

Hmm.. let's try again..

```
How old are you? 23
Welcome.
```

What if you want to repeat Welcome, so it fills the screen? You have **goto** or **go to** to do that.

```
flood:
    ? "Welcome!!"
    goto flood
```

flood is a **name label**. You could have also put a **line number**, like

```
10
    ? "Welcome!!"
    goto 10
```

Let's see what happens if I run the program..(if you are stuck, hit **CTRL-D**)

```
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
Welcome!!
(oops, this keeps going on forever...)
(I have to break the program with CTRL-D )
```

This is horrible! Welcome!! keeps repeating forever -- what a welcome that is. Let's try instead to write it just 5 times...

```
for count = 1 to 5
   ? "Welcome!!!"
next count
```

This is a **for**-**next** loop... Let's run the program..

```
Welcome!!!
Welcome!!!
Welcome!!!
Welcome!!!
Welcome!!!
```

Cool..Here, that's it, you know the basics of the BASIC language. To learn more, get a decent book or an online tutorial. If you don't want to bother with that, you can learn a lot by opening running and examining the examples that come with METAL. When you are not sure what a command does, hit the **"help" key** or choose "METAL help..." from the Help menu to get help in METAL. Then you can lookup the instruction in the **METAL Instructions Index**. Copy the code from the examples and modify it, try to add something cool to it. Just change the things even if you don't have the faintest idea what they do. Trust me, this is **tinkering** -- the best way to learn anything **:)**
If you think you are hardcore enough, you can proceed to the following chapters, which are written for the people who know some sort of BASIC already.

==========

2 Programming Control

```
( flow of control and other programming things )
```

==========

Comments

rem - anything written after rem is a comment: remove; remember; rem hehe
'

Loops

for *variable* = *start* **to** *end* **[step** *increment* **]**
  commands
next

do
  commands
loop

while *condition*
  commands
wend

repeat

commands
until *condition*

do until *condition*
commands
loop

do while *condition*
commands
loop

If-Then
If-Then-Else
If-Then-Endif
If-Then-Else-Endif

if *condition* **then**
commands
else
commands
endif or **end if**

Program Termination

end
stop - for now behaves equally to end

Brute Flow-control

goto or **go to** *label*
gosub *label*
return
pop - returns from all the gosub calls to the main program

Case-based Flow-control

on *variable* **goto** *label1* , *label2* , ... , *labelN* - if *variable* = 2 , jumps to *label2*
on *variable* **gosub** *label1* , *label2* , ... , *labelN*

select case *variable*
case *value1*
commands1
case *value2*
commands2
...
case *valueN*
commandsN
case else
commandsE

end select

Variables

swap - swaps the two variables. nifty.

Predefined Data

read
restore - sets the data read pointer to the given label
data

Arrays

dim - if done twice on the same array, it clears it and reallocates
clear - clears all arrays and variables

User-defined functions

def fn - defines a custom math function. See the Hat example.

==========

3 Number Crunching

```
( Functions for manipulating numbers, math functions, etc. )
```

==========

Number Systems

$*value* - returns the hexadecimal value of the number
!*value* - returns the octal value of the number
#*value* - returns the binary value of the number

Signs and Intergers

sgn
abs
round - rounds the number to the nearest integer
int - truncates the number's integer part.

Powers

sqr - takes the square root
^ - raises the number to the power, i.e 2^3 which is 8.

Random Numbers

rnd - returns a random number between 0 and 1 (including 0 and 1)
random - returns a random integer between 0 and the given number.
randomize - pass the seed so that **rnd** doesn't return the same sequence all the time
randomize timer - the most common form of using **randomize**

Bases

These functions deal with base e.

exp
log

Trigonometry

cos
sin
tan

atn - arcus tangent
acos
asin
atan2 - does the arcus tangent with respect to the coordinate axes

==========

4 String Functions

```
( Functions for manipulating and creating strings )
```

==========

Basic

left$ - string , n . returns first n characters of the string
mid$ - string, pos, len . extracts the given substring out of a string
right$
len - string length

Searching

instr - searches the given string for the given substring
rinstr - same as **instr** but starts from the right side

Manipulation

ucase$ - converts the string into all uppercase letters
lcase$ - converts the string into all lowercase letters
flip$ - returns the string reversed

String-To-Number and vice versa

val - tries to evalute the given string as a number
str$ - converts the given number to a string
hex$ - converts the given number to a string with a hexadecimal representation

Generating Strings

space$
string$
chr$ - generates a character with the given ASCII code
asc - returns the ASCII code of a given string

sprint using - prints the formatted string into the given string
(See **print using** in the Metal Help)

==========

5 Console I/O

```
( Simple Input/Output and other things about the METAL Console )
```

==========

METAL Console tries to have all the "features" of the 1980's text screens. Fortunately, it also supports graphics. Here are the commands you can use for the Console I/O. Remember, if you are not sure how it works, just hit the "help" key when you are in METAL, or choose "METAL Help..." from the Help menu.

Simple Input/Output

print or **?**
input
line input - inputs a whole line from the console.
print using - This command is used for formatting output. See Metal Help.
input$ - inputs the given number of characters from the console

Keystrokes

inkey$ - returns the current key that is pressed (if any)
wait key - waits for a keystroke
wait key up - waits for the key to be depressed

Keymap group (for games):

keymap scan - no parameters. use this first to get the keymap of the keyboard
keymap key - use this to test if a certain key is pressed on the keymap
keymap bit - use this to test for special keys (ctrl, shift, etc.) See help.

NOTE! Try to make sure that all keyboards have the bits you are testing for.

Cursor positioning

cls - clears the screen
home - the same. Introduced for compatibility with AppleSoft Basic
locate - sets the cursor position (y,x)
pos - returns the cursor's x position
csrlin - returns the cursor\s y position

Mouse related stuff

button - returns (-1) if the button is pressed
getmousexy - returns the x and y coordinates of the mouse
getmousex
getmousey
wait button - waits for the mouse button to be pressed
wait button up - waits for the mouse button to be depressed
hidecursor - hides the mouse cursor.
showcursor - shows it

The Console Window

resize console - resize the console window
set console title to - changes the console title
hide console - hides the console window
show console - shows it
disable done - disables the blinking '<DONE>' message.

User alerts

alert error - shows the error alert with the given message.
alert note - shows the note alert with the given message.

==========

6 System Stuff

```
( Interaction with METAL Runtime and MacOS )
```

==========

These commands help you fine-tune your METAL program with the system environment.

User-Response

disable break - disables the user from breaking the program with CTRL-D

System CPU Time (events)

system breathe - gives the system some air..useful
system time - lets the system handle a couple of events

Memory

fre - returns the available memory in the Runtime Heap.
malloc - allocates a block of memory and returns its address
calloc - clears and allocates a block of memory
free - frees the block of memory

Low-Level Memory Access

peek - gets the byte from the given address
poke - address, byte. writes the given byte to the given address.
deek
doke
leek
loke

What's the Time?

timer - returns the number of seconds since 1970.
wait - waits the given number of seconds.
(you can also use stuff like wait 0.3)

Resource Management

]rsrcmerge - merges the resource fork of the given file.
attach resources - attaches the resource fork of the given file.
count resources - counts the number of the resources of the given type

Interprocessor

See METAL Help for more info.

]minversion
]spacedtokens
]ignorecase
]note

==========

7 Files

```
( File manipulating commands )
```

==========

Opening Files and file cursor positioning

open file - opens the file with the given file name. Returns a file-reference.
close file - closes the file

set to start - sets the file's read/write pointer to the beginning of the file
set to end - sets the file's R/W pointer to the end of file (EOF).

Binary Files

bopen file - opens the file with the given file name in the **binary mode**.

get file size - gets the size of the file's data fork (in bytes)

set file to - sets the file to the given position (specified in bytes).

byte read - writes a byte out into a binary file
short read - writes 2 bytes out into a binary file
int read - writes 4 bytes out into a binary file
byte write - writes a byte out into a binary file
short write - writes 2 bytes out into a binary file
int write - writes 4 bytes out into a binary file

file poke - analogous to **poke** , puts a byte into the specified location in the file
file doke
file loke

file peek - analogous to **peek** , gets a byte from the specified location in the file
file deek
file leek

block read - reads in a block of information from a binary file
block write - writes out a block of information into a binary file

bload - loads the complete binary file into memory. Use with caution.
bsave - dumps the memory block into a file. Use with caution.

Reading and Writing from and to Files

fread - reads numbers and strings from the file
fwrite - writes similar things to the file
fprint - behaves exactly like print, except it prints to a file
fprint using - use this to format your output and write to the file
line input - inputs a whole line from the file
input$ - inputs the given number of characters from the file
eof - returns (-1) if the end of file (EOF) is reached.

Managing Files (copying, renaming, deleting)

copy file
rename file
delete file

Type and Creator Code Management

set file creator
set file type
get file creator$
get file type$

Handling File Lists (folders and directories)

exists - returns (-1) if the given file exists.
is folder - returns (-1) if the given pathname is a folder.
file count - returns the number of files in the given folder
dir - lists the current folder (directory). Very powerful. See METAL Help.
curdir$ - returns the absolute path of the current folder.

Open and Save Dialogs
Also, check out the **open preview dialog$** command in the **QuickTime(TM)** chapter.

open dialog$ - opens up the standard file open dialog. Returns the pathname.
save dialog$ - opens up the standard file save dialog. Returns the pathname.

==========

8 Graphics

`( Graphics. )`

==========

Foreground and Background color

forecolor - sets the foreground color that is used by most of the gfx commands.
backcolor - sets the background color. Used by cls and text drawing routines.
opcolor - sets the operational color, for use with **copyrect** and similar routines

Dots, Lines and Shapes

plot
get pixel - gets the red, green and blue values of a pixel.
line
moveto - sets the graphics cursor to the given coordinates.
lineto - draws a line from the graphics cursor to the given (x,y) point.
circle
ellipse
fcircle - draws a filled circle
fellipse - draws a filled ellipse
rect - draws a filled box
poly - draws a polygon using the specified points

This group is used for drawing polygons with large amount of points:

poly set - sets the first point for the polygon
poly to - sets other points for the polygon
poly end - draws the polygon

Drawing Text with QuickDraw (TM)

text - x,y,message$ draws the given message on the given position.
textsize - sets the text font size.
textmode - sets the text font mode.
textfont - sets the text font. Stuff like textfont "Courier" also works.
textface - changes the text typeface style (bold, italic and such). See METAL Help.
string width - returns the length of the string in pixels

Checking the Screen Resolution

screen width
screen height
screen bitdepth - returns 8 (256 colors), 16 (thousands) or 32 (millions of colors mode)

Changing the Screen Resolution

fullscreen mode - width, height, bitdepth . Will not report an error.
normal screen - back to normal

Working with PICT files (pictures)
Also see **load quicktime pict** in the **QuickTime(TM)** chapter.

loadpict or **load pict** - loads a **PICT** file.
load pict rsrc - loads and displays a 'PICT' resource.
get pict size - gets the width and height of the given PICT file.
save pict - saves the picture enclosed by the given rectangle to disk as a pict file.
plot cicon - draws an icon from the given 'cicn' resource

Copying Blocks of Screen

copyrect - very powerful. Copies areas of screens. see METAL Help.
copyrect masked - same, but also uses a transparency mask.
copyrect deepmasked - not yet implemented.
scroll - scrolls the screen.

Virtual Screens

init screen - opens up a virtual screen with the given dimensions.
kill screen
set screen to - switches the current drawing port to the given screen.
set screen to console - switches the port back to the METAL Console.

???

setpixel - unimplemented and obsolete.

==========

9 Sound

(Playing sounds and beeps; speech)

==========

Initializing

init sound - not mandatory, but a good thing to do before using sound.

Speech Synthesis

say - uses the Speech Manager to say the given line.

System Beeps

beep - plays the system beep.

Playing Sound

Also, see **QuickTime(TM)** which you can also use to play music files.

play aiff - **OBSOLETED**. Will not work in METAL v1.7a. See help.
play rsrc - plays the given 'snd ' resource.

==========

10 QuickTime(TM)

(Commands for using this piece of technology.)

==========

Opening Movies

A **movie** is anything that QuickTime is able to open.
QuickTime can open JPEGs, GIFs, PNGs, MooVs, AVIs, MPEGs, MPGs (MPEG Layer III music), AIFFs, WAVs and much more.

load quicktime - optional. use this to preload QuickTime into memory.

open movie - opens the given movie file. Returns a movie-reference.
open preview dialog$ - opens up an open dialog with a preview box.
close movie - closes the given movie file.
movies error - use this to check if there is any errors after opening.

Playing Movies and Controling Movie Playback

start movie - starts playing the movie.

movies task - you must use this to give the QuickTime some time for playing.
is movie done - returns (-1) if the given movie finishes playing.
stop movie
rewind movie to start
rewind movie to end
rewind movie - sets the movie to the given time position. See **get movie fps**.

Movie Time/Position

get movie duration - returns the movie duration in the movie time units.
get movie time - returns the current movie time in the movie time units.
get movie fps - returns the number of movie units per second. See METAL Help.

Movie Preview Track(s)

play movie preview - plays the movie preview track.
get preview time - gets the total playing time of the movie's preview track.

Tweaking the Movie Playing Properties

set movie rect - sets the bounding rectangle for the movie display.
get movie wh - gets the movie's width and height
set movie screen - sets the movie drawing screen
set movie interpolation - suggests interpolation to QuickTime
set movie interlace - suggests interlacing to QuickTime
set movie volume
get movie volume

Movie Pictures

show movie poster - shows the movie poster picture
get poster rect
set poster rect

get movie pict - gets the picture from any movie frame.

Loading Pictures with QuickTime

load quicktime pict
get quicktime pict size

Misc

get movie name$ - returns "Untitled" for now..

==========

11 Sprites

(For games. A little bit about those things from C64.)

==========

The whole sprite system works in the **absolute coordinate system**. This means that all of the screens you use for sprite rendering should **start at (0,0)** .
Remember, using the **METAL Console** for sprite grabbing or rendering is **not recommended**.
Instead **grab and render sprites on virtual screens** and then copy them to the console screen.

Initializing the Sprite System

You must use these initialization routines before using the sprite system.

set sprite render port to - sets the screen where the sprites will be drawn
set sprite back port to - sets the background screen
no sprite background - tells the sprite system that you will draw the background.
set sprite mask color to - sets the transparent color. Usually black (0,0,0) .

Sprite Grabbing and Cloning

grab sprite - use this to store a sprite in the sprite system memory.
clone sprite - creates an identical copy of the sprite. Saves memory.

Moving and Manipulating Sprites

set sprite - defines the sprite's position.
set sprite priority - use this to draw the sprite before or after the others
hide sprite - hides the sprite. Use **set sprite** to show it again.
kill sprite - deletes the sprite from the sprite system memory.
render sprites - renders all the visible sprites to the render port.
render sprite - you will rarely need to use this.

Sprite Collision Detection

sprite collides - returns (-1) if the two given sprites collide.

A Fresh Start

reset sprites - deletes all the sprites and lets you re-set all the ports.

==========

12 Direct Screen Access

(If you render a lot.)

==========

WARNING: YOU MUST USE SET DPORT TO BEFORE ATTEMPTING TO USE DPORT COLOR OR DPLOT. METAL WILL NOT WARN YOU IF YOU FORGET TO DO THIS. IF YOU FORGET TO USE SET

DPORT TO, YOUR PROGRAM WILL CRASH.

Direct Screen Plotting

set dport to - sets the direct plotting port. Pass 0 for console.
dport color - works the same way **forecolor** does.
dplot - draws the point at x,y

• **Example:**

```
cls
set dport to 0 : 'don't forget to use set dport to
for y = 0 to 49
for x = 0 to 49
dport color rnd*65535,0,0 : 'a random shade of red
dplot x,y
next
next
```

Direct Screen Blitting (copying)

dblit - a less compatible, but slightly faster version of **copyrect**.