Macro Language Reference

PatternMaker Macro Language

by Gary Pickrell

How to use the Macro Language.

Table of Contents

1 Concepts

The Macro Language is based on the C Language. It has many things removed that aren't needed, like pointers and structures. It has a couple of special variable types to deal with PatternMaker objects such as points or objects. It has some powerful features for running PatternMaker commands. Commands can be joined together or new functions can be customized.

1.1 Functions

The most basic function looks like:

```
main()
{
  ;
}
```

main is the name of the function. It has a pair of () after it's name. After the () follows a pair of {}. The code inside the {} is run when the function is called. The *main* function is special. It is the function that is called with the macro starts.

Function names must start with a letter. After the first character the can also include numbers and underscores.

Variables can be passed two a function do something. See Variables for more information. The variables must be inside the (). It looks like.

```
print(string to_print)
{
  prompt(to_print);
}
```

They follow the format of variable type then variable name. A function can except multiple variable by separating them by comma's.

```
print(string first_string, second string)
{
  prompt(first_string);
  prompt(second_string);
}
```

To call a function you use it's name and put the variables you want inside the (). For example.

```
main()
{
  print("Hello World");
}

print(string to_print)
{
  prompt(to_print);
}
```

Functions can do logic and return a value. Unlike C they don't have to prototype their return value. It is implicitly converted to what to receiving variable is. Here is an example.

```
main()
{
  print(hello_world() );
}

hello_world()
{
  return "Hello World");
}
```

When return is called the function stops being run and returns control to the calling function.

```
main()
{
  print(hello_world() );
}

hello_world()
{
  return "Hello World");
  prompt("This will never be called");
}
```

1.2 Variables

Name

Variable names start with _ or a letter. After that they can also contain a number.

Types

The following are the variable types used in the macro language.

- **double** - Any type of number. See Working With Doubles.
- **string** - Text. See Working With Strings.
- **object** - A PatternMaker object. It can be a text object, a polygon object or a dimension line. See Working With Objects.
- **point** - PatternMaker objects are built from points. See Working With Points.
- **selection** - A group of objects or points.
- **colortype** - A color. See Working With Colors.
- **linetype** - A line pattern. See Working With Lines.
- **patterntype** - A fill pattern. See Working With Patterns.

Declaration

To declare a variable use the type then variable name. To declare a double named my_variable do:

```
double my_variable;
```

Scope

There are 3 types of variables.

1. Global
2. Local
3. Passed Parameters

Globals can be accessed anywhere in the program. They are defined anywhere outside of a function. To declare *global_var* as a global do:

```
double global_var;

main()
{
  ...
}
```

Locals are defined within a block (between {}). They are defined within functions. They can only be used within the function. To declare *local_var* as a local do:

```
main()
{
  double local_var;
  ...
}
```

Passed Parameters are used with functions. They can only be used in the function like local variables. See Functions. To declare *passed_var* as a passed parameter do:

```
function(double passed_var)
{
...
}
```

Sometimes an example is worth a 1000 words. This example demonstrates the difference between global and local variables.

```
double global_var;

main()
{
   double local_var;

  //global_var can be used here
  debug(global_var);

  //local_var can be used here
  debug(local_var);
}

func()
{
  //global_var can be used here
  debug(global_var);

  //local_var can not be used here
```

```
}
```

1.2.1 Working With Doubles

Doubles are numbers. They are floating point but can be used to represent integers. Use integer to truncate a number. Doubles support the following operators.

- (subtraction and negative sign)
+ (addition and positive sign)
=
+=
-=
*=
/=
||
&&
==
!=
<
>
<=
>=
+
*
/
%
!
-
+
++ (prefix and postfix)
-- (prefix and postfix)

1.2.2 Working With Strings

Strings are words. This is one place the macro language is different than C. String literals are contained in double quotes. For example:

```
main()
{
  string text = "Hello World";
  prompt(text);
}
```

Since the string is ended by a double quote \ is used to indicate the string isn't ending. For example:

```
main()
{
  string text = "We wrote \"Hello World\"";
  prompt(text);
}
```

Use + to concatenate strings. For example

```
main()
```

```
{
  string hello = "Hello";
  string world = "World";

  prompt(hello + " " + world);
}
```

A double concatenate with a string is converted to a string. For example:

```
main()
{
  double height = 72;

  prompt("You are " + height + inches tall");
}
```

1.2.3 Working With Objects

Objects represent PatternMaker objects. Creating objects is a multiple step process. See Creating_an_object.

If an object isn't intialized it will be equal to 0. This is useful for doing logic tests and looping. There are several ways to write this kind of test. The most common way to write it this is:

```
if (obj)
  prompt("Object exists");
```

or

```
if (!obj)
  prompt("Object doesn't exist");
```

1.2.4 Working With Points

Points make up PatternMaker objects. Points have several different parts. See Creating_a_point.

All points have an x and y coordinate. They also Names, They also have notches which can be added or changed with notch_type. They also may have a Grading_Arrows.

Like objects points are set to 0 when they don't exist. Example:

```
if (pnt)
  prompt("Point exists");
```

or

```
if (!pnt)
  prompt("Point doesn't exist");
```

1.2.5 Working With Selections

Selection sets are a group of either points or objects. A command like group requires several objects which are stored in selection set. Another use is to loop through several objects.

The process for creating a selection set is.

1. Create a selection set with newselection.
2. Add items (points or objects) to the set using select.
3. Perform logic on the selection set.
4. Finish with deselect

Creating a selection set

To create a point selection set do:

```
selection sel = newselection("POINT");
```

To create an object selection set do

```
selection sel = newselection("OBJECT");
```

Adding items

Use select to add items. To add an object to selection set do:

```
select(sel, obj);
```

To add a point to a selection set do:

```
select(sel, pnt);
```

Another way to add objects to a selection set is to have the user select items with get_user_selection. This suspends the macro and has the user select either points or objects depending on the selection set type. See Get User Selection Example.

Logic

Some commands require a selection set. An example of this is group.

Another options is looping on the selection set and doing the logic by hand. Each loop has a different item. Point selection sets contain the object that the point belongs to. Here are examples:

For an object selection set:

```
while (loop_sel(sel))
  {
  obj = get_sel_obj();

  //Do your logic
  ...
  }
```

For an point selection set

```
while (loop_sel(sel))
  {
  pnt = get_sel_point();
  obj = get_sel_obj();
```

```
  //Do your logic
  ...
  }
```

See Selection Set Examples

Finish

In PatternMaker when you select an object it is highlighted. When your command is finished it is unhighlighted. When selecting objects in a macro PatternMaker has to be told when to unhighlight. To do this use deselect. In real deselect does some more bookkeeping so it is important to call it.

Running Commands

One of the more impressive things that can be done with selection sets is running commands. A macro can run a command and tell PatternMaker which objects to use via sel_push. The commands effected by a user command can be retrieved with get_result_selection. See Running a Command Example, User Input Example, Getting Command Results Example and Running_PatternMaker_Commands.

Selection sets created by macros don't have the flexibility that ones used by PatternMaker do. PatternMaker's *ChangeFont* command only selects text objects. This same restriction can be done in a macro.

1.2.6 Working With Colors

Colors use rgb (red-green-blue) values. There are two ways to initialize a color by a number or by name.

Initialization by Name

The possible names are: "USELAYERCOLOR", "BLUE", "GREEN", "CYAN", "RED", "MAGENTA", "BROWN", "LIGHTGRAY", "DARKGRAY", "LIGHTBLUE", "LIGHTGREEN", "LIGHTCYAN", "LIGHTRED", "LIGHTMAGENTA", "YELLOW" and "WHITE"

Initialization by Number

To convert a rgb value into a number use the formula.

red*256*256 + green*256 + blue

If the number is negative then the color is the layer color.

Examples

```
main()
{
  colortype red = "RED";
  color(red);
}
```

See Initalizing Colors Example

1.2.7 Working With Lines

There are 5 line types.

Initialization by Name

The possible names are: "SOLID_LINE", "DOTTED_LINE", "CENTER_LINE", "DASHED_LINE" and "USERBIT_LINE"

Initialization by Number

The possible values are 0, 1, 2, 3 and 4

Example

```
main()
{
  linetype line = "SOLID_LINE";
  line(line);
}

main()
{
  linetype line = 2;
  line(line);
}
```

See Initalizing Line Style Example

1.2.8 Working With Patterns

There are 9 fill patterns.

Initialization by Name

The possible names are: "NONE", "SOLID", "LINE", "LTSLASH", "SLASH", "BKSLASH", "LTBKSLASH", "HATCH" and "PICTURE"

Initialization by Number

The possible values are 0, 1, 2, 3, 4, 5, 6, 7 and 8

Example

```
main()
{
  patterntype pattern = "NONE";
  pattern(pattern);
```

```
}

main()
{
  patterntype pattern = 2;
  pattern(pattern);
}
```

See

1.3 Flow Control

There are several ways to branch and loop. Use the following commands:

if
else
while () {...}
for
do {...} while

Blocks

It is important to understand what a code block is before continuing. A block can be a single line of code ending with a ; or a lot of code contained within {}

A single line of code looks like.

```
prompt("Hello World");
```

Several lines of code looks like

```
{
  prompt("Hello");
  prompt("World");
}
```

IF

If tests if a piece of logic is true then runs a block. For tests 0 is false, anything else is true.

```
if (1)
  prompt("hello");
```

Or

```
if (1)
  {
  prompt("hello");
  prompt ("world");
  }
```

ELSE

Often it is convenient to do something when an if statement is false as well as true. The else statement is used for this. It looks like:

```
if (1)
  prompt("true");
else
  prompt("false");
```

FOR

The most common type of loop is a for loop. It has 3 parts.

1. Initialization
2. Logic
3. Variable change

Each part is separated by a comma. If I want to count from 1 to 10 it looks like:

```
for (count=1; count<=10;  count++)
  prompt(count);
```

WHILE

If you don't need the setup and variable change contained in a for loop you can use while. It only tests if something is true. It looks like:

```
while (test_logic())
  {
  prompt("Thinking");
  }
```

If test logic is false it will never display the prompt.

DO

In order to make sure the logic is run once no matter what use a do while loop. It looks like

```
do
  {
  prompt("Thinking");
  } while(test_logic);
```

1.4 Running PatternMaker Commands

One of the more power features of the macro language is the ability to run PatternMaker commands using run_command. The simplest example is running a single command. To start the *Rectangle* command do:

```
main
{
  run_command("RECT");
}
```

This is not very useful. Why use a macro to do what PatternMaker can do? Input can be sent to a command using var_push. To run *Rectangle* and force the first mouse click to be 0,0 do:

```
main
{
  var_push(0,0);
  run_command("RECT");
}
```

More than one input can be pushed. To create a complete rectangle do:

```
main
{
  var_push(0,0);
  var_push(1,1);
  run_command("RECT");
}
```

There are other ways to create a rectangle in a macro. It is more impressive to ask the user for a point using user_input. To have the first point be asked do:

```
main()
{
  user_input();
  var_push(1,1);
  run_command("RECT");
}
```

For the second point do:

```
main()
{
  var_push(0,0);
  user_input();
  run_command("RECT");
}
```

It is important to note that the variable inputs need to be put on the stack before running the command. The first item on the input stack is the first input used. In other words when you have user_input before var_push, the user_input is used first.

The results of a command can be gotten with get_result_selection. This places the results in a selection set. See Working With Selections To do this do:

```
main()
{
  selection current_selection;

  var_push(0,0);
  user_input();
  run_command("RECT");
```

```
  current_selection=newselection("OBJECT");
  get_result_selection(current_selection);
}
```

This selection set can be used in another command with sel_push as follows:

```
main()
{
  selection current_selection;

  var_push(0,0);
  user_input();
  run_command("RECT");

  current_selection=newselection("OBJECT");
  get_result_selection(current_selection);

  prompt("Move Rectangle");
  sel_push(current_selection);
  user_input();
  user_input();
  run_command("MOVE");
}
```

Any numbers of commands can be chained together this way allow.

There are a couple of limitations on what can be done with the macro language. Input can't be provided for a command that displays a dialog box. The user has to fill them.

In addition there are many restrictions on selection sets in PatternMaker. For example when running the edit text command only text objects can be selected. When using the macro language these limitations aren't implemented.

1.5 Dialogs

There are 4 types of dialogs. They are:

1. Message Dialog
2. Input Dialog
3. Style Dialog (radio box)
4. Measurement Dialog

Message Dialog

A message can be displayed using dialog_box using a single string. See Dialog Box Example.

Input Dialog

Strings or numbers can be asked for using dialog_box. Up to 10 fields can be asked for. If dialog_box is passed a string the field will be a string. If it is passed a double it will ask for a number. See Dialog Box Example.

Style Dialog

Style dialogs ask the user to select one option from a list. They can include pictures but they aren't required. Setting up style dialog requires several steps

1) start_pict_dialog is used to start the dialog. Pass it a title for the dialog
2) Add item via add_pict_item. This can be called several times. It requires a title string and the name of a picture. The picture needs to be in the same directory as the macro.
3) To display the style dialog use run_pict. It always displays a cancel button. In addition it can display one or two buttons depending on how many strings you pass it. If you pass two strings two buttons will appear. It will return 0 if the left button is click or 1 for the right button.
4) To determine which radio button was selected call get_pict_result. It returned 0 if the topmost radio button was clicked.

See Creating a style dialog.

Measurement Dialog

Measurement dialogs ask for any number of numbers. There are 3 steps to create one.

1) Call start_picture_input with the title to start setting up the dialog.
2) Add any number of measurements with add_picture_input. The field is initialized and the results are passed via the variable passed to the function.
3) Call run_picture_input to display the dialog.

See Creating a measurement dialog.

1.6 Creating a point

Points can be created by using addpoint and insert_point. Addpoint is most commonly used and adds a point to the end of the last object in the drawing. Insert_point inserts a point into an object before a specified point.

Every point has a name and an x,y coordinate.

Types

Points can have one of four types. They are:

LINE - A closed line
OPEN - An open point
XARC_START - The start of a curve
XARC_CORNER - The corner point in an object

PatternMaker is fussy about where you can and can't use different point types. If you mess up the order of points you can cause problems in PatternMaker so be very careful.

Here are the rules

1. Any point can be a LINE unless it follows an XARC_START.
2. Only the last point in an object can be OPEN
3. XARC_START is the start of a curve and must be followed by an XARC_CORNER
4. XARC_CORNER can only follow XARC_START

Notches

Points can have notches. They are set with notch_type. They can have the following types.

1. NONE - No notch
2. NOTCH - A regular notch
3. DBL_NOTCH - A double notch
4. TAB - A tab notch
5. DBL_TAB - A double tab
6. BTN - A Button Hole

Arrows

Grading arrows can be added to point using addarrow. See Grading_Arrows. They are only useful in the grading studio.

See also Creating_an_object

1.7 Creating an object

There are four types of objects. Polygones, text, insertions and dimension lines. Objects are created using newobject.

Polygons are a sequence of lines and curves. They can have any amount of points. See Creating a point to for the limitation on point types.

Text objects have only one point. They have text, height and width which have to be the same.

Insertions also have one point. Like text they have height and width which must be the same. They have a symbol name which is the refers to which symbol they want.

Dimension lines have 3 points. The first two points are the start and end to measure. The third point is the location to put the text displaying the measure.

1.8 Using Layers

Layers are one of Patternmaker's ways of keeping different groups of items separate. Every object in a Patternmaker pattern has a layer. There are certain built-in functions that make use of layers; specifically, the Grade functions automatically place each size they create on a different layer, and the Marker functions create different numbers of different sizes of a garment, also by layer.

Users can hide or show groups of objects quickly by turning layers on or off, but beyond this users find extensive use of layers to be rather tiime-consuming. however, layers are an excellent tool for the macro programmer to organize patterns containing large numbers of individual items so the user can find his or her way around. Keep in mind that, other than the grading and marking behavior, Patternmaker allows

you to use layers in whatever way best suits the work at hand. Since macros are often used for custom garments which do not need to be graded, the macro author does not need to worry about arranging layers by garment size. Examples of layer arrangements include:

- Text and construction instructions on separate layers
- Multiple garments in a single file, on different layers
- Fabric groups on different layers

1.9 Creating a Symbol

Symbols are groups of objects that are used repeatedly. Insertions represent them in the drawing.

To Create a symbol put all the objects in a selection set. See Working With Selections. Then use

If a symbol is created with a name that already exists the newly made symbol will replace the old

Each Symbol has an insertion point. This is the point in the symbol that will match the point in

See Symbol Example.

1.10 Grading Arrows

Grading arrows are attached to points. Each layer contains a different size (see Using_Layers). Arrows contain the change in size from on point to another. For example a point may move 1/4" up and to the left for size 2,4, and 6. For size 8,10 and 12 it moves 1/2". To accomplish this an arrow is created that with x=0.25 and y=0.25. Then a breakpoint is added at size 8 with the value of x=0.25 and y=0.25.

Use is_break to test if an arrow has a break point at a specific layer. The value at a layer value can be found using arrow_x and arrow_y. The name of an arrow can be found using get_arrow_name.

1.11 Markers

Marking is the process of assembling garment pieces to be cut from a bolt of cloth in a manufacturing setting. The marking process requires, first, that certain Patternmaker objects be designated as pieces of the garment (as distinguished from things such as grain lines darts, etc. which will not be cut out of fabric), and second, that other parameters relating to the marking job be set. The following marker functions implement the various functions elated to creating and arranging markers.

[****]
marker_piece
Declares a Patternmaker drawing object to be a marker piece. In general, any object of type Poly can be a marker piece.
is_marker
Returns true if the object is a marker piece.
marker_area
Toggles (show or hide) the marker area.
get_markerarea
Returns true if the marker area is on (showing).

marker_width
Sets the width of the marker area (normally, the width of the cloth bolt or cutting table to be used).
get_marker_width
Returns the width of the marker area.
marker_style
Sets the marker style.
get_marker_style
Returns the marker style.
marker_fabric
Sets the marker fabric type.
get_marker_fabric
Returns the marker fabric type.
plaid_point
DISABLED
get_plaid_point
DISABLED
set_layer_cutratio
Sets the cut ratio for the given layer.
get_layer_cutratio
Returns the cut ratio for the given layer.

2 Angle Macro

Karina Beeke created her own macro that calculated the angle between 3 points. It is a good simple example of how to use the macro language. I've used it here as a starting point to build on and add programming concepts.

2.1 Starting

The origin

Karina Beeke posted to PMUG a macro that asks the user for three points then it prompts the user with the angle.

I'm very impressed with Karina's initiative. It is exciting to see someone be this resourceful. I want to encourage her and anyone else like her to create there own macros. There are some things that can be done to clean up the macro. I'm going to post these changes.

Here is her macro.

```
// To find the angle subtended at Pt0 by Pt1 and Pt2
// Written by Karina Beeke 2009.
//
//    | Pt2       /Pt1
//    |          /
//    |         /
//    |        /
//    |       /
//    |      /
//    |     /
//    |    /
//    |   /
```

```
//    |  /
//    | /
//    |/Pt0
//

double x[3], y[3], Theta_rad, Theta_deg, Rad_to_deg;
main()
{
  Rad_to_deg = 57.295779513082320876798154814105;

  prompt_point("Select apex of Angle");
  x[0] = getresultx();
  y[0] = getresulty();

  prompt_point("Select end of first line");
  x[1] = getresultx();
  y[1] = getresulty();

  prompt_point("Select end of second line");
  x[2] = getresultx();
  y[2] = getresulty();

  Theta_rad = angle(x[0], y[0], x[2], y[2]) - angle(x[0], y[0], x[1], y[1]);
  Theta_deg = Theta_rad * Rad_to_Deg;

  If (Theta_deg<-180){Theta_deg=Theta_deg+360;}
  If (Theta_deg>180){Theta_deg=Theta_deg-360;}

  prompt_point("Angle is " + Theta_deg + " degrees");
}
```

Ifs

PatternMaker 7.5 is picker about syntax than 7.0. It won't allow mixed case key words. This means this code:

```
  If (Theta_deg<-180){Theta_deg=Theta_deg+360;}
  If (Theta_deg>180){Theta_deg=Theta_deg-360;}
```

won't work in the future. It needs to be replaced with:

```
  if (Theta_deg<-180){Theta_deg=Theta_deg+360;}
  if (Theta_deg>180){Theta_deg=Theta_deg-360;}
```

Normalization

The purpose of the if lines is to make sure the resulting angle is between -180 and 180. This is called angle normalization. Remember when dealing with angles 0=360=720. Forcing the angle to be within this range makes it easier to read angles. There is a macro language function that does this. Using it we can replace the lines:

```
  Theta_deg = Theta_rad * Rad_to_Deg;
```

```
  If (Theta_deg<-180){Theta_deg=Theta_deg+360;}
  If (Theta_deg>180){Theta_deg=Theta_deg-360;}
```

with the line:

```
  Theta_deg = normalizeangle(Theta_rad)*Rad_to_deg;
```

In this line we are converting the angle from Radians (PatternMaker uses radians rather than degrees) in addition to normalizing the angle.

Prompting

The last line of the macro prints the resulting angle. It is:

```
  prompt_point("Angle is " + Theta_deg + " degrees");
```

This prints a prompt then asks the user to click a point. We don't need the click a point. Instead we need to just print the angle. In order to do this we use the prompt command.

```
  prompt("Angle is " + Theta_deg + " degrees");
```

After all this clean up we have the following macro

```
// To find the angle subtended at Pt0 by Pt1 and Pt2
// Written by Karina Beeke 2009.
//
//     | Pt2        /Pt1
//     |           /
//     |          /
//     |         /
//     |        /
//     |       /
//     |      /
//     |     /
//     |    /
//     |   /
//     |  /
//     | /
//     |/Pt0
//

double x[3], y[3], Theta_rad, Theta_deg, Rad_to_deg;

main()
{
  Rad_to_deg = 57.295779513082320876798154814105;

  prompt_point("Select apex of Angle");
  x[0] = getresultx();
```

```
  y[0] = getresulty();

  prompt_point("Select end of first line");
  x[1] = getresultx();
  y[1] = getresulty();

  prompt_point("Select end of second line");
  x[2] = getresultx();
  y[2] = getresulty();

  Theta_rad = angle(x[0], y[0], x[2], y[2]) - angle(x[0], y[0], x[1], y[1]);
  Theta_deg = normalizeangle(Theta_rad)*Rad_to_deg;

  prompt("angle is " + theta_deg + " degrees");
}
```

2.2 Functions

We've started with an angle macro from Karina Beeke. Here we are going to do some more cleanup. We are going to add some functions to break down the logic and make it more organized. In a macro this small it may seam pointless, but for bigger macros it becomes important.

What to functionize

Looking at the code we notice the code for asking for a point is basically the same. This is ideal for putting into a function.

The part that does the calculations also can be put into a function. It does not reduce the code but it makes the code more readable.

To summarize we have 2 things to do

- Mouse Inputs
- Logic

Converting prompts

When we look at the first input we have:

```
  prompt_point("Select apex of Angle");
  x[0] = getresultx();
  y[0] = getresulty();
```

The string to prompt and the array index are the parts that differ for each point. We need to pass them to the function. The syntax for this is:

```
ask_for_point(double index, string prompt_string)
```

Breaking down each part

- *ask_for_point* is the name of the function
- index is the first parameter
- double is the type of variable index is
- **prompt_string** is the string to display to the user
- string means **prompt_string** is a string

```
ask_for_point(double index, string prompt_string)
{
  prompt_point(prompt_string);
  x[index] = getresultx();
  y[index] = getresulty();
}
```

To call our function we can use the following code

```
ask_for_point(0, "Select apex of Angle");
```

The following happens when we do this.

- It finds the function *ask_for_point*()
- index becomes 0
- **prompt_string** becomes “Select apex of Angle”
- It prompts for the point with the message **prompt_string**, ie “Select apex of Angle”
- It sets x[0] to *getresultx()*
- It sets y[0] to *getresulty()*

main() can be simplified to

```
main()
{
  Rad_to_deg = 57.295779513082320876798154814105;

  ask_for_point(0, "Select apex of Angle");
  ask_for_point(1, "Select end of first line");
  ask_for_point(2, "Select end of second line");

  Theta_rad = angle(x[0], y[0], x[2], y[2]) - angle(x[0], y[0], x[1], y[1]);
  Theta_deg = normalizeangle(Theta_rad)*Rad_to_deg;
```

```
  prompt("angle is " + theta_deg + " degrees");
}
```

This is more readable than our original function.

Doing logic

Another simplification we can make is putting the logic into a function. It would look like.

```
calculate_angle()
{
  Theta_rad = angle(x[0], y[0], x[2], y[2]) - angle(x[0], y[0], x[1], y[1]);
  Theta_deg = normalizeangle(Theta_rad)*Rad_to_deg;
}
```

There are some simplifications we can make. **Rad_to_deg** is only used in this function. It makes sense to declare and initialize it in the function rather than at the start of the program. This can be done as follows.

```
calculate_angle()
{
  double Rad_to_deg;
  Rad_to_deg = 57.295779513082320876798154814105;

  Theta_rad = angle(x[0], y[0], x[2], y[2]) - angle(x[0], y[0], x[1], y[1]);
  Theta_deg = normalizeangle(Theta_rad)*Rad_to_deg;
}
```

Scope

Now I need to explain a complicated concerpt scope. If you don't fully follow the following it is ok.

The code

```
  double Rad_to_deg;
```

Declares the variable ***Rad_to_deg***. Since it is defined with the function *calculate_angle*() it is only defined with *calculate_angle*() If you try to used it within *main*() you'll get an error.

x and **y** are defined outside of every function so they can be used in all functions.

Return

Functions can be used to calculate values. The return command is used to pass the calculated values back and use them in a different function. Using this *calculate_angle*() can be rewritten as:

```
calculate_angle()
{
  double Theta_rad, Rad_to_deg;
  Rad_to_deg = 57.295779513082320876798154814105;
```

```
  Theta_rad = angle(x[0], y[0], x[2], y[2]) - angle(x[0], y[0], x[1], y[1]);
  return normalizeangle(Theta_rad)*Rad_to_deg;
}
```

To use this within main() we use the following code:

```
  Theta_deg = calculate_angle();
```

The Final Implementation

After refactoring we get:

```
// To find the angle subtended at Pt0 by Pt1 and Pt2
// Written by Karina Beeke 2009.
//
//    | Pt2       /Pt1
//    |          /
//    |         /
//    |        /
//    |       /
//    |      /
//    |     /
//    |    /
//    |   /
//    |  /
//    | /
//    |/Pt0
//

double x[3], y[3];

main()
{
  double Theta_deg;

  ask_for_point(0, "Select apex of Angle");
  ask_for_point(1, "Select end of first line");
  ask_for_point(2, "Select end of second line");

  Theta_deg = calculate_angle();

  prompt("angle is " + Theta_deg + " degrees");
}

calculate_angle()
{
  double Theta_rad, Rad_to_deg;
  Rad_to_deg = 57.295779513082320876798154814105;

  Theta_rad = angle(x[0], y[0], x[2], y[2]) - angle(x[0], y[0], x[1], y[1]);
  return normalizeangle(Theta_rad)*Rad_to_deg;
}
```

```
ask_for_point(double index, string prompt_string)
{
  prompt_point(prompt_string);
  x[index] = getresultx();
  y[index] = getresulty();
}
```

The important things to note are:

- **x** and **y** are defined outside *main*() so they have global scope. They can be used inside all functions. They are used inside *calculate_angl*() and *ask_for_point()*
- **Theta_rad** and **Rad_to_deg** are defined within *calculate_angle*() and can only be used within it
- **index** and **prompt_string** are passed parameters to *calculate_angle*(). They can only be used with *calculate_angle*(), but the values are set by the call to the function, ie *ask_for_point(0, "Select apex of Angle")*;
- *calculate_angle*() returns a value which can be used within *main*()

2.3 Draw Angle

What we plan to do

It would be nice to be able to see the angle we are measuring. We can do that by adding a line object for each line. That is what we plan to do.

Creating an Angle Object

Everything drawn on the screen is an object. To display the angle we need to create an object for it. Here is the code to create an object.

```
  newobject("Poly", "LIGHTBLUE");
  addpoint(x[1], y[1], "LINE");
  addpoint(x[0], y[0], "LINE");
  addpoint(x[2], y[2], "OPEN");
```

newobject() creates an object. "Poly" tell PatternMaker to create a polygon object. A polygon is a series of lines. Our angle is two lines so it fits the bill. "LIGHTBLUE" is the color for the object. If we don't specify the color it will use the default drawing color.

addpoint() adds a point to the end of the last created object. It requires the coordinates of the point and the point type. There are four point types LINE, OPEN, XARC_START and XARC_CORNER. LINE creates a line from the point being added to the next point. If all the points were LINEs we would have a triangle. To convert the triangle into an angle we convert the last point to OPEN. The last point in a line is the only legal place to have an OPEN point.

We have added an object to the object list. If we leave it there it will become part of the drawing. This isn't what we want. In order to prevent this we need to store the object in a variable so we can remove it at the end of the macro. *newobject()* returns an object for such a case. We can store as follows.

```
AngleObject = newobject("Poly", "LIGHTBLUE");
```

The variable **AngleObject** needs to be defined as a global so it can be accessed in another function It's declaration is.

```
object AngleObject;
```

Just because we have added an object to the drawing doesn't mean it gets drawn immediately. We need to force it to redraw the screen using *refresh_drawing()* This command doesn't exist in older version of PatternMaker. To test if PatternMaker has a function use the *is_function()* command.

The function to draw an angle looks like

```
object AngleObject;

draw_angle()
{
  AngleObject = newobject("Poly", "LIGHTBLUE");
  addpoint(x[1], y[1], "LINE");
  addpoint(x[0], y[0], "LINE");
  addpoint(x[2], y[2], "OPEN");

  if (is_function("refresh_drawing"))
    { refresh_drawing(); }
}
```

Erasing An Object

After them measured angle has been displayed the angle object need to be erased. We've stored the angle object in **AngleObject**. To erase the object we need to put the object into a selection set. Selection sets are a group of objects or points. The design pattern for this is:

1. Create the selection set
2. Add the object/point to the selection set
3. Do something with the selection set

The code for erasing the angle is:

```
erase_angle()
{
  selection Sel;

  Sel=newselection("OBJECT");
  select(Sel, AngleObject);
  erase(Sel);
}
```

Putting it all together

We've written the subroutines for adding an angle object and erasing it. They are *draw_angle()* and *erase_angle().* If we use *prompt()* to display the angle the angle object will be drawn and erase in the blink of an eye. We will replace *prompt*() with *dialog_box*() which displays a popup dialog. It will be placed between *draw_angle()* and *erase_angle().* This yields:

```
main()
{
  double Ang;

  ask_for_point(0, "Select apex of Angle");
  ask_for_point(1, "Select end of first line");
  ask_for_point(2, "Select end of second line");

  Ang = calculate_angle();
  draw_angle();

  dialog_box("angle is " + Ang + " degrees");

  erase_angle();
}
```

The Final Macro

```
 // To find the angle subtended at Pt0 by Pt1 and Pt2
// Written by Karina Beeke 2009.
// Modified by Gary Pickrell
//
//    | Pt2       /Pt1
//    |          /
//    |         /
//    |        /
//    |       /
//    |      /
//    |     /
//    |    /
//    |   /
//    |  /
//    | /
//    |/Pt0
//

double x[3], y[3];
object AngleObject;

main()
{
  double Ang;

  ask_for_point(0, "Select apex of Angle");
```

```
  ask_for_point(1, "Select end of first line");
  ask_for_point(2, "Select end of second line");

  Ang = calculate_angle();
  draw_angle();

  dialog_box("angle is " + Ang + " degrees");

  erase_angle();
}

erase_angle()
{
  selection Sel;

  Sel=newselection("OBJECT");
  select(Sel, AngleObject);
  erase(Sel);
}

draw_angle()
{
  AngleObject = newobject("Poly", "LIGHTBLUE");
  addpoint(x[1], y[1], "LINE");
  addpoint(x[0], y[0], "LINE");
  addpoint(x[2], y[2], "OPEN");

  if (is_function("refresh_drawing"))
    { refresh_drawing(); }
}

calculate_angle()
{
  double Radians, Conversion;
  Conversion = 57.295779513082320876798154814105;

  Radians = angle(x[0], y[0], x[2], y[2]) - angle(x[0], y[0], x[1], y[1]);
  return normalizeangle(Radians)*Conversion;
}

ask_for_point(double Index, string PromptString)
{
  prompt_point(PromptString);
  x[Index] = getresultx();
  y[Index] = getresulty();
}
```

2.4 Lines Command

In PatternMaker when you rotate an object it draws an object as you input the angle to rotate by. It

would be nice to create an object like this. There are two ways to do this.

1. Modify our angle object to add points as they are clicked on rather than after all of them have been entered
2. Use the line command to draw two lines as they are entered.

The first is a very good way of doing it. Using the line command allows us to learn how to run a command from the macro language. This can be very useful.

Running the line command

To run the line command we must tell it what type of input to use. For the first line we want the user in enter a point. Use the *user_input()* for this. After we've set up the input run the command. At first this seams backwards. The reason we setup the inputs first is because when we run the command the macro stops and runs the command. The macro needs to have given the command all the information it needs by this time. The code looks like

```
  user_input();
  user_input();
  run_command("LINE");
```

Getting line results

The line command adds an object at the end of the object list. To get the result use the command *last_obj()* An object has a list of points just like the drawing has a list of objects. We can get the first and last points of an object using *first_point()* or *last_point().* The code looks like

```
  StartLine = last_obj();
  StartPoint = first_point(StartLine);
  ApexPoint = last_point(StartLine);
}
```

We need to make sure to declare a global object and point. It uses the code.

```
object StartLine;
point ApexPoint;
```

First Line Function

Here is what the function for the first line looks like. Note we've added a *prompt()*

```
get_first_line()
{
  prompt("Select start of first line");
  user_input();
  user_input();
  run_command("LINE");

  StartLine = last_obj();
  StartPoint = first_point(StartLine);
  ApexPoint = last_point(StartLine);
```

```
}
```

The Second Line Function

The second line needs to start at the Apex point. Having the user enter is point is redundant. Instead we can tell PatternMaker to use a specific point. To do this we use *var_push()* command rather than *user_input()*. In this case it takes two values, a x coordinate and a y coordinate. We will also want to save the object and user entered point into different variables. It looks like:

```
get_second_line()
{
  prompt("Select end of second line");
  var_push(pointx(ApexPoint), pointy(ApexPoint) );
  user_input();
  run_command("LINE");

  EndLine = last_obj();
  EndPoint = last_point(EndLine);
}
```

Calculating the angle

Rather than having our points stored in a **x** and **y** array they are stored in the point variables StartPoint, ApexPoint and EndPoint. To get the x and y values out of a point use *pointx()* and *pointy().*

```
  Radians = angle(pointx(ApexPoint), pointy(ApexPoint), pointx(StartPoint), pointy(StartPoint)) - angle
(pointx(ApexPoint), pointy(ApexPoint), pointx(EndPoint), pointy(EndPoint));
```

Erasing Objects

There are two object to delete. We have to add this to our *erase()* function. It requires adding a second object to the selection set we erase. The code looks like.

```
erase_angle()
{
  selection Sel;

  Sel=newselection("OBJECT");
  select(Sel, StartLine);
  select(Sel, EndLine);
  erase(Sel);
}
```

The Final Macro

Incorporating the above changes in yields:

```
// To find the angle subtended at Pt0 by Pt1 and Pt2
// Written by Karina Beeke 2009.
// Modified by Gary Pickrell
//
```

```
//    | Pt2       /Pt1
//    |          /
//    |         /
//    |        /
//    |       /
//    |      /
//    |     /
//    |    /
//    |   /
//    |  /
//    | /
//    |/Pt0
//

object StartLine, EndLine;
point StartPoint, ApexPoint, EndPoint;

main()
{
  get_first_line();
  get_second_line();

  dialog_box("angle is " + calculate_angle() + " degrees");
  erase_angle();
}

get_first_line()
{
  prompt("Select start of first line");
  user_input();
  user_input();
  run_command("LINE");

  StartLine = last_obj();
  StartPoint = first_point(StartLine);
  ApexPoint = last_point(StartLine);
}

get_second_line()
{
  prompt("Select end of second line");
  var_push(pointx(ApexPoint), pointy(ApexPoint) );
  user_input();
  run_command("LINE");

  EndLine = last_obj();
  EndPoint = last_point(EndLine);
}

erase_angle()
{
  selection Sel;
```

```
  Sel=newselection("OBJECT");
  select(Sel, StartLine);
  select(Sel, EndLine);
  erase(Sel);
}

calculate_angle()
{
  double Radians, Conversion;
  Conversion = 57.295779513082320876798154814105;

  Radians = angle(pointx(ApexPoint), pointy(ApexPoint), pointx(StartPoint), pointy(StartPoint)) - angle
(pointx(ApexPoint), pointy(ApexPoint), pointx(EndPoint), pointy(EndPoint));
  return normalizeangle(Radians)*Conversion;
}
```

2.5 Final Touches

Translation

Not everyone in the world speaks English so we must translate our macro if we want everyone to use it. To determine the what language PatternMaker is running in use *get_language()*. It's return value is:

0=English
1=Finnish
2=Dutch
3=German
4=Spanish
5=French

It is useful to put that value of *get_language()* into a global variable the macro can be tested in another language by change it's value. Then the prompt commands need to be replaced with a function that will prompt it the correct language. An example of this for the first line's prompt looks like:

```
first_line_prompt()
{
  string text;

  if (lang==0) text = "Select start of first line";
  else if (lang==1) text = "Ensiluokkainen alku -lta edellä asettaa riviin";
  else if (lang==2) text = "Selecteer begin van eerste lijn";
  else if (lang==3) text = "Wählen Sie Anfang der ersten Linie vor";
  else if (lang==4) text = "Seleccione el comienzo de la primera línea";
  else if (lang==5) text = "Choisissez le début de la première ligne";

  prompt(text);
}
```

The same idea here works for a dialog_box, just replace *prompt()* with *dialog_box()*

Making the lines Cyan

In the move command PatternMaker creates a tempary object to show how far the object is being moved. The line is always a solid cyan line. Our object uses the default color and line style. It would be nice to be able to control this. Fortunately we can set the default color and line style using *color()* and *line()*. The code looks like:

```
  color("CYAN");
  line("SOLID_LINE");
```

Because this changes the default colors we need to save the default color and line style. We can do that with *get_color()* and *get_line().* We have to declare variables to store the returned values in. The code looks like:

```
  colortype SaveColor;
  linetype SaveLine;

  SaveColor = get_color();
  SaveLine = get_line();
```

The last thing to be done is to use the *color()* and *line()* to reset the original values. All the above code can be added to main. The result is:

```
main()
{
  colortype SaveColor;
  linetype SaveLine;

  lang = get_language();

  SaveColor = get_color();
  SaveLine = get_line();
  color("CYAN");
  line("SOLID_LINE");

  get_first_line();
  get_second_line();

  color(SaveColor);
  line(SaveLine);

  angle_dialog_box();
  erase_angle();
}
```

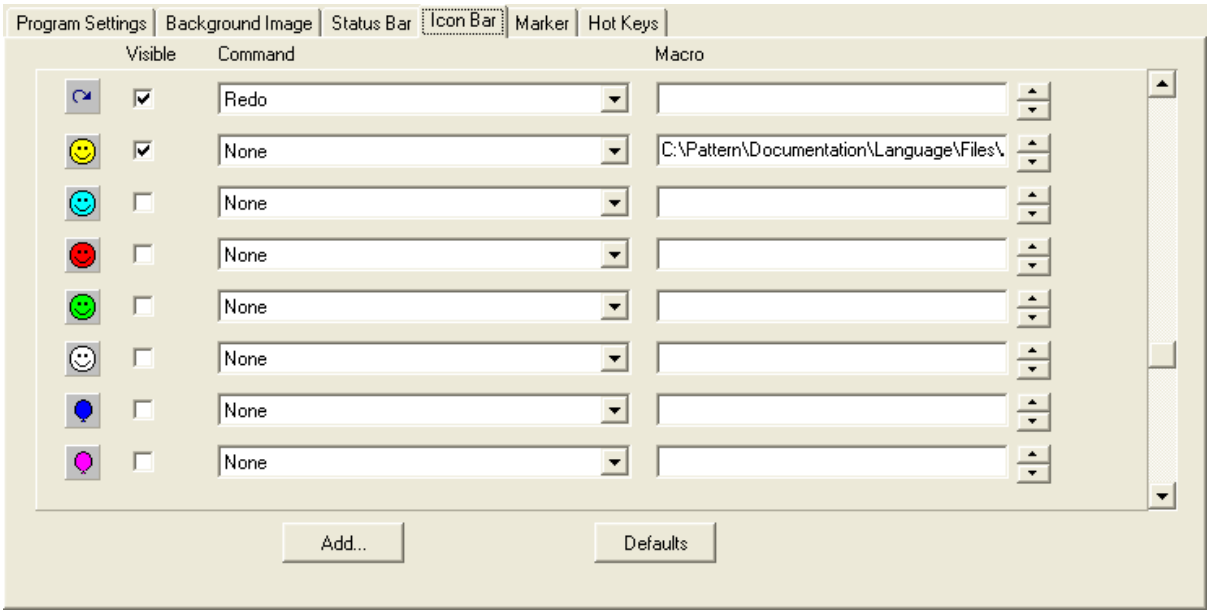
Making a Icon for the Macro

It is inconvenient to have to run our macro through the macro command. Fortunately you can add a macro to the icon bar as follows:

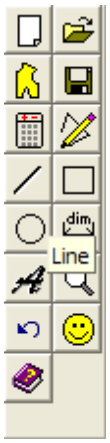
Setting->Configure

If Icon Bar is hidden click Advanced

1. Find an icon you like. In this case it is the yellow smiley face.
2. Double click in the macro edit field and select Angle Macro Part5-Finishing Touches.mac
3. Ensure Visible is checked
4. Click ok



The icon is added to the icon be. See below



The Final Macro

```
// To find the angle subtended at Pt0 by Pt1 and Pt2
// Written by Karina Beeke 2009.
// Modified by Gary Pickrell
//
//    | Pt2       /Pt1
//    |          /
//    |         /
//    |        /
//    |       /
```

```
//    |      /
//    |     /
//    |    /
//    |   /
//    |  /
//    | /
//    |/Pt0
//

object StartLine, EndLine;
point StartPoint, ApexPoint, EndPoint;
double lang;

main()
{
  colortype SaveColor;
  linetype SaveLine;

  lang = get_language();

  SaveColor = get_color();
  SaveLine = get_line();
  color("CYAN");
  line("SOLID_LINE");

  get_first_line();
  get_second_line();

  color(SaveColor);
  line(SaveLine);

  angle_dialog_box();
  erase_angle();
}

first_line_prompt()
{
  string text;

  if (lang==0) text = "Select start of first line";
  else if (lang==1) text = "Ensiluokkainen alku -lta edellä asettaa riviin";
  else if (lang==2) text = "Selecteer begin van eerste lijn";
  else if (lang==3) text = "Wählen Sie Anfang der ersten Linie vor";
  else if (lang==4) text = "Seleccione el comienzo de la primera línea";
  else if (lang==5) text = "Choisissez le début de la première ligne";

  prompt(text);
}

second_line_prompt()
{
  string text;
```

```
  if (lang==0) text = "Select end of second line";
  else if (lang==1) text = "Ensiluokkainen häntäpää -lta avustaa asettaa riviin";
  else if (lang==2) text = "Selecteer eind van tweede lijn";
  else if (lang==3) text = "Wählen Sie Ende der zweiten Linie vor";
  else if (lang==4) text = "Seleccione el extremo de la segunda líne";
  else if (lang==5) text = "Choisissez l'extrémité de la deuxième ligne";

  prompt(text);
}

angle_dialog_box()
{
  string text;

  if (lang==0) text = "The angle is " + calculate_angle() + " degrees";
  else if (lang==1) text = "kalastaa on " + calculate_angle() + "-KIRJAIN arvo";
  else if (lang==2) text = "de hoek is de graden van " + calculate_angle();
  else if (lang==3) text = "Winkel ist " + calculate_angle() + "-Grad";
  else if (lang==4) text = "el ángulo es grados de " + calculate_angle();
  else if (lang==5) text = "l'angle est des degrés de " +calculate_angle();

  dialog_box(text);
}

get_first_line()
{
  first_line_prompt();

  user_input();
  user_input();
  run_command("LINE");

  StartLine = last_obj();
  StartPoint = first_point(StartLine);
  ApexPoint = last_point(StartLine);
}

get_second_line()
{
  second_line_prompt();
  var_push(pointx(ApexPoint), pointy(ApexPoint) );
  user_input();
  run_command("LINE");

  EndLine = last_obj();
  EndPoint = last_point(EndLine);
}

erase_angle()
{
  selection Sel;

  Sel=newselection("OBJECT");
```

```
  select(Sel, StartLine);
  select(Sel, EndLine);
  erase(Sel);
}

calculate_angle()
{
  double Radians, Conversion;
  Conversion = 57.295779513082320876798154814105;

  Radians = angle(pointx(ApexPoint), pointy(ApexPoint), pointx(StartPoint), pointy(StartPoint)) - angle
(pointx(ApexPoint), pointy(ApexPoint), pointx(EndPoint), pointy(EndPoint));
  return normalizeangle(Radians)*Conversion;
}
```

3 Examples

3.1 Drawing Examples

3.1.1 Layer Example

```
main()
{
  object obj;
  double value;

  obj = newobject("POLY");
  addpoint(10,10, "LINE");
  layer(obj,3);
  value = getlayer(obj);

  debug(value);
}
```

The output is

value = 3.0

3.1.2 Group Example

```
object current_object;
selection current_selection;
object group_obj[2];

main()
{
  string name;

  current_object=newobject("POLY","USELAYERCOLOR","NONE","SOLID_LINE",0,1);
  addpoint(5,3,"LINE");
  addpoint(5,5,"LINE");
  group_obj[1]=current_object;
```

```
  current_object=newobject("POLY","USELAYERCOLOR","NONE","SOLID_LINE",0,1);
  addpoint(4,2,"LINE");
  addpoint(4,4,"LINE");
  group_obj[0]=current_object;

  current_selection=newselection("OBJECT");
  select(current_selection,group_obj[0]);
  select(current_selection,group_obj[1]);
  group(current_selection);
  deselect(current_selection);
  name_group(group_obj[0],"Group");

  name = get_group_name(group_obj[0]);
  debug(name);
}
```

The output is:

name = Group

3.1.3 System Settings Example

```
main()
 {
   debug(get_line());
   line("DOTTED_LINE");
   debug(get_line());

   debug(get_pattern());
   pattern("SOLID");
   debug(get_pattern());

   debug(get_color());
   color("GREEN");
   debug(get_color());
   color(192);
   debug(get_color());
 }
```

The output is:

get_line() = SOLID_LINE
get_line() = DOTTED_LINE
get_pattern() = NONE
get_pattern() = SOLID
get_color() = 0.0
get_color() = 49152.0
get_color() = 192.0

3.2 Grading Examples

3.2.1 Arrow Example

```
main()
{
```

```
  object obj;
  point pnt, pnt2;
  double x, y;
  string name;

  obj = newobject("POLY");
  addpoint(1,10, "LINE");
  pnt = addpoint(2,10, "LINE");
  addpoint(3,10, "OPEN");

  addarrow(pnt, obj, 2,3, "Hello");
  arrow_break(pnt, 3, -3, -4);

  name = get_arrow_name(pnt);
  debug(name);

  x = arrow_x(pnt, 0);
  y = arrow_y(pnt, 0);
  debug(x);
  debug(y);

  x = arrow_x(pnt, 3);
  y = arrow_y(pnt, 3);
  debug(x);
  debug(y);
}
```

The output is:

name = Hello
x = 2.0
y = 3.0
x = -3.0
y = -4.0

3.2.2 Breakpoint Example

```
main()
{
  object obj;
  point pnt;
  double value;

  obj = newobject("POLY");
  addpoint(1,10, "LINE");
  pnt = addpoint(2,10, "LINE");
  addpoint(3,10, "OPEN");

  addarrow(pnt, obj, 2,3, "Hello");
  arrow_break(pnt, 3, -3, -4);

  value = is_break(pnt,3);
  debug(value);
}
```

The output is:

value = 1.0

3.3 Math Examples

3.3.1 Arc Line Intersection Example

```
main()
 {
   double x, y, z;

   z = arc_line_intersect(0,1,  1,1,  1,0,  0,0, 1,1  ,1);
   x  = getresultx();
   y  = getresulty();
   debug(x);
   debug(y);
 }
```

The output is

x = 0.7071067811865475
y = 0.7071067811865476

3.3.2 Arc Arc Intersection Example

```
main()
{
  double x,y,z;

  z = arc_arc_intersect(0,1,  1,1,  1,0,  0,0, 1,0, 1,1  ,1);
  x  = getresultx();
  y  = getresulty();
  debug(x);
  debug(y);
}
```

---

x = 0.8646973235024038
y = 0.5

3.3.3 Math Function Example

```
main()
{
 debug(sqrt(2));
 debug(sin(3.14/2));
 debug(cos(0));
 debug(acos(0));
 debug(asin(1));
 debug(atan(1,1));
 debug(angle(0,0,1,1));
 debug(distance(0,0,1,1));
}
```

The output is:

```
sqrt(2) = 1.4142135623730951
sin(3.14/2) = 1.0
cos(0) = 1.0
acos(0) = 1.5707963267948966
asin(1) = 1.5707963267948966
atan(1,1) = 0.7853981633974483
angle(0,0,1,1) = 0.7853981633974483
distance(0,0,1,1) = 1.4142135623730951
```

3.3.4 Circle Intersection Example

```
 main()
 {
   double x,y;

   circlesintersect(0,0,1,  1,1,1, 0);
   x = getresultx();
   y = getresulty();
   debug(x);
   debug(y);

   circlesintersect(0,0,1,  1,1,1, 1);
   x = getresultx();
   y = getresulty();
   debug(x);
   debug(y);
 }
```

The output is:

```
x = 1.0
y = 0
x = 0
y = 1.0
```

3.3.5 Intercept Line Example

```
main()
{
   double dist;

   dist = interceptline(0,1,2);
   debug(dist);
}
```

The output is:

```
dist = 1.7320508075688772
```

3.3.6 Line Line Intersection Example

This is an example of intersecting the lines (0,0)(24,24) and (5,17)(15,9) ignoring end point intersections.

```
main()
{
  double x;
  double y;

  intersect(0, 0,  24, 24,  5, 17,  15, 9,  0);

  x  = getresultx();
  y  = getresulty();
  debug(x);
  debug(y);
}
```

The result is:

x = 11.666666666666666
y = 11.666666666666666

3.3.7 Polar Example

```
main()
{
  double x,y;

  polar(0, 0, 3.14/2, 5);
  x  = getresultx();
  y  = getresulty();
  debug(x);
  debug(y);
}
```

The output is:

x = 0
y = 5

3.3.8 Place Corner Example

```
double x, y;

main()
{
  place_corner(0, 10,
               3, 5,
               10, 0,
               0,10,
               10,0);
  x=getresultx();
  y=getresulty();
  debug(x);
```

```
   debug(y);
 }
```

The output is:

x = 3
y = 5

3.4 Marker Examples

3.4.1 Create Marker Example

```
main()
{
   object obj;
   double x;

   obj = newobject("POLY");
   addpoint(10,10, "LINE");
   marker_piece(obj, 2);

   debug( is_marker(obj));
   debug(get_bundle(obj));
}
```

The output is:

is_marker(obj) = 1.0
get_bundle(obj) = 2.0

3.4.2 Marker Area Example

```
main()
 {
   double area;

   marker_area("ON");
   area = get_markerarea();
   debug(area);

   marker_area("OFF");
   area = get_markerarea();
   debug(area);

   marker_width(36);
   debug(get_marker_width());

   marker_style("Dress");
   debug(get_marker_style());

   marker_fabric("Twill");
```

```
  debug(get_marker_fabric());
}
```

```
area = 1.0
area = 0.0
get_marker_width() = 36.0
get_marker_style() = Dress
get_marker_fabric() = Twill
```

3.4.3 Plaid Point Example

```
main()
 {
  object obj;

  obj = newobject("POLY");
  addpoint(10,10, "LINE");
  marker_piece(obj, 2);

  plaid_point(obj,5,15 , 1);

  get_plaid_point(obj);
  debug( getresultx());
  debug( getresulty());
}
```

```
getresultx() = 5.0
getresulty() = 15.0
```

3.4.4 Cut Ratio Example

```
main()
 {
   lock_layer(5);

   set_layer(0,"ON","My Layer","WHITE");
   name_layer(1, "Second Layer");
   debug(get_layer_name(0));
   debug(get_layer_name(1));

   set_layer_cutratio(2, 5);
   debug(get_layer_cutratio(2));

   debug(get_layer_color(1));
   color_layer(1, "RED");
   debug(get_layer_color(1));
 }
```

The output is:

get_layer_name(0) = Layer0
get_layer_name(1) = Second Layer
get_layer_cutratio(2) = 5.0
get_layer_color(1) = 49152.0
get_layer_color(1) = 192.0

3.5 Object Examples

3.5.1 Name Object Example

```
main()
{
  object obj;
  string name;

  obj = newobject("POLY");
  addpoint(1,10, "LINE");
  addpoint(2,10, "LINE");

  name_object(obj, "Fred");
  name = get_object_name(obj);

  debug(name);
}
```

The output is:

name = Fred

3.5.2 Looping through the drawing

```
main()
{
  object obj;
  point pnt;
  double x;

  obj = newobject("POLY");
  addpoint(10,10, "LINE");

  newobject("POLY");
  addpoint(20,20, "LINE");

  newobject("POLY");
  addpoint(30,30, "LINE");

  //print out the first point in each object
  for (obj = first_obj(); obj!=0; obj = next_obj(obj))
    {
    pnt = last_point(obj);
    x = pointx(pnt);
    debug(x);
    }
}
```

The output is:

x = 10.0
x = 20.0
x = 30.0

3.5.3 Last Object Example

```
main()
 {
   object obj;
   point pnt;
   double x;

   newobject("POLY");
   addpoint(10,10, "LINE");

   newobject("POLY");
   addpoint(20,20, "LINE");

   obj = last_obj();
   pnt = last_point(obj);
   x = pointx(pnt);

   debug(x);
 }
```

The output is:

x = 20

3.5.4 Object Pattern Example

```
main()
 {
   object obj;
   string style;

   obj = newobject("POLY");
   addpoint(7,4,"NO_TYPE");
   addpoint(1,1,"NO_TYPE");
   addpoint(7,5,"NO_TYPE");

   style = get_obj_pattern(obj);
   debug(style);

   set_obj_pattern(obj, "SOLID");
   style = get_obj_pattern(obj);
   debug(style);
 }
```

The output is:

style = NONE
style = SOLID

3.5.5 Object Color Example

```
main()
 {
   object obj;
   double style;

   obj = newobject("POLY");
   addpoint(7,4,"NO_TYPE");
   addpoint(1,1,"NO_TYPE");
   addpoint(7,5,"NO_TYPE");

   style = get_obj_color(obj);
   debug(style);

   set_obj_color(obj, "RED");
   style = get_obj_color(obj);
   debug(style);
 }
```

The output is:

style = 0.0
style = 192.0

3.5.6 Object Line Type Example

```
main()
{
  object obj;
  string style;

  obj = newobject("POLY");
  addpoint(7,4,"NO_TYPE");
  addpoint(1,1,"NO_TYPE");
  addpoint(7,5,"NO_TYPE");

  style = get_obj_line(obj);
  debug(style);

  set_obj_line(obj, "DOTTED_LINE");
  style = get_obj_line(obj);
  debug(style);
}
```

The output is:

```
style = SOLID_LINE
style = DOTTED_LINE
```

3.5.7 Is Clockwise Example

```
main()
 {
   object obj;
   double x;

   obj = newobject("POLY");
   addpoint(10,10, "LINE");
   addpoint(10,20, "LINE");
   addpoint(30,20, "LINE");
   addpoint(30,10, "LINE");

   x = is_clockwise(obj);
   debug(x);

   obj = newobject("POLY");
   addpoint(10,10, "LINE");
   addpoint(30,10, "LINE");
   addpoint(10,20, "LINE");

   x = is_clockwise(obj);
   debug(x);
 }
```

```
x = 1.0
x = 0.0
```

3.5.8 Creating Text Example

```
main()
 {
   object obj;
   string word, font;
   double size, angle;

   obj = newobject("POLY");
   addpoint(10,10, "LINE");
   debug(is_text(obj));
   obj=newobject("TEXT","Hello",0.785398,1,1,"PatternMaker");
   addpoint(1,1,"LINE");
   debug(is_text(obj));
   word = get_text(obj);
   debug(word);

   set_text(obj,"Goodbye");
   word = get_text(obj);
```

```
  debug(word);

  font = get_font(obj);
  debug(font);

  set_font(obj, "Big");
  font = get_font(obj);
  debug(font);
  size = get_width(obj);
  debug(size);

  set_width(obj,5);
  size = get_width(obj);
  debug(size);

  angle = get_angle(obj);
  debug(angle);

  set_angle(obj,5);
  angle = get_angle(obj);
  debug(angle);
}
```

The input is:

```
is_text(obj) = 0.0
is_text(obj) = 1.0
word = Hello
word = Goodbye
font = PatternMaker
font = Big
size = 1.0
size = 5.0
angle = 0.785398
angle = 5.0
```

3.5.9 Creating a Dimension Line Example

```
main()
 {
  object obj;

  obj=newobject("DIM",-1,"A","SOLID_LINE",0);
  addpoint(1,3,"LINE");
  addpoint(1,1,"LINE");
  addpoint(2,2,"LINE");

  debug(is_dim(obj));
  debug(is_poly(obj));

  obj = newobject("POLY");
```

```
  addpoint(10,10, "LINE");

  debug(is_dim(obj));
  debug(is_poly(obj));
}
```

The output is:

```
is_dim(obj) = 1.0
is_poly(obj) = 0.0
is_dim(obj) = 0.0
is_poly(obj) = 1.0
```

3.5.10 Symbol Example

```
object current_object;
 point current_point;
 selection current_selection;
 string Name;

 LoadSymbolTester()
 {
   object symbols[2];

   current_object=newobject("POLY");
   addpoint(2,-1,"LINE");
   addpoint(2,2,"LINE");
   symbols[0]=current_object;
   debug(is_insertion(current_object));

   current_object=newobject("POLY");
   addpoint(1,-2,"LINE");
   addpoint(1,1,"LINE");
   symbols[1]=current_object;

   current_selection=newselection("OBJECT");
   select(current_selection,symbols[1]);
   select(current_selection,symbols[0]);
   symbol(current_selection,"Tester",0,0);
   erase(current_selection);
 }

 LoadSymbolBob()
 {
   object symbols[2];

   current_object=newobject("POLY");
   addpoint(2,-1,"LINE");
   addpoint(2,2,"LINE");
   symbols[0]=current_object;

   current_selection=newselection("OBJECT");
   select(current_selection,symbols[0]);
   symbol(current_selection,"Bob",3,3);
   erase(current_selection);
 }
```

```
main()
{
 double size;

  LoadSymbolTester();
  LoadSymbolBob();
  if (block_loaded("Tester") && block_loaded("Bob"))
    {
    debug("Symbol Loaded");
    current_object=newobject("SYMBOL","Tester",0,1,1,"USELAYERCOLOR","NONE","SOLID_LINE",0);
    addpoint(2,2,"NO_TYPE");
    debug(is_insertion(current_object));

    size = get_height(current_object);
    debug(size);

    set_height(current_object,5);
    size = get_height(current_object);
    debug(size);

    Name = get_symbol_name(current_object);
    debug(Name);

    //This changes the printed symbol from Tester to Bob
    set_symbol_name(current_object, "Bob");
    Name = get_symbol_name(current_object);
    debug(Name);

    debug(block_loaded("Tester"));
    debug(block_loaded("Bob"));
    debug(block_loaded("New Name"));
    }
  else
    {
    debug("Symbol not Loaded");
    }
}
```

---

is_insertion(current_object) = 0.0
"Symbol Loaded" = Symbol Loaded
is_insertion(current_object) = 1.0
size = 1.0
size = 5.0
Name = Tester
Name = Bob
block_loaded("Tester") = 1.0
block_loaded("Bob") = 1.0
block_loaded("New Name") = 0.0

3.5.11 Looping Through an Object Example

```
//This loops through all the points in an object and prints them out.

 main()
{
   selection current_selection;
   object obj;
```

```
  point pnt;

  prompt("Select an object to print out the points on.");

  current_selection=newselection("OBJECT");
  get_user_selection(current_selection);

  while (loop_sel(current_selection))
    {
    obj = get_sel_obj();

    pnt=first_point(obj);
    do
      {
      prompt("Point = (" + pointx(pnt)+ "," + pointy(pnt) + ")");
      pnt=next_point(obj, pnt);
      } while (pnt!=first_point(obj))
        }
 deselect(current_selection);
}
```

3.6 Point Examples

3.6.1 Getting a Points Value Example

```
main()
{
  object obj;
  point pnt;
  double x,y;

  obj = newobject("POLY");
  pnt = addpoint(1,2, "LINE");

  x=pointx(pnt);
  y=pointy(pnt);

  debug(x);
  debug(y);
}
```

The output is:

```
x = 1.0
y = 2.0
```

3.6.2 Naming the last point in an object

```
main()
 {
   object obj;
   point pnt;
   string name;
```

```
   obj = newobject("POLY");

   addpoint(27,30, "LINE");
   addpoint(10,1, "LINE");

   pnt = last_point(obj);
   name_point(pnt,"Hello");
   name = get_point_name(pnt);

   debug(name);
   }
```

The output is:

name = Hello

3.6.3 Previous Point Example

```
main()
{
  object obj;
  point pnt;
  double x, y;

  obj = newobject("POLY");
  addpoint(1,10, "LINE");
  pnt = addpoint(2,10, "LINE");
  addpoint(3,10, "OPEN");

  pnt = prev_point(obj, pnt);
  x = pointx(pnt);
  y = pointy(pnt);
  debug(x);
  debug(y);
}
```

The output is:

x = 1.0
y = 10.0

3.6.4 First Point Example

```
main()
{
  object obj;
  point pnt;
  double x,y;

  obj = newobject("POLY");
  addpoint(27,30, "LINE");
  addpoint(10,1, "LINE");

  pnt = first_point(obj);
```

```
  x=pointx(pnt);
  y=pointy(pnt);

  debug(x);
  debug(y);
}
```

The output is:

x = 27.0
y = 30.0

3.6.5 Next Point Example

```
main()
 {
 object obj;
 point pnt;
 double x, y;

 obj = newobject("POLY");
 addpoint(1,10, "LINE");
 pnt = addpoint(2,10, "LINE");
 addpoint(3,10, "OPEN");

 pnt = next_point(obj, pnt);
 x = pointx(pnt);
 y = pointy(pnt);

 debug(x);
 debug(y);
 }
```

The output is:

x = 3.0
y = 10.0

3.6.6 Remove Point Example

```
main()
{
  object obj;
  point pnt;
  double x,y;

  obj = newobject("POLY");

  addpoint(27,30, "LINE");
  addpoint(10,1, "LINE");

  pnt = last_point(obj);
  remove_point(obj, pnt);
  pnt = last_point(obj);

  x = pointx(pnt);
```

```
  y = pointy(pnt);

  debug(x);
  debug(y);
}
```

The output is:

x = 27.0
y = 30.0

3.6.7 Get Point Type Example

```
main()
 {
  object obj;
  point pnt;

  obj = newobject("POLY");

  pnt = addpoint(0,0, "LINE");
  debug(get_point_type(pnt));
  pnt = addpoint(1,0, "XARC_START");
  debug(get_point_type(pnt));
  pnt = addpoint(1,1, "XARC_CORNER");
  debug(get_point_type(pnt));
  pnt = addpoint(0,1, "OPEN");
  debug(get_point_type(pnt));
 }
```

---

get_point_type(pnt) = LINE
get_point_type(pnt) = XARC_START
get_point_type(pnt) = XARC_CORNER
get_point_type(pnt) = OPEN

3.6.8 Notch Example

```
main()
 {
  object obj;
  point pnt;
  double notch;

  obj = newobject("POLY");
  addpoint(0,10, "LINE");
  pnt = addpoint(10,10, "LINE");
  addpoint(10,10, "OPEN");

  notch_type(pnt,"NOTCH", 1);
  notch = get_notch_dir(pnt);
```

```
  debug(notch);
  debug( get_notch_type(pnt));
}
```

The output is:

notch = 1.0
get_notch_type(pnt) = NOTCH

3.6.9 Point Type Example

```
main()
{
  object obj;
  point start, mid, end;

  obj = newobject("POLY");

  start = addpoint(0,0, "LINE");
  end = addpoint(1,1, "OPEN");

  //insert a point between start and end
  mid = insert_point(obj, end);
  set_line(mid);
  set_x(mid,1);
  set_y(mid,0);

  //We have an open triangle.  Lets verify it's correct
  if (is_line(start) && is_line(mid) && is_open(end))
       prompt("Triangle Ok");
  else
       prompt("Triangle Broken");

  //Convert triangle to a curve
  set_xarc(start);
  set_corner(mid);
  set_line(end);

  //We have an closed curve.  Lets verify it's correct
  if (is_xarc(start) && is_corner(mid) && is_line(end))
       prompt("Curve Ok");
  else
       prompt("Curve Broken");

  //Make curve open
  set_open(end);
  if (is_open(end))
       prompt("Open OK");
  else
       prompt("Open Broken");

  //Convert the object to a close triangle
  set_point_type(start, "LINE");
  set_point_type(mid, "LINE");
  set_point_type(end, "LINE");
```

```
  if (is_line(start) && is_line(mid) && is_line(end))
       prompt("Closed Triangle OK");
  else
       prompt("Closed Triangle Broken");

  //Convert the object to a close triangle
  set_point_type(start, "XARC_START");
  set_point_type(mid, "XARC_CORNER");
  set_point_type(end, "OPEN");

  if (is_xarc(start) && is_corner(mid) && is_open(end))
       prompt("Open Curve OK");
  else
       prompt("Open Curve Broken");
}
```

Triangle Ok
Curve Ok
Open OK
Closed Triangle OK
Open Curve OK

3.7 Dialog Box Examples

3.7.1 Dialog Box Example

```
main()
{
  dialog_box("I'm writting my own code!");
}
```

```
main()
{
  double x, y;

  dialog_box("Title", "X Value", x, "Y Value", y);
  debug(x);
  debug(y);
}
```

3.7.2 Creating a style dialog

```
main()
{
  double result1, result2;

  //Create a two button dialog
  start_pict_dialog("A Number");
  add_pict_item("Item 1","Item1.jpg");
  add_pict_item("Item 2","Item2.jpg");
  run_pict("Done");
```

```
  result1=get_pict_result();
  debug(result1);

  //Create a three button dialog
  start_pict_dialog("A Letter");
  add_pict_item("Item A","A.jpg");
  add_pict_item("Item B","B.jpg");
  if (run_pict("Back", "Done))
    {
   result2=get_pict_result();
   debug(result2);
    }
  else
    prompt("Back Clicked");
}
```

3.7.3 Creating a measurement dialog

```
main()
{
  double x, y;

  //This will assume we've created the following files
  //XFocus.jpg
  //XFocus.txt
  //YFocus.jpg
  //YFocus.txt

  start_picture_input("My exciting Title");
  add_picture_input("X Value", x, "XFocus");
  add_picture_input("Y Value", y, "YFocus");
  run_picture_input();

  debug(x);
  debug(y);
}
```

3.7.4 Creating a radio box

```
main()
{
  double result;

  result = pop_up("One x", "Two", "Three");
  debug(result);
}
```

3.8 Selection Set Examples

3.8.1 Object Selection Set Example

```
object current_object;
 selection current_selection;
object group_obj[3];

 main()
 {
   string name;

   current_object=newobject("POLY","USELAYERCOLOR","NONE","SOLID_LINE",0,1);
   addpoint(4,2,"LINE");
   addpoint(4,4,"LINE");
   group_obj[0]=current_object;
   name_object(group_obj[0], "First Object");

   current_object=newobject("POLY","USELAYERCOLOR","NONE","SOLID_LINE",0,1);
   addpoint(5,3,"LINE");
   addpoint(5,5,"LINE");
   group_obj[1]=current_object;
   name_object(group_obj[1], "Second Object");

   current_object=newobject("POLY","USELAYERCOLOR","NONE","SOLID_LINE",0,1);
   addpoint(6,4,"LINE");
   addpoint(6,6,"LINE");
   group_obj[2]=current_object;
   name_object(group_obj[2], "Third Object");

   current_selection=newselection("OBJECT");
   select(current_selection,group_obj[0]);
   select(current_selection,group_obj[1]);
   select(current_selection,group_obj[2]);

   while (loop_sel(current_selection))
     {
     current_object = get_sel_obj();
     name = get_object_name(current_object);
     debug(name);
     }
  deselect(current_selection);
 }
```

The output is:

```
name = First Object
name = Second Object
name = Third Object
```

Note: The order does not matter

3.8.2 Point Selection Set Example

loop_sel
get_sel_point
select
newselection

```
object obj;
selection current_selection;
point pnt[3];
point name_pnt;

main()
{
  string name;

  obj = newobject("POLY","USELAYERCOLOR","NONE","SOLID_LINE",0,1);
  pnt[0] = addpoint(4,2,"LINE");
  pnt[1] = addpoint(4,4,"LINE");
  pnt[2] = addpoint(4,6,"LINE");

  name_point(pnt[0], "Red");
  name_point(pnt[1], "White");
  name_point(pnt[2], "Blue");

  current_selection=newselection("POINT");
  select(current_selection, obj, pnt[0]);
  select(current_selection, obj, pnt[1]);
  select(current_selection, obj, pnt[2]);

  while (loop_sel(current_selection))
    {
    name_pnt = get_sel_point();
    name = get_point_name(name_pnt);
    debug(name);
    }
deselect(current_selection);
}
```

The output is:

name = Red
name = White
name = Blue

Note: The order does not matter

3.8.3 Get User Selection Example

```
selection current_selection;

 main()
 {
   string name;

   current_selection=newselection("OBJECT");
   get_user_selection(current_selection);

   while (loop_sel(current_selection))
     {
     name = get_object_name(get_sel_obj());
     debug(name);
     }
  deselect(current_selection);
 }
```

3.9 Running Commands Examples

3.9.1 Running a Command Example

```
selection current_selection;

 main()
 {
   current_selection=newselection("OBJECT");
   get_user_selection(current_selection);
   sel_push(current_selection);
   run_command("ERASE");
   deselect(current_selection);
 }
***
```

3.9.2 Generated Mouse Input Example

```
//This will run a rectangle with the first click at 1,1 and then
//asks the user for the next input
main()
{
  //Emulate clicking at 1,1 (in inches)
  var_push(1,1);

  //Ask the user to input something
  user_input();

  //Now that the inputs have been setup run the command
  run_command("RECT");
}
```

3.9.3 User Input Example

```
selection current_selection;

//This will run a rectangle with the first click at 1,1 and then
//asks the user for the next input
main()
{
  //Setup a rectangle
  var_push(1,1);
  var_push(2,2);
  run_command("RECT");

  //Ask the user to select an object
  current_selection=newselection("OBJECT");
  get_user_selection(current_selection);

  //Now prepare to run the rotate command
  //The order is
  //Selection set
  //Point to rotate around
  //Point to start rotating
  //Point to end rotating
  sel_push(current_selection);
  var_push(0,0);
  var_push(1,0);
  var_push(1,1);
  run_command("ROTATE");
}
```

***

3.9.4 Getting Command Results Example

```
selection current_selection;

//This will run a rectangle with the first click at 1,1 and then
//asks the user for the next input
main()
{
  //Setup a rectangle
  var_push(1,1);
  var_push(2,2);
  run_command("RECT");

  //Get the resulting selection set.  It will contain the rectangle
  current_selection=newselection("OBJECT");
  get_result_selection(current_selection);

  //Now prepare to run the rotate command
  //The order is
  //Selection set
  //Point to rotate around
```

```
  //Point to start rotating
  //Point to end rotating
  sel_push(current_selection);
  var_push(0,0);
  var_push(1,0);
  var_push(1,1);
  run_command("ROTATE");
}
```

3.9.5 Generating Single Value Example

```
selection current_selection;

//This will run a rectangle with the first click at 1,1 and then
//asks the user for the next input
main()
{
  //Setup a rectangle
  var_push(0,0);
  var_push(2,2);
  run_command("RECT");

  //Get the resulting selection set.  It will contain the rectangle
  current_selection=newselection("OBJECT");
  get_result_selection(current_selection);

  //Now prepare to run the rotate command
  //The order is
  //Selection set
  //Point to rotate around
  //The angle to rotate
  sel_push(current_selection);
  var_push(0,0);
  var_push(45);
  run_command("ROTATE");
}
```

3.10 Misc Examples

3.10.1 English or Metric Example

```
main()
{
  if (unit_mode() == 1)
    {
    dialog_box("Metric");
    }
  else
    {
    dialog_box("English");
    }
}
```

3.10.2 Get Language Example

```
main()
{
  debug(get_language());
}
```

3.10.3 Command Line Example

```
main()
{
  double x;
  string name;

  x = 3.14;
  name = "Fred";

  debug(12);
  debug("Hello");
  debug(x);
  debug(name);
  x = 12;
  name = "Hello";
  prompt(name);
  name = "Twelve is " + x;
  prompt(name);
}
```

The output is:

```
12 = 12.0
"Hello" = Hello
x = 3.14
name = 1.0
Hello
Twelve is 12.0
```

3.10.4 Logic Example

Here is a test for all the number logic combinations. The should only prints Works, never Broken.

```
main()
{
  double x, y;

  x = 100;
  y = 0;

  if (x == 100)
    debug("Equal Works");
  else
    debug("Equal Broken!!!");

  if (x == 101)
    debug("Equal Fail Broken!!!");
  else
```

```
  debug("Equal Fail Works");

if (x != 100)
  debug("Not Equal Works");
else
  debug("Not Equal Broken!!!");

if (x != 100)
  debug("Not Equal Fail Broken!!!");
else
  debug("Not Equal Fail Works");

if (x >= 100)
  debug("Greater Than and Equal Works");
else
  debug("Greater Than and Equal Broken!!!");

if (x >=  99)
  debug("Greater Than and Equal Fail Broken!!!");
else
  debug("Greater Than and Equal Fail Works");

if (x <= 100)
  debug("Less Than and Equal Works");
else
  debug("Less Than and Equal Broken!!!");

if (x <=  101)
  debug("Less Than and Equal Fail Broken!!!");
else
  debug("Less Than and Equal Fail Works");

if (x > y)
  debug("Greater Than Works");
else
  debug("Greater Than Broken!!!");

if (y > x)
  debug("Greater Than Fail Broken!!!");
else
  debug("Greater Than Fail Works");

if (x >= y)
  debug("Greater Than or Equals Works");
else
  debug("Greater Than or Equals Broken!!!");

if (y >= x)
  debug("Greater Than or Equals Fail Broken!!!");
else
  debug("Greater Than or Equals Fail Works");

if (y < x)
  debug("Less Than Works");
else
  debug("Less Than Broken!!!");

if (x < y)
  debug("Less Than or Equals Fail Broken!!!");
else
  debug("Less Than or Equals Fail Works");
```

```
  if (y <= x)
    debug("Less Than Works");
  else
    debug("Less Than Broken!!!");

  if (x < y)
    debug("Less Than or Equals Fail Broken!!!");
  else
    debug("Less Than or Equals Fail Works");
}
```

3.10.5 Recursion Example

```
main()
 {
   double x;
   x = Sub(3);
   debug(x);
 }

 Sub(double pass)
 {
 if (pass <= 0)
   return 0;
 debug(pass);
 return Sub(pass-1);
 }
```

The output is

```
pass = 3.0
pass = 2.0
pass = 1.0
x = 6.0
```

3.10.6 Initalizing Colors Example

```
main()
{
  colortype color;

  color = "RED";

  if (color == "RED")
          prompt("String init with == Works");
  else
          prompt("String init with == Broken");

  if (color != "BLUE")
```

```
        prompt("String init with != Works");
 else
        prompt("String init with 1= Broken");

 color = 192;
 if (color == 192)
        prompt("Number init with == Works");
 else
        prompt("Number init with == Broken");

 if (color != 193)
        prompt("Number init with != Works");
 else
        prompt("Number init with 1= Broken");
}
```

```
String init with == Works
String init with != Works
Number init with == Works
Number init with != Works
```

3.10.7 Initalizing Line Style Example

```
main()
{
 linetype line;

 line = "SOLID_LINE";
 line(line);


 if (line == "SOLID_LINE")
        prompt("String init with == Works");
 else
        prompt("String init with == Broken");

 if (line != "DOTTED_LINE")
        prompt("String init with != Works");
 else
        prompt("String init with != Broken");

 line = 1;
 if (line == 1)
        prompt("Number init with == Works");
 else
        prompt("Number init with == Broken");

 if (line != 2)
        prompt("Number init with != Works");
 else
        prompt("Number init with 1= Broken");
}
```

```
String init with == Works
String init with != Works
Number init with == Works
Number init with != Works
```

3.10.8 Initalizing Fill Pattern Example

```
main()
{
  patterntype pattern;

  pattern = "SOLID";

  if (pattern == "SOLID")
        prompt("String init with == Works");
  else
        prompt("String init with == Broken");

  if (pattern != "NONE")
        prompt("String init with != Works");
  else
        prompt("String init with 1= Broken");

  pattern = 1;
  if (pattern == 1)
        prompt("Number init with == Works");
  else
        prompt("Number init with == Broken");

  if (pattern != 2)
        prompt("Number init with != Works");
  else
        prompt("Number init with 1= Broken");
}
```

```
String init with == Works
String init with != Works
Number init with == Works
Number init with != Works
```

4 Command Reference

This is a complete list of all the Macro Language commands.

4.1 acos

The inverse cosine

Syntax

double acos(double value)

Parameters

value: a number between -1 and 1

Return

An angle in radians between pi and -pi

Example

y=acos(x)

See Math Function Example

4.2 addpoint

Adds a point to an object. By default it adds a point at the end of the object on the end of the object list. See Creating_an_object and

Syntax

point addpoint(double x, double y, string type, object obj=LAST_OBJECT, point pnt=LAST_POINT)

Parameters

x: The x coordinate of the point
y: The y coordinate of the point
type: The type of point to add. The valid values are
OPEN - An open point
LINE - A closed line
XARC_START - The start of a curve
XARC_CORNER - The corner point in an object

obj: The object to add the point to. The default is the last object in the list
pnt: The point on obj that will be before the newly added point

Return

The newly created point

Example

```
pnt=addpoint(0,0,"LINE");
addpoint(1,1,"LINE", obj, pnt);
```

4.3 arc_arc_intersect

Finds the intersection between two arcs. It is possible to have up to 4 intersections. A parameter determines which intersection is selected

Syntax

```
double arc_arc_intersect(double arc1_start_x, double arc1_start_y
                         double arc1_corner_x, double arc1_corner_y,
                         double arc1_end_x, double arc1_end_y,
                         double arc2_start_x, double arc2_start_y,
                         double arc2_corner_x, double arc2_corner_y,
                         double arc2_end_x, double arc2_end_y,
                         double which_intersection=0)
```

Parameters

arc1_start_x The x coordination of the start of the first arc
arc1_start_y The y coordination of the start of the first arc
arc1_corner_x The x coordination of the corner of the first arc
arc1_corner_y The y coordination of the corner of the first arc
arc1_end_x The x coordination of the end of the first arc
arc1_end_y The y coordination of the end of the first arc
arc2_start_x The x coordination of the start of the second arc
arc2_start_y The y coordination of the start of the second arc
arc2_corner_x The x coordination of the corner of the second arc
arc2_corner_y The y coordination of the corner of the second arc
arc2_end_x The x coordination of the end of the second arc
arc2_end_y The y coordination of the end of the second arc

which_intersection A value between 1-4 corresponding to the which possible intersection. 1 is the first intersection, 2 is the second..etc. If there isn't an intersection at this location the return value is 0. The default is 1.

Return

0 If there isn't an intersection
1 There is an intersection at this location

The results are stored in getresultx() and getresulty().

The order of the intersections is along the second arc in the direction of start-corner-end.

Example

```
z = arc_arc_intersect(0,1,  1,1,  1,0,  0,0, 1,0, 1,1  ,1)
```

See Arc Arc Intersection Example

4.4 arrow_break

Sets the value of an arrows break point. See Grading_Arrows

Syntax

arrow_break(point arrow, double layer, double x, double y)

Parameters

arrow - The point with the arrow on it
layer - The layer the breakpoint is on
x - The x value of the breakpoint
y - The y value of the breakpoint

Return

none

Example

```
arrow_break(pnt, 3, -3, -4);
```

See Breakpoint Example

4.5 arrow_x

Gets the x value of an arrow point. See Grading_Arrows

Syntax

double arrow_x(point arrow, double layer)

Parameters

arrow: The point the contains the value
layer: The layer value of the arrow to return

Return

The x value

Example

```
x = arrow_x(pnt, 0);
```

See Arrow Example

4.6 arrow_y

Gets the y value of an arrow point. See Grading_Arrows

Syntax

double arrow_y(point arrow, double layer)

Parameters

arrow: The point the contains the value
layer: The layer value of the arrow to return

Return

The y value

Example

```
y = arrow_y(pnt, 0);
```

See Arrow Example

4.7 arc_line_intersect

Finds the intersection of an arc and a line

```
PointClass *Start = ArcObj.addPoint(Params->toDouble(0), Params->toDouble(1), PointClass::
XARC_START);
```

```
        ArcObj.addPoint(Params->toDouble(2), Params->toDouble(3), PointClass::XARC_CORNER);
        ArcObj.addPoint(Params->toDouble(4), Params->toDouble(5), PointClass::CLOSED_LINE);
        PointClass Temp;

        ObjectClass LineObj(ObjectClass::POLY);
        PointClass *StartLine = LineObj.addPoint(Params->toDouble(6), Params->toDouble(7),
PointClass::CLOSED_LINE);
        LineObj.addPoint(Params->toDouble(8), Params->toDouble(9), PointClass::CLOSED_LINE);
```

Syntax

double acos(double arc_start_x, double arc_start_y, double arc_corner_x, double arc_corner_y, double arc_end_x, double arc_end_y, double line_start_x, double line_start_y, double line_end_x, double line_end_y, double which_intersection=0)

Parameters

arc_start_x: The x coordinate of the arc's starting point.
arc_start_y: The y coordinate of the arc's starting point.
arc_corner_x: The x coordinate of the arc's corner point.
arc_corner_y: The y coordinate of the arc's corner point.
arc_end_x: The x coordinate of the arc's end point.
arc_end_y: The y coordinate of the arc's end point.
line_start_x: The x coordinate of the line's start point.
line_start_y: The y coordinate of the line's start point.
line_end_x: The x coordinate of the line's end point.
line_end_y: The y coordinate of the line's end point.
which_intersection: There can be two intersections between the line and an arc. If this value is 1 it will find the second intersection otherwise it will find the first.

Return

0 If there isn't an intersection
1 There is an intersection at this location

The results are stored in getresultx() and getresulty().

Example

z = arc_line_intersect(0,1, 1,1, 1,0, 0,0, 1,1 ,1);

See Arc Line Intersection Example

4.8 add_pict_item

Adds a picture to a style dialog box. See start_pict_dialog, run_pict and get_pict_result.

Syntax

void add_pict_item(string title, string picture)

Parameters

title: The name of the item
picture: The name (including extension) of the picture to be displayed when this radio box is selected.

Return

None

Example

add_pict_item("Item 1","Item1.jpg");

See Creating a style dialog

4.9 add_picture_input

Adds picture and description information to a style measurement box. See start_picture_input and run_picture_input.

Syntax

void add_picture_input(string title, variable var string filename)

Parameters

title: The name of the item
var: A variable that's value is modified by the user. It's value will be used as the default value.
filename: The name (without extension) of the picture (.jpg) and text file (.txt) to be displayed when this edit box is in focus.

Return

None

Example

add_picture_input("X Value", x, "XFocus");

See Creating a measurement dialog

4.10 addarrow

Adds an arrow point to a point. See Grading_Arrows

Syntax

void addarrow(point pnt, object obj, double x, double y, string name="")

Parameters

pnt: The point to add the arrow to
obj: The object that pnt belongs to
x: The X value of the new arrow point
y: The Y value of the new arrow point
name: The arrows name

Return

The x value

Example

addarrow(pnt, obj, 2,3, "Hello");

See Arrow Example

4.11 angle

Find the angle between two points.

Syntax

double angle(double x1, double y1, double x2, double y2)

Parameters

x1: The x value of the first point
y1: The y value of the first point
x2: The x value of the second point
y2: The y value of the second point

Return

An angle in radians.

Example

ang = angle(0,0,1,1)
See Math Function Example

4.12 asin

The inverse sine

Syntax

double asin(double value)

Parameters

value: a number between -1 and 1

Return

An angle in radians between pi and -pi

Example

y=asin(x)

See Math Function Example

4.13 atan

The inverse tangent

Syntax

double atan(double y, double x)

Parameters

y: The vertical value
x: The horizontal value

Return

An angle in radians between pi and -pi

Example

ang =atan(y, x)

See Math Function Example

4.14 block_loaded

Checks if a symbol has been loaded. See Creating_a_Symbol, symbol and is_insertion.

Syntax

double block_loaded(string symbol_name)

Parameters

symbol_name: The name of the block to check if is loaded

Return

0: The block doesn't exist
1: The block does exists

Example

```
if (block_loaded("Tester"))
  {
  ...
  }
```

See Symbol Example

4.15 circlesintersect

Finds the intersection between two circles a and b.

Syntax

```
double circlesintersect(double circle_a_center_x,
                        double circle_a_center_y,
                        double circle_a_radius,
                        double circle_b_center_x,
                        double circle_b_center_y,
```

double circle_b_radius,
double which_intersection)

Parameters

circle_a_center_x: The x coordinate of the center of the first circle (a).
circle_a_center_y: The y coordinate of the center of the first circle (a).
circle_a_radius: The radius of the first circle (a).
circle_b_center_x: The x coordinate of the center of the second circle (b).
circle_b_center_y: The y coordinate of the center of the second circle (b).
circle_b_radius: The radius of the second circle (b).
which_intersection: 0=The first intersection. 1=The second intersection.

Return

The number of intersections

The results are stored in getresultx() and getresulty().

Example

circlesintersect(0, 0, 2, 1,1,2, 1);

See Circle Intersection Example

4.16 check_overlap

Checks if two marker pieces overlap. If they do it moves them apart so they no longer overlap. See Working With Selections.

Syntax

void check_overlap(selection objects)

Parameters

objects: The selection set containing the marker objects

Return

None

Example

check_overlap(current_selection)

4.17 check_reg

Determines if a macro has been registered or not. If it is not registered the macro displays an error and exits. The system for registering macros is set up in Collection Creator.

Syntax

void check_reg(double unused, string author, string macro, string message)

Parameters

unused: Not used
designer: The macro designer. See collection creator documentation
macro: The macros internal name
message: The message displayed on error.

Return

Nothing

Example

check_reg(0,"Leena", "Women's Volume One", "You haven't registered this macro");

4.18 color

Sets the drawing color. This is the default color that objects will be drawn in. See Working With Colors.

Syntax

void color(colortype value)

Parameters

value: The new color

Return

Nothing

Example

color("GREEN");
color(256);

See System Settings Example

4.19 color_layer

Sets the color of a layer. See get_layer_color.

Syntax

void color_layer(double layer, colortype color);

Parameters

layer: The layer
color: The color to set the layer

Return

None

Example

color_layer(1, "RED");

See Cut Ratio Example

4.20 cos

The cosine of an angle

Syntax

double cos(double angle)

Parameters

angle: The angle

Return

The cosine of an angle

Example

y=cos(ang)

See Math Function Example

4.21 debug

Prints out the name of a variable and it's content. This is used for debugging a macro. Also see prompt

Syntax

void debug(variable to_print)

Parameters

to_print: The variable to be printed.

Return

Nothing

Example

```
debug(x);
debug("Hello World");
```

See Command Line Example

4.22 deselect

Deselects a selection set to tell the program your no longer using it. See Working With Selections.

Syntax

void deselect(selection sel)

Parameters

sel: The selection set your done with.

Return

Nothing

Example

deselect(current_selection);

See Group Example

4.23 dialog_box

Displays a message box or a dialog box with up to 10 fields.

Syntax

void dialog_box(string title)
void dialog_box(string title, [string caption, variable var]x10)

The string or caption part can be repeated up to 10 times.

Parameters

caption: The caption to be displayed for the input.
var: A string or double variable. It will be modified by the user.

Return

None, but see var under parameters

Example

dialog_box("Hello World");
dialog_box("Title", "X Value", x, "Y Value", y);
dialog_box("Title", "Your Name", name);

See Dialog Box Example.

4.24 distance

Finds the distance between two points

Syntax

double distance(double x1, double y1, double x2, double y2)

Parameters

x1: The x coordinate of the first point

y1: The y coordinate of the first point
x2: The x coordinate of the second point
y2: The y coordinate of the second point

Return

The distance between the points

Example

distance(0,0,1,1)

See Math Function Example

4.25 erase

Erases or deletes all the objects in a selection set. See Working With Selections.

Syntax

void erase(selection objects)

Parameters

objects: The objects to erase

Return

Nothing

Example

erase(current_selection);

See Group Example

4.26 fill_picture

This is a prototype function. It sets the fill style to a background picture

Syntax

void fill_picture(object obj, string name, double stretch, double tile_x, double tile_y, double x_offset, double y_offset)

Parameters

obj: The object to set the fill picture on
name: The name of the image
stretch: Is the image to be stretched
tile_x: Is the image repeated horizontally
tile_y: Is the image repeated vertically
x_offset: Offset in the x direction
y_offset: Offset in the y direction

Return

Nothing

Example

4.27 first_point

Returns the first point in an object. It can be used to loop through all the points in an object. See next_point, last_point and Looping Through an Object Example.

Syntax

point first_point(object obj)

Parameters

obj: The object that has the first point

Return

The first point in obj

Example

pnt = first_point(obj);

See First Point Example

4.28 first_obj

Returns the first point in the drawing. This will appear on the bottom with all the other objects stacked on top. It is used for looping through all the objects in the drawing. See next_obj, last_obj

Syntax

object first_obj()

Parameters

none

Return

The first object in the drawing.

Example

obj = first_obj();

See Looping through the drawing

4.29 get_angle

Returns the angle an text object or symbol is at.

Syntax

double get_angle(object obj);

Parameters

obj: The object

Return

The angle from the x axis in radians.

Example

angle = get_angle(obj);

See Creating Text Example

4.30 get_arrow_name

Returns the name of an arrow. See Grading_Arrows

Syntax

void get_arrow_name(point pnt)

Parameters

pnt: The point that contains an arrow

Return

The name of the arrow

Example

name = get_arrow_name(pnt);

See Arrow Example

4.31 get_bundle

Returns the bundle number for an object. See Markers

Syntax

double get_bundle(object obj);

Parameters

obj: The object

Return

The bundle number.

Example

get_bundle(obj);

See Create Marker Example

4.32 get_color

Returns the default color for drawing objects. See Working With Colors.

Syntax

colortype get_color()

Parameters

None

Return

The default drawing color

Example

user_color = get_color()

See System Settings Example

4.33 get_font

Returns the font name in a text object

Syntax

string get_font(object obj);

Parameters

obj: The object

Return

The object's font name

Example

font = get_font(obj);

See Creating Text Example

4.34 get_group_name

Gets the name of a group. See name_group and group.

Syntax

string get_group_name(object obj)

Parameters

obj: The object that is a member of the group.

Return

The name of the group. If obj isn't part of a group the default is "".

Example

get_group_name(current_object);

See Group Example

4.35 get_height

Returns the height an text object or symbol is at.

Syntax

double get_height(object obj);

Parameters

obj: The object

Return

The width of the text or symbol in inches

Example

size = get_height(obj);

See Creating Text Example

4.36 get_language

Returns a number that corresponds to what language patternmaker is running in.

Syntax

double get_language()

Parameters

None

Return

0=english, 1=finish, 2=dutch, 3=german, 4=spanish, 5=french

Example

get_language()

See Get Language Example

4.37 get_layer_name

Returns the name of a layer. See name_layer.

Syntax

string get_layer_name(double layer);

Parameters

layer: The layer to get the name of

Return

The layers name.

Example

get_layer_name(0);

See Cut Ratio Example

4.38 getlayer

Returns the layer of an object. See layer

Syntax

double getlayer(object obj)

Parameters

obj: The object to get the layer of.

Return

The layer number of layer the object is on.

Example

value = getlayer(obj);

See Layer Example

4.39 get_layer_color

Returns the color of a layer. See color_layer

Syntax

colortype get_layer_color(double layer);

Parameters

layer: The layer to get the color of

Return

The layers color.

Example

get_layer_color(0);

See Cut Ratio Example

4.40 get_line

Returns the default line type for drawing objects. See Working With Lines.

Syntax

linetype get_line()

Parameters

None

Return

The default drawing line type

Example

user_line = get_line()

See System Settings Example

4.41 get_marker_fabric

Gets the marker fabric. See Markers

Syntax

string get_marker_fabric()

Parameters

None.

Return

The value of the marker fabric.

Example

get_marker_fabric()

See Marker Area Example

4.42 get_marker_width

Returns the marker width. See Markers

Syntax

double get_marker_width()

Parameters

None.

Return

The marker width

Example

get_marker_width();

See Marker Area Example

4.43 get_notch_type

Gets the notch type. See notch_type and get_notch_dir.

Syntax

string get_notch_type(pnt);

Parameters

pnt: The point with a notch

Return

The side the notch faces. It can be "NONE", "NOTCH", "DBL_NOTCH", "TAB", "DBL_TAB" or "BTN".

Example

notch = get_notch_dir(pnt);

See Notch Example

4.44 get_obj_color

Sets the color of an object. See set_obj_color and Working With Colors.

Syntax

colortype get_obj_color(obj);

Parameters

obj: The object

Return

The color of the object

Example

get_obj_color(obj);

See Object Color Example

4.45 get_obj_line

Sets the line type of an object. See set_obj_line and Working With Lines.

Syntax

linetype get_obj_line(object obj);

Parameters

obj: The object

Return

The line type of the object

Example

get_obj_line(obj);

See Object Line Type Example

4.46 get_object_name

Gets the name of an object. See name_object

Syntax

string get_object_name(object obj)

Parameters

obj: The object to get the name of.

Return

The objects name

Example

name = get_object_name(obj);

See Name Object Example

4.47 get_layer_cutratio

Gets the cut ratio for a layer. See Markers

Syntax

double get_layer_cutratio(double layer)

Parameters

layer: The layer to set

Return

The cut ratio

Example

get_layer_cutratio(2)

See Cut Ratio Example

4.48 get_markerarea

Determines if the marker area is on. See marker_area and Markers

Syntax

double marker_area()

Parameters

None

Return

0=Marker area off
1=Marker area on

Example

```
if (get_markerarea())
  {
  ...
  }
```

See Marker Area Example

4.49 get_marker_origin

This has been obsoleted and does nothing.

Syntax

get_marker_origin();

Parameters

None.

Return

None.

Example

4.50 get_marker_style

Returns the marker style. See Markers

Syntax

string get_marker_style();

Parameters

None

Return

The marker style.

Example

get_marker_style();

See Marker Area Example

4.51 get_notch_dir

Gets the direction the notch faces. See notch_type and get_notch_type.

Syntax

double get_notch_dir(point pnt);

Parameters

pnt: The point with a notch

Return

The side the notch faces. It can be either 0 or 1

Example

notch = get_notch_dir(pnt);

See Notch Example

4.52 get_obj_pattern

Sets the color of an object. See set_obj_pattern and Working With Lines.

Syntax

patterntype get_obj_pattern(object obj);

Parameters

obj: The object

Return

The objects pattern type

Example

get_obj_pattern(obj);

See Object Pattern Example

4.53 get_plaid_point

Creates a plaid point. Currently they are disabled in PatternMaker. See Markers

Syntax

void get_plaid_point(object obj);

Parameters

obj: The object the plaid point belongs to

Return

The coordinates are returned in getresultx() and getresulty().

Example

get_plaid_point(obj);

See Plaid Point Example

4.54 get_point_type

Returns a points type.

Syntax

string get_point_type(pnt);

Parameters

pnt: The point

Return

The point type. The following values are:

LINE
XARC_START
XARC_CORNER
OPEN

Example

get_point_type(pnt);

See Get Point Type Example

4.55 get_point_name

Sets the name of a point. See name_point and Creating_a_point

Syntax

void name_point(point pnt, string name)

Parameters

pnt: The point to name
name: The name of the point

Return

None

Example

name_point(pnt,"Hello");

See Naming the last point in an object

4.56 get_pict_result

Gets the result of a a style dialog box. See add_pict_item, start_pict_dialog and run_pict.

Syntax

double get_pict_result()

Parameters

None

Return

The position of the checked radio button. 0=first button, 1=second button

Example

result1=get_pict_result();

See Creating a style dialog

4.57 get_pattern

Returns the default pattern for drawing objects. See Working With Lines.

Syntax

patterntype get_pattern()

Parameters

None

Return

The default drawing fill pattern.

Example

user_pattern = get_pattern()

See System Settings Example

4.58 getresultx

Returns the x value of a function that returns a point or two values. See getresulty

Syntax

double getresultx();

Parameters

None:

Return

The x value calculated by another function.

Example

x = getresultx();

4.59 getresulty

Returns the y value of a function that returns a point or two values. See getresulty.

Syntax

double getresulty();

Parameters

None:

Return

The y value calculated by another function.

Example

y = getresulty();

4.60 get_result_selection

Gets the objects modified by the last command run by the user. See Running_PatternMaker_Commands

Syntax

void get_result_selection(selection sel);

Parameters

sel: The selection set to copy the changed items into

Return

None:

Example

get_result_selection(current_selection);

See Getting Command Results Example

4.61 get_sel_obj

Gets an object from a selection set. It needs to be used with loop_sel. See Working With Selections.

Syntax

object get_sel_obj();

Parameters

sel: The selection set

Return

The current object set by loop_sel

Example

```
current_selection=newselection("OBJECT");
select(current_selection, current_object);
...

while (loop_sel(current_selection))
  {
  current_object = get_sel_obj();
  ...
  }
```

See Object Selection Set Example, Point Selection Set Example

4.62 get_sel_point

Gets an point from a selection set. It needs to be used with loop_sel. The object the point belongs to can be found with get_sel_obj. See select, get_user_selection, deselect and newselection.

Syntax

point get_sel_point();

Parameters

sel: The selection set

Return

The current point set by loop_sel

Example

```
current_selection=newselection("OBJECT");
select(current_selection, current_point);
…

while (loop_sel(current_selection))
 {
 current_object = get_sel_obj();
 current_point = get_sel_point();
 …
 }
```

See Object Selection Set Example, Point Selection Set Example

4.63 get_user_selection

Asks the user to select points or objects that are added to a selection set. The alternative is select. See Working With Selections..

Syntax

void get_user_selection(selection sel);

Parameters

sel: The selection the items are added to

Return

None.

Example

get_user_selection(current_selection);

See Get User Selection Example, User Input Example and Running a Command Example

4.64 get_symbol_name

Gets the name of the symbol.

Syntax

string get_symbol_name(object obj)

Parameters

obj: An insertion object

Return

The insertions name

Example

name = get_symbol_name(current_object);

See Symbol Example

4.65 get_text

Returns the displayed text in a text object

Syntax

string get_text(object obj);

Parameters

obj: The object to find the text of

Return

The object's displayed text

Example

font = get_font(obj);

See Creating Text Example

4.66 get_width

Returns the width an text object or symbol is at.

Syntax

double get_width(object obj);

Parameters

obj: The object

Return

The width of the text or symbol in inches

Example

size = get_width(obj);

See Creating Text Example

4.67 grid

Turns on or off the grid

Syntax

void grid(string state);

Parameters

state: It can be "ON" or "OFF"

Return

None.

Example

grid("ON");
grid("OFF");

4.68 group

Forms a group out of several objects. See name_group and get_group_name.

Syntax

void group(selection objects)

Parameters

objects: The objects to group together

Return

Nothing

Example

group(current_selection);

See Group Example

4.69 is_break

Checks if the layer on an arrow point is a break point. See Grading_Arrows

Syntax

is_break(point arrow, double layer)

Parameters

arrow - The point with the arrow on it
layer - The layer the breakpoint is on

Return

1=the point has breakpoint on this layer
0=the point doesn't have a breakpoint on this layer

Example

value = is_break(pnt,3);

See Breakpoint Example

4.70 integer

Converts a double to an integer by truncating the values after the decimal point. This can be used in math functions.

Syntax

double integer(double value)

Parameters

value: The number to convert.

Return

Value with nothing after the decimal.

Example

integer(4.5);

4.71 intersect

Finds the intersection between two lines segments.

Syntax

double intersect(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4, double endpoint_intersect)

Parameters

x1: The x coordinate of the start of the first line
y1: The y coordinate of the start of the first line
x2: The x coordinate of the end of the first line
y2: The y coordinate of the end of the first line
x3: The x coordinate of the start of the second line

y3: The y coordinate of the start of the second line
x4: The x coordinate of the end of the second line
y4: The y coordinate of the end of the second line
endpoint_intersect:

Return

1=The line segments intersect
0=The lines are parallel or the intersection happens off the line segments.

The intersection points are found in getresultx() and getresulty();

Example

intersect(0, 0, 24, 24, 5, 17, 15, 9, 0);

See Line Line Intersection Example

4.72 insert_point

Inserts a point into an object. See addpoint

Syntax

point = insert_point(object obj, point position);

Parameters

obj: The object the point belongs to
position: The point we are going to add a point before

Return

The newly created point that is before position.

Example

mid = insert_point(obj, end);

See Point Type Example

4.73 interceptline

Determines the intersection of a vertical line with a circle.

Syntax

double interceptline(double x, double line, double radius)

Parameters

x: x coordinate of the point
line: x coordinate of a line if it is vertical
radius: The radius of the circle

Return

sqrt (radius*radius - (line - x)*(line - x))

Example

```
interceptline(0,1,2);
```

See Intercept Line Example

4.74 is_corner

Determines if a point is a corner point. See Creating_an_object, is_open, is_line, is_xarc, set_corner, set_xarc, set_line, set_open, get_point_type and set_point_type.

Syntax

double is_corner(point pnt);

Parameters

pnt: The point to test

Return

0: pnt is not a corner point.
1: pnt is a corner point.

Example

```
if (is_corner(current_point))
  {
  ...
  }
```

See Point Type Example

4.75 is_clockwise

Determines if the points in an object are clockwise or counter clockwise.

Syntax

double is_clockwise(object obj);

Parameters

obj: The object to check the direction of.

Return

0: Is clockwise.
1: Not clockwise.

Example

```
if (is_clockwise(current_object))
  {
  ...
  }
```

See Is Clockwise Example

4.76 is_dim

Checks if an object is an insertion. See is_poly and is_text.

Syntax

double is_dim(object obj);

Parameters

object: The object to check

Return

0: Not a dimension line
1: Is a dimension line

Example

```
if (is_dim(current_object))
  {
  ...
  }
```

See Creating a Dimension Line Example

4.77 is_function

Test if a function exists. This is useful when using a new function that might not be supported in an older version. See version.

Syntax

double is_function(string function)

Parameters

function: The name of the function to see if exists

Return

0: The function doesn't exist
1: The function exists

Example

```
if (is_function("refresh_drawing"))
   {
   refresh_drawing();
   }
```

4.78 is_insertion

Checks if an object is an insertion. See block_loaded, symbol, is_dim, is_marker, is_open, is_poly, is_text.

Syntax

double is_insertion(object obj);

Parameters

object: The object to check if is an insertion.

Return

0: Not an insertion
1: Is an insertion.

Example

```
if (is_insertion(current_object))
  {
  ...
  }
```

See Symbol Example

4.79 is_marker

Tests if an object is a marker piece or not. See Markers

Syntax

double is_marker(object obj);

Parameters

obj: The object to test

Return

0=The object isn't a marker piece.
1=The object is a marker piece.

Example

is_marker(obj);

See Create Marker Example

4.80 is_open

Determines if a point is an open point. See Creating_an_object, is_corner, is_line, is_xarc, set_corner, set_xarc, set_line, set_open, get_point_type and set_point_type.

Syntax

double is_open(point pnt);

Parameters

pnt: The point to test

Return

0: pnt is not a open .
1: pnt is a open.

Example

```
if (is_open(current_point))
  {
  …
  }
```

See Point Type Example

4.81 is_poly

Checks if an object is an insertion. See is_dim and is_text.

Syntax

double is_poly(object obj);

Parameters

object: The object to check

Return

0: Not a polygone
1: Is a polygone

Example

```
if (is_poly(current_object))
  {
  ...
  }
```

See Creating a Dimension Line Example

4.82 is_line

Determines if a point is a line point. See Creating_an_object, is_corner, is_open, is_xarc, set_corner, set_xarc, set_line, set_open, get_point_type and set_point_type.

Syntax

double is_line(point pnt);

Parameters

pnt: The point to test

Return

0: pnt is not a corner point.
1: pnt is a corner point.

Example

```
if (is_line(current_point))
  {
```

```
...
}
```

See Point Type Example

4.83 is_text

Checks if an object is an insertion. See is_dim and is_poly.

Syntax

double is_text(object obj);

Parameters

object: The object to check

Return

0: Not a text object
1: Is a text object

Example

```
if (is_text(current_object))
 {
 ...
 }
```

See Creating Text Example

4.84 is_xarc

Determines if a point is an arc start point. See Creating_an_object, is_corner, is_open, is_line, set_corner, set_xarc, set_line, set_open, get_point_type and set_point_type.

Syntax

double is_xarc(point pnt);

Parameters

pnt: The point to test

Return

0: pnt is not an arc point.
1: pnt is an arc point.

Example

```
if (is_xarc(current_point))
 {
 ...
 }
```

See Point Type Example

4.85 layer

Returns the layer of an object. See getlayer

Syntax

void layer(object obj, double new_layer)

Parameters

obj: The object to set the layer
new_layer: The new layer for the object.

Return

None

Example

layer(obj,3);

See Layer Example

4.86 line

Sets the default line type for drawing objects. See Working With Lines.

Syntax

```
line(linetype value);
```

Parameters

value: The new linetype

Return

None:

Example

```
line("DOTTED_LINE");
```

See System Settings Example

4.87 last_obj

Returns the last object in the drawing. This will appear on top of all other objects. See first_obj and next_obj.

Syntax

object last_obj()

Parameters

None

Return

The last object in the drawing.

Example

obj = last_obj();

See Last Object Example

4.88 last_point

Returns the last point in an object. See first_point and next_point

Syntax

point last_point(object obj)

Parameters

obj: The object to find the last point of.

Return

The last point in the object.

Example

pnt = last_point(obj);

See Naming the last point in an object

4.89 lock_layer

Locks a layer so it can't be edited.

Syntax

void lock_layer(double layer);

Parameters

layer: The layer to lock.

Return

None

Example

lock_layer(5);

See Cut Ratio Example

4.90 loop_sel

Loops through all the objects or points in a selection set. Use get_sel_obj or get_sel_point to get the object from the selection set. Use select or get_user_selection to add objects or points to the set. See Working With Selections.

Syntax

double loop_sel(selection sel)

Parameters

sel: The selection set

Return

0=Nothing left in the set

1=There are more objects.

The current object in set is stored in get_sel_obj.
The current point in the set is stored in get_sel_point

Example

```
while (loop_sel(current_selection))
 {
 ...
 }
```

See Object Selection Set Example, Point Selection Set Example

4.91 marker_area

Turns on or off the marker area. See get_markerarea and Markers

Syntax

marker_area(string state)

Parameters

state: "ON"=show marker area. "OFF"=hide marker area.

Return

None.

Example

```
marker_area("ON");
```

See Marker Area Example

4.92 marker_fabric

Sets the marker fabric. See Markers

Syntax

marker_fabric(string fabric);

Parameters

fabric: The new fabric

Return

None.

Example

marker_fabric("Twill");

See Marker Area Example

4.93 marker_origin

This has been obsoleted and does nothing.

Syntax

marker_origin(double x, double y);

Parameters

x: Not used
y: Not used

Return

None.

Example

4.94 marker_notes

Sets a marker note.

Syntax

marker_notes(double position, string value);

Parameters

position: The position of the marker note. It can be 0,1 and 2.
value: The new value of this note

Return

None.

Example

marker_notes(0, "Test");

4.95 marker_piece

Creates a marker piece out of an object. See Markers

Syntax

marker_piece(object obj, double bundle)

Parameters

obj: The object to create a marker from
bundle: The number of pieces the curratio command creates.

Return

None.

Example

marker_piece(obj, 2);

See Create Marker Example

4.96 measure_table

Displays a measurement table dialog box asking the user to select a table that matches the master measurement table code that is passed it. If the user selects a table the variables that match the table will be read in from the table.

Syntax

double measure_table(string id)

Parameters

id: The master measurement table code.

Return

0=The user didn't select a table
1=The user selected a table.

Variables that match the table will be read in.

Example

```
if (!measure_table("MyWomensMeasurements"))
 {
 //display a measurement dialog
 ...
 }
```

4.97 marker_style

Sets the marker style. See Markers

Syntax

marker_style(string style);

Parameters

style: The new style

Return

None.

Example

marker_style("Dress");

See Marker Area Example

4.98 marker_width

Sets the marker width. See Markers

Syntax

void marker_width(double width);

Parameters

width: The new width of the marker area

Return

None.

Example

marker_width(36);

See Marker Area Example

4.99 macgen_write

Used so PatternMaker can communicate with MacroGenerator.

Syntax

void macgen_write(string data);

Parameters

data: The value to pass

Return

None.

Example

macgen_write("Some data");

4.100 name_group

Sets the name of a group. See get_group_name and group.

Syntax

void name_group(selection objects)

Parameters

objects: The objects that make the group.

Return

None

Example

name_group(current_selection);

See Group Example

4.101 name_object

Sets the name of a group. See get_object_name and Creating_an_object.

Syntax

void name_object(obj, name);

Parameters

obj: The object to name
name: The name

Return

None

Example

name_object(obj, "Bodice");

See Name Object Example

4.102 name_layer

Returns the name of a layer. See name_layer.

Syntax

string get_layer_name(double layer);

Parameters

layer: The layer to get the name of

Return

The layers name.

Example

get_layer_name(0);

See Cut Ratio Example

4.103 name_point

Sets the name of a point. See get_point_name and Creating_a_point

Syntax

string get_point_name(point pnt)

Parameters

pnt: The point to get the name of.

Return

The points name,

Example

name = get_point_name(obj)

See Naming the last point in an object

4.104 newobject

Adds an object to the end of the drawing. The object can be a symbol, a dimension line, text or a poly object. Each has it own syntax. See Creating_an_object

Syntax

The only required parameter is the type of object. All other parameters can be left out.

For a polygone.
object newobject("poly", colortype color=drawing_color, patterntype pattern=drawing_fill_type, linetype line=drawing_line_type, double layer=active_layer, double line_width=drawing_line_width)

For text.
object newobject("text", double angle=0, double width=1, double height=1, string font=drawing_font, colortype color=drawing_color, double layer=active_layer)

For a symbol:
object newobject("symbol", double angle=0, double width=1, double height=1, double layer=active_layer)

For a dimension line:
object newobject("dim", colortype color=drawing_color, string dim_type="", double layer=active_layer)

Parameters

color: The objects color
pattern: The objects fill pattern
line: The objects line type
layer: The objects layer
line_width: The objects line width
angle: The angle the text or symbol is oriented at from the x axis in radians.
width: The width of the text or symbol in inches
height: This must match width
font: The name of the font to use
dim_type: not used

Return

The newly created object

Example

For a polygone.

```
newobject("POLY");
current_object=newobject("POLY","USELAYERCOLOR","NONE","SOLID_LINE",0,1);
```

See Layer Example and Group Example

For text.

```
obj=newobject("TEXT","Hello",0.785398,1,1,"PatternMaker");
```

See Creating Text Example

For a symbol:

```
current_object=newobject("SYMBOL","Tester",0,1,1,"USELAYERCOLOR","NONE","SOLID_LINE",0);
```

See Creating_a_Symbol

For a dimension line:

obj=newobject("DIM",-1,"A","SOLID_LINE",0);

See Creating a Dimension Line Example

4.105 next_obj

Returns the first point in the drawing. This will appear on the bottom with all the other objects stacked on top. It is used for looping through all the objects in the drawing. See first_obj and last_obj

Syntax

object first_obj()

Parameters

none

Return

The last object in the drawing.

Example

obj = last_obj();

Looping through the drawing,

4.106 next_point

Returns the point after the current point or 0 if no point exists. See first_point, last_point, prev_point and Looping Through an Object Example.

Syntax

void next_point(object obj, point pnt)

Parameters

obj: The object that contains pnt
pnt: The next point in obj that follows pnt. If this doesn't exist or is the last point it will return 0

Return

The next point in the object

Example

pnt = next_point(obj, pnt);

4.107 newselection

Creates a selection set. There are two types point sets and object sets. See Working With Selections.

Syntax

selection newselection(string type)

Parameters

type: The selection set type. It can be "OBJECT" or "POINT"

Return

The created selection set

Example

current_selection=newselection("OBJECT");
current_selection=newselection("POINT");

See Object Selection Set Example, Point Selection Set Example

4.108 normalizeangle

Makes sure an angle is between 0 and two pi to make it more human readable.

Syntax

double normalizeangle(double ang)

Parameters

ang: The angle in radians

Return

The same angle between 0 and two pi.

Example

ang = normalizeangle(ang);

***

4.109 notch_type

Creates a notch on a point. See get_notch_dir, get_notch_type and Creating_a_point.

Syntax

void notch_type(point pnt,string type, double direction);

Parameters

pnt: The point to attach the notch to
type: The notch type. It can be "NONE", "NOTCH", "DBL_NOTCH", "TAB", "DBL_TAB" or "BTN"
direction: Which face should the notch be on.

Return

None

Example

notch_type(pnt,"NOTCH", 1);

See Notch Example

4.110 origin

Sets the screen origin. This allows the position of the screen to be stored in a save file.

Syntax

void origin(double x, double y);

Parameters

x: The x coordinate.
y: The y coordinate.

Return

None.

Example

origin(0,0);

4.111 origin_arrange

Moves a group of objects so they are aligned to the origin (0,0).

Syntax

void origin_arrange(selection objects);

Parameters

objects: The objects to move.

Return

None.

Example

origin_arrange(objs);

4.112 pattern

Sets the default pattern type for drawing objects. See Working With Lines.

Syntax

```
pattern(patterntype value);
```

Parameters

value: The new pattern type

Return

None:

Example

```
pattern("SOLID");
```

See System Settings Example

4.113 place_corner

This is used when cutting an arc in two. You pass it the original arc and the beginning and ending of the new arc. It returns the corner point that best matches the original arc.

Syntax

place_corner(double orig_arc_start_x, double orig_arc_start_y,
double orig_arc_corner_x, double orig_arc_corner_y,
double orig_arc_end_x, double orig_arc_end_y,
double new_arc_start_x, double new_arc_start_y,
double new_arc_end_x, double new_arc_end_y)

Parameters

orig_arc_start_x: The x coordinate of the start point of the original arc
orig_arc_start_y: The y coordinate of the start point of the original arc
orig_arc_corner_x: The x coordinate of the corner point of the original arc
orig_arc_corner_y: The y coordinate of the corner point of the original arc
orig_arc_end_x: The x coordinate of the end point of the original arc
orig_arc_end_y: The y coordinate of the end point of the original arc
new_arc_start_x: The x coordinate of the start of the new arc
new_arc_start_y: The y coordinate of the start of the new arc
new_arc_end_x: The x coordinate of the end of the new arc
new_arc_end_y: The y coordinate of the end of the new arc

Return

The created point is stored in getresultx() and getresulty()

Example

place_corner(0, 10, 3, 5, 10, 0, 0,10, 10,0);

Place Corner Example

4.114 plaid_point

Creates a plaid point. Currently they are disabled in PatternMaker. See Markers

Syntax

void plaid_point(object obj, double x, double y , double is_active);

Parameters

obj: The object to attach the plaid point to
x: The x coordinate of the plaid point
y: The y coordinate of the plaid point
is_active: 0=inactive. 1=active

Return

None.

Example

plaid_point(obj,5,15 , 1);

See Plaid Point Example

4.115 pointx

Returns the x value of a point. See pointy and Creating_a_point

Syntax

double pointx(point pnt)

Parameters

pnt: The point

Return

The x value of the point

Example

x=pointx(pnt);

See Getting a Points Value Example

4.116 pointy

Returns the x value of a point. See pointx and Creating_a_point

Syntax

double pointy(point pnt)

Parameters

pnt: The point

Return

The y value of the point

Example

y=pointx(pnt);

See Getting a Points Value Example

4.117 polar

Finds the polar coordinates from a cartesian point.

Syntax

void polar(double x, double y, double angle, double distance)

Parameters

x: The x coordinate to go from
y: The y coordinate to go from
angle: The angle to go away from x,y
distance: The distance from x,y in angle direction.

Return

Nothing. The results are stored in getresultx() and getresulty();

Example

polar(0, 0, 3.14/2, 5);

See Polar Example

4.118 pop_up

This creates a box with several radio buttons. This has been replaced. See Creating a style dialog

Syntax

double pop_up(string option1, string option2...)

Parameters

There can be any number of options

option1: The text for the first radio button
option2: The text for the second radio button
...

Return

0=Cancelled
1=option1 was selected
2=option2 was selected
...

Example

result = pop_up("One x", "Two", "Three");

See Creating a radio box

4.119 printer_area

Turns on or off the print area.

Syntax

void printer_area(string state);

Parameters

state: It can be "ON" or "OFF"

Return

None.

Example

printer_area("ON");

4.120 printer_origin

Sets the lower left corner of the printer area.

Syntax

void printer_origin(double x, double y);

Parameters

x: The x value of the printer origin.
y: The y value of the printer origin.

Return

None.

Example

printer_origin(0,0);

4.121 prev_point

The point before the passed point on the passed object. See first_point, last_point. and next_point

Syntax

```
point prev_point(object obj, point pnt);
```

Parameters

obj: The object that contains pnt
pnt: The previous point in obj that follows pnt. If this doesn't exist or is the first point it will return 0

Return

The point before this point.

Example

```
pnt = prev_point(obj, pnt);
```

See Previous Point Example

4.122 prompt

Prints a message in the command line. See debug and prompt_point.

Syntax

void prompt(string to_print)

Parameters

to_print: The information to print to the command line

Return

Nothing

Example

prompt("Hello World");

See Command Line Example

4.123 prompt_point

Prints a message in the command line then waits for the user to click on a point. See debug and prompt.

Syntax

void prompt_point(string to_print)

Parameters

to_print: The information to print to the command line

Return

The clicked on points coordinates are stored in getresultx() and getresulty().

Example

```
prompt_point("Select apex of Angle");
x = getresultx();
y = getresulty();
```

See Command Line Example

4.124 remove_point

The point before the passed point on the passed object.

Syntax

void remove_point(object obj, point pnt);

Parameters

obj: The object that contains pnt
pnt: The point to remove

Return

None.

Example

```
remove_point(obj, pnt);
```

See Remove Point Example

4.125 refresh_drawing

Redraws the drawing. This is used when running PatternMaker commands.

Syntax

void refresh_drawing();

Parameters

None.

Return

None.

Example

refresh_drawing()

4.126 run_command

Runs a PatternMaker Command. See Running_PatternMaker_Commands

Syntax

void run_command(string command);

Parameters

command: The name of the command to run.

Return

None.

Example

run_command("ERASE");

See Running a Command Example

4.127 run_pict

Displays a style dialog box so the user can make a selection. See add_pict_item, start_pict_dialog and get_pict_result.

Syntax

double run_pict(string first_button, string second_button="")

Parameters

first_button: The caption of the first button
second_button: The caption of the second button. If this isn't include the dialog box will have only one button

Return

0: The first button was clicked.
1: The second button was clicked.

Example

run_pict("Done");
run_pict("Done", "Back");

See Creating a style dialog

4.128 run_picture_input

Starts a measurement dialog. See add_picture_input and run_picture_input.

Syntax

void run_picture_input();

Parameters

None

Return

None

Example

run_picture_input();

See Creating a measurement dialog

4.129 scale

Sets the zoom scale. When PatternMaker is started it is set to 14.3

Syntax

void scale(double value)

Parameters

value: The new scale

Return

None.

Example

scale(13);
[****]

4.130 select

Adds a point or an object to a selection set. The other was to add items to a selection set is get_user_selection. See Working With Selections.

Syntax

void select(selection sel, object obj)
void select(selection sel, point pnt)

Parameters

sel: The selection set to add an item to
obj: The object to add
pnt: The point to add

Return

None.

Example

select(current_selection, current_object);
select(current_selection, current_point);

See Object Selection Set Example, Point Selection Set Example

4.131 set_angle

Sets the angle of a text object or symbol.

Syntax

void set_angle(object obj, double angle);

Parameters

obj: The object
angle: The angle from the axis in radians

Return

None.

Example

size = set_width(obj,3.14);

See Creating Text Example

4.132 set_corner

Makes a point a corner point. See Creating_an_object, is_corner, is_open, is_line, is_xarc, set_xarc, set_line, set_open, get_point_type and set_point_type.

Syntax

void set_corner(point pnt);

Parameters

pnt: The point to change

Return

None:

Example

set_corner(pnt);

See Point Type Example

4.133 set_font

Sets the font in a text object

Syntax

void set_font(object obj, string font);

Parameters

obj: The object
font: The new font

Return

None.

Example

set_font(obj,"Big");

See Creating Text Example

4.134 set_height

Sets the size in a text object

Syntax

void set_height(object obj, double height);

Parameters

obj: The object
height: The size of the object in inches

Return

None.

Example

size = set_height(obj,1);

See Creating Text Example

4.135 set_layer

Sets the values of a layer

Syntax

set_layer(double layer, string on_off, string layer_name, colortype color)

Parameters

layer: The layer.
on_off: Turns the layer on or off. Can be "ON" or "OFF".

layer_name: The name of the layer.
color: The color of the layer.

Return

None

Example

set_layer(0,"ON","My Layer","WHITE");

See Cut Ratio Example

4.136 set_layer_cutratio

Sets the cut ratio for a layer. See Markers

Syntax

void set_layer_cutratio(double layer, double cut_ratio);

Parameters

layer: The layer to set
cut_ratio: The cut ratio

Return

None

Example

set_layer_cutratio(2, 5);

See Cut Ratio Example

4.137 set_line

Makes a point a line point. See Creating_an_object, is_corner, is_open, is_line, is_xarc, set_corner, set_xarc, set_open, get_point_type and set_point_type.

Syntax

void set_line(point pnt);

Parameters

pnt: The point to change

Return

None:

Example

set_line(pnt);

See Point Type Example

4.138 set_marker

Makes an object a marker or converts it from a marker to a polygon.

Syntax

void set_marker(object obj, double value);

Parameters

obj: The object to change
value: 1=Make the object a marker object. 0=Make it a polygon.

Return

None:

Example

set_marker(obj, 1);

4.139 set_obj_color

Sets the color of an object. See get_obj_color and Working With Colors.

Syntax

void set_obj_color(object obj, colortype color);

Parameters

obj: The object to change the color of
color: The color to set the object to

Return

None

Example

set_obj_color(obj, "RED");

See Object Color Example

4.140 set_obj_line

Sets the line type of an object. See get_obj_line and Working With Lines.

Syntax

void set_obj_line(object obj, linetype line);

Parameters

obj: The object
line: The linetype to give obj

Return

None.

Example

set_obj_line(obj, "DOTTED_LINE");

See Object Line Type Example

4.141 set_obj_pattern

Sets the color of an object. See get_obj_pattern and Working With Lines.

Syntax

void set_obj_pattern(object obj, patterntype pattern);

Parameters

obj: The object to modify
pattern: The fill pattern to use

Return

None

Example

set_obj_pattern(obj, "SOLID");

See Object Pattern Example

4.142 set_open

Makes a point an open point. See Creating_an_object, is_corner, is_open, is_line, is_xarc, set_corner, set_xarc, set_line, get_point_type and set_point_type.

Syntax

void set_open(point pnt);

Parameters

pnt: The point to change

Return

None:

Example

set_open(pnt);

See Point Type Example

4.143 sel_push

Pushes a selection set onto the stack for use with run_command. See Running_PatternMaker_Commands

Syntax

void sel_push(selection sel);

Parameters

sel: The selection set to put on the stack

Return

None.

Example

sel_push(current_selection);

See Running a Command Example and User Input Example

4.144 set_point_type

Changes a points type. See Creating_an_object, is_corner, is_open, is_line, is_xarc, set_corner, set_xarc, set_line, set_open and get_point_type.

Syntax

void set_point_type(point pnt, string type);

Parameters

pnt: The point to change
type: The point to change to. The options are "XARC_START", "XARC_CORNER", "LNE" and "OPEN".

Return

None:

Example

```
set_point_type(pnt, "LINE");
```

See Point Type Example

4.145 set_symbol_name

Sets the name of a insertions symbol

Syntax

void set_symbol_name(object obj, string name);

Parameters

obj: An insertion object
name: The name of the symbol to use

Return

None.

Example

set_symbol_name(current_object, "Bob");

See Symbol Example

4.146 set_text

Sets the displayed text in a text object

Syntax

void set_text(object obj, string text);

Parameters

obj: The object
text: The text to display

Return

None.

Example

set_text(obj,"Hello World");

See Creating Text Example

4.147 set_width

Sets the size in a text object

Syntax

void set_width(object obj, double width);

Parameters

obj: The object
width: The size of the object in inches

Return

None.

Example

size = set_width(obj,1);

See Creating Text Example

4.148 set_x

Set the x coordinate of a point. See set_y and addpoint.

Syntax

void set_x(point pnt);

Parameters

pnt: The point to change

Return

None:

Example

set_x(pnt);

See Point Type Example

4.149 set_xarc

Makes a point an arc start point. See Creating_an_object, is_corner, is_open, is_line, is_xarc, set_corner, set_open, set_line, get_point_type and set_point_type.

Syntax

void set_xarc(point pnt);

Parameters

pnt: The point to change

Return

None:

Example

set_xarc(pnt);

See Point Type Example

4.150 set_y

Set the y coordinate of a point. See set_x and addpoint.

Syntax

void set_y(point pnt);

Parameters

pnt: The point to change

Return

None:

Example

set_y(pnt);

See Point Type Example

4.151 sin

The sin of an angle

Syntax

double sin(double angle)

Parameters

angle: The angle

Return

The sine of an angle

Example

y=sin(ang)

See Math Function Example

4.152 start_pict_dialog

Starts the creation of a style dialog box. See add_pict_item, run_pict and get_pict_result.

Syntax

void start_pict_dialog(string title)

Parameters

title: The title of the dialog box

Return

None

Example

start_pict_dialog("A Number");

See Creating a style dialog

4.153 start_picture_input

Starts a measurement dialog. See add_picture_input and run_picture_input.

Syntax

void start_picture_input(string title);

Parameters

title: The title of the dialog box

Return

None

Example

start_picture_input("My exciting Title");

See Creating a measurement dialog

4.154 symbol

Creates a symbol. See Creating_a_Symbol, block_loaded, is_insertion and Working With Selections.

Syntax

void symbol(selection objects, string name, double x, double y);

Parameters

objects: The objects to use to create the symbol
name: The name of the symbol
x: The x coordinate of the insertion point, ie the location that will match the insertion point when the symbol is put into the drawing
y: The y coordinate of the insertion point, ie the location that will match the insertion point when the symbol is put into the drawing

Return

none

Example

symbol(current_selection,"Tester",0,0);

See Symbol Example

4.155 sqrt

The square root of a number

Syntax

double sqrt(double value)

Parameters

value: A number

Return

The square root

Example

x=sqrt(x);

See Math Examples

4.156 user_input

Pushes a spot for the user to input a value onto the stack for use with run_command. See Running_PatternMaker_Commands

Syntax

user_input();

Parameters

None.

Return

None.

Example

var_push(0,0);
var_push(45);

See Generated Mouse Input Example

4.157 unit_mode

Returns 1 if the user is using metric coordinates

Syntax

double unit_mode()

Parameters

none

Return

0=English
1=Metric

Example

```
if (unit_mode() == 1)
  {
  ...
  }
```

See English or Metric Example

4.158 var_push

Pushes variables onto the stack for use with run_command. See Running_PatternMaker_Commands

Syntax

void var_push(double mouse_x, double mouse_y);
void var_push(double value);

Parameters

mouse_x: The x coordinate of a left mouse click
mouse_y: The y coordinate of a left mouse click

value: A number for a typed value

Return

None.

Example

var_push(0,0);

var_push(45);

See User Input Example and Generated Mouse Input Example

4.159 version

Returns the version of the macro language being used. It is useful for testing if a feature is implemented in the language. See is_function.

Syntax

double version()

Parameters

None.

Return

The version of PatternMaker being run

Example

```
if (version() == 7)
  {
  ...
  }
```