INSIDE SCION IMAGE FOR WINDOWS

General Information

About this Document

This document is intended for beginners who never studied programming but need to use software and get the most out of it. You might have an image processing application that needs doing and don't want to figure out all the aspects of Windows. You can build your image processing application into the Scion Image program and save yourself from a lot of wasted effort. Hopefully, this manual may help you on your route to writing simple or complex macros.

Macro Examples, Techniques & Operations

What is a macro and why write one?

A macro is text containing a sequence of calls or routines which Scion Image interprets and executes. To write a macro, you can choose "New" then "text window" to create a text window within Scion Image. You load the macro using "Load Macro". A rich set of example macro routines is distributed with the Scion Image program. You can try some of these out and borrow code from them in order to write your own macro.

Simple macros, such as the one below, are useful utilities to save time and effort. This macro is an example of a macro which follows the same operations that could be performed by you from the Scion Image menus. It's operation is to clear anything outside of the Region of Interest (ROI) which you draw. Macros can, of course, be much larger and can include looping, calculations and basically an entire imaging application.

```
MACRO 'Clear Outside [C]';
 {Erase region outside ROI.}
BEGIN
  Copy;
  SelectAll;
  Clear;
  RestoreRoi;
  Paste;
  KillRoi;
END;
```

Before you begin

It should not be hard for you to start writing a macro. You will want to do several things before you begin. Appendix A in the Scion Image users manual provides you with a complete list of all macro calls. The list is organized by the Scion Image menus and then is categorized in alphabetical order. After this locate the macros directory distributed with Scion Image. Open, load and examine some of the macros. Try using "Find" from the "Edit" menu on one of the open macros. "Find" is fairly useful in helping you debug a macro. It allows you to go to sources of error when you get error messages during the load or execution of a macro.

For the programming beginner

You probably don't need to study programming to write a macro. Depending on the complexity of your application, you might be able to pick up everything you need by examining some of the macros in the macros directory. To some a confusing aspect of writing macros is understanding what a function is and how it is used. A function returns a value or a boolean (true/false). In the example below, nPics is a function which will return an integer number of pictures open. KeyDown(‘control’) returns a true or false depending on whether you hold the control key down.

```
Macro 'Function demo';
begin
 {Here is an example use of the nPics function returning a value}
   showmessage('Number of images open: ',nPics);
 {Here is an example use of keydown function returning a boolean}
  If KeyDown('control') then putmessage('Number of images open: ',nPics);
end;
```

Macro global vs. local vars

Just as in pascal, C, or other programming languages, you can have a local or global variable. A global variable is declared at the top of the macro file and can be utilized by any procedure or macro in the file. A local variable is declared in the procedure or macro in which it is used. For the example macro set below, "A" and "B" are local to the 'Add numbers' macro. "Answer" is globally declared and used by both macros.

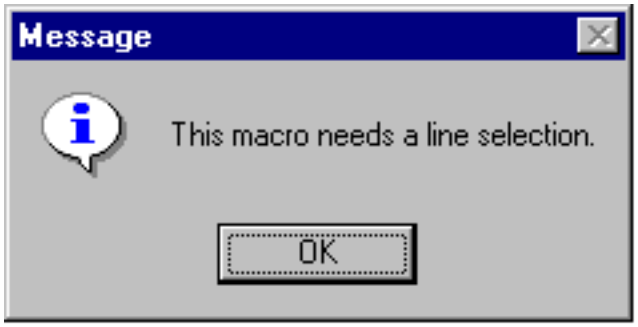
```
VAR
   Answer:real;

Macro 'Add numbers';
Var
   A,B: real;
begin
   A := Getnumber('Enter the first number',2.0);
   B := Getnumber('Enter the second number',3.14);
   Answer := A+B;
end;

Macro 'Show Answer';
begin
   ShowMessage(' The added result is: ', Answer:4:2);
end;
```

Putmessage, ShowMessage & Write

PutMessage



PutMessage is perhaps one of the easiest ways to provide feedback to users. To use putmessage you simply call the routine with the message or string you wish to give to the user.

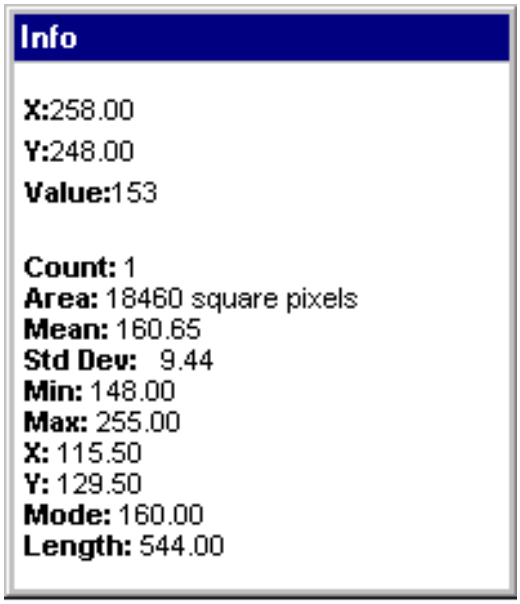
```
PutMessage('This macro requires a line selection');
```

You can pass multiple arguments with PutMessage if you needed to.

```
PutMessage('Have a ', 'Nice day');
```

ShowMessage

ShowMessage allows display of calculations, data, variables or whatever you caste as a string into the Info window.



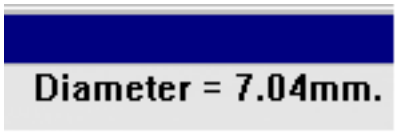
Here is a simple example of output to the Info window:

```
ShowMessage('x1 = ',x1);
```

You can use the backslash ('\') character to do a carriage return for macros:

```
ShowMessage('Average Size=',AverageSize:1:2,'\TotalCount=',TotalCount);
```

Write



You can also write data or info onto the image window with a macro call to Write or Writeln.

```
Diameter := Width / PixelsPerMM; {in MM.}
MoveTo(300,10);
Write('Diameter = ', Diameter:5:2,'  mm.');
```

Switching and choosing windows

There are a number of ways to switch between windows in a macro. For the most part you will need to use the PidNumber function to identify a unique ID for that window. Pidnumber is a function which returns a value. For example you might have:

```
var
  MyPicID:integer;
begin
   MyPicID := PidNumber;
   Duplicate('Duplicate image');
{some process}
   SelectPic(MyPicID); {To go back to the original}
```

Here the returned value from the PidNumber function was assigned to a variable called MyPicID. The variable MyPicID was then used later on in the macro to select the picture.

As an alternative to SelectPic, you could have used ChoosePic(MyPicID). This would have selected the picture but would not have made it the active front window. This is useful when you flip between many windows, but do not need to activate the window.

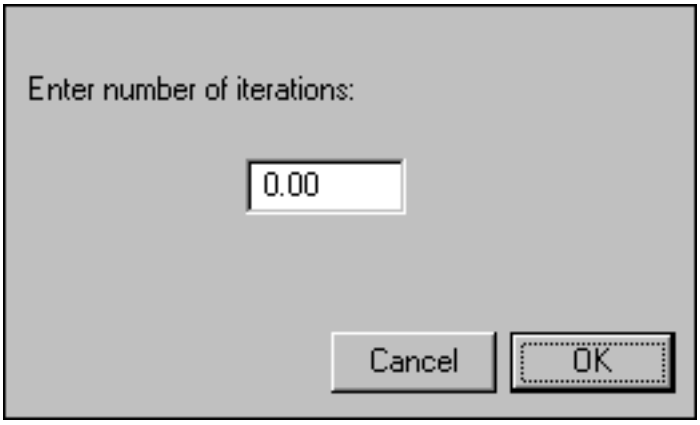
As a second alternative, you could can use SelectWindow('Window name') to select the window by its title.

How to input a number or string

Making a call to getnumber will allow you to enter a number into your macro. The GetNumber macro will return a real number, or if assigned to an integer variable, such as in this example, it will not pass the decimal digits.

```
var
 MyGlobalNumber:integer;

macro 'Number input';
begin
   myGlobalNumber:=GetNumber('Enter number of iterations:',0);
end;
```



The idea is the same for entering a string

```
var
 MyString:string;

macro 'String input';
begin
   MyString:=GetString('What name?','Data');
end;
```

Looping

The Scion Image macro language has the standard set of pascal loops. This includes "for" loops and "while" loops.

The Scion Image macro language is (almost) a subset of Pascal and the Pascal FOR statement does not have a BY option. Instead, use a WHILE loop. For example:

```
i:=1;
while i<=fred do begin
  {process}
   i:=i+10;
```

```
end;
```

Regions of Interest (ROI)

Before you start looking at macro ROI's an introduction to coordinates is worthwhile. See the picture below for a general guideline. Regions of interest are characterized by 'marching ants' which surround a selection.

Getting ROI information

GetRoi(left,top,width,height)
You will want to call this macro routine if you need any information about the current ROI. The routine returns a width of zero if no ROI exists.

ROI creation

SelectAll
The Selectall macro command is equivalent to the selects all of the image and shows the ROI's 'marching ants'.

MakeRoi(left,top,width,height)
This is as straight forward as the name implies.

MakeOvalRoi(left,top,width,height)
Not terribly differing to implement from MakeROI. If you want a circular ROI set width and height to the same value. See the example below.

Altering an existing ROI

MoveRoi(dx,dy)
Use to move right dx and down dy.

InsetRoi(delta)
Expands the ROI if delta is negative, Shrinks the ROI if delta is positive.

Other routines involving ROI's

RestoreROI,KillRoi
These are opposities.

Copy,Paste,Clear,Fill,Invert,DrawBoundary

Detecting the press of the mouse button

The example below shows a macro which operates until the mouse button is pressed. Button is your basic true or false boolean and becomes true when the button is pressed.

```
macro 'Show RGB Values [S]';
var
 x,y,v,savex,savey:integer;
begin
 repeat
  savex:=x; savey:=y;
  GetMouse(x,y);
  if (x<>savex) or (y<>savey) then begin
    v:=GetPixel(x,y);
    ShowMessage('loc=',x:1,', ',y:1,
     '\value=',v:1,
     '\RGB=',RedLUT[v]:1,', ',GreenLUT[v]:1,', ',BlueLUT[v]:1);
    wait(.5);
  end;
 until button  ;
end;
```

Detecting press of shift and control keys

The macro "KeyDown(key)" (Key = 'shift', or 'control') returns a boolean true or false. It returns TRUE if the specified key is down. The example macro below can be run on any stack, using shift to delay more or control to delay less.

```
macro 'Animate Stack';
var
 i,delay:integer;
begin
 RequiresVersion(1.56);
 i:=0;
 delay:=0.1;
 repeat
  i:=i+1;
  if i>nSlices then i:=1;
  Wait(delay);
  SelectSlice(i);
  ifKeyDown('shift')   then delay:=1.5*delay;
  if delay>1 then delay:=1;
  ifKeyDown('control')   then delay:=0.66*delay;
  ShowMessage('delay=',delay:4:2);
 until button;
end;
```

Measurement and rUser Arrays

There are a number of arrays in macros, but there are two varieties the measurement arrays and the rUser arrays. You can store macro data and results in the rUser arrays. These arrays are not affected by the Measurement counter (rCount) which works with measurements arrays such as rMean[rCount], rArea, etc. The current rCount for these is changed by doing a measurement or calling SetCounter.

Example of storing data to the rUser arrays:

```
rUser1[1]:=SomeNumber;
rUser2[1]:=SomeOtherNumber;
```

If you have more than two sets of data which you'd like to keep, and because there are only two rUser arrays, then you can access other macro arrays. This includes rArea, rMean, rStdDev, rX, rY, rMin, rMax, rLength, rMajor, rMinor, and rAngle. However you will need to be careful because these arrays are affected by the rCount value and you could write over your data. An example use of measurement arrays outside the intended use is a snipet of code from the Export look up table macro:

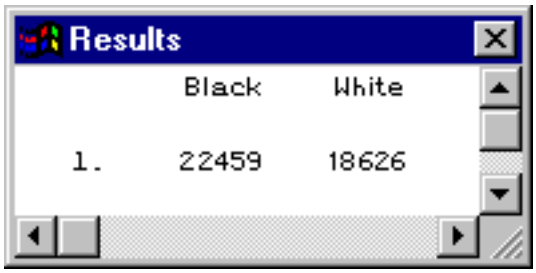
```
for i:=0 to 255 do begin
   rArea[i+1]:=RedLut[i];
   rMean[i+1]:=GreenLut[i];
   rLength[i+1]:=BlueLut[i];
 end;
```

Here rArea, rMean and rLength are used for Red, Green and Blue instead of area, mean and length.

Placing macro data in the "Results" window

If you have particular information, data, calculated results, or any type of numeric data which you want to keep, you can redirect it into the Results window. Use the SetUser label commands to title your field name. The rCount function keeps the current index of the measurement counter. Since rUser1 and rUser2 are arrays, you specify the index of the array with the rCount value. See below.

```
macro 'Count Black and White Pixels [B]';
{
Counts the number of black and white pixels in the current
selection and stores the counts in the User1 and User2 columns.
}
begin
 RequiresVersion(1.44);
 SetUser1Label('Black');
 SetUser2Label('White');
 Measure;
 rUser1[rCount]:=histogram[255];
 rUser2[rCount]:=histogram[0];
 UpdateResults;
end;
```



Saving results data to a tab delimeted file

You can also save data from the macro, to a tab delimeted text file by adding several commands in your macro:

```
SetExport('Measurements');
Export('YourFileName');
```

Operating on each image in a stack (SelectSlice)

By using a loop (for i:= 1 to nSlices) you can operate on a series of 2D images. The nSlices function returns the number of slices in the stack.

```
macro 'Reduce Noise';
var
 i:integer;
begin
   if nSlices=0 then begin
   PutMessage('This window is not a stack');
    exit;
 end;
 for i:= 1 to nSlices do begin
   SelectSlice(i);
   ReduceNoise;  {Call any routine you want}
 end;
end;
```

See the series of stack macros distributed with the Scion Image program for more examples.

Extracting a substring from a string

Below is an example macro that will allow you to pull a substring out of a string.

```
{
   An example routine to return a substring from a string in Scion Image macro.
}
var
    ReturnString:string;

procedure copystring(SourceString:string,index:integer,count:integer);
begin;
```

```
    ReturnString:=SourceString;
    if index > 0 then Delete(ReturnString,0,index);
    Delete(ReturnString,count+1,length(ReturnString)-count);
end;

macro 'test copystring'
var TestString:string;
begin
   TestString:='This is a test';
   copystring(TestString,11,4);
   PutMessage('The Returned String is :  ' ReturnString);
end;
```

Accessing bytes of an image

The macro commands GetRow, GetColumn, PutRow and PutColumn can be used for accessing the image on a line by line basis. These macro routines use what is know as the LineBuffer array.

The example below is a macro which uses the linebuffer array. If you are interested in using a macro to get at image data, this example should be fairly clear.

```
Macro 'Invert lines of image'
var
 i,j,width,height:integer;
begin
 GetPicSize(width,height);
 for i:=1 to height do begin
   GetRow(0,i,width);
   for j:=1 to width do begin
    LineBuffer[j] := 255-LineBuffer[j];
   end;
   PutRow(0,i,width);
 end;
```

Reading from disk (importing)

One simple way to load data from disk is to create a window and dump information to it. An example of this is a macro which imports files created by the IPLab program. The macro reads the first 100 bytes from the file into a temporary window. It erases the window when it is through finding useful header information.

```
macro 'Import IPLab File';
var
  width,height,offset:integer;
begin
  width:=100;
  height:=1;
  offset:=0;
  SetImport('8-bit');
  SetCustom(width,height,offset);
  Import(''); {Read in header as an image, prompting for file name.}
  width := (GetPixel(8,0)*256) + GetPixel(9,0);
```

```
  height := (GetPixel(12,0)*256) + GetPixel(13,0);
  Dispose;
  offset:=2120; {The IPLab offset}
  SetImport('16-bit Signed; Calibrate; Autoscale');
  SetCustom(width,height,offset);
  Import(''); {No prompt this time; Import remembers the name.}
end;
```

Batch Processing

It's easy to write a macro to process a series of images in a directory as long as the file names contain a numerical sequence such as 'file01.pic', 'file02.pic', 'file03.pic', etc. Below is an example macro that does this.

```
macro 'Batch Processing Example';
{
Reads from disk and processes a set of images too large to
simultaneously fit in memory. The image names names must be
in the form 'image001', 'image002', ..., but this can be changed.
}
var
 i:integer;
begin
 for i:=1 to 1000 do begin
    open('image',i:3);
   {process;}
    save;
    close;
  end;
end;
```

Avoiding a macro dialog box

You should be able to process many files and only have to see one dialog box. For example, only one dialog box appears when you run the following macro as long as 'A', 'B' and 'C' are in the same directory.

```
macro 'test';
begin
  Open('A');
  Invert;
  Save;
  Close;
 Open('B');
end;
```

Another way to avoid the dialog box is to use full directory paths as in the following example.

```
macro 'test';
begin
 Open('c:\images\A');
```

```
 Invert;
 Save;
 Close;
 Open('c:\images\B');
 Invert;
 Save;
 Close;
 Open('c:\images\C');
 Invert;
 Save;
 Close;
end;
```

TickCount

Ticks are counted at the rate of 60 per second. You can varify this by running the enclosed macro and timing the interval between beeps.

```
macro 'TickCount Test';
{"Beeps" every 10 seconds}
 var
   interval,ticks:integer;
begin
 interval:=600;
 ticks:=TickCount+interval;
 repeat
  if TickCount>=ticks then begin
    beep;
    ticks:=ticks+interval;
  end;
 until button;
end;
```

Accessing an image Look Up Table (LUT)

You can modify the way an image appears by altering the RedLUT, GreenLUT and BlueLUT. This is simple and straightforward enough. You can access the RedLUT, GreenLUT and BlueLUT arrays from both macros and from Pascal.

Here is an example macro which finds any gray or black components in a color image and sets them to white. It's useful for seperating certain kinds of medical data.

```
macro 'Remove Equal RGB [V]';
{Changes only the LUT, removes gray component from an image}
var
 i,Value:integer;
begin
  for i:=1 to 254 do begin
    If ((RedLUT[i] = BlueLUT[i]) and (RedLUT[i] = GreenLUT[i]))
   then begin
     RedLut[i] :=255;
     BlueLut[i] := 255;
     GreenLut[i] :=255;
  end;
```

```
  end;
 ChangeValues(255,255,0); {remove black}
 UpdateLUT;
end;
```

Placing time and date into your data

Here is a macro that writes the current date and time to a text window. It also stores results into the text window. Here is what the output from this macro looks like:

Date=94:5:31
Time=14:45:24
Area=10000.000
Mean=80.198

```
macro 'Write Results to Text Window';
var
  year,month,day,hour,minute,second,dow:integer;
begin
  GetTime(year,month,day,hour,minute,second,dow);
  Measure;
  NewTextWindow('My Results');
  writeln('Date=',year-1900:1,':',month:1,':',day:1);
  writeln('Time=',hour:1,':'minute:1,':',second:1);
  writeln('Area=',rArea[rCount]:1:3);
  writeln('Mean=',rMean[rCount]:1:3);
end;
```

PlotData notes

A command exists in the macro language for making profile plot data available to macro routines. It has the form "GetPlotData(count,ppv,min,max)", where count is the number of values, ppv is the number of pixels averaged for each value, and min and max are the minimum and maximum values. The plot data values are returned in a built-in real array named PlotData, which uses indexes in the range 0-4095. The macro "Plot Profile" in the "Plotting" file illustrates how to use GetPlotData and PlotData.

To help explain further....

1. For a count value of n the PlotData array will have meaningful values from 0 to n-1 (higher array values are accessible but will contain old/meaningless results).

2. Count is equal to the line length, in pixels, rounded to the nearest integer value.

3. Substantially more pixels are usually highlighted by a line selection, and this seems to have only an approximate corelation with the pixels used by PlotData.

4 The PlotData array contains real-numbers (not integers) which presumably are derived from a weighted average of pixels rather than being the values of single pixels - even when ppv is 1. Because of this it is not possible to relate PlotData values to single locations.

5. My conclusion after some experimentation is that;

```
after   GetLine(x1,y1,x2,y2,lw);
and     GetPlotData(count,ppv,min,max);
```

The following function will probably return the centre of the location used to derive PlotData[c]:

```
ypos:=y1+(c+0.5)/(count)*(y2-y1);
xpos:=x1+(c+0.5)/(count)*(x2-x1);
```