Brief Add-Ons Explaination...

No application could possibly have every option, every function, and all the flexibility anyone would ever want. MacPhase has one feature which will help you add some flexibility.

Have you ever just wanted to do some sort of processing on data file but were procrastinating because you didn't want to spend the time learning how to read the file and then display your results. Well, now you can let MacPhase do the work and you can spend your valuable time programming that processing option you always wanted to perform.

There are two ways in which you can “interface” to MacPhase, through the Macintosh clipboard and by loading in code moduals. Later a third method will be added which will allow programming by way of the MacPhase macro language.

Macintosh Clipboard Interface

MacPhase will send and receive to and from your application a special clipboard called a MPOP clipboard. Now if you are worried about how to read and write to and from the clipboard, don't. Programming templates have been included on your MacPhase application disk in Think Pascal™ and Think C™. Further information below.

Macintosh Code Resource Interface

If you would like to make life as easy as selecting a menu option, then the MacPhase Add-On is what you always wanted. Code Resources can be compiled and loaded into MacPhase on launching. Programming templates have been included on your MacPhase application disk in Think Pascal™ and Think C™. Further information below.

Using the MPOP Clipboard

First, what you'll have to do is startup MacPhase and select the Clipboard Options from the Edit menu. In the Clipboard Options dialog box, select the following items: Clip data as Binary Data (MPOP), Convert MPOP clipboards to data, and Make Incoming clipboard a new window. Now in Multifinder or System 7.0, switch out to the Finder and startup your programming application, for this example Think Pascal™ will be used.

Two functions are provided which can help you copy and paste to the System clipboard. This clipboard can not be used by any standard Macintosh application, only those which support MacPhase. This specific clipboard makes communicating between your program and MacPhase so simple.

The function provided to read the System clipboard is named ReadClip. The statement is shown below.

- **`function ReadClip (p: ptr; width, height: longint; max, min: real; mpVarCode: integer): boolean;`**

ReadClip accepts the several arguments which are:

Argument	Type	Description	Returns
ReadClip	boolean	Function to read clipboard	TRUE if read OK
p	Ptr	Pointer to your data array	Unchanged
width	longint	width of the data expected	width or error
height	longint	height of the data expected	height read
max	real	maximum of the data read	maximum read
min	real	minimum of the data read	minimum read
mpVarCode	integer	data type code	actual type read

ReadClip Arguments

Type	Code
Byte	1
Integer	2
Real	4
Longint	5

MacPhase Variable Codes

The function provided to write to the System clipboard is named WriteClip. The statement is shown below.

- `function WriteClip (p: ptr; var width, height: longint; var max, min: real; var mpVarCode: integer): boolean;`

WriteClip accepts the several arguments which are:

Argument	Type	Description	Returns
ReadClip	boolean	Function to read clipboard	TRUE if read OK
p	Ptr	Pointer to your data array	N/A
width	longint	width of the data	error code
height	longint	height of the data	N/A
max	real	maximum of the data	N/A
min	real	minimum of the data	N/A
mpVarCode	integer	data type code	N/A

WriteClip Arguments

The maximum and minimum are not needed when writing the clipboard, but you will have to Rescale the window when you enter into MacPhase.

Loading Code Resources

As with transfering MacPhase data through the Macintosh clipboard, loading code resources allows you to concentrate on your specific processing task. You won't have to spend long hours writing code to read, visualize, print, and store data... You can use MacPhase for that. Use your time to code your specific processing task, compile it to a code resource file, and drag that file to the MacPhase Add On folder.

Programming examples have been included in THINK™ Pascal and C, extensions to other compilers should be relatively straight forward. If you have any problems, feel free to get in touch with us.

Add On Introduction

Before you begin to write your Add-On you should understand how these will be "Added-On" to MacPhase.

Before MacPhase can use any Add-On you have created, it must first know where to look for your procedures. You will have to set the Add-On folder from the Preferences...Folders sub menu, see the File Menu chapter. Once this has been selected, each time MacPhase is executed, any Add-On file found in this selected Add-On folder will be "added on".

Add On Principles

Once loaded, each Add-On file name found will be added to the Extra menu. This will be a hierarchical menu with 1 to 10 MPCD code resource names as sub menu options. This allows you to organize common processing procedures in one menu.

Most compilers only allow single code resources to be compiled. You will have to use ResEdit™ to copy multiple code resources to a single resource file. Make certain each resource has a name.

Add Ons • How they're done

When a specific Add-On menu option is selected, MacPhase will call your MPCD code resource passing some of the active windows information to your code procedure as:

```
• function main (info: MPTypePtr ): MPTypePtr ;
```

The pointer passed to your code resource, namely info, points to a MPType record structure of the following format.

```
MPTypePtr = ^MPType;
MPType = record
    data: Handle;
    mpVarCode: integer;
    width, height: longint;
    max, min: real;
    dataRgn : RgnHandle;
```

```
codeRefNum: integer;
infoWindow: WindowPtr;
```

```
end;
```

Argument	Type	Description
data	Handle	Handle to your data array
mpVarCode	integer	data type code
width	longint	width of the data expected
height	longint	height of the data expected
max	real	maximum of the data read
min	real	minimum of the data read
dataRgn	RgnHandle	Region of the data array selected
codeRefNum	integer	Your code resource refNum
infoWindow	WindowPtr	Region of the data array selected

MPType Format

In addition to information about the active window, two other variables are passed. The **codeRefNum** is the reference number to your resource file containing the MPCD code resource. If you call any other resources found in that particular Add-On file, call **UseResFile(codeRefNum)**, this will assure that you will be accessing your resources and not MacPhase's. A WindowPtr to the Info Window is passed so that you may include progress statistics or other important information about your processing.

Data is accessed in much the same way as in the MPOP clipboard processes, see above.

On completion of your procedure, you have two basic options. You can return the processed data to the active window or you can create a new window. If you write your processed data to the data structure pointed to by **data**, your results will be returned to the active window. But, pass a different **data** Handle back through the fuctions return, a new window will be created. You can do this by creating a new Handle and replacing **data**with that new Handle.

Quick MPCD Code Example

The pascal code is provided for you in the event that the MacPhase diskettes are out of reach. This should be enough to get you started. As you progress, you can increase the complexity of the Macintosh interface by calling up dialogs and other resources.

```
unit MakeCode;
interface

  function main (info: ptr): ptr;
    { This is the main function call which is }
    { called from MacPhase }

implementation

  type
    { Some types which are used through }
    { this code procedure }
```

```
  IntegerPtr = ^Integer;
  RealHandle = ^RealPtr;
  RealPtr = ^RealArray;
  RealArray = array[1..1] of real;

  MPTypePtr = ^MPType;
  MPType = record
     data: Handle;
     mpVarCode: integer;
     width, height: longint;
     max, min: real;
     codeRefNum: integer;
     infoWindow: WindowPtr;
    end;

function main (info: ptr): ptr;
  { This is where everything takes place }
  { all the processing is done within  }
  { this function }

const
   NewDataWindow = TRUE;
     { Whether or not a new window is }
     { created on return to MacPhase }
  var
   ok: boolean;  { Memory allocation ok ? }
   newData: Handle; { handle for new data }
   infoP: MPTypePtr; { typed ptr }

{ • }
  procedure DoCalculation;
   { Very simple calculation of adding }
   { 1.0 to every point in the passed }
   { data.  Use this as a guide for }
   { more complicated calculations }
 var
     x, y, xp: longint;
     mx, mn, ans: real;
  begin
   with infoP^ do
    begin
      mx := -1e20;
      mn := 1e20;
      HLock(data);
      HLock(newData);
        { Lock all handles to be on the safe side }
      for y := 1 to height - 1 do
         for x := 1 to width do
          begin
            xp := (y - 1) * width + x;
            ans := RealHandle(data)^^[xp] + 1.0;
            RealHandle(newData)^^[xp]:=ans;
            if ans > mx then
              mx := ans;
            if ans < mn then
              mn := ans;
          end;
```

```
        min := mn;
        max := mx;
        HUnLock(newData);
        HUnLock(data);
      end; { with }
   end; { Normalize }
{ • }

 begin
   infoP := MPTypePtr(info);
     { Coerse to a typed pointer to make things }
     { a bit simpler }
   ok := TRUE;

   if NewDataWindow then
      { If you want to make a new data window }
      { on return to MacPhase you must return }
      { a different data handle. }
    begin
      newData := NewHandle(infoP^.width *
 infoP^.height * sizeof(real));
      ok := (MemError = noErr) and (newData <> nil);
      if not ok then
        SysBeep(0);
    end;

   if ok then
     DoCalculation;
   if NewDataWindow and ok then
     infoP^.data := newData;
       { If you do want to create a new window on }
       { return, you must set the data handle to }
       { the new data handle you created. }
   Main := info;
 end; { main }

end. { unit }
```