User External Documentation
Externals Docs Part II

Hermes BBS v1.0.7

by Frank Price, III

First Draft
Call the support BBS.....(213)275-6975

Introduction

Writing user externals can be simple and it can be very difficult depending on what you are doing. The possibilities are absolutely endless of what you could do with this interface. Source code has been included with this document for a very small example external which allows users to change their phone number. It does nothing special. It doesn't hook into a main menu command, but uses the standard externals menu for access. This is just one of many ways your external can be accessed. For the most part, you can do anything that I myself could do inside Hermes. However, this is limited by several issues which we will find out about.

Basics

Sysop externals and user externals are two different beasts. They must be compiled separately and they are called differently by Hermes. But the external file that is used by sysop externals may also contain a user external. Whereas the sysop external code is contained in a resource of type 'XHRM' ID=10000, a user external is an 'XHRM' ID=10001. An example of an external that has no sysop external associated with it is the Update Phone external. An example of a sysop external with no user external is the Options external. If we were to combine these two externals as can easily be done, we could create quite powerful changes to Hermes. The user code for GFiles is internal in Hermes. This is because it was programmed before the user external interface was finished. But if it wasn't, we could make something exactly the same completely external. The necessary resources for a Hermes external file to be recognized as a user external are:

XHRM, 10001 Code resource
HRMS, 100 Information record

Also, as stated in the sysop external docs, the file must be of type "XHRM" and creator "HRMS". A file can have both a sysop external and a user external, or either one. But it must meet these conditions. When placed in the Externals folder, Hermes will automatically recognize it on startup.

The HRMS 100 Resource

In Pascal, the structure for the HRMS ID=100 resource that must accompany any user external is as follows:

```
eInfoRec = record
    allTime: boolean;
    minSLforMenu: integer;
    Restriction: char;
   end;
```

This record is also contained in the HermHeaders.p file distributed with this document. You can either program a sysop external to fill the fields of this record with the desired values or set them manually. A definition of these fields is in order here:

allTime: When true, the external will be called on every single user idle loop in Hermes. This feature will be explained more later. If you external does nothing beyond getting invoked from the external menu, doing its thing, and then returning, you will want to set this to false. For most other externals however, this will be set to true. For instance, if you wanted your external to be invoked by the "V" command on the main menu, you would continually check every time you are called in this idle loop to see if the user has typed "V" at the main menu prompt. You would then take control. A side note here would be that you would first have to add "V" to the acceptable main menu commands list in STR# resource ID=16.

minSLforMenu: This number represents the minimum security level required for your external to be seen on the externals menu accessed by the "." command from the main menu. Setting this above 255 will make sure it is never seen on the externals menu.

Restriction: If this is not equal to a null character, it must be equal to a capital character from "A" to "Z" representing the restriction a user is required to have in order to see this external on the externals menu. Setting this to a null character will cancel the checking for the restriction.

It is reccomended that you give users a sysop external with which to configure these commands and any other options you can provide in your external. Unless of course that is not appropriate for your external. The Update Phone external has no sysop external and simply leaves all the fields in the HRMS resource at 0 because it is not of much importance.

We have now completed our discussion of the general file interface for user externals. The next topic is the actual calling method for the external.

Calling Method

All of the interfaces included are in Pascal. These should also work fine with MPW Pascal, and could be converted to C. Source to the Update Phone external has been included and is a very simple external that should provide most of the information you need. With it is included the latest version of the HermHeaders.p file. This file now contains tons of data structures that you will never need. But it just represents the power now available with user externals. The immense structure in this file called the **HermUserGlobsRec** is the same record that is used by Hermes to store every single variable that is saved for each node. These are passed to you the external in the **UserXIPtr**. The procedure prototype for a user external is this:

```
procedure MAIN (theHermStuff: UserXIPtr);
```

You will find the **UserXIPtr** as the last structure in the **HermHeaders.p** file. It directly proceeds the Hermes procedures interfaces. Each of the procedures listed at the bottom of the file are available to be called by you and are defined individually later in this document. For now, let's take a look at the **UserXIPtr**:

```
UserXIPtr = ^UserXInfoRec;
 UserXInfoRec = record
   privates: myPrivsHand;
   extID: integer;
   totalNodes: integer;
   message: integer;
   curNode: PtrToWord;
   curUGlobs: PtrToLong;
   ResShared: integer;
   DataShared: integer;
   HSystPtr: SystPtr;
   HMDescPtr: MDescPtr;
   HDirDataPtr: dirDataPtr;
   HGFilePtr: GFileSecPtr;
   filesPath: StringPtr;
   HermUsers: UListHand;
   n: array[1..10] of HermUserGlobPtr;
   procs: array[0..0] of procPtr;
  end;
```

This record is passed to your external each time it is called. An explanation field by field follows:

privates: This field is where you store your own variables. It is currently declared as myPrivsHand. This is arbitrary. The myPrivsHand structure can be found in the HermHeaders.p file and is a reccomended start for what variables you will need. When your external is first called, this field will be set to 0. As shown in the Update Phone external, you should then allocate the block for your variables as a handle and use this field to store that handle. It will be preserved across calls to

you. It is used much like the refCon field in much of the Macintosh toolbox.

extID: This field contains the number of your external according to Hermes' internal list of externals. It is useful for causing a switch of control to your external as explained later under the message field.

totalNodes: This is simply an integer which indicates how many nodes are allocated and active currently. You cannot change this, it serves only to tell you the number.

message: This field can contain one of the following values:

```
ACTIVEEXT = 1;{values for message field to user external}
IDLE = 2;
CLOSENODE = 3;
CLOSEEXTERNAL = 4;
```

The **activeExt** value means your external is currently being called because through some condition the BBS thinks you have been selected from the externals menu. This can be caused in two ways. The first is by a user actually selecting you from the externals menu. The second is if you set the **BoardSection** variable for a particular node to **EXTERNAL** and set the **activeUserExternal** field to your external number contained in the field **extID**. If you set these two variables in your idle routine, you will then be called with the activeExt message. The **Idle** message is sent only to externals which have the **allTime** variable set to true in their HRMS 100 resource. If this variable is true, you will be called with the Idle message on every event loop of Hermes. Directly after you are called with this message, Hermes calls its own IdleUser routine which handles carrier detect, timeouts, and serial port tending. The **closeNode** message is sent to an external when the user on the node curNode has logged off or some other condition has caused this node to stop your external such that it cannot continue to execute it for this node. This does not mean you should dispose of everything. Just stop any planned actions for that particular node. The **closeExternal** message is sent to your external when Hermes quits. You should clean up immediately and dispose of any structures you have allocated.

curNode: This field is a pointer to an integer which contains the active node number that you are being called to handle. This field is invalid when you are called with the Idle message. But it is necessary when called with the activeExt message. The reason it is a pointer is because the variable it points to is the same variable Hermes uses to determine the active node. You should beware of switching it, but that power is available.

curUGlobs: This field is a pointer to a pointer. The reason for this is similar to the reason curNode is a pointer. It points to an actual global variable in Hermes that is used by almost every single routine. The value pointed to will be exactly the same as the value of n[curNode^]. The only reason you would ever change this

variable would be to switch between nodes. For instance, if you had a routine that ran every idle message that would output a line on any node that was at a certain BoardSection, you would have to tell Hermes exactly which ndoe you are talking about when you use OutLine since curNode is meaningless during the Idle message. This is what you would do if you want to set the current node to node 4:

```
curNode^:=4;
curUGlobs^:=n[4];
```

You must set both of those variables in order for procedures like OutLine and bCR to understand which node to work with.

ResShared: This integer is the resource file reference number for the Hermes Shared file. You may not close this file, but feel free to read and write from it.

DataShared: This integer is the data fork file reference number for the Hermes Shared file. The UserRec structure found in the HermHeaders.p file is stored here consecutively by user number. Feel free to read and write from this file. Do not close it.

HSystPtr: This is a pointer identical to the identically named one passed to sysop externals. For a listing of the fields in this record, search for it in the HermHeaders.p file. This record contains the fields set by the Options sysop external.

HMDescPtr: This is a pointer identical to the identically named one passed to sysop externals. For a listing of the fields in this record, search for it in the HermHeaders.p file. This record contains the fields set in the Messages area under the Sysop menu.

HDirDataPtr: This is a pointer identical to the identically named one passed to sysop externals. For a listing of the fields in this record, search for it in the HermHeaders.p file. This record contains the fields set by the Transfers command under the Sysop menu.

HGFilePtr: This is a pointer identical to the identically named one passed to sysop externals. For a listing of the fields in this record, search for it in the HermHeaders.p file. This record contains the fields set by the GFiles sysop external.

filesPath: This is a pointer to the full pathname to the Hermes Files folder. For instance, “Hard Drive:BBS:Hermes Files:” would be a possible value for this.

HermUsers: This field is included for convenience. It is a handle to an index of all users on the system. The actual structures used can be found in the HermHeaders.p file.

n: This 10 pointer array contains one pointer for each node. They are in order, and you should not use the fields above the value of totalNodes. They point to the variables structure **HermUserGlobsRec** found in the HermHeaders.p file.

procs: This 0-based array is a variable length list of pointers to procedures. At the current time, there are 13 available procedures you can call in Hermes each of which are defined later in this document. Each procedure has a “selector” which is an index into this array. Further procedures will be added in future versions. These additions will not affect already existing externals. Do not rely on the size of the UserXIRec.

Hermes Procedures

The user externals interface provides many procedures which you can call. These procedures are inside Hermes and will act as if they had been called by Hermes. Several of these procedures are demonstrated in the Update Phone sample external. First, let’s take a look at the bCR procedure. It is defined as follows in the HermHeaders.p file:

```
procedure bCR (theRout: ProcPtr);   {selector = 0}
```

This procedure acts like pressing a return. It will cause to the cursor to move down one line and back to horizontal position zero. If the cursor is at the bottom of the screen, the screen will scroll up. This procedure automatically outpus the necessary characters to reproduce this on the remote screen. Specifically, a carriage return (13) followed by a line feed(10). Since there is no jump table like in a normal application, the procs field of the UserXIrec serves as our pseudo jump table. When you call bCR, you must pass its address as the last parameter. The inline code in HermHeaders.p will handle the call from there. So, you would do this to call the procedure:

```
bCR(procs[0]);
```

Each procedure is listed at the bottom of HermHeaders.p with its appropriate selector. Just pass Procs[] with that selector for theRout parameter. Now let’s take a look at the other procedures available to you.

```
procedure OutLine (goingOut: str255; NLatBegin: boolean; typeLine: integer; theRout: ProcPtr);
{selector=1}
```

The OutLine procedure handles almost all character output to the user screen. It outputs the string out the port, draws it on the screen, puts it in the buffer, and handles ANSI for you. The **goingOut** parameter is the actual text you want to output. It should not contain control characters, and should be limited to standard text. The **NLatBegin** field is just a convenience. If this is true, the bCR procedure will be called directly before outputting the text in goingOut. The **typeLine** field represents the ANSI color for this text. If this field is a number from 0-7, the text will be drawn in the user color corresponding to that number. If it is equal to -1,

the current ANSI configuration will not be changed. For more complete control over ANSI, see the ANSICode procedure described later.

procedure HangUpAndReset (theRout: ProcPtr); {selector = 2}
This procedure has no parameters besides its jump vector. Call this if you want to force an immediate logoff and reset of the current node. This procedures handles hanging up the modem and returning to waiting mode.

procedure PromptUser (whichNode: integer; theRout: ProcPtr); {selector = 3}
You will rarely use this procedure directly. Usually, you will use on of the procedures described later for your prompts. This procedure is provided mainly for extremely complex prompts that do not fit into any of the standard categories used by Hermes. If you call this directly, it is your responsibility to set the fields of the prompting record that can be found in the HermHeaders file. Each node has its own prompting record in a variable named **myPrompt** which you may feel free to modify. When you call any prompting procedure, you should return to Hermes quickly. You will be called again when the prompt has been completed. If you receive idle messages, you will be called during the prompt, but you should not take any actions on a node being prompted. If you think you need this method, see the ReprintPrompt procedure below. The user's input to the prompt will be contained in the node variable **curPrompt**. See the Update Phone external for a demonstration of how to prompt a user.

procedure LettersPrompt (prompt, accepted: str255; sizeLimit: integer; auto, wrap, capital: boolean; replace: char; theRout: ProcPtr); {selector=4}
This is a very common prompting procedure, and it is very powerful. It is easier than filling all the fields of the promptRec yourself, and provides most functions. The **prompt** field is the text of your prompt. The **accepted** field is a string containing each character that will be allowed to be typed. This does not apply to numbers, but does include every other character including *&^%$#@!()'", etc... The **sizeLimit** field specifies how many characters the maximum length for the prompt input will be. The **auto** field indicates whether Hermes should auto-accept input when it deems it accpetable. If the **wrap** field is true, the prompt will have word-wrapping abilities. The wrapped text will be placed in the node variable **excess**. You must then output that variable when Hermes calls you again to effect the actual wrapping. The **capital** field set to true will cause all letters typed to the prompt to be capitalized before being processed. The **replace** field is for hidden entry type prompts like entering passwords where the screen should show characters different from what is entered. Whatever this character is will replace all input on the remote and local screens. But the **curPrompt** variable will still contain the accurate entry.

procedure NumbersPrompt (prompt, accepted: STR255; high, low: integer; theRout: ProcPtr); {selector=5}
If your prompt is number-based, you should use the NumbersPrompt procedure instead. Even if your prompt accepts some characters, NumbersPrompt is often more appropriate. The **prompt** field is the text of your prompt. The **accepted** field is a string containing the characters allowed to be typed. If this is an empty

string, all characters are allowed. The **high** field is the maximum numerical value this field may return. Hermes will enforce this at the prompt. The **low** field can be either 0 or 1 depending on what you want. Setting high lower than low will cause all numerical input to be disallowed. If numerical input has been entered, it will be returned in the **curPrompt** string as with all other prompt and you should then use the toolbox procedure StringToNum to convert it to a number if necessary.

procedure YesNoQuestion (prompt: STR255; yesIsDefault: boolean; theRout: ProcPtr); {selector=6}
This prompt is for standard yes/no questions. The **prompt** field acts like it does with the other prompts. The **yesIsDefault** field controls what happens when a return is pressed. If this field is true, a return will cause a yes answer, otherwise it will default to no.

procedure GoHome (theRout: procPtr); {selector=7}
This procedure is for convenience only. Basically, it sets the appropriate variables so that the current board position is at the main menu. The **inTransfer** variable controls whether this is the transfer main menu or the normal main menu. This procedure consists essentially of two commands. BoardSection:=MainMenu, and MainStage:=MenuText. Use this if you ever want to exit out of your external back to the main menu instead of to the external menu.

procedure BackSpace (howMany: integer; theRout: procPtr); {selector=8}
If for some reason you want to cause an immediate backspacing that deletes the previosu character typed, use this procedure. It handles all drawing and output to both remote and local screens. The **howMany** variable will loop through this procedure howMany times.

procedure ANSIPrompter (numChars: integer; theRout: ProcPtr); {selector=9}
This procedure is used often in Hermes. It uses the user's ANSI color number 4 which defaults with a blue background to draw **numChars** spaces which creates the effect of a special background being typed on. For instance, this is used when entering the name of a file to download in Hermes. Call this immediately after your prompt procedure and then exit back to Hermes if you want this effect.

function ReadTextFile (fileName: str255; storedAs: integer; insertPath: boolean; theRout: ProcPtr): boolean; {selector=10}
This function reads all text files. The **fileName** parameter either contains a file name on disk or a resource name for a Hermes HTxt resource. The **storedAs** parameter controls what type of text entity is being read. If this equals 0, the filename refers to a disk file, if it equals 1 the fileName is a resource of type HTxt. The **insertPath** variable will insert the Hermes Files path before the filename for disk based files. It has no effect when reading HTxt resources. When this procedure completes, the text is ready to be listed. You can cause the text file to be listed to the user by setting the **BoardAction** variable to ListText. It will then automatically be listed out. You should exit your external after setting that variable.

procedure RePrintPrompt (theRout: ProcPtr); {selector=11}
If for some reason you had reason to take over during a prompt and want to restore the state of the system, just call this routine. It handles reproduction of the prompt and whatever may have already been typed. It also handles the appropriate ANSI colors.

procedure OutChr (theChar: char; theRout: ProcPtr); {selector=12}
This is for outputting all the things that OutLine does not cover. But this only accepts one character. But it has lots of special functions. If called with ASCII character 12, this will clear the screen. If the user is using ANSI, this character will be translated to the ANSI screen clear command. If called with ASCII character 7, it will cause the remote computer to beep.

procedure ANSICode (theCode: str255; theRout: str255); {selector=13}
This procedure handles raw ANSI. Just put your ANSI code in **theCode** and it will execute the proper actions both on the remote screen and local screen. You must check to make sure the user has ANSI turned on before using this. That variable is contained in **thisUser.canANSI**, and if you plan to use color you must check **thisUser.ANSIcolor**. Do not include the escape-bracket sequence before your ANSI commands. I.E. to clear the screen you would use:
ANSICode('2J', procs[13]);

procedure PAUSEPrompt (prompt: str255; theRout: ProcPtr); {selector=14}
This procedure acts much like the other prompting procedures. In Hermes, this is always called with **prompt** equal to "[PAUSE]". If you want to make it say something else, feel free to do so. This procedure is used when you want a pause to be hard coded. Hermes will automatically pause the screen if you have not prompted after the user's specified screen height has scrolled up.

function FindUser (SearchString: str255; **var** UserFound: UserRec; theRout:ProcPtr): boolean; {selector=15}
This procedure is often used in Hermes to find user accounts and load them from disk. The **SearchString** parameter can be either a user number or a user name. The input from the "NN" prompt at logon is passed directly to this procedure, as is the input whenever Hermes wants to look up a user. This procedure is highly optimized, and should be used whenever you have to do this kind of thing. The entire user account will be passed back in **UserFound**. Remember that **UserFound** is a var parameter, and you should allocate this either as a local variable or in your PrivsHandle.

Last Words

I could go on forever describing every little variable in the nodes variables. But this would end up confusing many and enlightening few. I have tried to point out the major variables in this document. BoardSection is probably the most important. If set to EXTERNAL, along with the variable activeUserExternal set to your extID, you will be called with the activeExt message. BoardAction can be either prompt, listText, or several others. When it is set to none, your external will be called. The best idea is to thoroughly go over the Update Phone external and

familiarize yourself with the HermUserGlobsRec. For many externals, you wont have to know 90% of the information contained here. I would be glad to answer specific questions about externals on the support BBS. Any serious external author will want to be on this BBS at (213)275-6975.