

マイコン部 2023

講習会資料

第4回

目次

第 4 章 発展内容

4-1 オブジェクト・ウィンドウ ID

4-2 関数

4-3 ファイル操作

4-4 割り込みラベルジャンプ

4-4.1 割り込み処理

4-4.2 キー・マウスの読み取り発展版

4-5 プラグイン

4-5.1 HSP のプラグインとモジュール

4-5.3 それ以外のプラグイン

第 4 章 発展内容

第 0 章から第 3 章までは HSP の基本的な機能、命令、簡単なプログラムの作成を行ってきました。そこで、第 4 章ではこれまで触れなかった追加の内容や HSP の裏側の処理、HSP 以外の外部プラグインなど発展的な内容を解説します。どれも内容的にはかなり高度なものとなるのでまだ HSP に慣れていない人はそこまで気にしなくても大丈夫です。HSP の扱いに慣れてきたらゆっくり読み進めてみてください。

4-1 オブジェクト・ウィンドウ ID

まずは、オブジェクト ID とウィンドウ ID の詳細についての説明です。

オブジェクト ID

1 つ目はオブジェクト ID です。第 2 回のオブジェクトの説明の際に少しだけ触れました。オブジェクト ID は HSP でオブジェクト (button 命令、checkbox 命令、mesbox 命令、input 命令、combox 命令、listbox 命令) や win32 で適用される命令設置するごとに、0 から順番に割り当てられる数値です。HSP では複数のオブジェクトを扱うため、objprm 命令や clrobj 命令などを適用するために特定のオブジェクトを指定したいといった場合にどのように指定すればいいのか困ってしまいます。その場合に使えるのが、通し番号であるウィンドウ ID というわけです。(オブジェクト命令については第 2 章オブジェクト節参照)

オブジェクト ID は HSP 内では非常によく使われるものですが、このオブジェクト ID は HSP が独自に用意しているもので、Windows 上では「ウィンドウハンドル」といわれる内部識別番号がオブジェクトを配置するたびに割り振られています。したがって、HSP のウィンドウ ID は、そのままでは後述する外部 API で扱うことができません。そのためには該当するオブジェクトのオブジェクト ID からウィンドウハンドルを取得する必要があります。

外部 API(DLL)等にウィンドウオブジェクトのハンドル(HWND)を渡す場合などでこのウィンドウハンドルを取得する場合は、objinfo 命令で第 2 パラメータに 2 番を指定することで実現することができます。

第4章 発展内容

```
val = objinfo(p1,p2) //ウィンドウオブジェクト情報の取得(object infomation)
```

p1=0～：ウィンドウオブジェクト ID

p2=0～：取得するタイプ

また、objinfo 命令は p2 に入力する値を変えることで、様々な情報を取得することができます。

p2 の値	内容
0	モード(下位 16bit)及びオプションデータ(上位 16bit)
1	object が配置されている BMSCR 構造体のポインタ
2	ウィンドウオブジェクトのハンドル(HWND)
3	owid (内部汎用データ)
4	owsize (内部汎用データ)
5~8	代入変数の情報
9~11	ウィンドウオブジェクトコールバックの情報

ウィンドウ ID

続いては、ウィンドウ ID です。といっても内容自体はほとんどオブジェクト ID と変わりません。ウィンドウ ID も screen 命令でウィンドウが作られた場合 0 から順番に割り当てられる ID で、HSP が独自に用意しているものです。windows 上ではウィンドウハンドルとして管理しています。

objinfo 命令と同じように、ginfo 命令の p1 に 2 番を指定することで、現在アクティブになっているウィンドウ ID を取得することができます。(HSP 以外のウィンドウの場合返り値は-1) さらに、p1 の値を変更することで、様々なウィンドウ情報を取得することができます。p1 で設定できる値は非常に多いため(約 30 種類)、詳細な内容はドキュメントライブラリなどで確認してください。

```
val = ginfo(p1) //ウィンドウ情報の取得
```

p1=0～：取得するタイプ

第4章 発展内容

注意点として、ginfo 命令は objinfo 命令と違い、特定のウィンドウ ID の情報を取得するものではなく PC のスクリーン全体の情報を取得する命令だということに気を付けてください。

4-2 関数・プリプロセッサ命令

次は関数についてです。関数とは、与えられた値をもとに、定められた独自の処理を実行し、その結果を返す命令のことです。つまり、これまで使用してきた様々な命令のオリジナル ver を作成することができる機能のようなものです。HSP にも使い勝手は特殊ですが一応関数のようなものは存在します。

HSP で関数を利用するにはプリプロセッサ命令を利用します。プリプロセッサ命令とはその名の通り、事前に（プリ）処理をする（プロセッサ）命令のことです。この命令はほかの命令と異なる特殊な動きをします。HSP 上では hspcmp.dll により実現されており、コンパイル（オブジェクトファイル作成）に先駆けて行われる前処理を行います。

プリプロセッサ命令には以下のようなものがあります。

↓プリプロセッサ命令の例

```
#comfunc #defcfunc #deffunc #define #func #global #module #modcfunc #modinit  
#modterm #addition #include
```

中には、第3章で用いた #module 命令も含まれています。これらはそれぞれ様々な機能を持ちますが、この節では関数に関係する #module #global #deffunc #defcfunc を説明します。

命令の登録

まずは #deffunc 命令です。これは、新規に「命令」を定義する際に使用します。

```
#deffunc p1 p2 p3,... //新規命令を登録する
```

p1 : 割り当てられる命令の名前

p2 p3~ : パラメータタイプ名・エイリアス名

上記のままではわかりにくいので、実際の使われ方を下に示します。

第4章 発展内容

#deffunc 命令の名前 パラメータ

命令のためのプログラム

return

1行目に#deffunc 命令を記述し、2行目以降にその命令のプログラムを記述します。すべて記述し終わったら、最後に return を置いて命令のためのプログラムを囲います。

p2のパラメータタイプ名やエイリアス名には引数となるパラメータタイプやエイリアスが入りますが、指定するエイリアス名は渡されたパラメータの内容を示すためほぼ変数と同じように使うことができます。パラメータ名に関しては指定できるのは以下の通りです。

int	: 整数値	str	: 文字列
var	: 変数(配列なし)	double	: 実数値
array	: 変数(配列あり)	local	: ローカル変数

注意点として、deffunc 命令は必ず return とセットで扱うこと、命令の名前がすでにあるものと重複しないこと、後述する module~global 内で使用することなどがあります。

それでは実際に deffunc 命令が使用されたプログラムを見てみましょう。

```
1 #module //モジュール始まり
2 #deffunc Hi //Hi命令
3     mes "Hi there!" //命令が実行されると"Hi there!"と表示
4     return
5
6 #deffunc japan int Hour //Japan命令
7     if (Hour < 6 or Hour > 18){ //6時より前か18時より後ならこんばんは
8         mes "こんばんは" //それ以外はこんにちはと表示
9     }else{
10         mes "こんにちは"
11     }
12     return
13
14 #deffunc Australia int Jikan, int Hun //Australia命令
15     H = jikan //オーストラリアの時刻を表示する
16     M = Hun
17     if (M >= 30){
18         M -= 30
19     }
20     else{
21         M += 30
22         if (H = 0){
23             H = 23
24         }else{
25             H -= 1
26         }
27     }
28     mes "オーストラリアの時間は" + H + "時" + M + "分です"
29     return
30 #global //モジュール終わり
31
32 Hour = gettime(4)
33 Minute = gettime(5)
34 mes "現在は" + Hour + "時" + Minute + "分"
35 Hi //Hi命令
36 japan Hour //Japan命令
37 Australia Hour, Minute //Australia命令
```

Hot Soup Processor ver.3.6
現在は6時47分
Hi there!
こんにちは
オーストラリアの時間は6時17分です

第4章 発展内容

上記のプログラムは、モジュール内に「実行されると『Hi there!』と表示してくれる命令 Hi」と「実行されると現在の時間によって『こんにちは』か『こんばんは』のどちらか表示してくれる命令 Japan」、「実行されると今の 30 分前の時刻を表示してくれる命令 Australia」の 3 つを定義しています。そして、35 行目～37 行目でそれぞれの命令を実行しています。プログラムを見ると分かるように、作成した命令にはパラメータを定義しない/2 つ以上のパラメータを定義することもできます。

関数の登録

次に#defcfunc 命令です。これは、新規に「関数」を定義する際に使用します。

```
#defcfunc p1 p2 p3,⋯ //新規関数を登録する
```

p1 : 登録する関数の名前

p2 p3～ : パラメータタイプ名・エイリアス名

こちらも先ほどと同じように、実際の使われ方を記述すると以下ようになります。

```
#defcfunc 関数の名前 パラメータ
```

関数のためのプログラム

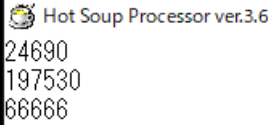
```
return 返却値
```

#deffunc 命令の時とは違い、return の後に返却地（戻り値）が追加されました。それ以外の使用や注意点は#deffunc 命令とほぼ同じです。

それでは関数を利用したプログラムを見てみましょう。

第4章 発展内容

```
1 #module
2
3 #defcfunc bai int p1 //入力された値の2倍の値を返す関数bai
4 return p1 * 2
5
6 #defcfunc tasu array para //入力された配列の1番目と2番目を足した
7     goukei = para.0 + para.1 //値を返す関数tasu
8 return goukei
9
10 #global
11
12     val1 = 12345
13     mes bai(val1)
14
15     val2 = 98765
16     mes bai(val2) //関数は再利用することができる
17
18     value = 12345, 54321
19     mes tasu(value) [EOF]
```



上記のプログラムはモジュール内に「実行されると入力された値の2倍の値を返す関数 bai」と「入力された配列の1番目と2番目を足した値を返す関数 tasu」を定義し、13行目、16行目、19行目でそれぞれ実行してその結果を表示しています。ちなみに、基本的に return には1つの変数しか指定できませんが、複数の戻り値が欲しい場合は配列変数(array 型)を使用することで複数の戻り値を配列として指定することができます。

コラム：命令と関数の違い

これまで、#deffunc 命令と#defcfunc 命令を確認しました。この2つの違いは新規に「命令」を登録するか「関数」を登録するかでしたが、この違いはいったい何なのでしょう。

関数と命令、お互いに指定して実行されると何かしらの計算をする機能ですが、少し違いがあります。何が違うのかというと、コンピュータに計算をさせた後に結果（戻り値）を返してくれるかどうかの違いです。

命令：命令が読み込まれる→コンピュータが命令を実行（計算）→終了

関数：関数が読み込まれる→関数を実行→**戻り値を返却する**→終了

これまでに勉強した命令の中に、戻り値が帰ってくるものがあつたかと思います。（input 命令（入力された内容）など）もしそれらを自作する場合は関数となり、反対に mes 命令や stop 命令など戻り値のないものは命令となります。（例外的に deffunc 命令のパラメー

第4章 発展内容

タに変数である var を設定すると結果を受け取ることができますが、defcfunc 命令を使用した方がスッキリと記述できると思います。)

モジュール

先ほど説明した#deffunc, #defcfunc 命令は基本的に、モジュール区間内に記述することが推奨されています。なぜなら、モジュール内に入れることによって局所変数（モジュール内のみで利用できる変数のこと）を利用することができ、メインのプログラムと変数が被ったりすることを避けることができるためです。

```
#module モジュール名 変数名 1,... //モジュールの開始
```

モジュール名：新規モジュール名（18 文字以内・重複なし）

変数名：登録するモジュール変数名

```
#global //モジュールの終了
```

#module 以降の区間をモジュールとして別な空間に割り当て、#global でモジュール区間を終了し、以降を通常のプログラム領域に戻します。また、モジュール内の変数やラベルは、モジュール外のものからは独立したものになります。さらに、モジュールのパラメータは省略することができます。

注意点として、モジュール必ず#module と#global をセットで扱うようにしてください。

新規命令や新規関数を#module~#global でくる場合は以下を参考にしてください

第4章 発展内容

通常プログラム

```
#module //ここからモジュール
```

```
    #deffunc a b //モジュール内で定義された命令
```

```
        新規命令 a の処理
```

```
    return
```

```
    #defcfunc c d //モジュール内で定義された関数
```

```
        新規関数 c の処理
```

```
    return e
```

```
#global //ここまでモジュール
```

通常プログラム

4-3 ファイル操作・文字列操作

HSP でツールを作成したりゲームのセーブデータを保存する場所を作成したり、実際にセーブデータをメモ帳に保存したりする場合、ファイル操作が必要になります。ここではそのファイル操作について解説します。

ファイル操作命令

ファイル操作命令には以下のようなものがあります。

命令	内容
bcopy	ファイルのコピー
blood	バッファにファイルをロード
bsave	バッファにファイルをセーブ
chdir	ディレクトリ移動
delete	ファイル削除
dirlist	カレント（現在の）ディレクトリのファイル一覧を取得
exist	ファイルのサイズ取得（ファイルがあるか確認）
memfile	メモリストリーム設定

第4章 発展内容

mkdir	ディレクトリ作成
-------	----------

今回は、ファイル操作命令の中でも比較的によく使う `dirlist`, `exist`, `mkdir` 命令についてみてみましょう。

まずは `dirlist`(ディレクトリリスト)命令です。

```
dirlist p1,"filemask",p2 //ディレクトリ一覧を取得
```

p1=変数 : ディレクトリ一覧を格納する文字列型変数

"filemask": 一覧のためのファイルマスク (基本は *.*)

特定の拡張子のファイルのみを取得するといった場合に使う

p2=0~(0) : ディレクトリ取得モード

この命令は、実行されるとカレントディレクトリ（現在のディレクトリ、標準では HSP プログラムのファイルがある場所）のファイル一覧を取得して、p1 で指定した変数に代入します。また、p2 のモードを変更することで取得する内容を変更することができます。

モード :	取得される内容
0	: すべてのファイル
1	: ディレクトリを除くすべてのファイル
2	: 隠し属性・システム属性を除くすべてのファイル
3	: ディレクトリ・隠し属性・システム属性以外のすべてのファイル
5	: ディレクトリのみ
6	: 隠し属性・システム属性ファイルのみ
7	: ディレクトリと隠し属性・システム属性ファイルのみ

続いては `exist` 命令です。

```
exist "filename" //ファイルのサイズ取得
```

"filename": サイズを調べるファイルの名前

第4章 発展内容

filename"で指定したファイルが存在するかをチェックして、そのファイルサイズを取得します。exist 命令が実行されると、システム変数 strsize に結果が反映されます。ファイルが存在する場合は、そのファイルサイズが strsize に代入されます。もしファイルが存在しなかった場合は、-1 が strsize に代入されます。この仕様を利用して、そのファイルが存在するかどうかを調べるためによく利用されます。

最後に mkdir 命令です。

```
mkdir "dirname" //ディレクトリ作成
```

"dirname": 作成するディレクトリ名

"dirname"で指定した名前でディレクトリを作成します。ディレクトリは1階層先までしか作成することができません。mkdir 命令を実行する前に 必ず dirlist 命令でフォルダの有無を確認するようにしてください。（もしフォルダがない場合はエラーが出る）

文字列操作命令

また、メモ帳に文字を記入したり内容を読み込んだりする命令もあります。それらは文字列操作命令と呼ばれています。文字列操作命令として代表的なものには以下のようなものがあります。

命令	内容
cnvsstow	通常文字列を Unicode に変換
cnvwtos	Unicode を通常文字列に変換
noteadd	指定行の追加・変更
notedel	行の削除
noteget	指定行を読み込み
noteload	対象バッファ読み込み
notesave	対象バッファ保存
notesel	対象バッファ指定
strrep	文字列の置換をする
instr	文字列の検索をする

第4章 発展内容

この文字列操作命令と前述のファイル操作命令を組み合わせることによって、ゲームのセーブデータを保存する機能などを作ることができます。

4-4 割り込みラベルジャンプ

HSP には、クリックが行われたりキー入力が行われたり、エラーが発生したり、windows のダイアログが出た瞬間に事前に指定したラベルへジャンプする（割り込み処理）命令が存在します。これらの命令は一度実行されると、その後 stop や wait、await で一時停止している間にずっとキーやマウスの入力を監視してくれます。

4-4.1 割り込み処理

割り込み処理の関数には以下のようなものがあります。

命令	内容
onclick	クリック時割り込み実行指定
oncmd	windows メッセージ割り込み実行指定
onerror	エラー発生時割り込み指定
onexit	終了後（ウィンドウ右上×ボタン）割り込み指定
onkey	キー割り込み実行指定

oncmd 命令以外の表記方法はほぼ同じ(oncmd の場合のみラベル名の後ろにメッセージ ID を入力できる)なので、代表として onclick を見てみましょう。

```
onclick goto/gosub *label //クリック割り込み実行指定
```

*label: ラベル名

onclick はマウスのボタンを押したときに自動でジャンプする場所を指定します。

同じように、onerror では HSP 上のエラーが出たときにジャンプする場所を指定し、onexit はウィンドウ右上にある赤い×印が押されたとき、onkey はキーボードのキーが押されたときにジャンプする場所を指定します。

注意点として onexit 命令は一度実行すると end 命令以外に終了する方法がなくなる（一応タスクキルで対処可能）ので必ず end 命令を記述してください。

第4章 発展内容

また、割り込み関数はいずれも命令の直後に 0 を入力して実行すると一時的に割り込みを停止し、1 を入力すると割り込みを再開させることができます。

例：onclick の場合の割り込み処理の ONOFF

`onclick 0` //割り込み一時停止

`onclick 1` //停止した割り込みの再開

また、各命令を使用して割り込みジャンプを行うと、システム変数 iparam, wparam, lparam に値が代入されます。各命令によって代入される内容は以下の通りです。

割り込み 要因	iparam		wparam		lparam
onclick	マウスボタン ID		ボタン番号		マウスポインタの クライアント座標 位置 下位ワードに X 座 標、上位ワードに Y 座標(mousex, mousey と同じ)
	0	左ボタン	1	マウスの左ボタン	
	3	右ボタン	2	マウスの右ボタン	
	6	中ボタン	4	+Shift キー	
			8	+Ctrl キー	
			1 6	マウスの中ボタン	
oncmd	メッセージ ID		window メッセージ※として 渡されたパラメータ		左に同じ
onerror	0(なし)		エラー番号		エラー発生行番号
onexit	終了要因		終了の通知を受けたウィン ドウ ID		0(なし)
	0=ユーザーの意思によ る通常の終了 1=システムによる終了 (再起動または電源 OFF)				
onkey	文字コード (getkey と同じ)		仮想キーコード (getkey と同じ)		いくつかのキー押 し関連情報がビッ ト単位で返る

oncmd 命令は、割り込みが発生したウィンドウ ID を ginfo 関数で取得することができます。

第4章 発展内容

※Window メッセージとは、Windows がアプリケーションウィンドウに対して送る通知のことで、アクティブなウィンドウに対して何かしらの操作（マウス、キーボード、ボタン押下など）を行ったときに Windows が検出してアプリケーションに知らせます。したがって、ウィンドウに対して何かしらのアクションがあった場合に oncmd 命令は動作します。

Window メッセージの例

定数名	値	意味
WM_CREATE	0x0001	ウィンドウが作成された
WM_DESTROY	0x0002	ウィンドウが破棄されようとしている
WM_PAINT	0x000F	ウィンドウを再描画する必要がある
WM_SIZE	0x0005	ウィンドウサイズが変更された
WM_LBUTTONDOWN	0x0201	マウスの左ボタンが押された
WM_LBUTTONUP	0x0202	マウスの左ボタンが離された
WM_LBUTTONDOWNBLCLK	0x0203	マウスの左ボタンがダブルクリックされた
WM_RBUTTONDOWN	0x0204	マウスの右ボタンが押された
WM_RBUTTONUP	0x0205	マウスの右ボタンが離された
WM_RBUTTONDOWNBLCLK	0x0206	マウスの右ボタンがダブルクリックされた
WM_KEYDOWN	0x0100	キーが押された
WM_KEYUP	0x0101	キーが離された
WM_CHAR	0x0102	文字が入力された

4-4.2 キー・マウスの読み取り発展版

割り込み処理はプログラムが wait、await、stop のいずれかにいるときはいつでもイベントを監視しているため、これらの割り込み処理を活用して、第2章で行ったキーやマウスの読み取りをより高度に行うことができます。

```
1 //押されたキーをウィンドウに表示するプログラム
2 onkey *jump
3 /*
4 ★この部分に好きな処理を書くことができる★
5 */
6 stop
7
8 *jump//何かキーが押されるとジャンプする
9     pos xvalue, 0
10
11     mes strf("%c", iparam)//押されたキーの文字コードを表示
12     xvalue += ginfo(14)//x座標を1文字分ずらす
13
14     stop[EOF]
```

Hot Soup Processor ver.3.6
A1UEOKONNITIHA

上記のプログラムは押下されたキーボードのキーの文字を表示させるプログラムです。実行されたプログラムは6行目のストップで停止しますが、何かキーが押されたときに8行目の jump サブルーチンに飛びます。その後押されたキーを表示して14行目でストップしますが、もう一度何かキーを押すと再び8行目にジャンプします。このように、プログラムの実行中にいつでも押されたキーを取得したり、判別したり、キーが押されたときに特別な処理を行ったりすることができます。

4-5 プラグイン

ここまではすべて、HSP に初めから登録されている、いわゆる「標準命令」のみを取り扱ってきました。しかし HSP には標準命令以外にも利用することができる、プラグインと呼ばれる追加ファイルがあります。最後にそのプラグインについて見てみましょう。

4-5.1 HSP のプラグインとモジュール

HSP には最初からいくつかのプラグインやモジュールと呼ばれるものが同封されています。これらを使用することで新たな命令や関数などの機能を追加することができます。

プラグインやモジュールを利用するには include 命令を利用します。


```
#include "filename" //別ファイルを結合
```

"filename" : 結合するファイル名

HSP はプラグインやモジュールであっても、最初から同封されているものに関してはドキュメントライブラリや HSP アシスタント、インターネット等に説明があります。それらを参考にしてみましょう。

コラム：プラグインとモジュールの違い

プラグインとモジュールの違いは以下の通りです。

HSP 拡張プラグインは、標準で DLL または HPI の拡張子を持つファイルとなっています。このファイルは、必ず HSP 本体(HSP3.EXE)と同じディレクトリに存在していなければなりません。また、EXE ファイル作成の際に PACKFILE に入れて、1つの EXE ファイルにすることはできません。拡張プラグインを使用する EXE ファイルを作成した場合は、EXE ファイルと同じディレクトリに拡張プラグインを置いてください。

モジュールは、拡張プラグインと同様に機能を追加するためのファイルですが、ソーススクリプトに追加する形で使用します。HSP 拡張プラグインと違い、モジュールの場合は、EXE ファイルと同じディレクトリにファイルを置く必要はありません。

プラグインの例

名称	説明
HSPEXT.DLL	標準の機能以外を幅広くサポートする、機能拡張プラグインです。レジストリ操作、簡易数学関数、シリアル通信、拡張画像操作、拡張ファイルアクセスなど多くの機能が HSP からコントロールできるようになります。
HSPINET.DLL	インターネットアクセスのためのプラグインです。 http、ftp サーバーへ手軽にアクセスすることが可能になります。
HGIMG3.hrt	高性能 2D・3D 画像処理機能付きランタイムです。 2D・3D グラフィックス表示をサポートした DirectX 専用のプラグインになります。
OBAQ.DLL	物理エンジン(OBAQ)が持つ機能を手軽に利用することができるプラグインです。物理演算による自然な動きを実現することができるよう、2D に特化し、設定や命令もシンプルにまとめられています。
HSPCV.DLL	OpenCV が持つ機能を手軽に利用することができるプラグインです。
HSPDB.DLL	データベースアクセスのためのプラグインです。 ODBC を経由して各種データベースへ アクセスすることが可能になります。
HSP3IMP.DLL	HSP ランタイム機能を持つプラグインです。 パーツを貼り付ける要領で、HSP3 の画面を利用することが可能です。また、HSP だけでなく、C/C++といった一般的な言語で作成されたアプリケーションに HSP の機能を付加することができます。

モジュールの例

名称	説明																
d3module	HSP の標準ランタイムで簡易 3D 表示を行なうためのモジュールです。手軽に 3D 的な表示を利用したい時などに威力を発揮します。																
SQLele	HSP で SQLite を簡単に扱うためのモジュールです。 ディアプレイヤーのファイルリストや、アドレス帳やカレンダーのデータ、画像処理ソフトの Undo/Redo 管理、ゲームの中のキャラの台詞、セーブデータやハイスコアなどのアプリケーション内部のデータを SQL で管理することができます。																
hsp3util.as	HSP3 で使用するための便利な命令が定義されています。文字列の修飾や 1 文字ずつ表示、日付や時刻の文字列取得、配列と文字列の相互変換機能などが実装されます。																
hspmath.as	高度な数学計算を行なうための定数やマクロが定義されています。 円周率 π などの定数や、ラジアン・度の相互変換などの関数が追加されます。																
mod_img.as	画像を読み込むための imgload 命令が追加されます。 BMP,JPEG,GIF,ICO,PNG フォーマットを読み込むことが可能です。																
mod_menu.as	メニューバーを作成し、選択などの動作をサポートするためのモジュールです。Windows がサポートするメニューバーの基本的な機能を利用することができます。																
Win32API 定義	Win32API の呼び出し定義ファイルです。このファイルを#include 命令により取り込むことで、システム定義の関数を呼び出すことができます。 <table border="1" data-bbox="469 1391 1026 1783"> <tr> <td>advapi32.as</td><td>advapi32.dll 定義</td></tr> <tr> <td>comctl32.as</td><td>comctl32.dll 定義</td></tr> <tr> <td>comdlg32.as</td><td>comdlg32.dll 定義</td></tr> <tr> <td>gdi32.as</td><td>gdi32.dll 定義</td></tr> <tr> <td>kernel32.as</td><td>kernel32.dll 定義</td></tr> <tr> <td>shell32.as</td><td>shell32.dll 定義</td></tr> <tr> <td>user32.as</td><td>user32.dll 定義</td></tr> <tr> <td>winmm.as</td><td>winmm.dll 定義</td></tr> </table>	advapi32.as	advapi32.dll 定義	comctl32.as	comctl32.dll 定義	comdlg32.as	comdlg32.dll 定義	gdi32.as	gdi32.dll 定義	kernel32.as	kernel32.dll 定義	shell32.as	shell32.dll 定義	user32.as	user32.dll 定義	winmm.as	winmm.dll 定義
advapi32.as	advapi32.dll 定義																
comctl32.as	comctl32.dll 定義																
comdlg32.as	comdlg32.dll 定義																
gdi32.as	gdi32.dll 定義																
kernel32.as	kernel32.dll 定義																
shell32.as	shell32.dll 定義																
user32.as	user32.dll 定義																
winmm.as	winmm.dll 定義																

標準でサポートされているプラグイン/モジュールはほかにもいろいろあります。詳しくは HSP の作者おにたまさんの WEB サイト「HSP 拡張プラグイン・モジュール一覧」を確認してください。

4-5.3 それ以外のプラグイン

HSP 拡張プラグインは、仕様が公開されていて、ユーザーの方たちが制作した 多くのプラグインが存在します。HSP の可能性を大きく広げるプラグインが、HSP オフィシャルホームページでも多数紹介されています。

また、拡張プラグインやモジュールを自作することができます。作成するための詳細な仕様については、HSP 開発キット(HSPSDK)フォルダを参照してください。