světelné panely.cz

PANEL II
Programování plug-ins

Obsah:

1. Popis rozhraní

Program Panel II je od verze 5,0 doplněn o rozhraní plug-ins. Pomocí tohoto rozhraní lze program rozšířit o další funkce. Program po spuštění prohledá direktorář PlugIns a načte všechny dll knihovny a následně je inicializuje. Načtené moduly a jejich pracovní funkce jsou přidány do menu „**Nástroje**“. Jedna knihovna může obsahovat více pracovních funkcí. V plug-in je možné volat funkce knihovny PDriver.dll.

Program může při spuštění pracovní funkce předat část nebo celý text z editoru programu, toto je možné zvolit parametrem při inicializaci funkce. Po ukončení pracovní funkce se může zvolit způsob vložení nového textu. V knihovně může být použito vlákno pro zpracování na pozadí (například cyklicky načítat teplotu ze všech připojených panelů).

2. Popis funkcí

2.1 Inicializace

```
_declspec(dllexport) int __cdecl PlugInInit( int *ver, LPCSTR path );
```

Tato funkce je volána po načtení knihovny. Vrací počet funkcí které knihovna obsahuje, dále v parametru vrací verzi. Druhým parametrem program předává direktorář, ve kterém se nachází. Pokud program zjistí, že s touto verzí neumí pracovat, uvolní ji z paměti a více s ní pracovat nebude.

V této funkci může být umístěn kód inicializace, jako například vytvoření proměnných, spuštění vlákna a podobně.

```
char folder[MAX_PATH];              // direktorář aplikace

int PlugInInit( int *ver, LPCSTR path )
{
     *ver = 0x0100;                 // typ - verze plug-in
     strcpy( folder, path );        // direktorář programu Panel II

          // inicializace plug-ins

     return 2;                      // vrací počet funkcí plug-in
}
```

2.2 Ukončení

```
_declspec(dllexport) void __cdecl PlugInDestroy( );
```

Funkce je volána při ukončení programu, před uvolněním knihovny z paměti. Funkce nemá žádné parametry, a má odstranit dočasné proměnné případně ukončit pracovní vlákno, vytvořené v knihovně.

```
void PlugInDestroy( )
{
     // zrušení proměnných, ukončení vlákna …
```

```
}
```

2.3 Nastavení jazyka

```
_declspec(dllexport) void __cdecl PlugInLanguage( LPCSTR lang );
```

Tato funkce je volána po inicializaci a následně vždy po změně jazyka v programu. V parametr funkce předává jméno zvoleného jazyka. Pokud plug-in nemá volbu jazyka, nebude tato funkce provádět nic.

Jméno, které program předává je v souboru *.lang na prvním řádku a je ohraničeno znaky < >.

Příklad: Jazykový soubor programu **czech.lang**

<czech> česky
[STRING-PANEL2]
……..

Program ve parametru pošle string „**czech**"

```
void PlugInLanguage( LPCSTR lang )
{
  char file[MAX_PATH];
                                    // jméno souboru
  sprintf( file, "%s\\Language\\%s_plug-in.ln", folder, lang );

  LoadLanguage( file );             // načteni jazyka
}
```

2.3 Načtení pracovních funkcí

```
_declspec(dllexport) FNC_SETING* __cdecl PlugInGetFnc( int iFnc, void
                                                       **ptrfnc );
```

Po inicializaci a nastavení jazyka si program vyžádá nastavení jednotlivcích pracovních funkci, které dll knihovna obsahuje. Funkce je provedena tolikrát, kolik plug-in obsahuje pracovních funkcí. Funkce vrací pointer na strukturu exportované funkce. Dále funkce předá tabulku pointerů funkcí knihovny PDriver.dll. Každá exportovaná funkce je přiřazena do menu „**Nástroje**" v programu. Ve struktuře nastavení lze zvolit předání textu. Pokud je dotazováno číslo funkce, které knihovna neobsahuje, musí vrátit „***NULL***"

Volba předání textu v parametru

Definice cmd	Kód	Popis
PLUG_NO_TEXT	0x0000	Bez textu.
PLUG_TEXT_ALL	0x0002	Celý text.
PLUG_TEXT_FOCUS	0x0004	Označený text.
PLUG_TEXT_CMD	0x0008	Příkaz na pozici kurzoru.

```
// Struktura definice exportované funkce
typedef struct FNC_SETING
{
public:
     int    Atribut;          // typ parametru
     int    iMenu;            // Nepoužito
     int    iSubMenu;         // Nepoužito
     int    iPopUpMenu        // nepoužitu
     char  *ptrMenu;          // pointer na text menu
     char  *ptrPopupMenu;     // nepoužito
     char  *ptrMessageString; // pointer na text na stavové řádce
     HICON *hIcon;            // nepoužito
}FNC_SETING;

FNC_SETING SetingFnc[2];      // Nastavení funkcí
void **ptrExtFnc;             // Tabulka funkci Driver.dll

const char szMenu1[30]   = { "Funkce 1" };
const char szMenu2[30]   = { "Funkce 2" };
const char szString1[50] = { "Bez textu" };
const char szString2[50] = { "S textem" };

FNC_SETING *PlugInGetFnc( int iFnc, void **ptrfnc  )
{
     ptrExtFnc = ptrfnc;

     switch ( iFnc )
     {
     case 0 :            // první funkce
                         // nepředává žádný text
          SetingFnc[0].Atribut = PLUG_NO_TEXT;
          SetingFnc[0].ptrMenu = szMenu1;
          SetingFnc[0]. ptrMessageString = szString1;
          return &SetingFnc[0];
     case 0 :            // druhá funkce
                         // nepředává žádný text
          SetingFnc[1].Atribut = PLUG_TEXT_ALL
          SetingFnc[1].ptrMenu = szMenu2;
          SetingFnc[1]. ptrMessageString = szString2;

          return &SetingFnc[1];
     default : break;
     }
     return NULL;        // funkce není definovaná
}
```

2.4 Pracovní funkce

```
_declspec(dllexport) int __cdecl PlugInFncRun(
                                    int iFnc, PLUGIN_PARAM *fncParam );
```

Je-li v menu nástroje zvolena příslušná položka, volá program tuto funkci. V prvním parametru program předá index pracovní funkce, která je volaná, dále parametrem předává pointer na strukturu obsahující parametry volané funkce.

Pracovní funkce obsahuje tři servisní funkce. Dvě pro pozastavení a opětovného spuštění zpracování na pozadí. Tyto funkce používá program v případě, že uživatel otevře nastavení programu. Je to z důvodu, že pokud uživatel provádí změnu v nastavení panelů, mohlo by dojít při běhu plug-in na pozadí k zhroucení nebo zaseknutí programu. Pokud plug-in nemá vlákno pro běh na pozadí, nebudou tyto funkce provádět nic jak je tomu v příkladu. Poslední funkcí se program dotazuje na jméno plug-in a stav běhu na pozadí.

Servisní funkce

Definice cmd	Kód	Popis
PLUG_FNC_STOP	100	Zastavení vlákna
PLUG_FNC_RESTART	200	Znovu spuštění vlákna
PLUG_FNC_IS_RUN	300	Stav plug-in

Návratové funkce

Definice cmd	Kód	Popis
RETURN_CANCEL	0x0001	Funkce byla zrušena
RETURN_OK	0x0002	Ukončení
RETURN_FNC_NOEXIST	0x0003	Číslo funkce neexistuje
RETURN_DLL_FREE	0x0100	Nepoužito
RETURN_RUN	0x0200	Vlákno spuštěno
RETURN_STOP	0x0400	Vlákno zastaveno
RETURN_NOTHREAD	0x0800	Plug-in neobsahuje vlákno

```
// Parametr prováděcí funkce
typedef struct PLUGIN_PARAM
{
     int       id;             // id zvoleného panelu
     int       AtributReturn; // return parametr (funkce vlozeni textu)
     int       iFirstChar;     // první znak označeného textu
     int       iEndChar;       // poslední znak označeného textu
     int       iCursor;        // pozice kurzoru
     int       iSize;          // velikost datového pole
     char     *cText;          // pointer na datové pole textu
                               // pokud neobsahuje je NULL
}PLUGIN_PARAM;


int PlugInFncRun( int iFnc, PLUGIN_PARAM *fncParam )
{
```

```
    CPlugInsDlg pluginsdlg;

    switch( iFnc )
    {
    case 0 :                              // prováděcí funkce
            pluginsdlg.DoModal( );
            return IDOK;
    case PLUG_FNC_STOP :                  // ukončení vlákna
            return RETURN_OK;
    case PLUG_FNC_RESTART :               // restart
            return RETURN_OK;
    case PLUG_FNC_IS_RUN :                // stav vlákna
            strcpy( fncParam->cText, "plug-in" );
            return RETURN_NOTHREAD;
    }
    return RETURN_FNC_NOEXIST;
}
```

Po ukončení prováděcí funkce je nutné zvolit jakým způsobem bude text vložen zpátky do textu programu. To provedete nastavením „AtributReturn“ ve struktůře PLUGIN_PARAM

Definice cmd	Kód	Popis
PLIG_NO	0x0000	Bez funkce – text ignoruje
PLUG_TEXT_INSERT	0x0100	Vložení textu na pozici kurzoru
PLUG_TEXT_REPLEACE_FOCUS	0x0200	Přepsání vybraného textu
PLUG_TEXT_REPLEACE_ALL	0x0400	Přepsání celého textu
PLUG_TEXT_ADD	0x0800	Přidání na konec
PLUG_VIEW_FOCUS	0x1000	Zvýraznění textu

2.5 Ukončení pracovní funkce

```
_declspec(dllexport)  void __cdecl PlugInFncFinished( int iFnc );
```

Poslední funkce je volána vždy po pracovní funkci, jakmile program převezme text a zpracuje jej. V této funkci je možné například zrušit dočasné proměnné, vytvořené pracovní funkcí a podobně.

```
void PlugInFncFinished( int iFnc )
{
    // zrušení proměnných
}
```

2.5 Postup volání funkcí.

Následující ukázka je pouze názorná pro lepší představu, v jakém pořadí program jednotlivé funkce volá

```
int ver;
FNC_SETING fset[10];

// Spuštění programu
```

```
// načtení dll knihovny

                // inicializace
  int count = PlugInInit( &ver, folder );
                // načtení pracovních funkci
  for (int i=0; ;i++ )
     if ((fset[i] = PlugInGetFnc( i, pdrvfnc )) == NULL) break
                // nastaveni jazyka
  PlugInLanguage( lang )

// zvoleni funkce v menu
  PlugInFncRun( 0, &fncParam );

// zpracovaní textu

  PlugInFncFinished( 0 );

// ukončeni programu
  PlugInDestroy( );

// uvolnění dll knihovny z paměti
```

3. Volání funkcí knihovny PDriver.dll

Funkce knihovny PDriver.dll nejsou volány přímo ale přes program Panel II. Program předá při inicializaci knihovně tabulku pointerů jednotlivých funkcí. Hlavní program se pomocí semaforu stará o to, aby nedošlo k současnému volání funkcí. Více o jednotlivých funkcí se dozvíte v „Popis funkcí knihovny PDriver“

Tabulka funkcí

Definice cmd		Funkce
PD_FILES	0	`Int pdFiles( int cmd, char *nfile, int size );`
PD_CONECT	1	`Int pdConnect( int cmd_id );`
PD_STRINGS	2	`int pdStrings( int cmd_id, char *cstr, int size, int iParam );`
PD_STATICLINE	3	`int pdStaticLine( int cmd_id, int *iParam, char *cstr, int size );`
PD_PANELCONTROL	4	`int pdPanelControl( int cmd_id, int iParam, BYTE *data, int size );`
PD_ARRDATA	5	`Int pdArrData( int cmd_id, char *name, int iParam );`
PD_CONTROL	6	`int pdControl_fnc( int control );`
PD_TRANS	7	`int pdTranslator( int cmd_id, char *data, int *size, int *nFirst, int *nEnd );`
PD_PANELFILE	8	`int pdPanelFile( int cmd_id, char *cstr, int size, int iParam );`
PD_PANELFILECONTROL	9	`int pdPanelFileControl( int cmd_id, char *cstr, int size, int iParam );`
PD_PANELFILESYSTEM	10	`int pdPanelFileSystem( int cmd_id, BYTE *bdata, int size, int iParam );`
PD_FONT	11	`int pdFont( int cmd_id, BYTE *data, int size, int *iParam1, int *iParam2 );`

PD_DIRECTSENDPANEL	12	int pdDirectSendPanel(int cmd_id, BYTE *data, int iParam1, int iParam2);
PD_VAR	13	Int pdVariable(int cmd_id, char *cstr, int size, int *iParam1, int *iParam2);
	14	Volné
	15	Volné
	16	Volné
	17	Volné
GETPTRPANEL	18	LPVOID GetPtrPanel(int id);

Definice typů jednotlivých funkcí:

```
typedef int (PD_FILES)( int cmd, char *nfile, int size );
typedef int (PD_CONECT)( int cmd );
typedef int (PD_STRINGS)( int cmd, char *cstr, int size, int iParam );
typedef int (PD_STATICLINE)( int cmd_id, int *iParam, char *cstr, int size );
typedef int (PD_PANELCONTROL)( int cmd_id, int iParam, BYTE *data, int size );
typedef int (PD_ARRDATA)( int cmd_id, char *name, int iParam );
typedef LPVOID    (GETPTRPANEL)( int id );
typedef int (PD_CONTROL)( int control );
typedef int (PD_TRANS)( int cmd_id, char *data, int *size, int *nFirst, int *nEnd );
typedef int (PD_PANELFILE)( int cmd_id, char *cstr, int size, int iParam );
typedef int (PD_PANELFILECONTROL)( int cmd_id, char *cstr, int size, int iParam );
typedef int (PD_PANELFILESYSTEM)( int cmd_id, BYTE *bdata, int size, int iParam );
typedef int (PD_FONT)( int cmd_id, BYTE *data, int size, int *iParam );
typedef int (PD_DIRECTSENDPANEL)( int cmd_id, BYTE *data, int iParam1, int iParam2 );
typedef int (PD_VAR)( int cmd_id, char *cstr, int size, int *iParam1, int *iParam2 );
```

Příklad definice funkce:

```
#define pdConnect( cmd_id ) ((int)((PD_CONECT*)ptrExtFnc[1])( cmd_id ) )
```

Výrobce: **RTG Mělník** tel/fax. 315624739 mob. 603261914
www.rtg-tengler.cz email: rtg@rtg-tengler.cz
www.svetelnepanely.cz

Software: **Vacek Luboš** tel. 606485842 email: vacek@svetelnepanely.cz