Plugin-DLL

Inhaltsverzeichnis

Für die Kommunikation mit LOTUS müssen je nach Bedarf festgelegte Methoden exportiert werden. Im Folgenden soll deshalb erläutert werden, wie der Variablen-Zugriff genau von statten geht und welche Methoden zu exportieren sind. Grundsätzlich bestehen aber große Freiheiten beim Anlegen der DLL; beispielsweise ist es möglich, ein zusätzliches Fenster mit VCL-Komponenten darzustellen und zu verwenden.

Da – abgesehen vom "AOwner" – auf die Verwendung von Objekten/Klassen verzichtet wurde, ist es vermutlich möglich, die Plugin-DLL auch in einer anderen Programmiersprache als Object Pascal zu entwickeln. Getestet wurde von mir aber nur Object Pascal (Delphi). Dementsprechend ist das folgende Beispiel auch in dieser Programmiersprache verfasst.

Welche Methoden aber konkret auf welche Weise deklariert und exportiert werden müssen, um verschiedene Kommunikationsaufgaben zu übernehmen, wird anhand des folgenden Beispiels erläutert. **ACHTUNG: Im Abschnitt "exports" muss dringend auf die korrekte Groß- und Kleinschreibung geachtet werden!**

Code

```
1.  library Test;
2.
3.  uses
4.    SysUtils,
5.    Dialogs,
6.    Classes,
7.    TestU in 'TestU.pas' {Form1};
8.
9.  {$R *.res}
10.
11. procedure PluginStart(AOwner: TComponent); stdcall;
12. begin
13.   form1 := TForm1.Create( AOwner );
14.   form1.Show;
15. end;
16.
17. procedure PluginFinalize; stdcall;
18. begin
19.   form1.Free;
20. end;
21.
22. procedure ReceiveVarFloat(varindex: word; value: single); stdcall;
23. begin
24.   case varindex of
```

```
  0:
  begin
  form1.Label2.Caption := floattostrF( value, ffFixed, 5, 1 );
  form1.Gauge1.Progress := round( value );
  end;
  end;
  end;
procedure ReceiveVarBool(varindex: word; value: boolean); stdcall;
  begin
  case varindex of
  0:
  begin
  form1.Label2.Caption := BoolToStr(value);
  end;
  end;
  end;
procedure ReceiveVarInt(varindex: word; value: integer); stdcall;
  begin
  case varindex of
  0:
  begin
  form1.Label2.Caption := IntToStr(value);
  form1.Gauge1.Progress := value;
  end;
  end;
  end;
procedure OnConnectingVehicle(name: shortstring); stdcall;
  begin
  form1.Caption := name;
  end;
procedure OnUnconnectingVehicle; stdcall;
  begin
  form1.Caption := 'no bus loaded';
  end;
function SetButton(eventindex: word): boolean;
  begin
  case eventindex of
  0: result := Form1.button1_pressed;
  end;
  end;
function SetFloat(eventindex: word): single;
  begin
  case eventindex of
  0: result := (Form1.TrackBar1.Position - 15)/15;
  end;
  end;
exports
  ReceiveVarInt,
  SetButton,
  SetFloat,
  PluginStart,
  OnConnectingVehicle,
  OnUnconnectingVehicle,
  PluginFinalize;
```

86. begin
87. end.

Alles anzeigen
Typen- und Funktionsdefinitionen für C, C++ und C#

1 Initialisierung und Finalisierung

Nach dem Laden der DLL wird "PluginStart" aufgerufen. In diesem Fall soll eine Form erstellt werden; für diesen Zweck bietet die Prozedur den AOwner als Parameter an.

Vor dem Entladen der DLL beim Schließen von LOTUS wird "PluginFinalize" aufgerufen. In diesem Beispiel wird die Form aus dem Speicher gelöscht.

Unmittelbar nachdem ein Fahrzeug in den Fokus genommen wird – also nach dem Start der Simulation, nach dem Neuplatzieren eines Fahrzeuges oder beim Wechsel auf ein anderes Fahrzeug – und somit die DLL an das neu ausgewählte Fahrzeug andocken soll, wird die Prozedur "OnConnectingVehicle" ausgeführt. Diese übergibt als Parameter den Namen des neu ausgewählten Fahrzeuges. In diesem Beispiel wird der Name des Fahrzeuges als Titel auf die Form geschrieben.

Bevor ein Fahrzeug aus dem Fokus genommen werden soll (ebenfalls beim Wechsel, beim Löschen oder bei Rückkehr zum Hauptmenü) wird "OnUnconnectingVehicle" ausgeführt.

2 Lesezugriff

Der Lesezugriff erfolgt mit den Prozeduren "ReceiveVarBool", "ReceiveVarInt", "ReceiveVarFloat" und "ReceiveVarString". Der erste Parameter ist jeweils "varindex: word", der zweite ist "value: boolean", "integer", "single" oder "PWideChar", je nach Prozedur.

Der Ablauf ist folgender: LOTUS durchläuft die jeweilige Variablenliste in der Plugin-Ini und ruft jeweils die entsprechende "ReceiveVar..."-Prozedur auf und übergibt ihr den Index der Variable in der *.ini-Liste und den aktuellen Wert. Im Normalfall wird dann in der Prozedur ein Case-Konstrukt verwendet, um die Werte der Variablen entsprechend unterschiedlich zu verarbeiten.

Im Beispiel wird in der Prozedur "ReceiveVarFloat" dafür gesorgt, dass der Wert der Variable, die in der Plugin-Ini im [ReadingVarsFloat]-Abschnitt unter "var.0" eingetragen ist, in das Label2 der Form geschrieben wird und die Gauge1 entsprechend ausschlägt.

3 Schreibzugriff

Der Schreibzugriff erfolgt dahingehend, dass Tastatur-/Gamecontroller-Events ausgelöst werden (und damit der Druck von Tastenkombinationen oder Joystick-Knöpfen simuliert wird) und dass Float-Events ausgelöst werden, denen der neue Float-Wert beigefügt ist (was somit der Verarbeitung von Joystick-Achsen entspricht).

Anders gesagt: Die Plugin-Schnittstelle funktioniert dahingehend wie ein frei konfigurierbarer und beliebig komplexer Gamecontroller. Die Event-Namen sind dieselben wie bei der Konfiguration von Tastatur und Gamecontroller.

Für die Events gibt es die beiden Funktionen "SetButton" und "SetFloat". Beide übergeben als Parameter den Index (aus der Plugin-Ini). Die Funktion kann dann je nach Index den gewünschten Wert zurückgeben. Für diesen Zweck wird wiederum üblicherweise ein Case-Konstrukt verwendet. In diesem Beispiel wird in "SetButton" das erste Tastatur-Event in der Liste in der *.ini-Datei von der Variable "Form1.button1_pressed" gesteuert. Außerdem wird das erste Float-Event in der Liste in der *.ini-Datei in "SetFloat" entsprechend der Stellung des Reglers "TrackBar1" gesetzt.