RDS Spy

Freeware RDS Decoder for Windows with Plugin Support

Plugin API

Revision 2022-11-12

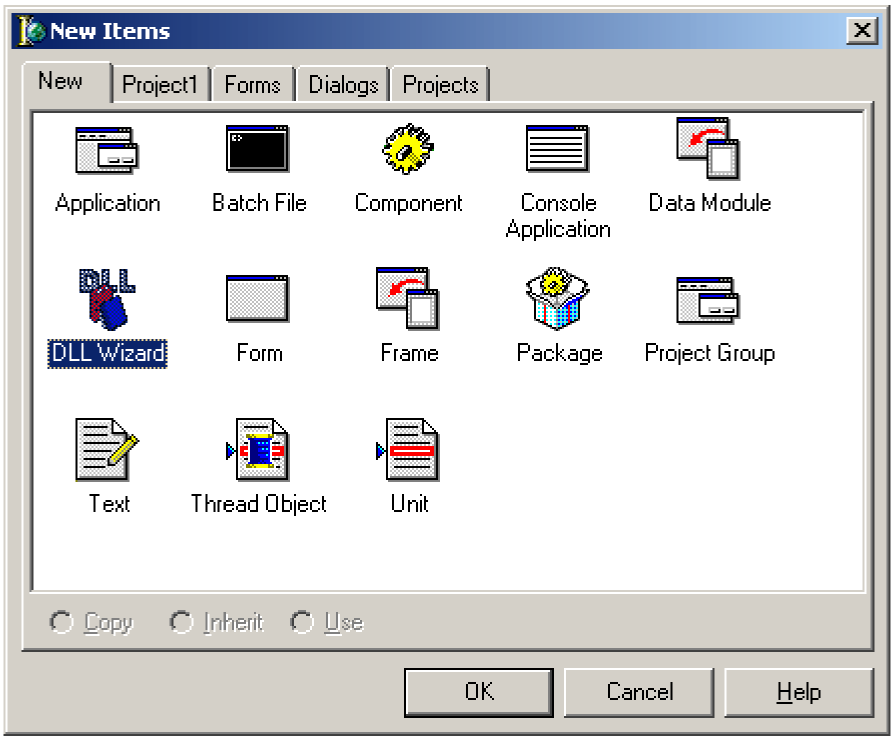
1. Creating Your First Plugin

A simple example is much better to understand than tens of boring pages of theory. Our first plugin will show a window with PI (program identification code) every time a new PI is detected.

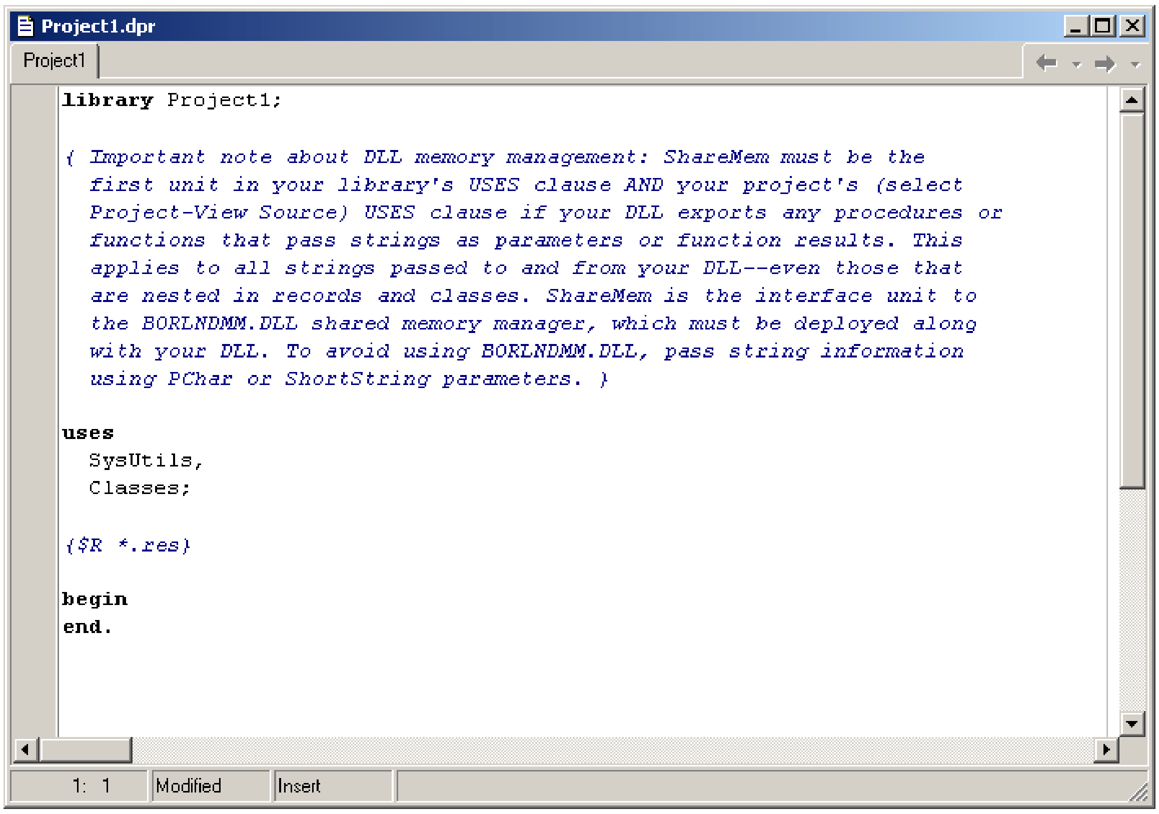
The example as well as the main application is created in Delphi. **Please don't ask us how to create plugins in C++, VB or any other programming language.** We have never tried it. But we are sure that it is as simple as in the Delphi. For anybody who has some experiences with programming, the conversion to his favourite programming language should be a piece of cake. Mr. Google provides many links related to this issue.

Are you ready? So run the Delphi and create a new DLL based project:

File / New / Other / DLL Wizard, OK



The basic DLL template will appear:



```
library Project1;

{ Important note about DLL memory management: ShareMem must be the
  first unit in your library's USES clause AND your project's (select
  Project-View Source) USES clause if your DLL exports any procedures or
  functions that pass strings as parameters or function results. This
  applies to all strings passed to and from your DLL--even those that
  are nested in records and classes. ShareMem is the interface unit to
  the BORLNDMM.DLL shared memory manager, which must be deployed along
  with your DLL. To avoid using BORLNDMM.DLL, pass string information
  using PChar or ShortString parameters. }

uses
  SysUtils,
  Classes;

{$R *.res}

begin
end.
```

Now replace the text in the window with the following:

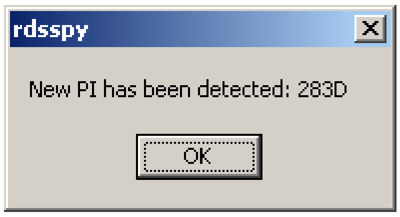
```
library myplugin;

uses
  SysUtils,
  Classes,
  Dialogs;

type
  P_RDSGroup=^TRDSGroup;
  TRDSGroup = record
    Year: word;
    Month: byte;
    Day: byte;
    Hour: byte;
    Minute: byte;
    Second: byte;
    Centisecond: byte;
    RFU: word;
    Blk1: integer;
    Blk2: integer;
    Blk3: integer;
    Blk4: integer;
  end;

var
  PI: integer;
  Group: TRDSGroup;
```

```
{$R *.res}

procedure RDSGroup(PRDSGroup: P_RDSGroup); stdcall;
begin
Group:=PRDSGroup^;
if (Group.Blk1>=0) then
  begin
  if (PI<>Group.Blk1) then
    begin
    PI:=Group.Blk1;
    ShowMessage('New PI has been detected: '+IntToHex(PI,4));
    end;
  end;
end;

procedure Command(Cmd, Param: PChar); stdcall;
var w: string;
begin
w:=UpperCase(string(Cmd));
if (w='CONFIGURE') then
  ShowMessage('Nothing to configure in this simple plugin.');
if (w='RESETDATA') then PI:=-1;
end;

function PluginName: PChar; stdcall;
begin
Result:='My First Plugin';
end;

Exports
  RDSGroup, Command, PluginName;

begin
PI:=-1;
end.
```

Save the project using File / Save project as, fill myplugin.dpr as the project file name.

Compile the project by pressing Ctrl+F9.

Now you should find the myplugin.dll file in the project directory. Copy this dll file into the RDS Spy plugins directory. Then run the RDS Spy. After tuning a station the window should appear:



New PI has been detected: 283D

That's all in this chapter. Much easier than in your imaginings, I think. Take advantage of this. Now it's the time to study the source code and make some experiments before reading the next chapter.

Please keep on mind:

- The main application calls the RDSGroup procedure in the plugin DLL each time a new RDS group is received or read from a file. The parameter of this procedure is a pointer to the actual group. The group structure is defined by the TRDSGroup type.
- Bad blocks are identified by negative value. These blocks shall be ignored.
- The main application calls additional Command procedure in some specific events. Parameters of this procedure are pointers to Cmd and Param null-terminated strings. The Param variable may be undefined for some commands.
- The plugins use standard calling convention (stdcall), also called WINAPI.

Update of the API specification

In the RDS Spy version 1.08 or later, the **RFU** word in the **TRDSGroup** structure contains following information:

Bits	Meaning	Value
0 to 1	RDS2 stream the group is coming from. Original RDS is represented by value of 0.	0 to 3
2 to 7	*Reserved for future use*	0
8	A value of 1 indicates that error correction was used for block 2 in the group. That group should be ignored as the group type may be potentially invalid.	0 or 1
9 to 15	*Reserved for future use*	0

2. List of Procedures and Functions

There is a set of procedures and functions that can be implemented in your plugin (procedure and function names are case-sensitive!). All these procedures and functions are optional.

```
procedure RDSGroup(PRDSGroup: P_RDSGroup); stdcall;
procedure Command(Cmd, Param: PChar); stdcall;
function PluginName: PChar; stdcall;
function Initialize(hHandle: THandle; DBPointer: PTDB): Longint;
stdcall;
```

Type definition:

```
type
  P_RDSGroup=^TRDSGroup;
  TRDSGroup = record
    Year: word;
    Month: byte;
    Day: byte;
    Hour: byte;
    Minute: byte;
    Second: byte;
    Centisecond: byte;
    RFU: word;
    Blk1: integer;
    Blk2: integer;
    Blk3: integer;
    Blk4: integer;
    end;

type
  TRecord = record
    Key: shortstring;
    Value: shortstring;
    end;

type
  TDB = record
    Count: integer;
    Records: array [0..255] of TRecord;
    end;
  PTDB = ^TDB;
```

Data types:
PChar – pointer to a null-terminated string.
ShortString - 0 to 255 characters long string type. While the length can change dynamically, its memory is statically allocated 256 bytes; the first byte stores the length of the string.
Word – unsigned 16-bit.
Integer, LongInt – signed 32-bit.

Procedure RDSGroup

Syntax:
```
procedure RDSGroup(PRDSGroup: P_RDSGroup); stdcall;
```

Description:
This procedure is called each time a new RDS group is received or read from a file. The parameter of this procedure is a pointer to the actual group. The group structure is defined by the TRDSGroup type. This type is a record of time information, a word reserved for future use and the blocks 1 to 4, where block 1 is always the PI. Bad blocks are indicated by its negative value and shall be ignored. No CRC or offsets are included.

Procedure Command

Syntax:
```
procedure Command(Cmd, Param: PChar); stdcall;
```

Description:
The main application calls the Command procedure in some specific events. Parameters of this procedure are pointers to Cmd and Param null-terminated strings. The Param variable may be undefined for some commands.

List of commands:

Cmd	**Meaning / Operation expected**
CONFIGURE	Show the plugin setup dialog. Param is ignored
RESETDATA	Reset all operational data. Param is ignored.
EXIT	Prepare the plugin for application exit. Param is ignored.
SAVEWORKSPACE	A request to save the plugin settings. Called also on the application exit. Param = current workspace file (*.rsw).
TUNE	The Tune button click event. Param = selected frequency.
OPENWORKSPACE	A request to load the plugin settings. Called also on the application start. Param = current workspace file (*.rsw).
MOVEX	The main application window has moved in X direction. Param = difference between new and old position in pixels.
MOVEY	The main application window has moved in Y direction. Param = difference between new and old position in pixels.
LEFT TOP RIGHT BOTTOM	Describes to the plugin the main application window border. Called on the application start. Param = the value in pixels.
SHOW	Set the visibility property to true and bring the window above others. Param is ignored.

SHOWHIDE	Toggle the plugin form visibility property. Param is ignored.
MINIMIZE	The user has minimized the main application window. Param is ignored.
RESTORE	The user has restored the main application window. Param is ignored.
REQUEST	If Param = DECODERDATA, the user requests a report create so the plugin should send data (if exist) to the Decoder Data database using the datxchng.dll procedure AddValue.

Any command whose implementation is not required shall be ignored.

Function PluginName

Syntax:
```
function PluginName: PChar; stdcall;
```

Description:
Returns a pointer to the plugin name. The plugin name is showed to the user in the main menu.

Function Initialize

Syntax:
```
function Initialize(hHandle: THandle; DBPointer: PTDB): Longint;
stdcall;
```

Description:
This function, if implemented, is the first function called in the plugin DLL on the application start. If the plugin contains a form, it should be created just here. The function returns the plugin form handle. This value is currently not used and may be *nil* as well.

Parameter meaning:
hHandle: the main application window handle.
DBPointer: a pointer to the Decoder Data database that is required when using some of the datxchng.dll functions.

3. DATXCHNG.DLL

This DLL contains useful functions that can be used in the plugins for simple data exchange between the plugin and the main application. It also contains a set of functions for load and save the plugin settings. Use of these functions is optional.

Decoder Data related functions:

```
procedure AddValue(Key, Value: PChar; DBPointer: PTDB); stdcall;
function ReadValue(Key: PChar; DBPointer: PTDB): PChar; stdcall;
function CountRecords(DBPointer: PTDB): integer; stdcall;
function ReadRecord(Index: integer; DBPointer: PTDB): TPRecord;
stdcall;
procedure ResetValues(DBPointer: PTDB); stdcall;
```

To imagine what the Decoder Data database means, click on View / Decoder Data in the main menu.

Load and Save related functions:

```
procedure SavePChar(Filename, Section, Key, Value: PChar);
stdcall;
procedure SaveInteger(Filename, Section, Key: Pchar; Value:
integer); stdcall;
procedure SaveBoolean(Filename, Section, Key: Pchar; Value:
boolean); stdcall;
function LoadPChar(Filename, Section, Key, DefaultValue: PChar):
PChar; stdcall;
function LoadInteger(Filename, Section, Key: Pchar; DefaultValue:
integer): integer; stdcall;
function LoadBoolean(Filename, Section, Key: Pchar; DefaultValue:
boolean): boolean; stdcall;
```

The functions are provided without exact description as the use should be clear enough from the examples.

Can my plugin send commands to other plugins?

Yes, this is currently possible by special use of the datxchng.dll function AddValue:

```
AddValue('COMMAND', Cmd, DBPointer);
```

where Cmd is a PChar type of the command string:
COMMAND
or
COMMAND+#13+Value

Examples (called from your plugin to send the command to all plugins):

```
AddValue('COMMAND', 'SHOW', DBPointer);
AddValue('COMMAND', 'YOURCOMMAND'+#13+'YOURVALUE', DBPointer);
```

4. Conclusion

Please download plugin examples from the website.

If you found some function missing or some part of this document is not clear enough, feel free to contact us or send a post to the online forum provided.

Please report all possible bugs!

Web:	http://www.rdsspy.com/
Email:	info@rdsspy.com