Rave DataLink Driver Specification

1: Creating DataLink Drivers

DataLink Drivers are DLL files with an RVD extension. By using Rave's DataLink architecture, drivers entail very little code and are quite quick to put together.

There are typically three units that need to be created to compile a DataLink driver:

DataLinkSample.dpr – Contains the exports interface to the Rave IDE and Rave Reporting Server
RvDLSample.pas – Contains the Driver, Connection and ResultSet classes
RvDLSampleCfg.pas and .dfm – Contains the connection configuration form

There are four areas of database support that will need to be plugged into a framework of classes to create a DataLink driver.

- Prompt for the parameters necessary to create a connection such as database name, user name and password
- Create a connection typically composed of one or more database, session or connection components
- Return a list of table names in the database for a given connection
- Open a TDataSet compatible object for a given connection and SQL string

In the following source samples, you should change 'Sample' to the identifier that you've chosen for your database connection type (e.g. BDE or ADO or dbExpress). All other items marked in yellow will require additional coding to be functional. You should review the BDE, ADO and dbExpress driver source that comes with Rave or the Add-On datalink drivers for more detailed examples of how to fill in these methods.

DataLinkSample.dpr

```
library DataLinkSample;

uses
  RvDLBase, RvDLSample, RvDLSampleCfg;

{$E rvd} // Forces the .rvd extension on the output file

exports
  DataInformation,
  DataConnectionConfigure,
  DataConnectionOpen,
  DataConnectionClose,
  DataGetTables,
  DataGetFields,
  DataResultSetOpen,
  DataResultSetClose,
  DataResultSetFirst,
  DataResultSetNext,
  DataResultSetEOF,
  DataResultSetGetRow,
  DataResultSetSetFilter,
  DataResultSetSetSort,
  DataResultSetConfigure,
  DataGetErrorText;

begin
end.
```

RvDLSample.pas

```
unit RvDLSample;

interface

uses
  Windows, SysUtils, Classes, Forms, RvDLCommon, RvDLBase, RvDLDataSet;

type
  TDLSampleDriver = class(TDLDataSetDriver)
  public
    function CreateConnection: TDLBaseConnection; override;
    function CreateResultSet(AConnection: TDLBaseConnection): TDLBaseResultSet; override;
    procedure Information(Params: PDLInformation); override;
  end;

  TDLSampleConnection = class(TDLDataSetConnection)
  private
    FDatabase: TSampleDatabase; // Database component(s) specific to the Sample Database system
  public
    procedure Connect(ADataSource, AUserName, APassword: string; AOptionList: TStringList); override;
    procedure Disconnect; override;
    procedure GetTableNames(List: TStrings); override;
    //
    property Database: TSampleDatabase read FDatabase;
  end; { TDLSampleConnection }

  TDLSampleResultSet = class(TDLDataSetResultSet)
  public
    function OpenDataSet(QueryStr: string): TDataSet; override;
  end; { TDLSampleResultSet }

implementation

{ TDLSampleDriver }

function TDLSampleDriver.CreateConnection: TDLBaseConnection;
begin
{ Create connection object }
  Result := TDLSampleConnection.Create;
end;

function TDLSampleDriver.CreateResultSet(AConnection: TDLBaseConnection): TDLBaseResultSet;
begin
{ Create result set object }
  Result := TDLSampleResultSet(AConnection).Create;
end;

procedure TDLSampleDriver.Information(Params: PDLInformation);
begin
{ Return installed state, version and driver names }
  Params.Installed := true; // Should return whether database client exists on this system or not
  Params.VerMajor := 1;
  Params.VerMinor := 0;
  StrPCopy(Params.InternalName,'Sample');
  StrPCopy(Params.DisplayName,'Sample Database Engine');
end;

{ TDLSampleConnection }

procedure TDLSampleConnection.Connect(ADataSource, AUserName, APassword: string; AOptionList: TStringList);
begin
{ Create a connection to the database.  Sample code below. }
  FDatabase := TSampleDatabase.Create(nil);
  Database.DataSource := ADataSource;
  Database.UserName := AUserName;
  Database.Password := APassword;
  Database.Option1 := AOptionList.Values['Option1'];
  Database.Option2 := AOptionList.Values['Option2'];
  Database.Connected := true;
end;
```

```
procedure TDLSampleConnection.Disconnect;
begin
{ Disconnect from the database and free any allocations.  Sample code below }
  FreeAndNil(FDatabase);
end;

procedure TDLSampleConnection.GetTableNames(List: TStrings);
begin
{ Return a list of table names in this connection... }
  Database.GetTableNames(List);
end;

{ TDLSampleResultSet }

function TDLSampleResultSet.OpenDataSet(QueryStr: string): TDataSet;
begin
{ Open a result set by creating query for QueryStr. Sample code below. }
  Result := TSampleQuery.Create(Application);
  try
    with TSampleQuery(Result) do begin
      Database := TDLSampleConnection(self.Connection).Database;
      SQL.Text := QueryStr;
      Open;
    end;
  except
    Result.Free;
    raise; // re-raise the exception so that it can be handled by the DataLink system
  end;
end;

initialization
  RegisterDriverClass(TDLSampleDriver);
end.
```

The added fields and properties of the TDLSampleConnection class should be specific to the database system that the driver is being written for. Some database systems will require only one component (such as TADOConnection for the ADO driver) while others will require more than one component (such as TDatabase and TSession for the BDE driver). It is recommended to create read-only properties for each field so that the other classes, specifically the TDLSampleResultSet.OpenDataSet method, can access the require database system components.

The Connect method is used to initialize the database system object fields from the DataSource, UserName, Password and OptionList values. The Disconnect method should close and free any objects that were created in the Connect method. Lastly for the TDLSampleConnection class, the GetTableNames method should fill up the List parameter with the available table names for the current connection. The code for this method is usually quite different from one database system to another.

The TDLSampleResultSet.OpenDataSet method is used to create and open a TDataSet compatible component initialized for the current connection settings and QueryStr parameter (SQL text).

One other method that may need to be overridden is the TDLBaseConnection.GetFields method. The standard declaration of this method is:

```
procedure TDLBaseConnection.GetFields(TableName: string);
begin
  with Driver.CreateResultSet(self) do try
  // Create an empty result set to get field info only
    Open('select * from ' + TableName + ' where 0=1');
    GetFields(Driver.Fields); // Global Driver FieldList
    Close;
  finally
    Free;
  end; { with }
end;
```

If the database system is not compatible with the above SQL statement (select * from TableName where 0=1) and supports another method to retrieve the field information for a specific tablename this method can be overridden. If the alternate method does not require the creation of a resultset object, then the code for TDLDataSetResultSet.GetFields, specifically the calls to AFields.AllocFieldList and AFields.SetFieldItem, should be copied and modified in the overridden method.

RVDLSampleCfg.pas and .dfm

```
unit RvDLSampleCfg;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, ComCtrls, ActnList, RvDLBase;

type
  TDLSampleConfigureForm = class(TDLConfigForm) // Make sure to change form type to TDLConfigForm!
  { ... Form Components and Events ... }
  private
  public
    procedure SetData(ADataSource, AUserName, APassword: string; AOptionList: TStringList); override;
    procedure GetData(var ADataSource, AUserName, APassword: string; AOptionList: TStringList); override;
  end;

implementation

{$R *.dfm}

uses
  RvDLCommon, RvDLSample;

procedure TDLSampleConfigureForm.SetData(ADataSource, AUserName, APassword: string; AOptionList: TStringList);
begin
{ Initialize form controls from ADataSource, AUserName, APassword and AOptionList here. Sample code below. }
  editDataSource.Text := ADataSource;
  editUserName.Text := AUserName;
  editPassword.Text := APassword;
  editOption1.Text := AOptionList.Values['Option1'];
  editOption2.Text := AOptionList.Values['Option2'];
end;

procedure TDLSampleConfigureForm.GetData(var ADataSource, AUserName, APassword: string; AOptionList:
TStringList);
begin
{ Assign new values to ADataSource, AUserName, APassword and AOptionList here.  Sample code below. }
  ADataSource := editDataSource.Text;
  AUserName := editUserName.Text;
  APassword := editPassword.Text;
  AOptionList.Values['Option1'] := editOption1.Text;
  AOptionList.Values['Option2'] := editOption2.Text;
end;

initialization
  RvDLBase.ConnectionConfigureForm := TDLSampleConfigureForm; // Assign the driver config form type
end.
```

The primary purpose of the configuration form is to edit the DataSource, UserName, Password and OptionList configuration variables. The OptionList variable should be used to store any configuration values other than the DataSource, UserName or Password by using the OptionList.Values property. The configuration form is displayed when a RaveDatabase component is first created or when the AuthDesign/AuthRun properties are edited. The configuration form should descend from the TDLConfigForm class and needs to override the SetData and GetData methods. The SetData method is called before the form is shown to the user and is the place where the form controls should be initialized to the values stored in the configuration variables. The GetData method is called after the form is closed (assuming the OK button was pressed) and is the place where the data from the form controls should be copied to the configuration variables.

The configuration form can contain any controls that are needed to allow the user to define a connection to a database however the overall layout of the form should be similar to the DataLink drivers that ship with Rave including the use of a TPageControl component to contain the separate configuration sections. A recommended addition to the configuration form is a button to test the configuration parameters. The code in the button should call the TestConnection method. This method takes a single TStrings parameter which will return a list of tables in the connection if the test is successful. Normally the list of table names is displayed in a TListBox component.