General page information

e asyReportPDF

Last modified : 06/09/2005 (Version 1.00)

Information

easyReportPDFis a library for developers to create **Reports in PDF** from any data source. It's based on the award winning aspEasyPDF library for rendering fast PDF documents trough the WEB and being optimized to use it on any application development that needs to use **real-time Data Reporting** by connecting to a database server.

But what really differs from aspEasyPDF library? We have simplified aspEasyPDF, added database connection, fast scripting interpretation and finally report rendering features; The result is that you design the report with VisualEasyPDF, which is a product designed for data reporting (included on the license) , saves the report and run it from your application without coding a line for the report. No other library makes it easier, take a look on the tutorials to learn more.

Key features

PDF Reports

Native PDF rendering by including the aspEasyPDF library inside the easyReportPDF which results a **fast real-time PDF Data Reporting library**.

Design reports

easyReportPDF includes a license of **VisualEasyPDF** to develop the reports. You don't need to code your application to render the report, just set the report file and render it directly to the printer.

Parameters

Create user parameters and automatically ask the user to input the value. The report will react with the user input data and will show the correct information that has requested for.

Grouping

Create up to **10 grouping level bands**.

Scripting

You need that the report reacts to certain values when rendering it, then just code with our fast scripting technology and debug it with VisualEasyPDF before posting the report to avoid any error.

Fast Maintenance

Found an error on your report and need to solve it fast; the reports may be altered directly trough the server and the changes will be effective immediately for your users after pressing save button in VisualEasyPDF.

Databases

It includes **msSQL** (© Microsoft Corporation) , **MySQL** (© MySQL AB.) , **Oracle** (© Oracle Corporation.) , **PostgreSQL** (© Portions. The PostgreSQL Global Development Group) , **SyBase** (© Sybase Inc.) , **FireBird** (© Firebird Project.) , **Access** (© Microsoft Corporation) and **ADO** native drivers to retrieve the data faster than any simple ODBC connection. Other databases may be accessed by using ADO native drive connection.

Versions

The library is **Shareware** which has one limitation; it displays a message on each page of the PDF document, the **Crippleware** technique is used to guaranty piracy protection. **The price? Starts at just US$ 159 per machine!**

The **Professional** version adds functions that helps you to alter object properties and connections settings in real-time; see help file to see which are those functions that are only available with the Professional version.

The SITE licensing option helps you achieve significant savings on volume purchase of MITData products. SITE license entitles you to install and use the software on unlimited number of workstations.

If you need all the aspEasyPDF functionality in the easyReportPDF then get the aspEasyPDF Enterprise version which adds all the features of easyReportPDF inside the aspEasyPDF.

Different versions of easyReportPDF

The different versions are all coded on the same DLL (reportpdf.dll) and the license file that you purchase sets the library to use one version or other. This way you may upgrade your license with just using a different license file.

Standard version is the basic version which loads the report and renders it. You can not use compression, encryption or other internal functions which interact with the report on your code. All those functions are marked with an asterisk in the help file.

Professional version does what the standard version do but adds compression properties, encryption and internal function that interacts with the objects from the report. Those functions are marked with an asterisk in the help file.

Feature	Standard	Professional
aspEasyPDF library coded	Yes	Yes
Support for Bitmaps, **GIF, TIFF, PNG, JP2000 and JPEG**	Yes	Yes
Chart capability	Yes	Yes
Use **code bars** (EAN, Code39, Code128, etc)	Yes	Yes
Supported Databases (ADO, msSQL, mySQL, FireFox, ODBC, Access and more)	Yes	Yes
Render reports directly to the printer or to the browser	Yes	Yes
Free license of VisualEasyPDF	Yes	Yes
Load VEP files done with the VisualEasyPDF design tool	Yes	Yes
Loads reports created by VisualEasyPDF	Yes	Yes
Adds all technology features from easyReportPDF into aspEasyPDF	Yes	Yes
Create dynamically reports	Yes	Yes
Add dynamic script to your report	Yes	Yes
Manage user parameters directly	Yes	Yes
Compressionand **security encryption** in 40 bits or 128 bits	No	Yes
Functions that interacts with VisualEasyPDF objects; **setDBConnection** ; Changes connection properties, this is useful to **setDBSQL** ; Changes the SQL dynamically before running the report. **getFirstObject** , **getNextObject**; retrieve objects from the report **getPropObj** , **getProperty** ; gets property objects **setPropObj** , **setPoperty** ; sets property objects	No	Yes

Release History

1.01 (X-September-2005)
Added the Clear function
Added NVersion function
Added Connection constants (see Constants help)

1.00 (1-September-2005)
Public release

[[TOPIC HERE]]

BinaryWrite

Function explain	
Name	**BinaryWrite**
Parameters	✖ **(none)**
Result	✖ **(none)**
From version	**1.00**

Warning: For using a similar function in .NET you should use the SaveVariant function and not the BinaryWrite

This is the same function as the Save method, but instead to save it to disk it will save it on memory and forwards it directly to the Internet Browser which will open the PDF reader (if it's registered in the machine).
Use this function for internet developments because you don't have to establish security permissions and you don't have to maintain PDF files in the hard disk.

Syntax

ERP .BinaryWrite

See also

Save SaveVariant Show Print

GetFirstObj

Function explain	
Name	**GetFirstObj**
Parameters	✓ **Object Type as Integer**
Result	✓ **Result as WideString**
From version	**1.00**
" Professional feature "	

Will return the first object found in the report memory. This may be used to retrieve the name of an object you don't known the exact name and then after knowing it's name change it's value. To get the next object use the GetNextObject function.

This tables show you the different object you may request:

Object Types	
ID	Description
1	Text
2	Shape
3	Graphic
4	MultiText
5	EditBox
6	Button
7	RadioButton
8	CheckBox
9	ListBox
10	ComboBox
11	DBField
12	DBFieldSum
21	Chart
30	Connection
40	Band
100	Bookmark

See also
GetNextObject

GetFirstObj

Function explain	
Name	**GetNextObj**
Parameters	✓ **Object Type** **as Integer, Object Name** **as WideString**
Result	✓ **Result** **as WideString**
From version	**1.00**
" Professional feature "	

Will return the next object found after the specified object name parameter in the report memory. This may be used to retrieve the name of an object you don't known the exact name and then after knowing it's name change it's value.

See the GetFirstObject function to known the Object types and a small sample.

See also

GetFirstObject

SetPropObj

Function explain	
Name	**GetPropObj**
Parameters	✓ **ObjectID** as WideString , **PropertyID** as Integer , **Value** as WideString
Result	✓ **Result** as WideString
From version	**1.00**
" **Professional feature** "	

The GetProperty gets a global parameter, the GetPropObj gets an object parameter defined in VisualEasyPDF.

Syntax

Result = PDF .GetPropObj ObjectID as string, ProperyNumber as Integer

Example in ASP

```
<%
' Create the component
set PDF = server.createobject("")
PDF.SetPropObj " form.listbox ", csPropObjCbxValues, " [ (AMX)(American Express)] [ (CBL)(Carte Blanche)] [
(DCL)(Diners Club)] [ (DSC)(Discover)] [ (ENR)(EnRoute)] (JCB)[ (MSC)(MasterCard)] [ (VIS)(Visa)] "

PDF.BinaryWrite
set pdf = nothing
%>
```

See also

SetProperty GetProperty SetPropObj GetPropObj

GetProperty

Function explain	
Name	**GetProperty**
Parameters	✓ **PropertyID as Integer**
Result	✓ **Value as String**
From version	**1.00**
" **Professional feature** "	

Returns the specified property. The PropertyID is the parameter identifier that we want to known, it's an integer value and we normally uses constants or defines to easily use names in one include file.
Remember to add the include instruction if you want to use name identifiers, if not, then use directly the value for the property you want to get.

Properties are defined in constant groups; text, graphics, documents, information, report, internal, etc. We call each property with the constant name, you should find an information on each constant defined in this help file under the constants help tree.

Info: When we started to design the library, we searched an easy way to change parameters for modifying the look of the PDF document, without using thousands of variables that could change between different versions without losing compatibilities with old version. Some libraries uses properties that on version x a property is n wide length and for example a string type, then it came version 2 and this version uses a double precision variable, making the old code being incompatible.
This is why we decided to implement two powerful functions that gets and sets values that modifies and interacts with the library. The value is always treated as a string, making the conversion inside the library and forgetting on incompatible version from version 1 to x.

Syntax

var = *PDF* .GetProperty PropertyID as Integer

Example in ASP:

```
const csPropTextFont = 100
' Write with Helvetica Font
Actual_Font = PDF.GetProperty csPropTextFont
PDF.AddText " This font is " & Actual_Font
```

See also

SetProperty GetProperty SetPropObj GetPropObj

SetPropObj

Function explain	
Name	**SetPropObj**
Parameters	✓ **ObjectID as WideString , PropertyID as Integer , Value as WideString**
Result	✗ **(none)**
From version	**1.00**
" **Professional feature** "	

The SetProperty changes a global parameter, the SetPropObj changes an object parameter.

Syntax
PDF .SetPropObj ObjectID as string, ProperyNumber as Integer , Value as string

Properties

See this link to read all properties that can be altered

Example in ASP

```
<%
' Create the component
set PDF = server.createobject(" aspPDF.EasyPDF ")
' Adds a Scroll list and introduces the values
PDF.AddFormObj 320, 400, 420, 450, " form.listbox ", " test ", " Introduce the value combo value ", foScrollList
PDF.SetPropObj " form.listbox ", csPropObjCbxValues, " [ (AMX)(American Express)] [ (CBL)(Carte Blanche)] [
(DCL)(Diners Club)] [ (DSC)(Discover)] [ (ENR)(EnRoute)] (JCB)[ (MSC)(MasterCard)] [ (VIS)(Visa)] "
PDF.BinaryWrite
set pdf = nothing
%>
```

See also
GetProperty SetPropety GetPropObj DrawVEP LoadVEPFile

Function explain	
Name	**SetProperty**
Parameters	✓ **PropertyID as Integer , Value as WideString**
Result	✗ **(none)**
From version	**1.00**

Returns the specified property. The PropertyID is the parameter identifier that we want to change, it's an integer value and we normally uses constants or defines to easily use names in one include file.
Remember to add the include instruction if you want to use name identifiers, if not, then use directly the value for the property you want to get.

Properties are defined in constant groups; text, graphics, documents, information, report, internal, etc. We call each property with the constant name, you should find an information on each constant defined in this help file under the constants help tree.

Info: When we started to design the library, we searched an easy way to change parameters for modifying the look of the PDF document, without using thousands of variables that could change between different versions without losing compatibilities with old version. Some libraries uses properties that on version x a property is n wide length and for example a string type, then it came version 2 and this version uses a double precision variable, making the old code being incompatible.
This is why we decided to implement two powerful functions that gets and sets values that modifies and interacts with the library. The value is always treated as a string, making the conversion inside the library and forgetting on incompatible version from version 1 to x.

Syntax

PDF .SetProperty ProperyNumber as Integer , Value as variant

Example in ASP:

```
const csPropTextFont = 100
' Write with Helvetica Font
PDF.SetProperty csPropTextFont, " F5
"
PDF.AddText " Hello world "
```

Example in C:

```
define csPropTextFont = 100;
' Write with Helvetica Font

main()
{
PDF.SetProperty (csPropTextFont, ' F5
');
PDF.AddText(' Hello world ');
}
```

Example in Delphi:

```
const csPropTextFont = 100;
' Write with Helvetica Font
PDF.SetProperty (csPropTextFont, ' F5
');
PDF.AddText(' Hello world ');
```

See also
SetProperty GetProperty SetPropObj GetPropObj

License

Function explain	
Name	**License**
Parameters	✓ **FileName as String**
Result	✓ **Result as Boolean**
From version	**1.00**

NOTE: Only for registered customers.

After purchasing the library you will receive a license file that should be used in conjunction with the library. What it does the license is to check if the machine is allowed to run the library by checking the IP address or the Unique identification.
You do not have to use the license method if you copy the license on the system32 path, the library will search on the actual path and on the system32 path if it exists the license, then it will load it.

The license function is an alternative way to load the license from a different location of different license name, this could be useful for multi host environments or because you don't have read permissions on the system32 path.

Call the license function after creating the library process.

Syntax
PDF .License LicenseFile as String

Example in ASP:

```
<%
' Create the component
set PDF = server.createobject(" aspPDF.EasyPDF ")
' Loads the license from a different place
PDF.License " C:\inetpub\wwwroot\licenses\easypdf.lic
"

' .... your code
%>
```

Example in Delphi:

```
var
   pdf : IEASYPDF;
begin
  // Create the COM object
  PDF := CreateComObject(CLASS_EASYPDF ) as IEASYPDF
;
  // Init the memory
  PDF.Create();
  // Load license
  PDF.License( ' C:\inetpub\wwwroot\licenses\easypdf.lic ' );
  // .... your code
end;
```

See also

SiteLicense Lic_Debug

L oadFromFile

Function explain	
Name	**LoadFromFile**
Parameters	✓ **FileName as String**
Result	✓ **Result as Boolean**
From version	**1.00**

Loads the VisualEasyPDF report to the memory, you need to always call this function before using any other commands. If it's succeed then returns true.

If you need to download the report from a remote site, (trough different internet connected servers), then use the LoadFromHTTP function, is slower than the LoadFromFile which will load the report locally, so try to avoid to use the LoadFromHTTP function.

See also
LoadFromHTTP Render

L oadFromFile

Function explain	
Name	**LoadFromHTTP**
Parameters	✓ **URL as String**
Result	✓ **Result as Boolean**
From version	**1.00**

Loads the VisualEasyPDF report to the memory, you need to always call this function before using any other commands. If it's succeed then returns true.

If you need to download the report from a remote site, (trough different internet connected servers), then use this function, is slower than the LoadFromFile which will load the report locally, so try to avoid to use the LoadFromHTTP function.

If you get a false return result, then check the error message which will contain the HTTP error code.

See also
LoadFromFile Render

Print

Function explain	
Name	**Print**
Parameters	✗ **(none)**
Result	✓ **FileName as String**
From version	**1.00**

Prints the report to the default printer. Returns the temporary file name that has created.

NOTE: This function should be never be used in ASP development, because then the server will print the PDF report and not the customer browser.

Syntax
FileName as String = *ERP* .Show

Example in VB:

```
<%
set PDF = server.createobject("")
' Your code
PDF.Save  Server.MapPath ("filename.pdf"
)
%>
```

See also
BinaryWrite   SaveVariant   Show   Print

Render

Function explain	
Name	**Render**
Parameters	✓ **Page as Integer**
Result	✗ **(none)**
From version	**1.00**

After loading the report into the memory and set all parameters to the connection, use the Render function to render the report in memory. You wont see the report until you issue the Show or Print function, this function just renders the VisualEasyPDF page you issue.

Setting page to 0 will draw all pages from the report.

You may call as many times the render function you need for drawing different reports with different parameters values..

See also
LoadFromFile LoadFromHTTP

SaveVariant

Function explain	
Name	**SaveVariant**
Parameters	✗ **(none)**
Result	✓ **Result as OleVariant (byte[])**
From version	**1.00**

This is the same function as the Save method, but instead to save it to disk it will return an array of bytes which you can be redirected to the client or save it to disk. This is the preferred way to work on interned developments because you don't have to establish securities permissions and you don't have to maintain PDF files on Disk.

NOTE FOR .NET DEVELOPERS: If you code in .NET you should add a Reference to your project, from the COM window list browse for the aspEasyPDF library, then add it. It should create the Interop dll that you will find it on your project.
You can also create manually your project, create a bin directory and copy from the installation directory the **Interop.aspPDF.dll** to the bin folder of your project.

Syntax in .NET
Byte[] = *ERP* .SaveString
Example in ASP.NET using language C#:

```
<% @Page Language=" C# " %>
<% @Import Namespace=" System " %>
<% @Import Namespace=" System.Web " %>
<% @Import Namespace=" aspPDF " %>
<script language=" C# " runat=" server ">
private void Page_Load(Object sender, EventArgs e) {

        aspPDF.EASYPDF pdf = new aspPDF.EASYPDF();
        pdf.Create();
        Response.Clear();
        Response.ContentType=" application/pdf ";
        Response.AddHeader( " content-disposition "," attachment; filename=MyPDF.PDF ");
        Response.BinaryWrite( (byte[]) pdf.SaveVariant());





}
</script>
```

See also
BinaryWrite SaveVariant Show Print

Save

Function explain	
Name	**Save**
Parameters	✓ **FileName as String**
Result	✓ **FileName as String**
From version	**1.00**

Saves the report to the disk, if you left blank the parameter then it will create a temporary PDF file in the temp directory and it will return the name of the file that has generated.

NOTE: If you use ASP or ASP.NET development you should always check your write permissions to the IIS_users for write permissions on the folder that you will use to save your document. If the IIS anonymous user doesn't have sufficient privileges then it will fail to write to disk.

Syntax

ERP .Save FileName as String

Example in ASP:

```
<%
set PDF = server.createobject("")
' Your code
PDF.Save  Server.MapPath ("filename.pdf"
)
%>
```

See also

BinaryWrite SaveVariant Show Print

SetDBConnection

Function explain	
Name	**SetDBConnection**
Parameters	✓ **Object Name** as String, **Connection Syntax** as String
Result	✓ **Result** as Boolean
From version	**1.00**
	" **Only in Professional version** "

Changes the connection dynamically before rendering the report. This may be helpful to dynamically change the server database connection or even the database type.

Programmers may also use this function for standard reports which you don't known which will be the final server connection.

Syntax parameter may only use the parameter which is changed, you don't need to set all parameters again, just the one that has been changed.

We have tried to use the MS notation for the connection string, each parameter should be separated with a semicolon ";" and blank spaces should be always set with the double quote ";

Connection parameters	
Driver	Sets the driver to be used for the connection, at this moment those are available and are all native, if you experience some problems then you may use the ADO connection which will use the ADO driver from the database. Normally Native drivers are faster than ADO. **ADO** - Sets an ADO connection which enables you to connect to practically to all databases which provides a driver for ADO. **FireBird** - Free Relation database which offers SQL-92 standard instruction sets. Runs in many platforms. Web page . **InterBase** - Borland® **InterBase**®, web page . **msSQL** - The Microsoft SQL server. Web information page . **MySQL** - The mySQL server from 3.12 to 5.x Web page. **Oracle** - Oracle. Web page. **Sqlite** - C library that implements a self-contained, embeddable, zero-configuration SQL **database** engine. Linux, Windows **.** Web page **.** **Sybase** - Sybase databaser. Web page.
Host	Specifies the host server. Can be introduced by IP address or by host name.
Port	Port address of the server. Set to 0 or do not introduce any value to use the default port connection.
Database	The database to be connected. Some databases can only use one database and this parameter may be ignore. For servers with multi-databases you should use this parameter to specify which database to use.
User ID	User ID for connection to the database. Use a valid user for the database authentication process-
Password	Sets the password for the connection user.
Properties	Sets additional properties for the connection. At this moment this only works with the ADO connection

Syntax

ERP .SetDBConnection Object as String, Connection as String

SetDBSQL

Function explain	
Name	**SetDBSQL**
Parameters	✓ **Object Name as String, SQL Syntax as String**
Result	✓ **Result as Boolean**
From version	**1.00**
	" **Only in Professional version** "

Changes the SQL connection dynamically before rendering the report. This is done to set complex values for the report or change report field values.
To known which is the name of the connection, open VisualEasyPDF, load the report and see on the connection window which is your object connection name.

Remember that parameter constants are defined with a double point (:) before the parameter name and should be defined before using VisualEasyPDF.

You may also use the SetParameter property which is included in the Standard version to set the parameter to a given value.

NOTE: This function does not check the SQL syntax, if wrong you will get an error message displaying where the error is. For WEB development check the Error property to see which error is.

Syntax
ERP .SetDBSQL Object as String SQL as String

SetDBConnection

Function explain	
Name	**SetDBConnection**
Parameters	✓ **Name as String, Value as String**
Result	✓ **Result as Boolean**
From version	**1.00**

Sets a new value for a given parameter object. Check with VisualEasyPDF which are the object parameters you have defined in the report.

If it successes then it returns true otherwise it will be false.

Show

Function explain	
Name	**Show**
Parameters	✗ **(none)**
Result	✓ **FileName as String**
From version	**1.00**

Saves the report to a temporary PDF file and opens it. Returns the temporary file name that has created.

NOTE: This function should be never be used in ASP development, because then the server will display the PDF report and not the customer browser.

Syntax

FileName as String = *ERP* .Show

Example in VB:

```
<%
set PDF = server.createobject("")
' Your code
PDF.Save  Server.MapPath ("filename.pdf"
)
%>
```

See also

BinaryWrite   SaveVariant   Show   Print

Acrobat_App

Property explain	
Name	**Acrobat_App**
Parameter type	**WideString**
Read / Write	✓ / ✓

Sets the acrobat binary program to see and print PDF documents. When starting the library it checks for the latest version of acrobat reader and assigns it to this variable, you may use any other version or different viewer by overriding this property.

The **Show** and **Print** functions uses this variable.

Debug

Property explain	
Name	**Debug**
Syntax in VB	***ERP*** **.Debug = True / False**
Parameter type	**Boolean**
Read / Write	✓ / ✓

Sets the library in debug mode. This is very useful for debugging your application if you get errors and wish to see what's happening.

If you activate this option when you made any operation on your program it will displays what's going on into a file or the screen. Default is set to False, so there is no debug information when running up.

To active the Debug to a file use the csPropIntDebugFile , by default it will output all information on the screen.

Note: Do not use the debug property in conjunction with ASP and BinaryWrite, it will output the information directly to the PDF reader and it will fail with an error. If you want to use the Debug option then use the Save function.

Example in ASP

```
<%
 ERP.Debug = True

%>
```

LastError

Property explain	
Name	**LastError**
Parameter type	**WideString**
Read / Write	✓ / ✓

Gets the last error from the library. The Error is a string that contains the description in english of the error that was issued by the last command executed.

To reset the error just assign it to an empty string.

Lic_ Debug

Property explain	
Name	**Lic_Debug**
Syntax in VB	***ERP*** **.Lic_Debug = True / False**
Parameter type	**Boolean**
Read / Write	✓ / ✓

This properties is only for registered users and outputs debug information on how it loads and checks the license. This is useful to check if there is a problem with the IP address of the machine and the licensed file or if the license is corrupted.

If it displays any problem, please send this debug information to support@mitdata.net to repair and send a new license file.

Example in ASP

```
<%
dim ERP
' Debug the license routine on the PDF version
' This is done when you get problems on the license machine
set ERP = server.createobject
("ReportPDF.easyReportPDF")
ERP.DEBUG = True
ERP.LIC_DEBUG = True
' Loads the license file, change it if you use a different path
ERP.License("reportPDF.lic")
response.write "<br>Version Information:<br>" &
ERP.Version
set erp = nothing
%>
```

See also
License SiteLicense

[[TOPIC HERE]]

SiteLicense

Property explain	
Name	**SiteLicense**
Parameter type	**String**
Read / Write	✗ / ✓

When you purchase the library you have two licensing methods, one is that you pay per server license (one license for one computer) and the other you pay once and you can install it everywhere or distributed it with your application without paying any royalties for that.

If you decide to purchase the SiteLicense option, then we can send you two different versions of the library, using the license file or compiling the DLL just for you with your information encoded inside the library:

Using License file:
This is the default method that will use our eCommerce system to compose the license, what is means is that you will have a license file encoded with the name of your company that must match on the source code by setting the SiteLicense property. See this example, imagine that your company is called MITData and you just purchased a site license this is what you should change on your code:

Example in ASP

```
<%
set ERP = server.createobject
("ReportPDF.easyReportPDF")
ERP.SiteLicense = " MITData "
' Loads the license file
ERP.License("easypdf.lic")
response.write "<br>Version Information:<br>" &
ERP.Version
set erp = nothing
%>
```

Notice that you must set the SiteLicense property after creating the library object and before calling the License function.

The license wont check any IP address or Unique Identification of the machine, just if the name matches the license.

Using a compiled DLL:
This is useful if you don't want that the library does not check any license file, you must contact us to compile a special version that will have encoded the company name inside the Library.

The bad part of this option is that if you want to upgrade the library to the latest version you must contact us, with the first option you only download the latest version from the registered users site.

See also
License Lic_Debug

ShowProgress

Property explain	
Name	**ShowProgress**
Syntax in VB	***ERP*** **. ShowProgress = True / False**
Parameter type	**Boolean**
Read / Write	✓ / ✓

Disable or enables a small progress window that appears when rendering the report.

By default is set to true, but if it detects that the library has been called from a WEB application, then it disables the windows because displaying it on the server it may hang the library and you will need to restart the machine to start again the IIS.

Version

Property explain	
Name	**Version**
Parameter type	**String**
Read / Write	✓ / ✗

Returns the version information of the library and the IP address from your network machine, this last information will be needed when buying the library.

Please support us and register it for developing better and new programming libraries.

Web_app

Property explain	
Name	**Web_App**
Syntax in VB	***ERP* . Web_App = True / False**
Parameter type	**Boolean**
Read / Write	✓ / ✓

Setting this property you are telling from where you are calling you application, if you set the Web_App to true, you will disable all error pop up messages and some user dialog interaction (like the parameters box). If you don't set the value properly then you may get unexpected results. So, if the application is always users unattended, like Web applications do, set it always to **True**.

By default is set to False, but if it detects that the library has been called from a WEB application then it sets to True.

Constant Definition	
ID	**900**
Name	**csPropERPParamName**
Type	**String**
GetPropObj	**Yes**
GetPropObj	**Yes**

Returns the parameter name of the parameter object.

Constant Definition	
ID	**901**
Name	**csPropERPParamValue**
Type	**String**
GetPropObj	**Yes**
GetPropObj	**Yes**

Returns the parameter name of the parameter object.

Constant Definition	
ID	**902**
Name	**csPropERPParamType**
Type	**String**
GetPropObj	**Yes**
GetPropObj	**Yes**

Returns the parameter type.

Constant Definition	
ID	**903**
Name	**csPropERPParamAskUser**
Type	**String**
GetPropObj	**Yes**
GetPropObj	**Yes**

Sets the parameter to be used on the dialog parameter user to ask a value for it. Set to 1 to enable it or 0 to disable it.

Constant Definition	
ID	**970**
Name	**csPropERPParamName**
Type	**String**
GetProperty	**Yes**
GetProperty	**Yes**

Enables you to activate or desactivate the dialog parameter that will display to the user to select a value. Use this parameter in visual application, not in web development.

Pascal reference

Pascal syntax
easyReportPDF executes internal event scripts written in Pascal syntax. This provides fast and easy programming in the reports.

Current Pascal syntax supports:

" begin .. end constructor
" procedure and function declarations
" if .. then .. else constructor
" for .. to .. do .. step constructor
" while .. do constructor
" repeat .. until constructor
" try .. except and try .. finally blocks
" case statements
" with statements
" as / is statements
" array constructors (x:=[1, 2, 3];)
" ^ , * , / , and , + , - , or , <> , >=, <= , = , > , < , div , mod , xor , shl , shr operators
" access to object properties and methods (ObjectName.SubObject.Property)

Script structure
Script structure is made of two major blocks: a) procedure and function declarations and b) main block. Both are optional, but at least one should be present in script. There is no need for main block to be inside begin..end. It could be a single statement. Some examples:

```
SCRIPT 1:
procedure DoSomething;
begin
CallSomething;
end;
begin
CallSomethingElse;
end;
SCRIPT 2:
begin
CallSomethingElse;
end;
SCRIPT 3:
function MyFunction;
begin
result:='Ok!';
end;
SCRIPT 4:
CallSomethingElse;
```

Like in pascal, statements should be terminated by ";" character. Begin..end blocks are allowed to group statements.

Identifiers
Identifier names in script (variable names, function and procedure names, etc.) follow the most common rules in pascal : should begin with a character (a..z or A..Z), or '_', and can be followed by alphanumeric chars or '_' char. Cannot contain any other character os spaces.
Valid identifiers:

VarName
_Some
V1A2
_____Some____
Invalid identifiers:
2Var
My Name
Some-more
This,is,not,valid

Assign statements
Just like in Pascal, assign statements (assign a value or expression result to a variable or object property) are built using ":=". Examples:

```
MyVar:=2;
Button.Caption:='This ' + 'is ok.';
```

Character strings
strings (sequence of characters) are declared in pascal using single quote (') character. Double quotes (") are not used. You can also use #nn to declare a character inside a string. There is no need to use '+' operator to add a character to a string. Some examples:

```
A:='This is a text';
Str:='Text '+'concat';
B:='String with CR and LF char at the end'#13#10;
C:='String with '#33#34' characters in the middle';
```

Comments
Comments can be inserted inside script. You can use // chars or (* *) or { } blocks. Using // char the comment will finish at the end of line.

```
//This is a comment before ShowMessage
ShowMessage('Ok');
(* This is another comment *)
ShowMessage('More ok!');
{
And this is a comment
with two lines }
ShowMessage('End of okays');
```

Variables
There is no need to declare variable types in script. Thus, you declare variable just using var directive and its name. There is no need to declare variables if scripter property OptionExplicit is set to false. In this case, variables are implicit declared. If you want to have more control over the script, set OptionExplicit property to true. This will raise a compile error if variable is used but not declared in script. Examples:

SCRIPT 1:

```
procedure Msg;
var S;
begin
S:='Hello world!';
ShowMessage(S);
end;
SCRIPT 2:
var A;
begin
A:=0;
A:=A+1;
end;
SCRIPT 3:
var S;
S:='Hello World!'
ShowMessage(S);
```

Note that if script property OptionExplicit is set to false, then var declarations are not necessary in any of scripts above.

Indexes

Strings, arrays and array properties can be indexed using "[" and "]" chars. For example, if Str is a string variable, the expression Str[3] returns the third character in the string denoted by Str, while Str[I + 1] returns the character immediately after the one indexed by I. More examples:

```
MyChar:=MyStr[2];
MyStr[1]:='A';
MyArray[1,2]:=1530;
Lines.Strings[2]:='Some text';
```

Arrays

Script support array constructors and support to variant arrays. To construct an array, use "[" and "]" chars. You can construct multi-index array nesting array constructors. You can then access arrays using indexes. If array is multi-index, separate indexes using ",".
If variable is a variant array, script automatically support indexing in that variable. A variable is a variant array is it was assigned using an array constructor, if it is a direct reference to a Delphi variable which is a variant array (see Delphi integration later) or if it was created using VarArrayCreate procedure.
Arrays in script are 0-based index. Some examples:

```
NewArray := [ 2,4,6,8 ];
Num:=NewArray[1]; //Num receives "4"
MultiArray := [ ['green','red','blue'] , ['apple','orange','lemon'] ];
Str:=MultiArray[0,2]; //Str receives 'blue'
MultiArray[1,1]:='new orange';
```

If statements

There are two forms of if statement: if...then and the if...then...else. Like normal pascal, if the if expression is true, the statement (or block) is executed. If there is else part and expression is false, statement (or block) after else is execute.
Examples:

```
if J <> 0 then Result := I/J;
if J = 0 then Exit else Result := I/J;
if J <> 0 then
begin
Result := I/J;
Count := Count + 1;
end
else
Done := True;
```

While statements

A while statement is used to repeat a statement or a block, while a control condition (expression) is evaluated as true. The control condition is evaluated before the statement. Hence, if the constrol condition is false at first iteration, the statement sequence is never executed. The while statement executes its constituent statement (or block) repeatedly, testing expression before each iteration. As long as expression returns True, execution continues. Examples:

```
while Data[I] <> X do I := I + 1;
while I > 0 do
begin
if Odd(I) then Z := Z * X;
I := I div 2;
X := Sqr(X);
end;
while not Eof(InputFile) do
begin
Readln(InputFile, Line);
Process(Line);
end;
```

Repeat statements

The syntax of a repeat statement is repeat statement1; ...; statementn; until expression where expression returns a Boolean value. The repeat statement executes its sequence of constituent statements continually, testing expression after each iteration. When expression returns True, the repeat statement terminates. The sequence is always executed at least once because expression is not evaluated until after the first iteration. Examples:

```
repeat
K := I mod J;
I := J;
J := K;
until J = 0;
repeat
Write('Enter a value (0..9): ');
Readln(I);
until (I >= 0) and (I <= 9);
```

For statements

Scripter support for statements with the following syntax:
for counter := initialValue to finalValue [step X] do statement

or:
for counter := initialValue downto finalValue [step X] do statement

For statement set counter to initialValue, repeats execution of statement (or block) and increment/decrement value of counter optionally with step X until counter reachs finalValue. Examples:

```
SCRIPT 1:
for c:=1 to 10 do
a:=a+c;
SCRIPT 2:
for i:=a to b do
begin
j:=i^2;
sum:=sum+j;
end;
```

Case statements
Scripter support case statements with following syntax:

```
case selectorExpression of
caseexpr1: statement1;
...
caseexprn: statementn;
else
elsestatement;
end
```

if selectorExpression matches the result of one of caseexprn expressions, the respective statement (or block) will be execute. Otherwise, elsestatement will be execute. Else part of case statement is optional. Different from Delphi, case statement in script doesn't need to use only ordinal values. You can use expressions of any type in both selector expression and case expression. Example:

```
case uppercase(Fruit) of
'lime': ShowMessage('green');
'orange': ShowMessage('orange');
'apple': ShowMessage('red');
else
ShowMessage('black');
end;
```

Function and procedure declaration
Declaration of functions and procedures are similar to Object Pascal in Delphi, with the difference you don't specify variable types. Just like OP, to return function values, use implicited declared result variable. Parameters by reference can also be used, with the restriction mentioned: no need to specify variable types. Some examples:

```
procedure HelloWord;
begin
ShowMessage('Hello world!');
end;
procedure UpcaseMessage(Msg);
```

```
begin
ShowMessage(Uppercase(Msg));
end;
function TodayAsString;
begin
result:=DateToStr(Date);
end;
function Max(A,B);
begin
if A>B then
result:=A
else
result:=B;
end;
procedure SwapValues(var A, B);
Var Temp;
begin
Temp:=A;
A:=B;
B:=Temp;
end;
```

Events

easyReportPDF is event oriented object, this means that for any band, page, or DBField in report that is printed before or after, will fire an event which the dessigned report may act in the user programmed way.

Report_OnStart

Before drawing the report it will fire this event script. There are no parameters set and does not return any value.

Details_OnPrintBefore

Before drawing the detail band it will fire this event script. There are no parameters set.

Note: You should always return true or false to that event. Set to True if you want to print the details or to false to not print it.

[OBJECTNAME FIELD]_OnPrintBefore

Before drawing the object field it will fire this event script. There is one parameter which points to the contents of the field

If you set a return value, then it will print the return value and not the contents of the field.

[OBJECTNAME FIELD]_OnPrintAfter

After drawing the object field it will fire this event script. There is one parameter which points to the printed value, not the field value.

No return values.

[OBJECTNAME TEXT]_OnPrintBefore

Before drawing the object text it will fire this event script. There is one parameter which points to the contents of the object text

If you set a return value, then it will print the return value and overrides the object text.

Internal functions

easyReportPDF supports some internal functions in the script that may be the same that you may use in the library.

TDataSet = DBDataSet ***(Connection_Name : string)***

The DBDataSet function returns the TDataSet object class (more information in the TDataSet class in the Global variable help menu) for the given connection name. If is not found then return nil.

TField = DBField ***(Field_Name : string)***

The DBField function returns a TField object class (more information in the TField class in the Global variable help menu) for the given field name. If is not found then return nil.

You may specify a connection name if you have more than one connection and the field maybe defined in both connection. To do that just put the name of the connection plus a dot separation and then the field name. (for example: **DBField('connection4.customernr')**)

string **=** getProperty ***(Property_ID : integer)***

Returns a property of the PDF document, to known which property to use see aspEasyPDF or easyReportPDF help manual.

string **=** getPropObj ***(ObjectName : string; Property_ID : integer)***

Returns a property of an object of the PDF document, to known which property and get more information on this function see aspEasyPDF or easyReportPDF help manual.

setProperty ***(Property_ID : integer; Value : string)***

Sets a property of the PDF document, to known which property to use see aspEasyPDF or easyReportPDF help manual.

setPropObj ***(ObjectName : string; Property_ID : integer; Value : string)***

Sets a property of an object for the PDF document, to known which property and get more information on this function see aspEasyPDF or easyReportPDF help manual.

Global variables

PosXCursor

Gets or set the position of the X Cursor.
Note: Using wrong this variable may render incorrectly the report.

PosYCursor

Gets or set the position of the Y Cursor.
Note: Using wrong this variable may render incorrectly the report.

PageNumber

Gets the actual page number.

PageCount

Gets the actual page count.
Note: Retrieves the actual page count, which may be not the final number.

Version

Returns the actual version of the library which uses to render the report.

NVersion

Returns the actual version number of the library which uses to render the report.
Note: Use this variable to use compatible functions on the script or request before rendering the report to avoid old version errors.

TDataSet Class

TDataSet is the ancestor class for TTable, TQuery, and TStoredProc (Delphi classes). As such, most properties, methods, and events that these classes use are actually defined by TDataSet. Because so many characteristics of the derived classes come from TDataSet, I'll list the primary properties, methods, and events of TDataSet here, and later I'll list the properties, methods, and events particular to each derived class.

Table 16.1 lists the most commonly used properties of the TDataSet class, Table 16.2 lists the primary methods.

TABLE 16.1. PRIMARY TDataSet PROPERTIES.

Property	*Description*
Active	Opens the dataset when set to True and closes it when set to False.
AutoCalcFields	Determines when calculated fields are calculated.
Bof	Returns True if the cursor is on the first record in the dataset and False if it isn't.
CachedUpdates	When True, updates are held in a cache on the client machine until an entire transaction is complete. When False, all changes to the database are made on a record-by-record basis.
CanModify	Determines whether the user can edit the data in the dataset.
DataSource	The DataSource component associated with this dataset.
DatabaseName	The name of the database that is currently being used.
Eof	Returns True if the cursor is at the end of the file and False if it isn't.
FieldCount	The number of fields in the dataset. Because a dataset might be dynamic (the results of a query, for example), the number of fields can vary from one dataset request to the next.
Fields	An array of TFields objects that contains information about the fields in the dataset.
FieldValues	Returns the value of the specified field for the current record. The value is represented as a Variant.
Filter	An expression that determines which records a dataset contains.
Filtered	When True, the dataset is filtered based on either the Filter property or the OnFilterRecord event. When False, the entire dataset is returned.
FilterOptions	Determines how filters are applied.
Found	Indicates whether a find operation is successful.
Handle	A BDE cursor handle to the dataset. This is used only when making direct calls to the BDE.
Modified	Indicates whether the current record has been modified.
RecNo	The current record number in the dataset.
RecordCount	Returns the number of records in the dataset.
State	Returns the current state of the dataset (dsEdit, dsBrowse, dsInsert, and so on).
UpdateObject	Specifies the TUpdateObject component to use for cached updates.
UpdatesPending	When True, the cached update buffer contains edits not yet applied to the dataset.

TABLE 16.2. PRIMARY TDataSet METHODS.

Method	*Description*
Append	Creates an empty record and adds it to the end of the dataset.
AppendRecord	Appends a record to the end of the dataset with the given field data and posts the edit.
ApplyUpdates	Instructs the database to apply any pending cached updates. Updates are not actually written until the CommitUpdates method is called.
Cancel	Cancels any edits to the current record if the edits have not yet been posted.
CancelUpdates	Cancels any pending cached updates.
ClearFields	Clears the contents of all fields in the current record.
CommitUpdates	Instructs the database to apply updates and clear the cached updates buffer.
Close	Closes the dataset.
Delete	Deletes the current record.
DisableControls	Disables input for all data controls associated with the dataset.
Edit	Enables editing of the current record.

EnableControls	Enables input for all data controls associated with the dataset.
FetchAll	Gets all records from the cursor to the end of the dataset and stores them locally.
FieldByName	Returns the TField pointer for a field name.
FindFirst	Finds the first record that matches the current filter criteria.
FindNext	Finds the next record that matches the current filter criteria.
FindLast	Finds the last record that matches the current filter criteria.
FindPrior	Finds the previous record that matches the current filter criteria.
First	Moves the cursor to the first record in the dataset.
FreeBookmark	Erases a bookmark set previously with GetBookmark and frees the memory allocated for the bookmark.
GetBookmark	Sets a bookmark at the current record.
GetFieldNames	Retrieves a list of the field names in the dataset.
GotoBookmark	Places the cursor at the record indicated by the specified bookmark.
Insert	Inserts a record and puts the dataset in edit mode.
InsertRecord	Inserts a record in the dataset with the given field data and posts the edit.
Last	Positions the cursor on the last record in the dataset.
Locate	Searches the dataset for a particular record.
Lookup	Locates a record by the fastest possible means and returns the data contained in the record.
MoveBy	Moves the cursor by the specified number of rows.
Next	Moves the cursor to the next record.
Open	Opens the dataset.
Post	Writes the edited record data to the database or to the cached update buffer.
Prior	Moves the cursor to the previous record.
Refresh	Updates the data in the dataset from the database.
RevertRecord	When cached updates are used, this method discards changes previously made to the record but not yet written to the database.
SetFields	Sets the values for all fields in a record.
UpdateStatus	Returns the current update status when cached updates are enabled.

TField Class

The TField class represents a field (column) in a database. Through the TField class, you can set a field's attributes. These attributes include the data type (string, integer, float, and so on), the size of the field, the index, whether the field is a calculated field, whether it is required, and so on. You can also access or set a field's value through properties such as AsString, AsVariant, and AsInteger.

Accessing Fields

Before you can get or set the field value, you need some way of locating a field. There are at least three ways to do this:

- By its pointer name

- By the Fields property of TDataSet
- By the FieldByName method of TDataSet

Accessing a field by its pointer name is probably the least used method. It works only if you have previously added fields to your project using the Fields Editor. When you add fields via the Fields Editor, Delphi creates a pointer for each field by combining the table name with the field name. If you have a table called Table1 and a string field called FirstName, Delphi would create a TStringField pointer called Table1FirstName. You could use this pointer to access a field:

Table1FirstName.Value := `Per';

The problem with this approach is that you don't always need to add fields using the Fields Editor.

The Fields property offers another way of accessing a field--by position. If you know that the LastName field is the first field in the table, you can use something like this:

Edit1.Text := Table1.Fields[0].Value;

The problem with this approach, of course, is that you have to know the exact order of fields.

Of the three ways of accessing fields, the most commonly used and reliable is the FieldByName method. Using FieldByName, you have to know only the name of the field to access the field:

Table1.FieldByName(`LastName').AsString := Edit1.Text;

FieldByName returns a TField pointer. To make it more understandable, let me break down the preceding line of code:

var Field : TField; begin Field := Table1.FieldByName(`LastName'); Field.AsString := Edit1.Text; end; In most cases, FieldByName is the way to go. Oh, you might be wondering which record is modified when you execute the preceding code. All these techniques retrieve the field from the current record.

Retrieving and Setting Field Values

After you obtain a pointer to a particular field, you can change its value by using the Value property or any of the As properties (by As properties I mean AsString, AsInteger, AsDateTime, AsBoolean, and so on). These properties perform conversions from one data type to another. Naturally, you can't always be assured that a conversion can be made. For example, if you try to convert a string field containing Smith to an integer, an exception will be thrown.

Setting a field's value is simple when you know the secret of FieldByName:

Table1.Edit; Table1.FieldByName(`LastName').AsString := Edit1.Text; Table1.Post;

First, the Edit method is called to put the table in edit mode. If you fail to call Edit, you will get an exception when you try to modify a field's value. After the table is put in edit mode, the field's value is set. In this case I used AsString instead of the Value property. For a string field, it's the same thing in either case. Finally, the Post method is called to post the edit to the database (or the update cache if CachedUpdates is on). That's all there is to it. Retrieving a field's value is just as easy:

var AcctNo : Integer; begin AcctNo := Table1.FieldByName(`ACCT_NBR').Value; { More code here. } end;

Samples

Example from the erp_orders tutorial:

- We define a total global variable which is a double type to hold the total price
- The function Report_OnStart is called when the report starts to render
- Before printing the text40 object it fires the text40_OnPrintBefore function, this calculates the discount price if it has and add the value to the total global value.
- See that the DBField function retrieves the actual value
- The text51_OnPrintBefore formats the total value to a monetory decimal display
- The text54_OnPrintBefore sets the version of the library and prints it on the report

```
{ Global variables
}
var
  total : double;
{ This is called just when the report start to render, before openning connection.
}
function Report_OnStart
begin
  total := 0;
end;
{ Calculate each line price with it's discount and add it to the total
}
function text40_OnPrintBefore
var
  price : double;
begin
  price := DBField(' UnitPrice ').asFloat * DBField(' Quantity ').asFloat;
  if DBField('Discount').asFloat <> 0 then
  begin
   price := price - ( ( price * DBField(' Discount ').asFloat ) / 100 );
  end;
  total := total + price;
  Result := Format( ' %.2n $ ', [price] );
end;
{ Before printing the total format it
}
function text51_OnPrintBefore
begin
  // Degug
  /// ShowMessage ( total );
  Result := Format( ' %.2n $ ', [total] );
end;
function text54_OnPrintBefore
begin
  Result := ' Report generated with ' + Version + ' version in real-time ';
end;
```

Example from the erp_invoices_month:

- This scripts shows you how to manipulate object property to make the report render dinamically.
- The function is called for each detail band print
- If the ExtendPrice is grater than 300 US dollars, then changes the Font to F2 (which is bold) and sets the color to blue
- if the price is lower than 300 dollars then restablish the font to F1 and the color to black
- Sets to true the result to say to the library to print the band.

```
{ This function is called before printing each record
}
function Details_OnPrintBefore
begin
  // Check the ExtendedPrice field if it's greater than 300 US$
  if DBField(' ExtendedPrice ').asFloat > 300 then
  begin
    // Change Font and color text of the field11 object
    SetPropObj(' field11 ', 100, ' F2 ');
    SetPropObj(' field11 ', 103, '# 0000FF ');
  end
  else
  begin
    // Restore Font and color text of the field11 object
    SetPropObj(' field11 ', 100, ' F1 ');
    SetPropObj(' field11 ', 103, '#000000');
  end;
  // Render the record
  Result := True;
end;
```

Example using sample invoice :

```
<%
' #$#Author#$#
' #$#Date#$#
' #$#VEPVER#$#
Response.Buffer = TRUE
Response.ContentType = " application/pdf "
%>
<%
' Create the component
set ERP = server.createobject(" ReportPDF.easyReportPDF ")
' Loads the VEP file
ERP. WEB_APP = True
ERP. LoadFromFile( Server.MapPath(" erp_invoices_month.vep
") )
ERP. SetParameter " YearNum ", 1997
' Render all pages from the VisualEasyPDF report
ERP. Render 0
ERP. BinaryWrite
'ERP.Save Server.MapPath("erp_invoice.pdf")
set ERP = Nothing
%>
```

Example using sample invoice but in Professional version :

```
<%
' #$#Author#$#
' #$#Date#$#
' #$#VEPVER#$#
Response.Buffer = TRUE
Response.ContentType = " application/pdf "
%>
<%
' Create the component
set ERP = server.createobject(" ReportPDF.easyReportPDF ")
' Loads the VEP file
ERP. WEB_APP = True
ERP. LoadFromFile( Server.MapPath(" erp_invoices_month.vep ") )
' Set compression to reduce the PDF report size ( PRO feature only )
ERP. SetProperty 507, 3
' Set Parameter year
ERP. SetParameter " YearNum ", 1997

' Change host from connection4 ( PRO feature only )
ERP. SetDBConnection " connection4 ", " host=127.0.0.1 "

' Render all pages from the VisualEasyPDF report
ERP. Render 0
ERP. BinaryWrite
'ERP.Save Server.MapPath("erp_invoice.pdf")
```

```
set ERP = Nothing
%>
```

[[TOPIC HERE]]

[[TOPIC HERE]]