Software License Agreement

ReportEase Plus

Version 10

1995-2022

Sub Systems, Inc.

All Rights Reserved

3200 Maysilee Street

Austin, TX 78728

512-733-2525

Software License Agreement

The Software is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The Software is licensed, not sold. This LICENSE AGREEMENT grants you the following rights:

A. This product is licensed per developer basis. Each developer working with this package needs to purchase a separate license.

B. When used this product within a desktop application, you are granted the right to modify and link the editor routine into your application. Such an application is free of distribution royalties with these conditions: the target application is not a standalone report designer or a standalone report executor; the target application uses the control for one operating system platform only; the target application is not a programmer's utility 'like' a report designer/executor; and the source code (or part) of the editor is not distributed in any form.

C. The DESKTOP LICENSE allows for the desktop application development. Your desktop application using this product can be distributed royalty-free. Each desktop license allows one developer to use this product on up to two development computers. A developer must purchase additional licenses to use the product on more than two development computers.

D. The SERVER LICENSE allows for the server application development. The server licenses must be purchased separately when using this product in a server application. Additionally, the product is licensed per developer basis. Only an UNLIMITED SERVER LICENSE allows for royalty-free distribution of your server applications using this product.

E. ENTERPRISE LICENSE: The large corporations with revenue more than $50 million and large government entities must purchase an Enterprise License. An Enterprise license is also applicable if any target customer of your product using the Software have revenue more than $500 million. Please contact us at info@subsystems.com for a quote for an Enterprise License.

F. Your license rights under this LICENSE AGREEMENT are non-exclusive. All rights not expressly granted herein are reserved by Licensor.

G. You may not sell, transfer or convey the software license to any third party without Licensor's prior express written consent.

H. The license remains valid for 12 months after the issue date. The subsequent year license renewal cost is 40 percent of the license acquisition cost. The license includes

General Overview

ReportEase Plus consists of two components. The first component is the Report Designer. The Report Designer allows you to develop report layouts. The second component is the report executor. The report executor is used to print a report using a specified report template.

ReportEase Plus provides a comprehensive set of features. The intuitive graphic Report Designer allows even a novice user to become productive quickly. For the sake of user friendliness, every input parameter offers a default value. The advanced features of ReportEase Plus can be used to generate sophisticated reports and documents.

Multiple File/Multiple Section

ReportEase Plus does not impose any limitation on the number of data files that can be used to supply data. A report or document can have up to 9 sort break sections. Your application supplies a list of data fields that can be used as the sort fields. In addition to the sort sections, you can also define the page header/footer, and report header/footer sections. The section footers can display subtotals, average, minimum, maximum and count fields.

The Report Designer also allows you to specify a selection criteria for the records to be printed. This feature allows your user to print the desired subset of the file.

The graphic Report Designer supports a drag and drop method of placing the report items. Various item arrangement tools can be used to align the items horizontally or vertically. Multiple items can be selected and manipulated. The items can be sized by simply pulling the sizing tabs;

A number of advanced features are also available. For example you can specify the calculated fields for a section break. A report section can be conditionally suppressed using a section selection criteria. A section can be instructed to print with every page break. The blank space before and after a section can be suppressed. You can specify page break criteria for every section. Moreover, multiple records can be printed across on the page.

Fields

The Report Designer supports the following types of fields: text, numeric, float, logical and date. A long text string field can be word wrapped for printing. The ReportEase Plus fields can come from one of the following sources:

Data Field: A field that is associated with a data record.

Calculation Field: Specified using constants, operators, functions and other fields.

System Field: Page number, current date, record number, etc.

Dialog Field: Used to prompt the user for data during the report execution. It can also be printed in the report for information purposes.

Word Wrapping

The memo fields can be word wrapped. The blank space after the section can be suppressed to support variable length memo fields. The memo fields can consist of multiple paragraphs.

Text formatting Options

The Report Designer allows multiple fonts, point sizes and character styles. You can select foreground and background colors for the text. The text can be centered or justified in the horizontal or vertical direction.

Line/Box, and Picture Items

The Report Designer supports lines at any angle. You can control the color, thickness and style of the line objects. ReportEase allows you to import pictures from the clipboard or bitmap files. The picture can be sized by simply pulling the sizing tabs.

A box item is treated as a special label item with blank label text. You can specify any shade or color for the box. You can specify boundary color and style for the box and embed a box within another box.

Printing

The report executor can print to a printer, or to a screen window. The user selects the printing device before the report execution session. The screen output is buffered. You can print selected pages from the screen to the printer.

Interface with Your Application:

You can interface with the product using two types of interface.

Simple Interface:

The simple method involves passing the data definition files (.df), and the corresponding data files (.db) from your application to the Report Designer and report executor. These are comma delimited text files. Your application is responsible for generating the definition and data files. ReportEase does any needed sorting of the records. Please refer to the customer.df, customer.db, sales.df, and sales.db files for an example.

Low-level interface:

Report Designer: When using the low-level interface, your application supplies a routine that allows the user to select the data fields when the users wishes to insert a field when designing a template in the Report Designer.

Report Executor: When using the low-level interface your application calls the print routine for each data record in the sorted data set. The Report Executor performs the record selection, sort breaks, calculations and printing functions.

Requirements

The total memory requirement for all ReportEase Plus modules is approximately 1 MB bytes. The product can be used with any application which can interface with an ActiveX control or a dll.

The product can also be used with an ASP.NET application to provide for report design and report execution at the client machine. The same feature is also available as server application.

Getting Started

This chapter describes the contents of the software diskettes and provides a step by step process of incorporating the TER routine into your application.

In This Chapter

License Key

Your License Key and License number are e-mailed to you after your order is processed. You would set the license information using the RepSetLicenseInfo static function. This should be preferably done before creating the converter session to avoid pop-up nag screens.

int RepSetLicenseInfo(LPBYTE LicenseKey, LPBYTE LicenseNumber, LPBYTE CompanyName);

LicenseKey:	Your license key is available in the product delivery email sent to you upon the purchase of the product. It consists of a string in the form of "xxxxx-yyyyy-zzzzz".
LicenseNumber:	Your license number is also available in the product delivery email. The license number string starts with a "srab" or "smo" prefix.
CompanyName:	Your company name as specified in your order.

Return Value: This method returns 0 when successful. A non-zero return value indicates an error condition. Here are the possible return values:

0	License application successful.
1	Invalid License Key.
2	Invalid License Number.
3	Ran out of available licenses. Please consider purchasing additional licenses.

Example:

result=RepSetLicenseInfo("xxxxx-yyyyy-zzzzz","srabnnnnn-n","Your Company Name")

Replace the 'xxxxx-yyyyy-zzzzz' by your license key, replace "srabnnnnn-n" with your license number, and "Your Company Name" with your company name as specified in your order.

Note: *RepSetLicenseInfo method should be called only once at the beginning of your application. Calling this method for each conversion would degrade the conversion performance.*

Also, you can use the RepGetLicenseStatus function at anytime to retrieve the license status.

Get the license status.

int RepGetLicenseStatus()

Return Value:

0 License application successful.

1	Invalid License Key.
2	Invalid License Number.
3	Ran out of available licenses. Please consider purchasing additional licenses.
4	The evaluation period has expired.

You can use the RepGetLicenseStatus function at anytime to retrieve the license status.

Creating Report Designer Control

This topic describes how to use ReportEase Plus to create a report designer control. There can be two methods of creating a designer window. The *simple method* requires a minimal programming, where as the low-level method requires a fair amount of programming to gain added flexibility. In this chapter, we will describe the *simple method*. The low-level programming method is described in another chapter in this help file.

The simple method can be used with the ActiveX control or direct DLL interface. The ActiveX control method should be used with Visual Basic, or other languages designed to use ActiveX controls.

Using ActiveX control to create a report designer window:

Please copy rep8.dll and roc8.ocx to the system directory or to a directory available at run time. Now register the roc8.ocx file using regsvr32 system utility or from within Visual Basic. The rep8.dll does *not* need registration.

Now create an application form and drop a ReportEase control on to the report template.

The following control properties are typically used to invoke the report designer:

DesignMode	Set this property to TRUE to create a report designer control. *This property should be set at the application design-time only.* Setting it at the run-time has no effect.
Standalone	Set this property to FALSE to embed the report control into your form (default value = FALSE). A TRUE value would create a floating report control instead. *This property should be set at the application design-time only.* Setting it at the run-time has no effect.
MasterDefFile	This property is set to the name of the master definition file. This file contains the field description for the master file. The master file is used for sort-field selection. Please refer to the Data Definition File Format topic for a description about the format of the master definition file.
DetailDefFile	This property is set to the name of the detail definition file. This file contains the field description for the detail file. The detail file has a many-to-one relationship with the master file. Please refer to the Data Definition File Format topic for a description about the format of the detail definition file.
OtherDefFiles	This property is set to the name(s) of the reference definition file(s). A reference file is used to provide addition information about an id in the master or detail file. The reference file has one-to-one relationship with the master or the detail file. When more than one reference files are used, the definition file name of the reference files are specified using a comma delimiter:

roc1.OtherDefFiles="file1.df,file2.df"

Please refer to the Data Definition File Format topic for a description about the format of the reference definition file.

DataMapPictFile	This picture file (jpeg or bmp) is used to schematically display the data-file relationship to the user. This picture is displayed when the user wishes to insert a data field in the report template.
TemplateFile	Use the property to set any initial report template file to edit. This is an optional property since the user can open a file using the toolbar. *However, if this property is set, it should be set after setting the MasterDefFile, DetailDefFile, OtherDefFile, and DataMapPictFile properties.*

When using the DLL interface, the above fields are set in the StrForm structure, and the RepOpenForm API is used to create the report design control. Please refer to the design_c.c file for an example.

Please refer to the design.vbp sample project to see a Visual Basic implementation.

Example 1:

```
  Roc1.MasterDefFile = "customer.df"

  Roc1.DetailDefFile = "sales.df"

  Roc1.OtherDefFiles = "states.df"      'reference file



  Roc1.DataMapPictFile = "DataMap.jpg"  'jpeg picture that
shows the data file relationship when inserting a data field
into the report



  ' Example of setting initial template

  'Roc1.TemplateFile = "sales.fpc"



  ' Example of specifying the data definition files from
internet

  'Roc1.MasterDefFile =
"http://www.subsystems.com/roc/customer.df"

  'Roc1.DetailDefFile =
"http://www.subsystems.com/roc/sales.df"

  'Roc1.OtherDefFiles =
"http://www.subsystems.com/roc/states.df"



  ' Example of setting initial template
```

```
   'Roc1.TemplateFile = "sales.fpc"


   ' Example of setting initial template from internet

   'Roc1.TemplateFile =
 "http://www.subsystems.com/roc/sales.fpc"
```

Example 2:

```
   'Example of specifying the data definition files from
 internet


   Roc1.MasterDefFile =

                     "http://www.subsystems.com/roc/customer.df"

   Roc1.DetailDefFile = "http://www.subsystems.com/roc/sales.df"

   Roc1.OtherDefFiles =
 "http://www.subsystems.com/roc/states.df"



   'Example of setting initial template from internet
```

Roc1.TemplateFile = "http://www.subsystems.com/roc/sales.fpc"

Creating Report Executor Control

This topic describes how to use ReportEase Plus to create a report executor control. There can be two methods of creating a executor window. The *simple method* requires a minimal programming, where as the low-level method requires a fair amount of programming to gain added flexibility. In this chapter, we will describe the *simple method*. The low-level programming method is described in another chapter in this help file.

The simple method can be used with the ActiveX control or direct DLL interface. The ActiveX control method should be used with Visual Basic, or other languages designed to use ActiveX controls.

Using ActiveX control to create a report executor window:

Please copy rep8.dll and roc8.ocx to the system directory or to a directory available at run time. Now register the roc8.ocx file using regsvr32 system utility or from within Visual Basic. The rep8.dll does *not* need registration.

Now create an application form and drop a ReportEase control on to the report template.

The following control properties are typically used to invoke the report executor:

DesignMode	Set this property to FALSE to create a report executor control. *This property should be set at the application design-time only.* Setting it at the run-time has no effect.
Standalone	Set this property to FALSE to embed the report control into your form (default value = FALSE). A TRUE value would be useful when printing or exporting the report without any user interface. *This property should be set at the application design-time only.* Setting it at the run-time has no effect.
AutoRun	Set this property to TRUE to run the report automatically using the definition and data files specified here (default value=TRUE). You would set this property to FALSE to use the Low-level Interface functions to pass data to the report executor.
MasterDefFile	This property is set to the name of the master definition file. This file contains the field description for the master file. The master file is used for sort-field selection. Please refer to the Data Definition File Format topic for a description about the format of the master definition file.
MasterDataFile	This property is set to the name of the master data file. The master data file is a comma delimited text file containing master data records. Please refer to the Data File Format topic for a description about the format of the master data file.

DetailDefFile	This property is set to the name of the detail definition file. This file contains the field description for the detail file. The detail file has a many-to-one relationship with the master file. Please refer to the Data Definition File Format topic for a description about the format of the detail definition file.
DetailDataFile	This property is set to the name of the detail data file. The detail data file is a comma delimited text file containing detail data records. The detail file has a many-to-one relationship with the master file. Please refer to the Data File Format topic for a description about the format of the detail data file.
OtherDefFiles	This property is set to the name(s) of the reference definition file(s). A reference file is used to provide addition information about an id in the master or detail file. The reference file has one-to-one relationship with the master or the detail file. When more than one reference files are used, the definition file name of the reference files are specified using a comma delimiter: roc1.OtherDefFiles="file1.df,file2.df" Please refer to the Data Definition File Format topic for a description about the format of the reference definition file.
OtherDataFiles	This property is set to the name(s) of the reference data file(s). A reference file is used to provide addition information about an id in the master or detail file. The reference file has one-to-one relationship with the master or the detail file. When more than one reference files are used, the data file name of the reference files are specified using a comma delimiter: roc1.OtherDataFiles="file1.db,file2.db" The sequence of the files names specified in the OtherDataFiles must match with their corresponding data definition files specified by the OtherDefFiles property. Please refer to the Data File Format topic for a description about the format of the reference data file.
PictureNameFile	This file is used to associate picture file names with picture ids. The data records passed to report executor may include a picture field. The picture field specifies a numeric picture id. This file contains the picture file names associated with the picture id. The picture file must be in BMP or JPEG format.
TemplateFile	Use the property to set any initial report template file to executet. This is an optional property since the user can run a report using the toolbar. *However, if this property is set, it should be set after setting all properties specified above.*

When using the DLL interface, the above fields are set in the StrRep structure, and the RepInit2 API is used to create the report design control. Please refer to the design_c.c file for an example.

Please refer to the report.vbp sample project to see a Visual Basic implementation.

Example:

```
  ' set data definition files

Roc1.MasterDefFile = "customer.df"

Roc1.DetailDefFile = "sales.df"

Roc1.OtherDefFiles = "states.df"


' set data files

Roc1.MasterDataFile = "customer.db"

Roc1.DetailDataFile = "sales.db"

Roc1.OtherDataFiles = "states.db"


' set the file name which contains picture ids and picture

  names to be used to print

Roc1.PictureNameFile = "PictNames.txt"


' Now run an initial report (optional)

' Note:  TemplateFile property should be set after assigning

'        the data and definition files. The assignment of the

'        TemplateFile property triggers the report execution.


'Roc1.TemplateFile = "saledate.fpc"      ' set the initial

                                           report file
```

Data File Description

This chapter describes the format of the files to be passed to the Report Designer and the report executor when using the product in the *Simple Mode*. These files are *not* used when using the *low-level interface* to send the data to the control.

You would pass the data definition file names to the Report Designer using the MasterDefFile, DetailDefFile, and OtherDefFiles properties. These files contain the description of the data-fields used in the file. The OtherDefFiles properties can be use to pass one or more than one reference data definition file names. When more than one reference file name is passed, the file names are comma delimited. Please refer to the customer.df, sales.df, and states.df files as examples. You can use Notepad or a text editor to view these files.

At the report run time, the data is passed to the report executor using the MasterDataFile, DetailDataFile, and OtherDataFiles properties. A data file consists of one or more data record. A data record consist of a list of comma delimited field-data, followed by a cr/lf terminator. The sequence and type of the field-data within a record must correspond to the format given in the corresponding data definition file. Please refer to the customer.db, sales.db and states.db files as examples. You can use Notepad or a text editor to view these files.

The third type of file that can be passed to the report executor is called Picture Name file. This file contains the picture ids and the corresponding picture file names. This file is used to resolve the picture id contained in a picture field to its corresponding picture file name. Please refer to the PictName.txt as an example.

In This Chapter

Data Definition File format

The data definition file is used to described the fields used in a data file. The data definition file names are passed to the Report Designer and report executor using the MasterDefFile, DetailDefFile, and OtherDefFiles properties.

The data definition files are used only when using the product in a simple mode. These files are not needed when using when product with low-level interface.

The data definition file uses a plain text format. Each data-field is described in one text line delimited by a cr/lf pair. Each line includes the following comma delimited items:

Field Name	The field name may consist of plain ASCII characters including 'a' to 'z', 'A' to 'Z', 0 to 9, and a '_' character. *Spaces are not allowed in the field name.*
Maximum width	For a text field, this item holds the maximum number of character that the field can hold. For a non-text field, this item holds the desired display character width for the field.
Field Type	The field type is denoted using one of these letters:

T	Text field
N	Numeric field (non-decimal)
F	Numeric field (decimal)
L	Logical field (Y or N value type field)
P	Dynamic picture field.

Decimal Places	For type 'F' field, this item specifies the number of decimal places used by the field. Set to 0 for other type of fields.

Please refer to the customer.df or sales.df files for examples of data definition files.

Example:

```
CUST_ID     ,04,T,0

DATE        ,08,D,0

ITEM        ,25,T,0

QUANTITY    ,03,N,0

AMOUNT      ,15,F,2

PROFIT      ,15,F,2

REF_NO      ,10,T,0
```

Data File format

The data file is used to pass data records to the report executor. The data file names are passed to the report executor using the MasterDataFile, DetailDataFile, and OtherDataFiles properties.

These data files are used only when using the product in a simple mode. These files are not needed when using when product with low-level interface.

The data file uses a plain text format. Each record is one text line terminated with a cr/lf pair. A record consists of a collection of data-field values separated with comma.Each data-field is described in one text line delimited by a cr/lf pair. The sequence and tyep of data field should be as described by the corresponding data definition file. The following table describes the how the field value is formatted for each field type:

T	Text	Text enclosed within a pair of comma characters.
N	Number	Non-decimal number
F	Number	Decimal Number
D	Date	Date is passed as a number in *YYYYMMDD* format.
L	Yes/No	Pass 'Y' for true, and 'N' for false.
P	Picture Id	Pass a numeric picture id. The 'Picture Name File' file is used to resolve the picture file name for the picture id specified in this field.

Please refer to the customer.db or sales.db files for examples of data files.

Example:

```
"0010",20050701,"Laser Cartridge",1,269.99,0,"AJ1355"

"0020",20050701,"Laser Cartridge",1,269.99,12,"AJ1282"

"0030",20050701,"Plotter Paper with gloss",1,369,6,"AJ1283"

"0040",20050702,"Computer Paper",1,169.12,12,"AJ1284"

"0050",20050703,"Laser Cartridge",2,161,4,"AJ1298"

"0070",20050706,"Laser Cartridge",1,269.99,6,"AJ1288"

"0080",20050708,"Computer Paper",1,101.05,4,"AJ1302"

"0090",20050710,"Pet Set (1K)",1,139.88,3,"AJ1319"

"0100",20050710,"Computer Paper",1,169.12,12,"AJ1293"

"0010",20050712,"Printer Ribbons",2,149,17.75,"AJ1320"

"0030",20050712,"Computer Paper",2,101.05,8,"AJ1321"
```

```
"0050",20050715,"Plotter Paper with gloss",1,369,3,"AJ1325"
```

Picture Name File

This file is used to associate picture file names with picture ids. The data records passed to report executor may include a picture field. The picture field specifies a numeric picture id. This file contains the picture file names associated with the picture id. The picture file must be in BMP or JPEG format. This file is assigned to the PictureNameFile property of the report executor control. Here is an example:

```
1,"rep_pict1.bmp"

2,"rep_pict2.jpg"

3,"rep_pict3.jpg"

4,"rep_pict4.bmp"

5,"rep_pict5.bmp"

6,"rep_pict6.bmp"

7,"rep_pict7.bmp"

8,"rep_pict8.bmp"

9,"rep_pict9.bmp"

10,"rep_pict10.bmp"
```

When picture id 1 is encountered, the report-executor will display the picture contained in the rep_pict1.bmp file at the specified picture rectangle. When picture id 3 is encountered report-executor will print rep_pict3.jpg picture at the specified picture rectangle.

Please refer to the PictNames.txt file for an example.

ASP .NET Interface

WebRep.dll is a wrapper to ROC8.OCX/REP.dll controls to be used in a Web application.

You can drop a WebRep object while designing a web page in Visual Studio .NET. As the page is loaded on the client machine, the resulting html page will include an 'object' tag which refers to the ROC ActiveX control.

This chapter describes the WebRep properties, and events.

In This Chapter

Web Demo Applications

In This Chapter

Report Designer

The demo_web_design.zip file includes a sample Design application. This application uses the WebRep and Roc controls.

To run the demo application, please unzip the demo_web_design.zip file to a folder called 'design' under the c:\Inetpub\wwwroot directory.

You will see the following files unzipped:

WebRep.dll	ReportEaser web control. This control generates the 'object' tag to host the Roc control within the html page generated by ASP.NET
Design.dll	Demo application dll
Design.aspx.cs	Source for Design.dll
Design.aspx	The aspx file to be loaded by the client machine.
AssemblyInfo.cs	The assembly file for the demo project.
Design.csproj	The demo project file

Now, copy the WebRep.dll and Design.dll files from the demo directory to the c:\Inetpub\wwwroot\design\bin directory.

The last file that you would need is called ROC8.cab. Please download this file from our web site: www.subsystems.com/roc/roc8.cab .Copy the ROC8.cab cab file to the c:\Inetpub\wwwroot\design directory.

Now you can access the demo from the client machine:

```
http://www.myserver.com/design/Design.aspx
```

You can also access the demo application from your own machine:

```
http://localhost/design/Design.aspx
```

Please note that the demo application refers to the data definition and data files on www.subsystems.com/roc site. To further customize the demo to your own environment, you would change the MasterDefFile, DetailDefFile, and other related properties to a location on your server.

Report Executor

The demo_web_report.zip file includes a sample report application. This application uses the WebRep and Roc controls.

To run the demo application, please unzip the demo_web_report.zip file to a folder called 'report' under the c:\Inetpub\wwwroot directory.

You will see the following files unzipped:

WebRep.dll	ReportEaser web control. This control generates the 'object' tag to host the Roc control within the html page generated by ASP.NET
report.dll	Demo application dll
report.aspx.cs	Source for report.dll
report.aspx	The aspx file to be loaded by the client machine.
AssemblyInfo.cs	The assembly file for the demo project.
report.csproj	The demo project file

Now, copy the WebRep.dll and report.dll files from the demo directory to the c:\Inetpub\wwwroot\report\bin directory.

The last file that you would need is called ROC8.cab. Please download this file from our web site: www.subsystems.com/roc/roc8.cab .Copy the ROC8.cab cab file to the c:\Inetpub\wwwroot\report directory.

Now you can access the demo from the client machine:

```
http://www.myserver.com/report/report.aspx
```

You can also access the demo application from your own machine:

```
http://localhost/report/report.aspx
```

Please note that the demo application refers to the data definition and data files on www.subsystems.com/roc site. To further customize the demo to your own environment, you would change the MasterDefFile, DetailDefFile, and other related properties to a location on your server.

WebRep Control

WebRep.dll control is a simple drop-in control to be used within an ASP.NET web application.

This dll is included in the demo_web_design.zip and demo_web_report.zip files. Please unzip one of these files and copy the WebRep.dll file to the c:\Inetpub\wwwroot\bin directory (or the bin directory of your application).

You will also need to download a signed cab file from our web site: www.subsystems.com/roc/roc8.cab . This cab file includes the files necessary to embed ReportEase control in the browser page. Copy this ROC8.cab file to the project directory (not the bin directory).

To use the WebRep.dll in a web application, create a new web application (or open an existing one) and right click on the 'Web Forms' tab in the Toolbox. Now select the 'Add/Remove new items...' and add the WebRep.dll control. Now the control will be available for selection in the toolbox.

Now you can select and drop an instances of the WebRep control on your web application page. Right click on the control to set the control properties as desired.

The WebRep control generates the 'object' tag to host the ReportEase Control in the web page. Here is an example of the 'object' tag generated by this control:

```
<object name="WebRep1" ClassId="clsid:41d2ac4b-a269-4c8c-a32c

-0a867e2dc594" codebase="roc8.cab" id="WebRep1"

style="background-color:Transparent;border-color:Blue;

border-style:Outset;height:627px;width:872px;Z-INDEX: 101;

LEFT: 23px; POSITION: absolute; TOP: 22px">

       <Param name="DesignMode"value=False>

       <Param name="VertScrollBar"value=True>

       <Param name="HorzScrollBar"value=True>

       <Param name="Toolbar"value=True>

       <Param name="RepKey" value="">

       <Param name="MasterDefFile"

              value="http://www.subsystems.com/roc/customer.df">

       <Param name="DetailDefFile"

              value="http://www.subsystems.com/roc/sales.df">

       <Param name="OtherDefFiles"

              value="http://www.subsystems.com/roc/states.df">
       <Param name="DataSetName" value="">

       <Param name="TemplateFile"
value="http://www.subsystems.com/roc/cust.fp">

       <Param name="Typeface" value="">

       <Param name="DataMapPictFile" value="">

       <Param name="MasterDataFile"
value="http://www.subsystems.com/roc/customer.db">

       <Param name="DetailDataFile"
value="http://www.subsystems.com/roc/sales.db">

       <Param name="OtherDataFiles"
value="http://www.subsystems.com/roc/states.db">

       <Param name="PictureNameFile"
value="http://www.subsystems.com/roc/PictNamesWeb.txt">

       <Param name="AutoRun"value=True>

       <Param name="SuppressPrintMessage"value=True>

       <Param name="UseCurrentPrinter"value=False>

       <Param name="TextPitch"value=0>

       <Param name="InWebRep"value=True>

</object>
```

When using TE Edit Control and SpellTime controls with ReportEase, a new cab file containing TOC13.dll and spell32.dll must be used.

The license keys must be assigned to the RepKey,properties using the property box. The license key for the product is available in a distribution file called key.txt.

WebRep Control Properties

The WebRep control uses the same properties as the underlying ActiveX control. Please refer to the ActiveX Properties topic for a list of available properties.

However, please note that file names assigned to the file related properties such as MasterDefFile, MasterDataFile must point to a location on your server for the server data to be available on the client computer. Example:

MasterDefFile="www.myserver.com/customer.df"

WebRep Control Events

WebTer control does not support any server-side event. However, all events listed in the ActiveX Events topic are available for the embedded ReportEase Control within the browser.

Here is an example of setting up an event handler for the UpdateToolbar event:

```
<script language="javascript" for="WebRep1"

                        event="UpdateToolbar()">

   <!--



   //-->

</script>
```

Here is an example of setting up an event handler for the Preprocess event:

```
<SCRIPT LANGUAGE="javascript" for="WebRep1"

                  event="Preprocess(ActionType,ActionId)">

  <!--

        alert(ActionType);

        alert(ActionId);

     -->

</SCRIPT>
```

In the examples above, WebRep1 refers the the 'object' hosting the ReportEase Control.

Low-level Interface

This chapter describes the low-level programming interface with Report Designer and report executor. You do not need to refer to this chapter if you are interfacing with Report Designer and Report Executor using the *Simple Mode*. Please refer to the Data File Description topic when using the *Simple Mode*.

In This Chapter

Report Designer Interface

The Report Designer allows the user to develop the report forms. The demo program shows an example of interfacing with the Report Designer. In particular, follow these steps to interface with the Report Designer:

1. Include the REP.H file into your application module which will interface with the Report Designer.

2. Create a list of fields used in your data base. The user will select the fields from this list to insert in the report template. For each field, you should know its name, default width (number of characters), field type, and number of decimal places (for numeric and float fields).

   For example, assume that your application uses two files. Further, assume that each file can have up to 15 fields. Define an array to store the field names and field properties for these two files:

   ```
   #define MAX_FILES  2

   #define MAX_FIELDS 15

   struct StrDataField {

   char name[35];  /* field name*/

   int  width;     /* field width */

   int  type;      /* field type */

   int  DecPlaces; /* decimal places */

   } DataField[MAX_FILES][MAX_FIELDS];
   ```

   width: The field width stores the default width of the field for printing. The user can modify the field width during the report template editing session.

   type: The field types are defined in the REP.H file. It can be one of the following:

   ```
   TYPE_TEXT    Text field

   TYPE_NUM     Numeric field

   TYPE_DBL     Float field

   TYPE_DATE    Date field

   TYPE_LOGICAL Logical field

   TYPE_PICT    Picture field
   ```

   Decimal Places: For numeric and float fields, you should also store the number of digits after the decimal point. The user can also change this parameter during the report template editing session.

3. Write a field selection routine. This routine will be called by the Report Designer whenever the user wishes to insert a data field in the report template. The field selection routine has the following prototype:

   *int FAR PASCAL UserFieldSelection(HWND hWnd, struct StrField far *field,int SortFieldNo)*

   The first parameter is the window handle of the Report Designer window. Your application may need to use this parameter to create a dialog box if necessary.

   The second parameter is a far pointer to a *field* variable. This routine should use the field pointer to store the data for the chosen field.

   The third Argument indicates whether the Report Designer intends to use this field as a sort field. For a regular field, this argument is set to zero. For a sort field, this argument is set to the sort level number for which the new field will be used. Your user field selection routine may like to restrict the number of fields that may be used for sorting. Or, your user selection routine may need to limit the number of sort levels that can be allowed.

   This routine should return a TRUE value (1), if the field selection is successful. Otherwise, it should return a FALSE value.

   Typically, a field selection routine should first display a list of files. After a file is selected, this routine should show the list of fields that are available for the file. The user can then choose the desired field. The file and field selection can be restricted if the SortFieldNo is not zero.

   The routine should return certain minimum information about the chosen field. This information should be written out to the field structure (argument #2). Although, the field structure contains a number of other variables, here we will discuss only those variables that must be assigned by this routine.

field->name	This variable should be set to the name of the field. If your application uses multiple files, the full field name should be provided. The '->' string should be used to separate the file name from the field name. For example, a customer name field in the customer file should be assigned as CUSTOMER->NAME. The file or field name must not contain any of these special characters: ()*/+#<=\'"$, or spaces. ReportEase Plus field names are not case-sensitive.
field->type	The field type must be one of the types described in step #4.
field->width	Initial width of the field.
field->DecPlaces	The number of digits to the right of the decimal point. This data must be specified for a numeric or float field.
field->ParaChar	*Needed only for a word-wrapped field with multiple paragraphs*. Specify the new paragraph indicator character in the first byte. When the report executor sees this character in the text stream, it will place the subsequent text in the next paragraph. We recommend ASCII 13 value for this field. Although not mandatory, it is advantageous to provide the

following two variables also:

field->FileId	An id associated with the file.
field->FieldId	An id associated with the field. These variables can be later used by your application during the report execution to identify the fields easily.

In the example used by step #4, all the above information can be provided very easily from the DataField structure.

4. Write a field verification routine. This routine will be called by the Report Designer whenever it needs to verify a data field name as entered by the user. The field verification routine has the following prototype:

*int FAR PASCAL VerifyField(HWND hWnd, struct StrField far *field,int SortFieldNo)*

The first parameter is the window handle of the Report Designer window. Your application may need to use this parameter to create a dialog box if necessary.

The second parameter is a far pointer to a field variable. The name of the field to verify is given by the name variable (field->name) within the field structure. The field name may contain the file prefix as well. The field name is always given in the upper case. This routine should verify that a field by this name exists in your application. If the field is found valid, this routine should use the field pointer to provide the additional data (as mentioned in step #5) for the field.

The third Argument indicates whether the Report Designer intends to use this field as a sort field. For a regular field, this argument is set to zero. For a sort field, this argument is set to the sort level number for which the current field will be used. Your field verification routine may choose not to allow all the fields as sort fields.

This routine should return a TRUE value (1), if the field is valid. Otherwise, it should return a FALSE value.

For a valid field, this routine should also provide additional information in the field structure. This information includes field->width, field->type, field->DecPlaces, field->FileId, field->FieldId. In the example used in step #4, all this information can be provided very easily from the DataField structure.

5. Define a structure variable of structure type *Str*form (defined in the REP.H file). This structure is used to pass the initial report template parameters to the report template function. Example:

struct Strform formParm;

The *Str*form structure is defined as following:

```
 struct Strform {

  int   x;

  int   y;

  int   widht;

  int   height;
```

```
    int    (FAR PASCAL *UserSelection)(HWND, struct StrField
                                          far*,int);
    int    (FAR PASCAL *VerifyField)(struct StrField far *,int);
    char   file[129];
    char   DataSetName[20];
    BOOL   ShowMenu;
    BOOL   ShowHorBar;
    BOOL   ShowVerBar;
    HANDLE  hInst;
    HANDLE  hPrevInst;
    HANDLE  hParentWnd;
    HANDLE  hFrWnd;
    DWORD style;
    char   FontTypeFace[31];
    LPCATCHBUF  Endform;
    BOOL   open;
    BOOL   modified;
    BOOL   ShowToolbar;
  }
```

6. Fill the *Str*form structure variables as following:

x:	Specify the initial X position (in device units) of the Report Designer window. You may specify CW_USEDEFAULT to use the default value.
y:	Specify the initial Y position (in device units) of the Report Designer window.
width:	Specify the initial width (in device units) of the window in device units. You may specify CW_USEDEFAULT to use the default value.
height:	Specify the initial height (in device units) of the editing window
UserSelection:	Specify the pointer to the data field selection routine developed in step #3. Example:

```
formParm.UserSelection = (void far *)
MakeProcInstance(UserFieldSelection,hInst);
```

	(This process instance should not be freed until the Report Designer window is closed)

VerifyField:	Specify the pointer to the data field validation routine developed in step #4. Example:
	`formParm.VerifyField = (void far *)`
	`MakeProcInstance(VerifyField,hInst);`
	(This process instance should not be freed until the Report Designer window is closed)

file:	Specify the name of the report template template file. The full path name is allowed in the file name.
DataSetName:	This field is used only when creating a new report template. Using this field, you can specify a name for the data set needed for this report template. The data set name for the report template is stored in the disk file. Note, that this field is not needed for report template editing and is never used internally by ReportEase Plus. When you initialize a report for execution, the report executor tells your application the data set name as you specify here. Your application can use this information to prepare the data to run the report.
ShowMenu:	Set to TRUE if you wish to use the Report Designer menu.
ShowHorBar:	Set to TRUE to show the horizontal scroll bar.
ShowVerBar:	Set to TRUE to show the vertical scroll bar.
hInst:	Specify the instance handle of your application.
hPrevInst:	Specify the instance handle of any previous invocation of your program, or specify NULL.
hParentWnd:	Specify your window's handle, or set to NULL. The Report Designer sends the REP_CLOSE message to this window before closing itself. Your application can then perform any necessary housekeeping tasks.
hFrWnd:	Set this field to 0. The Report Designer will place into this field the handle of the Report Designer window when it is created
style:	Use this field to specify the style word for the Report Designer window.
FontTypeFace:	Specify the typeface for the default font. Set this field to NULL if you wish the Report Designer to use the preset default typeface.
open:	Set this field to FALSE. The Report Designer will set this field to a TRUE value after opening the Report Designer window.
modified:	This flag is used internally and it should be set to FALSE.
ShowToolbar:	Set to TRUE if you wish to display the toolbar at the top of the Report Designer window.

7. Update the export section of your application's .DEF file to include the UserSelection and VerifyField functions. These exported function will be called by the ReportEase Plus DLL to accept and verify data fields.

8. Call the Report Designer routine as following:

   ```
   RepEditform(&formParm);
   ```

   This routine displays the selected report template in an editor window and allows the user to edit the report template. The routine returns a 0 value on a successful execution. Otherwise it returns an error code. For a list of error codes, refer to ERR_ constants in the REP.H file

9. Edit the link statement in your make file to include the REP8.LIB import library file.

   Please also refer to the *Analysis of the Demo Program* chapter for further help with the Report Designer interface.

Report Executor Interface

Your application uses the report executor to print a report or letter for a report template template. Your application provides the data records. The report executor applies the data records to the chosen report template to produce the output. The following functions have been provided to interface with the report executor.

*RepInit(struct StrRep far * RepParm):* Your application calls this routine to initialize the report executor for a report template. The name of the report template is provided using a variable in the StrRep structure. This function returns a unique report id as one of the member variables in the StrRep structure.

RepRec(int ReportId): Your application calls this routine for each record in the sorted data set. The 'ReportId' indicates the unique id for this reporting session. It is returned by the RepInit function calls as one of the member variables in the StrRep structure.

RepExit(int ReportId): Your application calls this routine to print the ending totals and free up the resources.

Follow these steps to interface with the report executor:

1. Include the REP.H file into your application module which will interface with the report executor.

2. Skip this step if your application does not use the picture type fields or the RTF memo fields. Otherwise write a picture drawing routine. This routine will be called by the report executor whenever it needs your application to draw a picture for a picture id or draw an RTF memo text. The picture drawing routine has the following prototype:

   ```
   int FAR PASCAL DrawPicture(HDC hDC, int PictId,

        int FileId, int FieldId, RECT far *rect, int ReportId)
   ```

   The hDC parameter specifies the device context of the report output device. This device context either belongs to a printer or to a screen metafile. The device context is in the ANISOTROPIC mode with the x and y resolutions set to UNITS_PER_INCH constant (defined in the REP.H file).

   The PictId parameter specifies the id of the picture to be drawn. The parameter has a value which was specified by your application in the field array before calling the RepRec function.

   The FileId parameter specifies the file name that contains this field.

   The FieldId parameter specifies the id of the field name.

   The rect parameter specifies the rectangle within which your application should draw the picture.

   The ReportId parameter specifies the unique id of the reporting session.

   Please call the RepDrawTerText function within DrawPicture function to display an RTF file. *This feature is available only when TE Developer's Kit is also installed.* Syntax:

```
int RepDrawTerText(int ReportId, LPBYTE FileName,

    HGLOBAL hBuf, long BufLen, LPBYTE FieldNames,

    LPBYTE FieldValues)
```

ReportId: Simply pass the value given by the 'ReportId' parameter for the DrawPicture function.

FileName:	Specify the rtf file name or the file path. If the data is passed via a global buffer, this set parameter to NULL.
hBuf:	The global buffer handle when the data is passed using a global memory buffer. Otherwise this parameter to NULL.
BufLen:	The length of the global memory buffer when the data is passed using a global memory buffer handle.
FieldNames:	The name of the mail merge fields separated by the '|' delimiter. Set this parameter to NULL when the mail merge feature is not used.
FieldValue:	The string values for the mail merge field names separated by the '|' delimiter. Set this parameter to NULL when the mail merge feature is not used.

Example:

```
RepDrawTerText(ReportId,"letter.rtf",NULL,0,NULL,NULL);

RepDrawTerText(ReportId,"letter.rtf",NULL,0,

               "FirstName|LastName", "Bob|Smith");
```

3. Define a structure variable using the StrRep structure. This structure is defined in the REP.H file. Example:

```
struct StrRep RepParm;
```

Fill in the structure variables as following:

file:	Specify the name of the report template file to execute. The full pathname is allowed for the file name.
device:	Your application uses this variable to specify the output device. It can be one of the following:

```
'P' = Printer

'S' = Screen

'F'= Native format File

'R'= RTF file (.RTF extension).
```

```
'H'= HTML file.

'T'= Ascii text file.

'C'= Comma delimted export file.

'B'= Tab delimited export file

'A' = Ask User.
```

If the device is specified as 'A', the report executor prompts the user to select a printer, screen or a disk file for output.

The following four variables are not used when printing to a printer.

x:	Specify the initial X position (in device units) of the output window. You may specify CW_USEDEFAULT to use the default value.
y:	Specify the initial Y position (in device units) of the output window.
width:	Specify the initial width (in device units) of the window in device units. You may specify CW_USEDEFAULT to use the default value.
height:	Specify the initial height (in device units) of the output window
struct StrField far *field:	(OUTPUT) Set to NULL. This variable is returned by the report executor function *RepInit*. After your application returns from the RepInit function, it should save the value of this variable for a later use. This variable provides a far pointer to the report executor's field array. Your application will need to use this variable to fill in the data for each field before calling the RepRec function.
TotalFields:	(OUTPUT) Set to 0. This variable is returned by the report executor. It tells your application the number of fields in the field pointer returned by the previous variable. Your application will need to use this variable to determine the number of fields to fill in.
struct StrField far *SortField:	(OUTPUT) Set to NULL. This variable is returned by the report executor. It points to a field array containing the fields needed to sort the data records. Your application will need to know the fields that are used for sorting.
TotalSortFields:	(OUTPUT) Set to 0. This variable is returned by the report executor. It tells your application the number of fields used for sorting. The description of each sort field is provide by the previous array variable.
DataSetName:	(OUTPUT) This variable is returned by the report executor. It tells your application about the data set needed to run this report. The value of this field is what your application provided to create this report template (see previous chapter, step #8). This variable does not have any significance for the internal

use of the report executor.

SwapDir:	Specify the directory path to store swapped screen pages. Set this variable to NULL, if you wish the report executor to use the current working directory for swapping.
hInst:	Specify the instance handle of your application.
hPrevInst:	Specify the instance handle of any previous invocation of your program, or specify NULL.
hParentWnd:	Specify your window's (if any) handle.
style:	Use this field to specify the style word for the output window.
DrawPicture:	If your application does not use picture fields or RTF memo fields, set this variable to a NULL value. Otherwise specify the pointer to the picture drawing selection routine developed in step #2. Example: `formParm.DrawPicture = (DRAW_PICTURE) MakeProcInstance(DrawPicture,hInst);` *(This process instance should not be freed until the Report Designer window is closed)*
SuppressPrintMessages:	Set to TRUE to suppress print progress messages.
UseCurrentPrinter:	Set to TRUE to print to the currently selected printer for the session. Set to FALSE to print to the printer selected in the report template. Please refer to the RepSetDefPrinterFlag function for further information about this flag.
ReportId:	(OUTPUT) This variable is filled by the RepInit function calls. It indicates a unique reporting session. You need to pass the report id to the subsequent RepRec and RepExit calls.
OutFile:	When the output device is a disk file (device='F'), this field denotes the disk file name. The report executor prompts the user for the file name selection when this field has a NULL in the first byte.
TextPitch:	Character pitch to use when the output is directed to an Ascii text file. The report executor uses this information to decide the number spaces to insert between the report columns. Use 0 for default.
ShowToolbar:	Set to TRUE to display the page number information band at the top of the report window.
DoPreprocessPass:	(OUTPUT) This flag is filled by the RepInit function calls. The reporter sets it to TRUE if the current report template uses the TotalPageCount field. When this flag is set, your application should feed the records twice to correctly print the total page count field. You can safely ignore this flag if you do not wish

to allow the total page count field in the report (see RepAllowPageCount). Please refer to the RepBeginPreprocessPass and RepEndPreprocessPass functions for further detail.

For information about variable types for the StrRep structure, refer to the REP.H file.

Now, call the RepInit function as following:

```
RepInit(&RepParm);
```

This function returns a 0 value on the successful execution. Otherwise, it returns an error code. The function can return an error condition either because of some internal error or when the user clicks the 'Cancel' button on any dialog box. If the function returns an error code, the RepRec and RepExit routines should NOT be called. The ERR_ constants in the REP.H file describes the error codes.

4. Prepare the data to run the report. The preparation involves joining multiple files (when more than one file is used) and sorting the records. The call to the RepInit function in the previous step returns the sort fields used by the report template. It also returns the data set used by the current report template. Depending upon how your application data is structured, the data set name may give you enough information for joining and sorting the records. Some applications may use index information for the records, which can simplify the data preparation.

Alternatively, the sort fields in the StrRep structure specifically provides you the sorting information. You can use the following code fragment to access the fields from the sort field pointer:

```
struct StrField far *fld;

  int i;

  fld=RepParm.SortField;

  for (i=0;i<RepParm.TotalSortFields;i++) {

  fld[i].name      /* sort field name */

  fld[i].FieldId   /* sort field id */

  fld[i].FileId   /* id of the file containing the sort

                 field */

}
```

The FieldId and FileId are the same information that your application provided using the field selection routine during the report template editing session (see the previous chapter, step 5). These fields are not of any internal significance to the report executor.

The ReportEase Plus package comes with a utility (UTIL or UTIL32) DLL to join two files or to sort a file. Please refer to a later chapter for a discussion on these utilities. You can use these utilities to prepare the data set.

5. The next step is to call the RepRec routine for each record in the data set. The RepRec routine does not take any argument. The data for the record is passed using the field pointer that your application retrieved by calling the RepInit function in step

#4. The RepInit function also returns the number of fields in the field structure. The data is stuffed into the field structure using one of these field variables:

LPSTR CharData;	Pointer to the text data, for the fields with type=TYPE_TEXT. The report executor allocates enough space for the default width of the variable. *The data may not exceed the allocated spaces. The text data should always be NULL terminated.*
long NumData;	For numeric data (TYPE_NUM), logical data (TYPE_LOGICAL) , date (TYPE_DATE) and picture id data (TYPE_PICT).
double DblData;	For Float data (TYPE_DBL).

The field structure contains many fields other than *data fields*. Your application MUST NOT change any value for any field other than the application data fields (source=SCR_APPL).

A data field may appear more than once within the field array. Your application must provide the data for each application data field within the array.

Use the following code fragment to carry out this step:

```
struct StrField far *fld;

int i,TotalFields;

fld=RepParm.field;

TotalFields=RepParm.TotalFields;

for each record in the data set {

  for each field in the data record {

    for (i=0;i<TotalFields;i++) {

      if (fld[i].source==SRC_APPL && fld[i].FieldId ==

                       record field id) {

        if (fld[i].type==TYPE_TEXT)

           lstrcpy(fld[i].CharData, record field data);

        else if (fld[i].type==TYPE_NUM)

           fld[i].NumData=record field data.

        else if (fld[i].type==TYPE_DBL)

           fld[i].DblData = record field data.

        else if (fld[i].type==TYPE_LOGICAL)

           fld[i].NumData = record field data.
```

```
            else if (fld[i]==TYPE_DATE)

               fld[i].NumData = record field data.

            else if (fld[i]==TYPE_PICT)

               fld[i].NumData = record field data (picture id)

          }

        }

      }

      if (RepRec(RepParm.ReportId)!=0) break;

  }
```

The logical data must be provided as a (long) 1 or (long) 0. The date information must be provided as YYMMDD or YYYYMMDD. Example, (long) 920430, or (long) 19920430.

6. Call the RepExit routine to end the report. This routine prints the ending totals and frees up the memory resources.

```
RepExit(RepParm.ReportId);
```

Please also refer to *Analysis of the Demo Program* chapter for further help with the ReportEase Plus interface.

Application Interface Functions

This section describes the ReportEase Plus application interface functions in an alphabetic order.

In This Chapter

RepAllowPageCountField

Allow or disallow the use of total page count field in the Report Designer.

BOOL RepAllowPageCountField(allow)

BOOL allow;	// Set to TRUE to allow the user to insert the total page count field, or set to FALSE to not allow the use of this field.

Comment: The Report Designer, by default, allows the user to insert the total page count field in the report. You can use this function to override this functionality.

Return Value: This function returns TRUE when successful.

RepBeginPreprocessPass

Allow or disallow the use of total page count field in the Report Designer.

BOOL RepAllowPageCountField(allow)

int ReportId;   // The report id of the current reporting session.

Comment: This function initiates the preprocess pass during report execution. The report executor sets the DoPreprocessPass member variable in the StrRep structure to TRUE when a report uses total page count field. The data records passed to the report executor during the preprocess pass are used to calculate the total number of pages in the report. The RepEndPreprocessPass function is used to end the preprocess pass and begin the actual printing process.

Example:

```
If  (RepParm.DoPreprocessPass) {

   RepBeginPreprocessPass(RepParm.ReportId);

   PrintRecords();  //feed data records to the report executor

   RepEndPreprocessPass(RepParm.ReportId);

}

PrintRecord(); feed data records to the report executor again
```

Return Value: This function returns TRUE when successful.

RepCanInsertSect

Allow or not allow a section in the insert-section dialog box.

BOOL RepCanInserrtSect(SectId, allow)

int SectId;	// Section id, 0 to MAX_SECTIONS-1.
BOOL allow;	// TRUE to allow this section to be included in the 'Insert New Section' dialog box.

Return Value: This function returns TRUE when successful.

RepCommand

Execute a menu command.

BOOL RepCommand(hWnd, CmdId)

BOOL RepCommand2(hWnd, CmdId, send);

HWND hWnd;	// Handle of the Report Designer window.
int CmdId;	// Command id to execute. The list of command ids follow after the function parameter description
BOOL send;	// TRUE to send the command or FALSE to post it.

ID_ADD_FONT_EXP	Add a font expression
ID_ADD_FONT_NAME	Add a named font
ID_ALIGN_HORZ_BOT	Align the bottom of the selected items horizontally.
ID_ALIGN_HORZ_CENTER	Align the center of the selected items horizontally.
ID_ALIGN_HORZ_TOP	Align the top of the selected items horizontally.
ID_ALIGN_VERT_CENTER	Align the center of the selected items vertically.
ID_ALIGN_VERT_LEFT	Align the left border of the selected items vertically.
ID_ALIGN_VERT_RIGHT	Align the right border of the selected items vertically.
ID_APPLY_DEF_FIELD_STYLE	Apply the field attributes of the default field.
ID_APPLY_DEF_ITEM_STYLE	Apply the item attributes of the default item.
ID_BOLD_ON	Toggle the Bold style.
ID_BORDER_COLOR	Set the item border color.
ID_BOT_JUSTIFY	Bottom align the text within the item rectangle.
ID_CENTER	Center the text within the item rectangle.
ID_CENTER_ITEM	Position the item centered horizontally on the page.
ID_COMPRESS_HORZ	Remove the horizontal space.
ID_COMPRESS_VERT	Remove the vertical space.
ID_COPY	Clipboard copy.
ID_CUT	Clipboard copy and delete.

ID_DEF_STYLE_FIELD	Toggle the default field designation for the current field.
ID_DEF_STYLE_ITEM	Toggle the default field designation for the current item.
ID_DEL_FONT_EXP	Delete a font expression.
ID_DEL_FONT_NAME	Delete a named font.
ID_DEL_ITEM	Delete an item.
ID_DLG_CREATE	Create a dialog field.
ID_DLG_DELETE	Delete a dialog field.
ID_DLG_MODIFY	Modify a dialog field.
ID_DOWN	Cursor down.
ID_EDIT_FLD	Edit the field attributes.
ID_EDIT_FLD_EXP	Edit the calculation field expression.
ID_EDIT_LABEL	Edit the field label.
ID_EDIT_LINE	Edit the line object.
ID_ERASE_CB	Erase the contents of the clipboard.
ID_EVEN_SIZE_HORZ	Evenly space the items horizontally.
ID_EVEN_SIZE_VERT	Evenly space the items vertically.
ID_EVEN_SPACE_HORZ	Evenly size the width of the items.
ID_EVEN_SPACE_VERT	Evenly size the height of the items.
ID_EXPAND_HORZ	Insert the space horizontally.
ID_EXPAND_VERT	Insert the space vertically.
ID_FILL_COLOR	Set the item background color.
ID_FIRST_PAGE	Position at the first report page.
ID_FONTS	Edit default font for the item.
ID_HELP	Invoke the help screen.
ID_INSERT_CALC	Insert a calculation field.
ID_INSERT_DATA	Insert a data field.
ID_INSERT_DATE_TIME	Insert the date/time system field.
ID_INSERT_DLG	Insert a dialog field.

ID_INSERT_LABEL	Insert a label.
ID_INSERT_LINE	Insert a line object.
ID_INSERT_PAGE_COUNT	Insert the page-count system field.
ID_INSERT_PAGE_NUMBER	Insert the page-number system field.
ID_INSERT_SYS	Insert a system field.
ID_ITALIC_ON	Toggle the Italic style.
ID_ITEM_BACKGROUND	Edit the item background.
ID_ITEM_FONT_EXP	Assign a conditional font.
ID_ITEM_OUTLINE	Edit the item outlines.
ID_JUSTIFY	Right and left align the text within the item rectangle.
ID_LAST_PAGE	Position at the last report page
ID_LEFT	Cursor left key.
ID_LEFT_JUSTIFY	Left align the text within the item rectangle.
ID_MOD_FONT_EXP	Edit a font expression.
ID_MOD_FONT_NAME	Edit a named font.
ID_NEXT_PAGE	Position at the next report page
ID_NEW	Begin a new report template.
ID_OPEN	Open a new report template for editing or report execution.
ID_PASTE	Clipboard paste operation.
ID_PGDN	Page down.
ID_PICT_FROM_CB	Insert the picture from clipboard.
ID_PICT_FROM_FILE	Insert the picture from a disk file.
ID_POS_ITEM	Position the text within the item box.
ID_PREV_PAGE	Position at the previous report page
ID_PRINT_OPTIONS	Printer setup for the current report.
ID_RIGHT_JUSTIFY	Right-align the text within the item rectangle.
ID_QUIT	Quit the Report Designer.
ID_REN_CALC_FLD	Rename a calculation expression.

ID_REN_FONT_EXP	Rename a font expression.
ID_REP_FILTER	Edit the report filter expression.
ID_REP_PARAM	Edit report parameters.
ID_RIGHT	Cursor right key.
ID_SAVE	Save the report template.
ID_SAVEAS	Save the report template to different name.
ID_SEC_BREAK	Set the section break field.
ID_SEC_EDIT	Edit the section properties.
ID_SEC_FILTER	Edit the section filter.
ID_SEC_NEW	Insert a new section.
ID_SEC_SORT	Set the section soft field.
ID_SNAP_TO_GRID	Snap to grid (toggle).
ID_TOP_JUSTIFY	Top align the text within the item rectangle.
ID_TEXT_COLOR	Set the text color.
ID_ULINE_ON	Toggle the Underline style.
ID_UNDO	Undo.
ID_UP	Cursor up.

Return value: This function returns TRUE when successful.

RepDeleteSection

Delete a report section.

int RepDeleteSection(hWnd, section)

HWND hWnd;	// Window handle of the Report Designer window
int section;	// A section id to delete. Please refer to the RepInsertSection for the list of section ids.

Return Value: This function returns TRUE when successful. Otherwise, it returns a FALSE value.

RepDrawBitmap

Draw a bitmap during report execution.

BOOL RepDrawBitmap(ReportId, hBM, hDC, rect, x, y, width, height)

int ReportId;	// The report id of the current reporting session.
HBITMAP hBM;	// The handle of the bitmap to draw.
HDC hDC;	// Destination device context.
LPRECT rect;	// Pointer to the destination rectangle.
int x;	// The x position in the source bitmap to begin drawing. This value is specified as the percentage of the total width of the source bitmap.
int y;	// The y position in the source bitmap to begin drawing. This value is specified as the percentage of the total height of the source bitmap.
int width;	// The width of the source bitmap to copy. This value is specified as the percentage of the total width of the source bitmap.
int height;	// The height of the source bitmap to copy. This value is specified as the percentage of the total height of the source bitmap.

Comment: This function is used by a Delphi application to draw a bitmap within the DrawPicture callback function. Please refer to the dmo_dlp project for an example.

Return Value: This function returns TRUE when successful.

RepDrawTerText

Display an external file for the current dynamic picture field.

int RepDrawTerText(ReportId, FileName, hBuf, BufLen, FieldNames, FieldValues)

HWND hWnd;	// Window handle of the Report Designer window
int ReportId;	// The report id of the current reporting session.
LPBYTE FileName;	// External file name. This file can be in the RTF format, the text format or the native TER format. Set this field to NULL (or "") when supplying the file data in a global buffer using the 'hBuf' variable.
HGLOBAL hBuf;	// External file data in a global buffer. Set this variable to NULL when supplying the data using the 'FileName' variable.
DWORD BufLen;	// Length of the global buffer when supplying the text data using the 'hBuf' variable.
LPBYTE FieldNames;	// The merge field names. Each field name within this string should be delimited by a '|' character. Set this parameter to NULL if the file does not contain the data field names.
LPBYTE FieldValues:	// The merge data field values. Each field value string should be delimited by a '|' character. Set this parameter to NULL if the file does not contain the data field names.

Comment: This function is used by your application to draw the text data from an external file. Your application calls this function from the DrawPicture callback function or the DrawPicture event. TE Developer's Kit must be installed to use this function.

Return Value: This function returns 0 when successful. Otherwise, it returns an error code (ERR_ constant defined in the rep.h file).

RepEditform

Create a new designer window for report editing.

int RepEditForm(FormInfo)

HWND hWnd;	// Window handle of the Report Designer window
Struct Strform far *formInfo;	// Report template information structure

Invoke the Report Designer.

int RepEditform(formInfo)

This function opens a report template for editing in a new report template window. Please refer to the Report Designer Interface chapter for a complete description for this function. Also please note that this function was formerly called 'report template' which is now obsolete. Please use the RepEditform function instead.

RepEnableCommand

RepEnableCommand

Enable or disable a menu command.

BOOL RepEnableCommand(CommandId, enable)

int CommandId;	// Command id of the menu item. All command ids and their numeric values are listed in the rep_cmd.h file.
BOOL enable;	// Set to TRUE to enable a menu item or FALSE to disable a menu item. All menu items are enabled in the beginning

Return Value: This function returns the previous 'enable' status of the given menu item.

RepEndPreprocessPass

End the preprocess pass during report execution

BOOL RepEndPreprocessPass(ReportId)

```
int ReportId;                    // The report id of the current reporting session.
```

Comment: This function ends the preprocess pass during report execution. The report executor sets the DoPreprocessPass member variable in the StrRep structure to TRUE when a report uses total page count field. The data records passed to the report executor during the preprocess pass are used to calculate the total number of pages in the report.

Example:

```
If (RepParm.DoPreprocessPass) {

RepBeginPreprocessPass(RepParm.ReportId);

PrintRecords(); //feed data records to the report executor

RepEndPreprocessPass(RepParm.ReportId);

}

PrintRecord(); //feed data records to the report executor again
```

Return Value: This function returns TRUE when successful.

.

RepExit

Terminate the report executor session.

int RepExit(RepInfo)

Please refer to the Report Executor Interface chapter for a complete description for this function.

RepGetCurPrinter

Retrieve the printer name set by the previous call to the RepSetPrinter function.

BOOL RepGetCurPrinter(LPBYTE PrtName)

```
LPBYTE PrtName;                    // The location to retrieve the printer name.
```

Return Value: This function returns a TRUE value when successful.

RepGetField

Retrieve the field structure for a field id.

BOOL RepGetField(hWnd, ReportId, FieldId, field)

HWND hWnd;	// The Report Designer window handle. Use the hFrWnd property for this argument when using the Report Designer as an OCX. Set to NULL when using the report executor.
int ReportId;	// The report id of the report during report execution. Set to 0 when using the Report Designer.
int FieldId;	// Field id to retrieve information (0 to TotalRepFields 1). Set to 1 to retrieve the information about the currently selected field.
Struct StrField far *field;	// The location to receive the field structure.

Return Value: This function returns TRUE when successful.

RepGetItemInfo

Retrieve information about a report item.

int RepGetItemInfo(hWnd, CurItem, label, x, y, width, height, type, AuxId, FieldType)

HWND hWnd;	// The Report Designer window handle. Use the hFrWnd property for this argument when using the Report Designer as an OCX.
int CurItem;	// Item id to retrieve the information. Set to 1 to retrieve the information about the currently selected item.
LPBYTE label;	// The location to retrieve the item label.
LPINT x;	// The location to retrieve the x position for the item. All position and size information is provided in 'native' unit where 1 inch equal 250 'native' units.
LPINT y;	// The location to retrieve the y position for the item.
LPINT width;	// The location to receive the width of the item.
LPINT height;	// The location to receive the height of the item.
LPINT type;	// The location to receive the item type:

LABEL:	Label object
FIELD:	Data field object
SECTION:	Report section object
LINE:	Line object
PICT:	Picture (static) object
GROUP:	Selection item object.

LPINT AuxId;	// The location to retrieve the auxiliary id for the item. This field retrieves the Field Id for the field type object and Section Id for the section type object. This field is not used for all other type of items.
LPINT FieldType;	// For the field type object, this argument returns the field type.

Comment: This function is valid only within a Report Designer session.

Return Value: This function returns the item id when successful. A return value of (1) indicates an error. This function also returns (1) when no items are selected and the function is called with the CurItem argument set to 1.

RepGetPageInfo

Retrieve page statistics during report execution.

BOOL RepGetPageInfo(hWnd, CurPage, TotalPages, RecCount)

HWND hWnd;	// The report executor window handle. Use the hFrWnd property for this argument when using the report executor as an OCX.
LPINT CurPage;	// The location to receive the current page number (zero based).
LPINT TotalPages;	// The location to receive the total number of pages in the document.
LPLONG RecCount;	// The location to receive the number of records processed

Return Value: This function returns TRUE when successful.

RepGetParent

Retrieve the parent window handle of the ReportEase Plus window.

HWND RepGetParent(hWnd, ReportId)

HWND hWnd;	// When invoking this function for a Report Designer window, specify the report template edit window handle. Otherwise set it to NULL.
int ReportId;	// When invoking this function during a report execution session, specify the repo id of the current session. Otherwise set it to 0.

Return Value: This function returns the parent window handle.

RepGetParentPtr

Retrieve the current parent pointer.

LPVOID RepGetParentPtr(hWnd, ReportId)

HWND hWnd;	// When invoking this function for a Report Designer window, specify the report template edit window handle. Otherwise set it to NULL.
int ReportId;	// When invoking this function during a report execution session, specify the report id of the current session. Otherwise set it to 0.

Return Value: This function returns the current parent pointer set by a previous call to the RepSetParentPtr function.

See Also
RepSetMsgCallback

RepIgnoreCommand

Ignore the current preprocess command.

```
BOOL RepIgnoreCommand(hWnd)

HWND hWnd;                    // Window handle to access
```

Description: This function can be used while processing the REP_PREPROCESS message or the 'Preprocess' event. This function sets a flag which instructs the editor to skip processing the current command.

Return Value: This function returns TRUE when successful.

RepInit

Initialize a report executor session.

int RepInit(RepInfo)

int RepInit2(hWnd, RepInfo)

The RepInit function initializes a report session (*.frc or *.fr) into a *new* report window. The RepView2 function initializes a report session into an *existing* report window. The 'hWnd' parameter specifies the handle of the report window in which to initialize the new report session. The new report file template is specified by a member variable in the RepInfo structure.

Please refer to the Report Executor Interface chapter for a complete description of the RepInfo structure and the RepInfo function.

RepInPreprocess

Check for preprocess mode.

BOOL RepInPreprocess(ReportId)

int ReportId;	// The report id of the current reporting session.

Return Value: This function returns TRUE if the report executor is in the preprocess mode

RepInsertField

Insert a new data field in the report template being edited.

int RepInsertField(hWnd, type, FieldName, sect, x, y, width, height, repaint)

int RepInsertField2(hWnd, type, FieldName, sect, x, y, width, height, formula, repaint)

HWND hWnd;	// Window handle of the Report Designer window	
int type;	// Field type:	
	SRC_APPL:	Application data field
	SRC_CALC:	Calculation field (use RepInsertField2 function to insert a calculation field)
	SRC_SYS:	System field
	SRC_DLG:	Dialog field
LPBYTE FieldName;	// The field name. **Example:** CUSTOMER->NAME A calculation field must begin with a 'calc->' prefix. Similarly, a system field must begin with a 'sys->' prefix, and a dialog field must begin with a 'dlg->' prefix.	
int sect;	// The section id where to insert the field. Please refer to the RepInsertSection function for a list of available section ids.	
int x;	// The X position of the field in mm unit	
int y;	// The Y position (mm) of the field relative to the top of the given section.	
int width;	// The width (mm) of the field box	
int height;	// The height (mm) of the field box	
LPBYTE formula;	// Calculation formula for a calculation field. Set the parameter to NULL for other field types.	
BOOL repaint;	// To repaint the screen after this operation	

Return Value: When successful this function returns the Item Id of the new field. Otherwise it returns 0.

RepInsertLabel

Insert a new label in the report template being edited.

int RepInsertLabel(hWnd, sect, x, y, width, height, flags, repaint)

HWND hWnd;	// Window handle of the Report Designer window
LPBYTE label;	// The text for the label
int sect;	// The section id where to insert the label. Please refer to the RepInsertSection function for a list of available section ids.
int x;	// The X position of the label in mm unit
int y;	// The Y position (mm) of the label relative to the top of the given section.
int width;	// The width (mm) of the label box
int height;	// The height (mm) of the label box
UINT flags;	// The following label flags are available:

OFLAG_HLEFT	Left aligned
OFLAG_HRIGHT	Right aligned
OFLAG_HCENTER	Centered horizontally
OFLAG_VTOP	Top aligned
OFLAG_VCENTER	Centered vertically
OFLAG_VBOTTOM	Bottom aligned
OFLAG_MULTILINE	Multiline label
OFLAG_VERT	Vertical label

More that one flags can be used by combining them using the logical OR operator.

BOOL repaint;	// To repaint the screen after this operation

Return Value: When successful this function returns the Item Id of the new label. Otherwise it returns 0.

See Also:
RepSetItemFont

RepInsertLine

Insert a line object in the report template being edited.

int RepInsertLine(hWnd, sect, x, y, width, repaint)

HWND hWnd	// Window handle of the Report Designer window
int sect;	// The section id where to insert the line. Please refer to the RepInsertSection function for a list of available section ids.
int x;	// The X position of the line in mm unit
int y;	// The Y position (mm) of the line relative to the top of the given section.
int width;	// The width (mm) of the line
BOOL repaint;	// To repaint the screen after this operation

Return Value: When successful this function returns the Item Id of the line object. Otherwise it returns 0.

RepInsertSection

Insert a new section in the report template being edited.

int RepInsertSection(hWnd, section, SortFieldName, repaint)

HWND hWnd	// Window handle of the Report Designer window
int section;	// New section to insert. Use one of the following constants:

SEC_HDR_PAGE	Page Header
SEC_HDR_REP	Report Header
SEC_HDR_LVL1	Sort Header 1
SEC_HDR_LVL2	Sort Header 2
SEC_HDR_LVL3	Sort Header 3
SEC_HDR_LVL4	Sort Header 4
SEC_HDR_LVL5	Sort Header 5
SEC_HDR_LVL6	Sort Header 6
SEC_HDR_LVL7	Sort Header 7
SEC_HDR_LVL8	Sort Header 8
SEC_HDR_LVL9	Sort Header 9
SEC_DETAIL1	Detail 1
SEC_DETAIL2	Detail 2
SEC_DETAIL3	Detail 3
SEC_DETAIL4	Detail 4
SEC_DETAIL5	Detail 5
SEC_DETAIL6	Detail 6
SEC_DETAIL7	Detail 7
SEC_DETAIL8	Detail 8
SEC_DETAIL9	Detail 9
SEC_FTR_LVL9	Sort Footer 9
SEC_FTR_LVL8	Sort Footer 8
SEC_FTR_LVL7	Sort Footer 7

SEC_FTR_LVL6	Sort Footer 6
SEC_FTR_LVL5	Sort Footer 5
SEC_FTR_LVL4	Sort Footer 4
SEC_FTR_LVL3	Sort Footer 3
SEC_FTR_LVL2	Sort Footer 2
SEC_FTR_LVL1	Sort Footer 1
SEC_FTR_REP	Report Footer
SEC_FTR_PAGE	Page Footer

This field can also be set to -1 to invoke a section selection dialog box.

LPBYTE SortFieldName;	// When then new section is one of the sort header sections, then this field should provide the sort field name for the sort header section.
BOOL repaint;	// To repaint the screen after this operation

Description: A higher order sort header section can only be created after creating all the lower sort header sections.

Return Value: When successful, this function returns the Item Id for the new section. Otherwise it returns 0.

RepListAppend

Add an item to the list box.

BOOL RepListAppend(hWnd, item)

HWND hWnd	// Window handle of the Report Designer window
LPBYTE item;	// The description of the item to add to the list box.

Return Value: This function returns TRUE when successful.

See Also
RepListInit
RepListShow

RepListInit

Initialize the list box.

BOOL RepListInit(hWnd, title, x, y)

HWND hWnd	// Window handle of the Report Designer window
LPBYTE title;	// The title for the list box.
int x;	// The x (pixel) position of the list box relative to the top/left of the screen. Set to -1 for default.
int y;	// The y (pixel) position of the list box relative to the top/left of the screen. Set to -1 for default.

Comment: This function is useful to create a modal list box within a programming environment such as MS Access that does not support pure modal list boxes. This function creates a empty list box. The 'RepListAppend' function is used to add items to the list box, and the 'RepListShow' function is used to show the list box.

Return Value: This function returns TRUE when successful.

See Also
RepListAppend
RepListShow

RepListShow

Display the list box.

int RepListShow(hWnd)

```
HWND hWnd                    // Window handle of the Report Designer window
```

Return Value: This function returns the index (zero based) of the selected item. It returns -1 if the user cancels the list box.

See Also
RepListAppend
RepListInit

RepMenuEnable

Retrieve the 'enable' status a menu command.

BOOL RepMenuEnable(hWnd, CmdId)

HWND hWnd	// Window handle of the Report Designer window
int CmdId;	// Command id for the menu command. Please refer to the RepCommand function for a list of menu commands.

Return Value: This function returns TRUE if the given menu item should be enabled. The FALSE value indicates that the menu item should be grayed.

RepMenuSelect

Retrieve the 'check' status a menu command.

BOOL RepMenuSelect(hWnd, CmdId)

HWND hWnd	// Window handle of the Report Designer window
int CmdId;	// Command id for the menu command. Please refer to the RepCommand function for a list of menu commands.

Return Value: This function returns TRUE when the given menu item should be 'checked'.

RepModified

Check if a report template is modified and needs to be saved.

```
BOOL RepModified(hWnd)

HWND hWnd                    // Window handle of the Report Designer window
```

Return Value: This function returns TRUE if the current report template is modified and needs to be saved to a disk file.

RepOpenform

Load a report template into the Report Designer window.

BOOL RepOpen0form(hWnd, formInfo)

HWND hWnd;	// The designer window handle
Struct Strform far *formInfo;	// report template information structure.

This function opens a report template in an *existing* Report Designer window. The report template file name is specified by a member variable in the Strform structure. The Strform stucture also contains many window creation parameters such as window location and size. These parameters are ignored by this function. Please refer to the *Report Designer Interface* chapter for a description of the Strform structure.

Return Value: This function returns TRUE when successful.

RepPrintFromPreview

Print the report to the specified printer from screen display.

BOOL RepPrintFromPreview(hWnd, name, driver, port, FirstPage, LastPage)

HWND hWnd	// The handle of the preview window
LPBYTE name;	// The printer name
LPBYTE driver;	// The printer driver name
LPBYTE port;	// The printer port
int FirstPage;	// The first page of the range of the pages to print. Set this variable to 0 to print all pages.
int LastPage;	// The last page to print. This parameter is ignored when the 'FirstPage' parameter is set to 0.

Return Value: This function returns TRUE when successful.

Example: Print to an HP Laserjet printer on the second printer port:

```
 RepPrintFromPreview(hWnd,

                    "HP LaserJet 4","HPPCL5MS","LPT2:",0,0);
```

RepQueryExit

Check if the Report Designer or the report editor window can be closed.

BOOL RepQueryExit(hWnd)

HWND hWnd // Window handle of the Report Designer window

This function is used when the report design or the report executor window is created in a non-standalone mode.

Return Value: This function returns TRUE if the current window can be closed.

RepRec

Supply a logical record to the report executor session.

int RepRec(ReportId)

```
HWND hWnd                    // Window handle of the Report Designer window
```

Please refer to the *Report Executor Interface* chapter for a complete description for this function.

RepSave

Save the report template file during report template editing session.

BOOL RepSave(hWnd, FileName)

HWND hWnd	// Window handle of the Report Designer window
LPSTR FileName;	// The file name for saving the current report template template.

Return Value: This function returns TRUE when successful.

RepSaveReport

Save the report from the report display window.

BOOL RepSaveReport(hWnd, Outputformat, FileName)

HWND hWnd	// The report executor window handle for the operation.
int Outputformat:	The output format can be specified using one of the following constants:

FILE_RTF:	RTF format
FILE_HTML:	HTML format
FILE_TXT:	Text format
FILE_NATIVE:	Internal format.

LPSTR FileName;	// The file name for saving the current report.

Return Value: This function returns TRUE when successful.

RepShrinkWrapFieldSpace

Shrink unused space in a variable height wrap field.

BOOL RepShrinkWrapFieldSpace(shrink)

BOOL shrink:	// TRUE to shrink wrap field space, FALSE to retain the original height of the text box.

Description: This function must be called before initializing a report run for this change to be effective.

Return Value: This function returns the previous value of this flag.

RepSetDefPrinter

Set the default printer for the report executor.

BOOL RepSetDefPrinter(name,driver,port)

LPBYTE name;	// The new default printer name. Set to "" to restore Windows original default printer.
LPBYTE driver;	// Printer driver name.
LPBYTE port;	// Printer port

Description: Use this function to override the Windows default printer. The new default printer is effective for the controls created after this function is called. The UseCurrentPrinter flag should be set to TRUE during report initialization to print to this new default printer.

Return Value: This function always returns TRUE.

Example: Set the default printer to HP Laserjet on the second printer port:

```
RepSetDefPrinter("HP LaserJet 4","HPPCL5MS.DRV","LPT2:");
```

RepSetDefPrinterFlag

Set the default printer selection flag.

BOOL RepSetDefPrinterFlag(UseDefaultPrinter)

BOOL UseDefaultPrinter;	// Set toTRUE to use the default printer. Set to FALSE to print to the printer specified in the report template.

Description: We will explain here the general printer selection mechanism in ReportEase and how it relates to the 'RepSetDefPrinterFlag' function.

The user can select a printer for the report template using the File->PrinterSetup dialog box while editing the report. This printer however may not always be the printer where the report is actually printed. The user can set the 'Always print to default printer' flag using the File->ReportParameters dialog box. When this option is 'checked', the report is printed to the current default printer when the report is run (instead of the printer selected during report design). If this option is not checked, then the developer can force the printing to go to the current printer using the 'UseCurrentPrinter' flag in the StrRep structure.

The RepSetDefPrinterFlag is used to specify the default value for the 'Always print to default printer' checkbox in the File->ReportParameter dialog box. This default value is used when a new report is created.

Return Value: This function always returns TRUE.

RepSetFlags

Set certain flags or retrieve the values of the flags.

DWORD TerSetFlags(hWnd, set, flags)

HWND hWnd	// The handle of the window to be accessed. Set this parameter to NULL when referring to a non-screen report session. If hWnd is set to NULL and ReportId is set to 1, then this function sets the common initialization flags. When a new window or report is initiated, it inherits all common flags.
int ReportId;	// Report id for the report. Set this parameter to 1 when report id is not known or not applicable.
BOOL set;	// TRUE to set the given flags, FALSE to reset the given flags
DWORD flags;	// Flags (bits) to set or reset. Currently, the following flag values are available:

REPFLAG_NO_PICT_PATH:	Do not write the picture path for the picture files during html file output.
REPFLAG_NO_OVERLAP_MSG:	Do not display the item overlap message during html or text output.

Return value: This function returns the new value of all the flags. Call this function with the 'flags' parameter set to zero to retrieve flag values without modifying it.

RepSetformFlags

Save the report flags during report template editing session.

UINT RepSetformFlags(hWnd, set, flags)

HWND hWnd	// Window handle of the Report Designer window
BOOL set;	// TRUE to set the given flags, or FALSE to reset them.
UINT flags;	// Report flags. The following report flags are available:

RFLAG_TRIAL:	Prompt for a trial record before printing data to adjust the report template printer. This option can be useful for adjusting the address labels in the printer.
RFLAG_REPORT_HDR_FIRST:	Print the report header before the first page header.
RFLAG_AUTO_COL_HDR:	Automatically create the column header labels for the fields.

Return Value: This function returns the new value of the flags.

RepSetField

Set the updated field structure for a field id.

BOOL RepSetField(hWnd, ReportId, FieldId, field)

HWND hWnd	// The Report Designer window handle. Use the hFrWnd property for this argument when using the Report Designer as an OCX. Set to NULL when using the report executor.
int ReportId;	// The report id of the report during report execution. Set to 0 when using the Report Designer.
int FieldId;	// Field id to set information (0 to TotalRepFields 1). Set to 1 to set the information about the currently selected field.
Struct StrField far *field;	// The location to receive the field structure.

Comment: To change the field structure information for a field id, first call the RepGetField, modify the desired items, then call the RepSetField function.

Return Value: This function returns TRUE when successful.

RepSetItemFont

Set the font attribute of an item the during report design session.

BOOL RepSetItemFont(hWnd, ItemId, typeface, pointsize, style, color, repaint)

HWND hWnd	// The Report Designer window handle. Use the hFrWnd property for this argument when using the Report Designer as an OCX.
int ItemId;	// The item id to set font information.
LPBYTE typeface;	// The new typeface for the item. Set this parameter to NULL to retain the current typeface.
int pointsize;	// The new pointsize for the item. Set this parameter to 0 to retain the current pointsize.
UINT style;	// The new style bits for the item. Select one or more from the following list:

REP_BOLD:	Bold
REP_ULINE:	Underline
REP_ITALIC:	Italic
REP_STRIKE:	Strikeout

COLORREF color;	// The new RGB color for the item.
BOOL repaint;	// Set to TRUE to repaint the screen after this operation.

Return Value: This function returns TRUE when successful.

See Also
RepInsertLabel
RepSetItemInfo

RepSetItemInfo

Set the item attributes.

BOOL RepSetItemInfo(hWnd, ItemId, flags, TextColor, FillColor, boder, BorderType, BorderThickness, BorderColor, repaint)

HWND hWnd	// The Report Designer window handle. Use the hFrWnd property for this argument when using the Report Designer as an OCX.
int ItemId;	// The item id to set information. Set the ItemId to 1 to use the currently selected item.
UINT flags;	// Item flags (more than one flags can be set using the logical 'or' operator):

OFLAG_FILL	Fill the item rectangle with a color
OFLAG_HLEFT	left aligned item
OFLAG_HRIGHT	right aligned item
OFLAG_HCENTER	Horizontally centered item
OFLAG_VTOP	Top aligned item
OFLAG_VCENTER	Vertically centered item
OFLAG_VBOTTOM	Bottom aligned item
OFLAG_MULTILINE	Multiline item
OFLAG_VERT	Vertically displayed item
OFLAG_INVISIBLE	Invisible item
OFLAG_DEF_STYLE	The item used to provide default styles to other items.
OFLAG_HLINK	Link item

COLORREF TextColor;	// The text color
COLOREF FillColor;	// The color to fill the item rectangle with. The OFLAG_FILL flag must also be set for this parameter to be effective.
int border;	// Select one or more borders to draw:

OUTLINE_LEFT	Draw left border

`OUTLINE_RIGHT`	`Draw right border`
`OUTLINE_TOP`	`Draw top border`
`OUTLINE_BOT`	`Draw bottom border`

int BorderType; // The pen selection to draw the border:

`PS_SOLID`	`Solid border`
`PS_DOT`	`Dotted line border`
`PS_DASH`	`Dashed line border`
`PS_DASHDOT`	`Dot-Dash pattern border`

int BorderThickness; // The border thickness in mm units.

COLORREF BorderColor; // The border color

BOOL repaint; // Set to TRUE to repaint the screen after this operation.

Return Value: This function returns TRUE when successful.

RepSetMsgCallback

Set a callback to receive the ReportEase Plus message.

BOOL RepSetMsgCallback(HWND, ReportId, ptr)

HWND hWnd	// When invoking this function for a Report Designer window, specify the report template edit window handle. Otherwise set it to NULL.
int ReportId;	// When invoking this function during a report execution session, specify the report id of the current session. Otherwise set it to 0.
MSG_CALLBACK ptr;	// A pointer to the callback function to receive the ReportEase Plus messages.

Comment: Normally, ReportEase Plus sends the messages to the parent window handle. Instead, you can set a callback using this function to receive those messages. A callback function must be prototypes as following:

LRESULT MsgCallback(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)

Return Value: This function returns TRUE when successful.

See Also
RepGetParentPtr

RepSetParentPtr

Set a parent pointer for this session.

BOOL RepSetParentPtr(HWND, ReportId, ptr)

HWND hWnd	// When invoking this function for a Report Designer window, specify the report template edit window handle. Otherwise set it to NULL.
int ReportId;	// When invoking this function during a report execution session, specify the report id of the current session. Otherwise set it to 0.
LPVOID ptr;	// A pointer to save for the parent window.

Comment: This function might be used by your application to store a pointer value for the current session. For instance, when wrapping a ReportEase Plus session within a class, your application can register the current class pointer.

Return Value: This function returns TRUE when successful.

See Also
RepGetParentPtr
RepSetMsgCallback

RepSetPrinter

Show user printer selection dialog box.

BOOL RepSetPrinter(hParentWnd, PrintSetup, ReportFile, persist)

HWND hParentWnd	// Parent window (report template) handle.
BOOL PrintSetup;	// Set to TRUE to show the print setup dialog, or set to FALSE to show the printer selection dialog box.
LPBYTE ReportFile;	// Name of the report template file. The header from the report template file is used to initialize the default printer parameters.
BOOL persits;	// Set to TRUE to retain the settings from this dialog box for all subsequent print jobs.

Description: This function should be called before Report Executor window is created.

Return Value: This function always returns TRUE.

See Also
RepGetCurPrinter

RepSetRtfAttach

Attach RTF file to print with the current report

BOOL RepSetRtfAttach(ReportId, FileName, before, FieldNames, FieldData)

BOOL RepSetRtfAttach2(ReportId, FileName, before, FieldNames, FieldData, continuous)

int ReportId;	// ReportId of the current report.
LPBYTE FileName;	// The name of the RTF file to attach
BOOL before;	// Set to TRUE to attach this RTF file at the beginning of the report. Set to FALSE to attach this file at the end of the report
LPBYTE FieldNames;	// You can use this field to conduct mail-merge on the file being attached. Please refer to TerMergeFields function in the TE Developer's Kit manual for the description of this field. Set to NULL if mail-merging is not desired.
LPBYTE FieldData;	// The data value strings for the field names contained in the FieldNames parameter. Set to NULL if mail-merging is not desired.
BOOL continuous;	// When multiple documents are attached. Set this field to TRUE to begin the current document on the same page as the previous attachment. This argument is ignored for the first attachment.

Description: You can call this function after calling the RepInit (or RvbInit) function to attach RTF documents to the current report. TE Developer's Kit must be installed to use this feature.

Return Value: This function always returns TRUE when successful.

RepSetModify

Set or reset the modification status for the current report template.

BOOL RepSetModify(hWnd, set)

HWND hWnd	// Window handle of the Report Designer window
BOOL set;	// Set to TRUE to turn-on the modification flag, or FALSE to reset the modification flag.

Normally, the editor sets the modification flag automatically when the user edits the report template. This function is used to override the modification flag.

Return Value: This function returns TRUE if successful.

Major Data Structures

This chapter describes the major data structures used by ReportEase Plus.

Strform

This structure is used to define the parameter list to call the Report Designer. This structure is defined in the REP.H file. The calling program must define a variable using this structure. The parameters must be passed using the pointer to this variable. Please refer to the *Report Designer Interface* chapter for a complete description of the structure members.

StrRep

This structure is used to define the parameter list to initialize a report execution session. This structure is defined in the REP.H file. The calling program must define a variable using this structure. The parameters must be passed using the pointer to this variable. Please refer to the *Report Executor Interface* chapter for a complete description of the structure members.

StrformHdr

This structure describes the report template file header. This structure is defined in the REP.H file. Your application can use this structure to read the report template name from the report template header. Your application can then display the available forms to the user to select. Member variables:

formSign:	(unsigned) A valid report template file will have a 2 byte code in the beginning of the file. The value of this code should be 0xDEBC (0xDEBA for win32).
name:	(char [52]) Name of the report template. The report template name may not exceed 50 characters.
DataSetName:	(char [20]) Data set to be used to produce the report. This value is of no internal significance to ReportEase Plus. However, your application can use this value to prepare the data before running the report.
TotalItems:	(int) Total number of screen items in the report template.
FieldCount:	(int) Total number of fields used in the report template.
BreakFieldCount:	(int) Total number of sort/break fields used in the report template.
FontCount:	(int) Total number of entires in the font table.
LeftMargin:	(float) Left margin (inches).
RightMargin:	(float) Right margin (inches)
TopMargin:	(float) Top margin (inches)
BottomMargin:	(float) Bottom margin (inches)

SelExp:	(int [52]) Report selection expression. This expression consists of fields, constants, functions and operator tokens. Only the first 50 integers may be used for this field.
Orientation:	(int) Specifies portrait (DMORIENT_PORTRAIT) or landscape (DMORIENT_LANDSCAPE) orientation for output.
PaperSize:	(int) Specified by using DMPAPER_* variables used by Windows' DEVMODE structure.
PaperLength:	(int) Specified in tenths of a millimeter. used only if the PaperSize variable is set to 0.
PaperWidth:	(int) specified in tenths of a millimeter. used only if the PaperSize variable is set to 0.
PrintQuality:	(int) Print quality specified using the DMRES_* variables used by Windows' DEVMODE structure.
PrinterName:	(char [52]) Name of the printer for which the current report template is designed.
PrinterDriver:	(char [52]) Driver name of the current printer.
flags:	(unsigned) The following flag bit can be set for a report (defined in REP.H):

```
RFLAG_TRIAL: Print trial records for report template

             adjustment.
```

Dateformat:	(int) Specifies the default date format. A 0 value for this field specifies MM/DD/YY format, where as a 1 value specifies the DD/MM/YY format.
RulerType:	(int) Specifies the ruler type used by the report template: RULER_INCH (inches), RULER_CM (centimeters), or RULER_OFF (ruler not used).
SecBannerHeight:	(int) Height of the section banner in 1/10 of millimeters.
reserved:	(char [148]) reserved for future use.

StrField

The StrField structure is used to define individual fields used in a report template. The Report Designer defines an array of fields using this structure. This structure is defined in the REP.H file. Member variables:

source: (int) This variable defines the source of the field. It can be set to one of the following values:

```
SRC_APPL:  Derived from your application.  This

           type of field will contain application

           data during the report execution.

SRC_CALC:  Calculation field.
```

```
SRC_SYS:   System field used for defining calendar
           date, time, record count, and paragraph
           break field, etc.
SRC_CONST: Defines the constants used in an
           expression.
SRC_NONE:  Indicates a deleted field.
SRC_DLG:   Dialog field.  Defined by the user using
           the dialog field menu option.
```

name:	(char [52]) This field contains the field name. If the file name is a part of the field name, it is separated from the field name using a '->' separator, i.e. CUSTOMER->ADDRESS. A field name may not exceed 50 characters.
FileId:	(int) This value is supplied by your application in the field selection routine. Your application can use this value during the report execution session to determine the file that contains a field.
FieldId:	(int) This value is supplied by your application in the field selection routine. Your application can use this value during the report execution session to determine a field.
type:	(int) This variable specifies the field data type. The field data types are defined in the REP.H file:

```
TYPE_NUM   Numeric field. Stored as a long variable.
TYPE_DBL   Float type numeric field.  Stored as a
           double variable.
TYPE_ALPHA Text field.  Stored using a character
           pointer.
TYPE_LOGICAL: Logical field.  Stored as a long
           variable.  The valid values are 0 or 1.
TYPE_DATE: Date Field.  Stored as a long variable
           in either YYMMDD format or YYYYMMDD
           format.
TYPE_PICT: Picture id field.  Stored as a long
           variable.  During the report execution
           time, your application routine
           (DrawPicture) is called to draw this
```

type of data.

width:	(int) default width of the field. The number of data characters in the text field may not exceed the value specified by this variable.
DecPlaces:	(int) Specifies the number of digits to the right of the decimal point for the numeric and double fields.
AllowChanges	(int) This field can be set to a FALSE value by your application to protect the field from any changes by the user.
InUse:	(int) This variable is set to a FALSE value when a field is deleted or if it is not being used.
flags:	(unsigned) The following flag bits can be set for a field (defined in the REP.H file):

```
FLAG_SUP_ZERO:   Do not print a numeric or float
                 field with a zero value.
FLAG_PAD_ZERO:   Insert zeroes in front of a numeric
                 or float field to yield the
                 required field width.
FLAG_CAPS:       Capitalize the text field.
FLAG_FIRST_CAP:  Capitalize the first character of
                 each word in a text field.
FLAG_SMALL:      Convert the text field to the lower
                 case letters.
FLAG_COMMA:      Specifies a comma format for a
                 numeric or float field.
FLAG_WRAP:       Specifies that the text field will
                 be wrapped if it is longer than the
                 display width.  An overflow field
                 must be defined underneath the text
                 field.
FLAG_WORD_WRAP:  Specifies that the text field will
                 be word wrapped if it is longer
                 than the display width.  An
                 overflow field must be defined
```

```
                  underneath the text field.
FLAG_RETAIN:      Specifies that a summary field will
                  retain its value after printing.
                  Normally, a summary field is
                  cleared after printing.
```

SumType:	(int) This variable defines the summary type:

```
SUM_NONE:   Print the value of the field.
SUM_TOTAL:  Print the total for the field.
SUM_AVERAGE:Print the average for the field.
SUM_COUNT:  Print the number of records between the
            breaks.
SUM_MAX:    Print the largest value for the field
            between the breaks.
SUM_MIN:    Print the smallest value for the field
            between the breaks.
```

SysIdx:	(int) For a system field, this variable specifies the index into the system field table. For the dialog fields, it specifies the index into the dialog field table.
Dateformat:	(int) The date format (for output) can be one of the following:

```
DT_MMDDYY:    Example: 4/30/92
DT_DDMMYY:    Example: 30/4/92
DT_MMDDYYYY:  Example: 4/30/1992
DT_MMMDDYYYY: Example: Apr 30, 1992
```

DateDelim:	(char [2]) This variable specifies two separators that are used in a date format. The default is '/'.
CurrencySymbol:	(char [4]) Allows you to specify the currency symbol ($, Rs, Fr. etc) for a numeric and float fields.
LogicalSymbols:	(char [2]) Allows you to specify the symbols to display logical values, such as Y,N,T,F,0,1.
NegSignPrefix:	(char [4]) The prefix to be printed for a negative value (example '-').
NegSignSuffix:	(char [4]) The suffix to be printed for a negative value (example 'CR').
PosSignPrefix:	(char [4]) The prefix to be printed for a positive value (example + or

	nothing).
PosSignPrefix:	(char [4]) The suffix to be printed for a positive value (example 'DR' or anything else).
CalExp:	(unsigned int [52]) This field contains the expression for a calculation field. The expression is defined in terms of other fields, constants, functions and operator tokens. An expression may not be greater than 50 integers.
CharData:	(LPSTR) This field is used by your application to supply text data. The report executor provides a valid character pointer in this variable for the text (TYPE_TEXT) fields. Your application copies the text data to the location pointed by the variable. The text data must be NULL terminated and should not be longer than the width specified by the 'width' variable.
NumData:	(long) This field is used by your application to provide numeric, logical, date, and picture id (TYPE_NUM, TYPE_LOGICAL, TYPE_DATE, TYPE_PICT) data.
DblData:	(double) This field is used by your application to provide float type data.
HoldNum:	(long) Used by the report executor to accumulate summary data for the numeric field.
HoldDbl:	(double) Used by the report executor to accumulate summary data for the float type field.
count:	(long) Stores the number of records within a sort break.
section:	(int) For the sort and break fields, this variable stores the section location of the field.
misc:	(int) Used for temporary calculations.
ParaChar:	(char [2]) For a word-wrapped text field with multiple paragraphs, use this variable to specify the new paragraph indicator in the first byte. The report executor examines the text data to search for the new paragraph indicator character. The text following the new paragraph indicator character is placed on the next paragraph. The second byte for this variable should be set to NULL.
reserved:	(char [18]) for future use.

StrBreakField

This structure contains the index to the sort and break (comparison) fields for each sort break. This structure is defined in the REP1.H file. Member variables:

SortField:	(int) Index of the corresponding sort field. The index points to a field in the field array.
CompField:	(int) Index of the corresponding comparison field. The index points to a field in the field array.
section:	(int) Index of the sort section. The index points to a section in the section array

StrSection

This structure defines the properties of each section in a report template. This structure is defined in the REP1.H file. Member variables:

InUse:	(int) A TRUE value of this variable indicates that the current section is being used in the report template.
flags:	(unsigned) Various bits in this flag can be set to indicate the following properties:

```
SFLAG_PAGE_BEF:    Advance to the new page before

                   printing this section.

SFLAG_PAGE_AFT:    Advance to the new page after

                   printing this section.

SFLAG_REPRINT:     Reprint this section on every

                   page break.

SFLAG_TRIM_BEFORE: Trim extra space before the top

                   most item in the section.

SFLAG_TRIM_AFTER:  Trim extra space after the

                   bottom most item in the section.
```

SelExp:	(int [52]) The selection criteria that must be met to print a section. A selection expression consists of fields, constants, functions and operator tokens.
selected:	(int) This variable stores the result upon the execution of the selection expression.
ScrItem:	(int) Screen item number of the section banner object.
height:	(int) The combined height (in logical units) of all lines in the section. This field is used by the report executor.
FirstItem:	(int) The position of the first screen item belonging to this section. This field is used by the report executor.
ItemCount:	(int) Number of screen items which belong to this section. This field is used by the report executor.
reserved:	(char [16]) Reserved for future use.

StrSysField

This structure maintains a table of system variables. The structure is defined in the REP1.H file. Member variables:

name:	(char [51]) The name of the system variable.

type:	(int) The variable type can have one of these values: TYPE_NUM, TYPE_DBL, TYPE_TEXT, TYPE_DATE, and TYPE_LOGICAL.
width:	(int) The initial width of the variable.

StrDlgField

This structure maintains the table of dialog fields created by the user. This structure is defined in the REP1.H file. Member variables:

InUse:	(int) Indicates a valid dialog field.
name:	(char [52]) The name of the dialog field as entered by the user.
prompt:	(char [52]) The text to be displayed to prompt the user for data.
type:	(int) The variable type can have one of these values: TYPE_NUM, TYPE_DBL, TYPE_TEXT, TYPE_DATE, and TYPE_LOGICAL.
PromptOrder:	(int) When more than one dialog field has been created, this variable specifies the order in which the user will be prompted for data.
CharData:	(char [52]) This variable stores the user entered text data for the TYPE_TEXT fields.
NumData:	(long) This variable stores the user entered numeric data for the TYPE_NUM, TYPE_DATE and TYPE_LOGICAL fields.
DblData:	(double) This variable stores the user entered numeric data for the TYPE_DBL fields.
x:	(int) X position where the dialog box will be displayed (future use).
y:	(int) Y position where the dialog box will be displayed (future use).
width:	(int) Width (in number of characters) of the dialog field data.
ValExp:	(int [52]) The dialog field validation expression. This field is not being used currently.
reserved:	(char [20]) Reserved for future use.

StrFont

This structure is used to define the fonts or picture bitmap used by a report template. This structure is defined in the REP1.H file. Member variables:

InUse:	(BOOL) Turned on when the font structure is in use.
IsPict:	(BOOL) TRUE if this font entry actually represents a picture bitmap.
hFont:	(HFONT) handle to the current font.
lFont:	(LOGFONT) Logical font structure.
hBM:	(HBITMAP) Handle to the current bitmap when the 'IsPict' flag is TRUE.

The following 8 variables are applicable only when using a bitmap.

ImageSize:	(DWORD) Size of the device independent bitmap image.
InfoSize:	(DWORD) Size of the device independent bitmap information header.
hImage:	(HANDLE) Handle to the bitmap data.
hInfo:	(HANDLE) Handle to the bitmap info header.
bmHeight:	(int) Actual height of the stored bitmap.
bmWidth:	(int) Actual width of the stored bitmap.
PictHeight:	(int) Height translated to the point size units.
PictWidth:	(int) Width translated to the point size units.
height:	(int) Height of the font or picture stored in logical units (1/10 mm).
BaseHeight:	(int) Ascent specified in the logical units (1/10 mm).
CharWidth:	(int [256]) Unit height of each character in the font.

ReportEase Plus File format

The ReportFile consists of 7 sections in the following sequence:

```
report template Header Block

Screen Object Block

Field Block

Break Field Block

Section Block

Dialog Block

Font Table
```

report template Header Block:

The file begins with a header block. The contents of the header block is defined by the StrformHdr structure. The size of the header block is equal to the sizeof(struct StrformHdr). The header is stored as a packed structure. In other words, there is no gap between adjacent header variables. The first 2 bytes of the header block contains a signature for ReportEase Plus:

```
First Byte =  0xBC

Second Byte =  0xDE
```

The *TotalItems* field in the header defines the total number of screen items in the screen object block. The *FieldCount* element defines the size of the Field Block. The *BreakFieldCount* element defines the size of the BreakField block. The *FontCount* defines the number of fonts used by the font.

Screen Object Block:

This block contains the screen objects used in the report template. The total number of screen items is defined by the TotalItems variable in the header structure.

Each screen object contains a 'font' field, which is an index into the StrFont array (last block in the file). The 'field' type screen object also includes a 'field' variable, which is an index into the field table (see Field Block). Each screen object also contains a 'section' variable which is an index into the section table (see Section Block).

Field Block:

This block contains the field table. The number of entries in the field table is defined by the FieldCount variable in the header structure. The size of the individual field table element is equal to the sizeof(struct StrField). The data is stored in the packed structure. In other words, there is no gap between the adjacent variables in the structure.

Break Field Block:

This section stores the sort/break field table. The number of entries in the break field table is defined by the BreakFieldCount variable in the header structure. The size of the individual break field element is equal to the sizeof(struct StrBreakField). The data is stored in the packed structure. The break field table summarizes the sort and break fields used by a report template.

Section Block:

This block stores the section attributes for all 23 sections allowed by the Report Designer. For a description of various sections allowed in the report template, refer to the SEC_ global constants in the REP_DEF.H file. The size of each section block is equal to the sizeof(StrSection). The data is stored in the packed structure.

Dialog Field Block:

This block stores the information about all 12 dialog fields allowed by the Report Designer. The size of each dialog field block is equal to the sizeof(StrDlgField). The data is stored in the packed structure.

Font Block:

This section stores the font table. It begins with a signature byte of value 0xBE. The signature byte is followed by the data for each font. The number of entries in the font table is given by the *FontCount* variable in the report template header structure. Each entry represents a font or a picture bitmap.

When the first integer byte of the font entry is non-zero, it is followed by picture bitmap information, as following:

```
Picture HeightWORD

Picture WidthWORD

Image   SizeDWORD

Info    SizeDWORD

Image   string of size Image Size

Info    string of size Info Size
```

When the first integer byte of the font entry is zero, it is followed by the LOGFONT structure (refer to Windows SDK for the description of the LOGFONT structure). The LOGFONT structure provides the font specification.

Sort and Join Utilities

The package includes a general purpose utility DLL (UTIL or UTIL32) containing the file sort and join API functions. Your program may use these functions but this DLL is not required by ReportEase Plus DLL. This chapter describes the syntax, usage and limitations of these APIs. Some applications have sorting and joining functions built into their database facility. However, if your application needs these features, please refer to the description below.

FileSort

Syntax:

```
int FileSort(LPSTR InputFile, LPSTR OutputFile, int NumKeys,

             LPINT KeyTable)
```

The first argument specifies the name of the input text file to sort. Each line in the file must be delimited by <CR> and newline characters. A line of text can contain a number of fields separated by the comma characters. The text field containing special characters must be enclosed within the quotation marks. The number of lines in the file must not exceed 30000 lines. The sort function reads the entire text file into memory. Therefore, the file size is also limited by the available memory.

The second argument specifies the output file name. The third argument specified the number of sort fields to be used for sorting. The fourth argument is a pointer to a table containing the field numbers for the sort fields. The sort field can be a number between 1 and the total number of fields in a text line. The sort field represented by the field number must be a text field. You can specify up to 10 sort keys. The sorting is always in the ascending order.

If the output file name is NULL, the output sort file name is constructed using the input file name prefix and a .SRT extension.

Return Value: This function returns TRUE when successful.

```
Examples:

1. int KeyTable[3]={3,5,6};

   FileSort("myfile", "outfile.srt",3,KeyTable);
```

This example sorts the *myfile* file to create *myfile.srt* as the sort file. The field numbers 3,5 and 6 are used for sorting.

```
2. int KeyTable[1]={1};

   FileSort("myfile.txt", "outfile.srt",1,KeyTable);
```

This example sorts the *myfile.txt* to create *myfile.srt* as the sort file. The first field is used for sorting.

FileJoin

Syntax:

```
int FileJoin(LPSTR InputFile1, int field1, LPSTR  InputFile2,

             int  field2, LPSTR OutputFile)

InputFile1: The first file to join

field1:     The common field in the first file.

InputFile2: The second file to join

field2:     The common field in the second file.

OutputFile: The name of the output file.
```

This function joins the second file to the first file creating an output file (Argument #5). The two files to be joined must have a common field. For Example, the CUSTOMER.DB and SALES.DB have a common field called customer id. The output file is sorted in the same order as the first file.

The input files must be in the text format. The fields within the text line must be separated by the comma character. The text fields containing special characters must be enclosed within the quotation characters. The input files may not exceed 30000 lines each. The FileJoin function reads both input files into memory before the join operation. Therefore, the file size is also limited by the available memory.

Return Value: This function returns TRUE when successful.

Examples:

```
1. FileJoin("customer.db" , 1 ,  "sales.db", 1 ,

            "customer.set");
```

This example joins the sales.db file to the customer.db file. The output file name is customer.set. Both input files have field number 1 as the common field.

```
2. FileJoin("test1",  5, "test2",  2, " test.set")
```

This example joins the test2 file to the test1 file. The output file name is test.set. The field number 5 in the test1 file represents the same data as the field number 2 in the test2 file.

Product Concepts

In This Chapter

User Commands

The Report Designer commands can be selected by using the menu or by using the speed keys. Each menu item also shows the speed key for the item. To get help on any menu item, highlight the menu item and hit the F1 function key. You can also use the Help menu option to see the index of help topics.

This chapter describes the Report Designer commands. We will discuss the commands by the order in which they appear in the menu.

In This Chapter

File Menu

This submenu contains the following selections:

Save

Use this command to save the changes to the current report template file. If a file name has not been assigned yet, this option will allow you to enter the file name.

Save As

Use this command to save a report template template to a new file. This option is used to create a copy of the exiting report template. The editor will prompt you for the name of the new report template file.

Report Parameters

The report parameters are entered using a dialog box. This option allows you to enter a description for the report template. In addition, you can specify the following parameters.

Page Margins:	You can specify top, bottom, left and right margins in inches. The Report Designer applies the margin information to the selected printer (see Printer Setup) to calculate the report width. The report width is indicated by the top ruler.
Date format:	This option lets you specify the default date format. Use an 'M' for the MM/DD/YY format or a 'D' for the DD/MM/YY format. This format is applicable to any date information entered by the user during report execution, and any date constants used in field expressions and filters.
Ruler Type:	Use this option to show the ruler in inches or centimeters. You can also turnoff the ruler.
Print Trial Records:	This option is useful when printing on a preprinted report template such as address labels. When this option is enabled, the report executor will print trial records to allow you to adjust the report template on the printer.
Always print to Default Printer:	This option would instruct the Report Executor to print the report to the current windows printer and ignore any selected printer when the report template was designed.

Edit Report Selection Criteria

This option allows you to specify a condition that must be met for a record to be selected by the report executor. In the absence of a selection criteria, all records are selected. The selection criteria is specified by entering a calculation expression. This expression must evaluate to a TRUE or FALSE value. During the report execution session, this expression will be evaluated for each record. A record will be selected if the expression evaluates to a TRUE value. For a detail description of calculation expressions, please refer to the *Calculation Expression* chapter.

Examples:

1. CUSTOMER->ID>="0010".AND.CUSTOMER->ID<="0040"

This expression will select records with the customer id between "0010" and "0040" inclusive.

2.SALES->AMOUNT>1000.

This expression will select records with the sales amount greater than $1000.

3. SALES->DATE>DLG->BEGIN_DATE

This expression will select records with the transaction date greater than the date specified by the dialog field BEGIN_DATE (see *Field Concepts*).

Printer Setup

The default printer is automatically assigned to a new report template. Use this option to select a different printer from the list of installed printers. This option also allows you to change the printer parameters for the selected printer. The selected printer and the corresponding setup parameters affect the width, height and orientation (portrait or landscape) of the printer output. Although you can opt to print to a screen window during the report execution session, a printer must always be associated with a report template. When screen output is selected during report execution, the printer information is used to provide wysiwyg screen output whenever possible

Edit Menu

This menu allows you to edit the appearance and placement of the screen objects. Every report template object (label, line, field, or picture) is enclosed in an object box. The object box boundary lines are invisible by default. This menu allows you to specify the attributes for the object boundaries, box color, item placement within the box, and text color and fonts. This menu also includes commands to insert or delete the spaces from the report template

Cut, Copy, Paste and Erase

These commands allow you to copy an item or a selection of items to the clipboard and paste the items from the clipboard to the current report template. The 'Erase' option allows you to erase the clipboard.

Undo

Use this function to undo the editing command.

Position Text

Use this option to position the text within the item boundaries. The text can be justified on the left, right, top, or bottom edges, or it can be centered horizontally or vertically. This option is valid for the 'label' and 'field' type items only.

Item Outlines

Use this option to select the item boundaries (left, right, top, bottom) to draw for one or more selected items. You can also specify the color and width of the boundary lines.

This dialog box also allows you to set the multi-line property for a label.

Item Background

Use this option to set the background color or pattern for one or more selected items.

Centering

This option is used to center horizontally one or more selected items. When more than one item are selected, the Report Designer first centers the selection rectangle and then moves the selected items such that the position of the selected items relative to the selection rectangle does not change.

Delete an Object

Use this option to delete one or more currently selected items.

If the current section is being deleted, the program asks for your confirmation before the deletion. All items within the section are also deleted.

Default Item Font

Use this function to change the default font and color for one or more selected objects. This option is valid for the field and label type objects only.

The default item font is used to display the item during the design mode. It is also used to print the item during report execution *except* when a conditional font is selected for the item.

Conditional Item Font

Use this function to select a font expression for one or more selected objects. This option is valid for the field and label type objects only.

During the report execution time, the report executor evaluates any conditional font expression specified for the item. When a font expression is evaluated, it results in a text string representing the Named Font to be used in place of the default item font. Consider a Name Font 'Font A' which represents an 'Arial' typeface, 12 point size, and red color font. Also consider the following font expression used for an item:

```
 .if.sales->amount<10.then."Font A".else.""
```

This font expression will result in "Font A" if the sales->amount is less than $10, otherwise will it will result in a blank string. This will force the report executor to use "Font A" when the sales amount is less than $10. The default item font will be used when the sales amount is $10 or larger than $10.

Name Font:

This submenu allows you to create, modify or delete a Name Font. A named font is used inside a font expression (see example above).

Font Expression:

This submenu allows you to create, modify, delete or rename a font expression. A font expression usually consists of a conditional statement to select a Named Font (see example above).

Please refer to the 'Detail Sales Report' in the demo program for an example of using the Named Font and Font Expression.

Default Style Item

A default Style Item is an item whose style attributes can be applied to other items. You can use this option to designate the current item as the default style item. A check mark on this menu item indicates a default style item. This option can also be used to reset the default style item designation for the current item.

Apply Default Item Style

This option can be used to apply the item attributes (such as borders, background color, etc) to the selected items.

Expand Horizontally

Use this option to create horizontal spaces by moving the items horizontally. For example, consider three items, A, B, and C placed horizontally. If you need to insert a new item between the items A and B, you can use this function to create the desired space between these two items and place the new item in the newly created space. To move the items B and C toward right, create a selection rectangle after the item A and select this option. The width of the selection rectangle specifies the movement of the items B and C toward the right (note that the selection rectangle does not need to include all items to be moved). All items toward the right of the selection rectangle and with the vertical placement between the vertical space spanned by the selection rectangle are moved.

Expand Vertically

Use this option to create additional vertical space by moving the items downward. For example, consider three items, A, B, and C placed vertically. If you need to insert a new item between items A and B, you can use this function to create the desired space between these two items and place the new item in the newly created space. To move the items B and C downward, create a selection rectangle below the item A and select this option. The height of the selection rectangle specifies the downward movement of the items B and C (note that the selection rectangle does not need to include all items to be moved). All items below the selection rectangle are moved.

This option also expands (vertically) the current section by the height of the selection rectangle.

Compress Horizontally

Use this option to delete extra horizontal space by moving the items horizontally. For example, consider three items, A, B, and C placed horizontally. You can use this function to bring the items B and C closer to the item A. To move the items B and C toward left, create a selection rectangle after the item A and select this option. The width of the selection rectangle specifies the movement of the items B and C toward left (note that the selection rectangle does not need to include all items to be moved). All items toward the right of the selection rectangle and with the vertical placement between the vertical space spanned by the selection rectangle are moved.

Compress Vertically

Use this option to delete vertical space by moving the items upward. For example, consider three items, A, B, and C placed vertically. You can use this function to bring the items B and C closer to the item A. To move the items B and C upward, create a selection rectangle below the item A and select this option. The height of the selection rectangle specifies the upward movement of the items B and C (note that the selection rectangle does not need to include all items to be moved). All items below the selection rectangle are moved.

This option also shrinks (vertically) the current section by the height of the selection rectangle.

Field Menu

When you select a 'field' type item, the corresponding field name is displayed on the status line. A field name typically contains a '->' separator. The text to the left of the separator indicates the file name, and the text to the right indicates the field name (within the file).

A field can be enlarged or reduced by simply pulling the sizing tabs. A field, like other screen items, can be moved by dragging and dropping at the desired location.

The field menu contains these options:

Insert New Field

Edit Current Field

Edit Field Expression

Dialog Field Table

Insert New Field

This submenu allows you to insert a field into the report template. This option will display a list of fields to choose from. When you select a field, the Report Designer displays a cursor rectangle. Use the mouse to position the cursor rectangle and click any mouse button. The new field is created where the cursor rectangle is positioned.

The submenu allows you to insert four types of fields (see also *Field Concepts*):

Data Field:	Data fields are associated with the data records. This submenu shows you selections for the data files and data fields.
Calculated Field:	A calculated field is specified using a calculation expression (see *Calculation Expression*). You must also provide a unique name for the calculation field. The calculated fields are used to print values that are not directly available by any data field. For Example, the profit amount can be calculated by multiplying the sales amount (data field) by the profit margin.
System Field	The system fields provide system depended information, such as calendar date, time, page number, record count, total page count, and paragraph break field. The calendar date and page number fields are typically printed on the page header. The paragraph break field can be used in a calculation expression to create multiple paragraph text (wrapped text). A system field called SORT_ITEM_COUNT can be used to print total number of detail records at any sort level. Consider a report which prints sales item by customer by state. You can use this system field to print the count of total sales items at the customer sort footer or at the state sort footer.
Dialog Field:	The dialog fields must be created before it can be selected. Use the *Dialog Field* option from the *Field* menu to create the dialog fields. The dialog fields are used to prompt the user for data before the report execution (example: report dates). The dialog fields can also be included in the report template. For example, you may create two dialog fields, BEGIN_DATE and END_DATE to prompt the user for the beginning and ending dates for

the report. You can then print these two dates on the report header by inserting them in proper places. The dialog fields can also be used in a report selection criteria.

When you insert a numeric or float field in a footer section, the Report Designer automatically assigns a 'total' attribute to the field. This attribute instructs the report executor to print the total for that field. You can change this attribute by using the 'Edit Current Field' option.

Edit Current Field

This selection is used to edit the specification about the currently selected field. This option presents different editing options for different types of fields. ReportEase Plus supports these types of fields:

Numeric

Float

Text

Date

Logical

Numeric and Float Fields: The following edit options are available for a numeric or float field:

Number of Decimal Places: This option determines the number of digits to the right of the decimal point.

Currency Symbol: You may wish to specify a currency symbol ($, Rs, Fr, etc) for fields that represent money.

Prefix and Suffix for Negative Values: This option allows you to decide the appearance of a negative value. For example, if you wish to enclose a negative value in parentheses, specify '(' for the prefix and ')' for the suffix. If you simply wish to show the '-' symbol, enter '-' for the prefix and nothing for the suffix.

Prefix and Suffix for Positive Values: This option allows you to decide the appearance of a positive value. For example, if you wish to enclose a positive value in parentheses, specify '(' for the prefix and ')' for the suffix. If you do not wish to show any symbol for the positive value, enter blanks for the prefix and the suffix.

Suppress Zero Fields: This option suppresses the printing of a field if it contains a zero value.

Suppress Trailing Zeroes: This option suppresses the trailing zeroes for a decimal field. For example, values 1.30 and 1.00 would print 1.3 and 1 respectively.

Pad With Zeros: This option will insert zeros before the field if the field value occupies less spaces than specified by the field width.

Invisible: You can specify an 'invisible' attribute for an item. When this attribute is set, the item is visible only during the design session. This option can be useful for creating intermediate calculation fields that you do not wish to display on the report.

The following two options are available for the fields located in a footer section only:

Print Value: Using this option you can instruct ReportEase Plus to print totals, average, maximum, minimum or count of a field (See *Field Concepts*). If you simply wish to print the field value for the last record before the footer section, specify 'value' for this option.

Retain Value After Printing: Normally, when a total (or average, maximum, minimum, count) is printed, the internal accumulator is cleared to start the next iteration of the section from zero. However, if you wish to print the running totals, select 'Y' for this option.

Text Fields: The following formatting options are available for a text field:

Capitalize All: This option will capitalize all characters in the field.

Cap First Letter: This option will capitalize the first letter of every word in the field.

Wrap and Word Wrap: These options are used to wrap a text field which is longer than the width allowed by the field on the report template. The *Wrap* option wraps the text that is larger than the field width. Whereas, the *Word Wrap* option breaks the text at the previous word boundary.

To specify more than one line for a wrapped field, simply pull the bottom sizing tab downward. When you release the mouse button, the Report Designer will show multiple lines in the field object box. Using this technique, you can increase the size of the wrap field such that it contains the desired number of lines. When a memo field is expected to contain a large number of lines, you can use the 'Variable number of lines' option. This option will compress the space after the last text line.

Variable Number of Wrapped Lines: Use this option with a wrapped field that may have *more or less* data than what can be contained in the field box. Normally, you should size the field box to contain the largest possible text data.

Invisible: You can specify an 'invisible' attribute for an item. When this attribute is set, the item is visible only during the design session. This option can be useful for creating intermediate calculation fields that you do not wish to display on the report.

Date Field: The following edit options are available for a date field:

Date format: The following date options are available:

```
format    Example

MMDDYY    4/30/92

DDMMYY    30/4/92

MMDDYYYY  4/30/1992

MMMDDYYYY Apr 30, 1992
```

Delimiter: The MMDDYY, DDMMYY and MMDDYYYY date formats use a delimiter to separate month, day and year. You can specify the value of this delimiter using this option.

Logical Field: This option allows you to specify the text that should be printed for the TRUE and FALSE value of a logical field.

A field can also be edited by simply double clicking on the desired field to edit its attributes.

Edit Field Expression

This selection allows you to edit the calculation expression used for the current calculation field. When you select this option, the Report Designer will display the current field expression and let you edit it. For a complete description of calculation expressions, refer to a later chapter.

Dialog Field Table

This selection is used to manipulate the dialog field table. A dialog field must be created before it can be inserted in the report template. A dialog field is used to prompt the user for data before running the report. The field can also be inserted in the report template to print the user selected values. This field can be used in the report selection criteria to select the records according to the user entered value for a dialog field.

This selection allows you to create new dialog fields, modify existing dialog fields, or to delete a dialog field.

Create a Dialog Field: This option lets you create a new dialog field. The user is prompted for the name of the dialog field and the field type. The field type can be one of the following:

```
Text

Numeric

Float

Date

Logical
```

Once a dialog field is created in the dialog table, it can be inserted in the report template by using the *Insert New Field* option from the field menu. A dialog field can appear in more than one place within a report template. When a dialog field is selected, the status area shows the dialog field name with a 'DLG->' prefix.

Modify a Dialog Field: This option can be used to modify the parameters for a dialog field in the dialog table. This option displays the list of fields from the dialog table and lets the user select a field to modify. You can modify the following parameters for a dialog field:

User Prompt: The text to be displayed to prompt the user for the data.

Prompt Order: When more than one dialog fields are used, this option allows you to enter the order in which the fields should be prompted.

Width: Width of the field given in number of characters.

Delete a Dialog Field: This option allows you to delete a field from the dialog table. The program shows the list of fields in the dialog table, and allows the user to select one field to delete. The chosen field is deleted from the dialog box. The dialog field must be deleted from the report template, and removed from any calculation expression, before it can be deleted from the dialog table.

Default Style Field

A default Style Field is a field whose attributes can be applied to other fields of the same field type (text, numeric, decimal, and date). You can use this option to designate the current field as the default style field. A check mark on this menu item indicates a default style field. There can be one default style field for each field type. This option can also be used to reset the default style field designation for the current field.

Apply Default Field Style

This option can be used to apply the field attributes (such as word-wrap, decimal position, etc) to the selected fields.

Rename Calculation Field

This option allows you to rename a calculation field. It is useful for renaming the calculation field pasted from the clipboard.

Section Menu

The ReportEase Plus forms consist of one or more sections:

Page Header and Footer sections

Report Header and Footer sections

Sort Headers and Footers sections

Detail Sections

The section menu allows you to create a new section, edit the parameters for an existing section, or to delete the current section

Insert New Section

When you select this option, the Report Designer shows a list of sections to choose from. This list contains the sections that do not already exist. Furthermore, the list will show a sort header or detail section only if the higher level section is already selected. This option will not let you select a sort footer section unless the corresponding section header is selected first.

When a header section is selected, the Report Designer shows you a list of sort fields to choose from. Highlight the sort field that you would like to associate with the sort header section. By choosing a sort field, you instruct the application to sort the record using that field. If your application has two sort sections, then the records will be sorted using those two sort fields. The sort field #1 will be the primary sort, and the sort field #2 will be the secondary sort.

When you create a new section, the Report Designer inserts the new section in the proper order within the report template.

Edit Current Section

To edit the section parameters for a section, select the section or select any item within the section. This option allows you to modify the following section parameters:

Advance page Before Section: This option instructs the report executor to advance to the next page before printing the data for the current section. For example, you may use this option to print a sort header section on a new page.

Advance page After Section: This option instructs the report executor to advance to the next page after printing all the items for the current section. For example, you may use this option to advance to the next page after printing the totals for the current section.

Compress Space Before the First Item: This option instructs the Report Designer not to print the additional space between the beginning of the section line and the topmost item.

Compress Space After the First Item: This option instructs the Report Designer not to print the additional space between the ending line of the section line and the bottom most item.

When a section includes a word wrapped field, this option is particularly useful to suppress the additional space when a memo field is smaller than the field rectangle. This technique allows you to create a wrapped field rectangle big enough to accommodate the biggest possible memo field data. The section will be automatically compressed when the text is smaller than the field rectangle.

Reprint With Page Header: This option is valid only for a header section. When this option is enabled, the current sort section header will be printed with every page header. For example, for a customer report with a large number of transactions, the customer name can be printed on every page.

Reset Page Number on Section Break: This option is valid only for a header section. When this option is enabled, the system *page number* field is reset to 1 after printing the corresponding footer for this header section.

Number of Records Across: This option is valid only for the detail sections (the detail section is used to print the individual records). Further, this option is not available when more than one detail section is used. This option can be used to print more than one record across the page. Please refer to the *Sales Summary by Date (SUMDATE.FP)* demo report for an example of printing more than one records across. This option can also be used to print labels when you wish to print more than one label across the page.

Delete

This option allows the user to delete the currently selected section.

Sort Field

This option is valid for a header section only. It is used to change the sort field associated with the sort header section. If you change the sort field, you may wish to change the break field also.

Break Field

This option is valid for a header section only. The break fields are used to determine the section break. In a typical report, the break field will be the same as the sort field. When you insert a new sort section, the Report Designer automatically creates a break field which is the same as the sort field. However, using this option, you can specify a different break field. Unlike the sort fields, a break field can be a data field or a calculated field.

Edit Selection Expression

Once a section is selected for a report template, you can still conditionally suppress the printing of the section by specifying a selection expression. This option allows you to enter an expression (see *Calculation Expression*). The selection expression must evaluate to a TRUE or FALSE value. Before printing a section, the report executor evaluates the selection expression, and suppresses the section print if the expression results in a FALSE value. In the absence of a selection expression, the section will always be printed.

Line, Label and Picture Commands

Create a Line

Use this option to draw a line. When you select this option, the Report Designer displays a positioning rectangle. Use the mouse to position the rectangle and click any mouse key. The line will be drawn within the position rectangle. The line size can be changed using the sizing tabs.

Edit Current Line

Use this option to edit the angle, color, and thickness of a 'line' type object.

Create a Label

Use this option to create a new label. When you select this option, the Report Designer displays a positioning rectangle. Use the mouse to position the rectangle and click any mouse key. The 'label' object will be created within the positioning rectangle. By default, the Report Designer inserts the text 'label' in the label item. The label text can be edited in the editing window.

You can turn-on the multi-line property for a label by selecting this option from the 'Outline' selection in the Edit Menu (or by double clicking on a label).

Edit Current Label

A label text can be edited by simply selecting the desired label item and clicking on the edit window.

When you insert or delete the text, the length of the label text changes. Normally, the Report Designer will automatically adjust the item box boundaries to completely enclose the new text. However, this automatic size adjustment ceases if you manually resized the item boundary by pulling on the sizing tab. This feature can be used to enclose the text in an item box larger than the default size.

Picture From Clipboard

Use this command to copy a picture bitmap from the clipboard.

When you select this option, the Report Designer creates a positioning rectangle equal to the dimensions of the picture. Use the mouse to position the picture rectangle and click any mouse key. The picture will be placed within the position rectangle. The picture size can be changed using the sizing tabs.

Picture From Disk File

Use this command to read a picture bitmap from a disk file.

When you select this option, the Report Designer creates a positioning rectangle equal to the dimensions of the picture. Use the mouse to position the picture rectangle and click any mouse key. The picture will be placed within the position rectangle. The picture size can be changed using the sizing tabs.

Object Arrangement Commands

This menu provides commands to help you position the report objects accurately. Select a set of objects to be arranged (see Object Selection Commands) and one of the following functions from the menu.

Align at Horizontal Top Edge

Use this option to horizontally align the top edge of the selected items to the top edge of the leftmost item in the selection.

Align at Horizontal Bottom Edge

Use this option to horizontally align the bottom edge of the selected items to the bottom edge of the leftmost item in the selection.

Align at Horizontal Center Line

Use this option to align the horizontal center line (imaginary) of the selected items to the center line of the leftmost item in the selection.

Align at Vertical Left Edge

Use this option to vertically align the left edge of the selected items to the left edge of the topmost item in the selection.

Align at Vertical Right Edge

Use this option to vertically align the right edge of the selected items to the right edge of the topmost item in the selection.

Align at Vertical Center Line

Use this option to align the vertical center line (imaginary) of the selected items to the center line of the topmost item in the selection.

Even Spacing Horizontally

Use this option to place the selected items horizontally at an equal distance from each other. The inter-item distance is equal to the distance between the first two leftmost items.

Even Spacing Vertically

Use this option to place the selected items vertically at an equal distance from each other. The inter-item distance is equal to the distance between the first two topmost items.

Set Even Width

Use this option to change the width of the selected items to the width of the topmost item.

Set Even Height

Use this option to change the height of the selected items to the height of the leftmost item.

Object Selection

Most Report Designer commands allow you to manipulate one or more selected items. To select a single item, simply click any mouse key on the desired item. The selected item is indicated by the 'dashed' boundary lines.

Multiple items can be selected using one of the two methods.

You can simply hold the control key and then click on the items to select. Click one more time to deselect a previously selected item. The selected items are indicated by the 'dashed' boundary lines. The primary item in the selection is indicated by black tab squares on it border.

The second method uses a a selection rectangle. To draw a selection rectangle, place the mouse cursor where you wish to begin the rectangle (mouse cursor must not be placed on an item) and click any mouse button. As the mouse button is depressed, move the cursor such that the rectangle includes the items that you wish to select, and release the mouse button. All items within the selection rectangle or 'touching' the selection rectangle are selected. To include or exclude additional items from the selection, hold the Shift key and click the mouse button on the desired item. The selected items are indicated by the 'dashed' boundary lines. The selection rectangle is indicated by a red color boundary.

You can stretch or compress the selection rectangle by pulling the sizing tabs with the mouse cursor. Thus it is possible to scroll the screen horizontally or vertically and include more items in the selection rectangle.

Report Executor Commands

These commands are available from the menu on the report executor window.

Copy

Copy selected pages to the clipboard.

Jump

Use this selection to position on a desired page number.

Print

This selection allows you to print a range of report pages to the current printer.

Preview

This selection turns on or off the print preview mode. In the print preview mode, the report is displayed one full page at a time

Save

This selection allows you to save the report to a disk file for later viewing or printing. The report can be saved in the RTF format (.RTF extension), HTML file (.HTM extension) or in the native format (.FR extension). The reports saved in the RTF format require an RTF viewer to display or print the report. The report saved in HTML format can be viewed using any standard HTML browser.

Field Concepts

A field represents a value to be printed in the report. This chapter discusses the placement of the fields, field value types, sources of fields and subtotals.

In This Chapter

Field Placement and Field Width

When a field is inserted using a menu option or the field button, the Report Designer displays a cursor rectangle. Use the mouse to position the cursor rectangle and click any mouse button. The new field is created where the cursor rectangle is positioned.

The field rectangle contains a text that represents the data type and the current format specification for the field. For a 'text' type field, the field rectangle contains a string of 'x' symbols. The 'x' symbols are capitalized if the capitalization is turned on for the field. The number of 'x' symbols is equal to the data width of the field or the maximum number of symbols that can be accommodated within the current rectangle. For a word-wrapped text field, you can increase the height of the field rectangle to specify multiple text lines containing the 'x' symbols.

For a numeric field, the field text can consist of the symbol '9', a decimal symbol and a set of comma symbols. The currency symbol is also shown when the field rectangle is large enough.

For a 'date' field, the field text describes the format of the date (example: mm/dd/yy, dd/mm/yy, mmm dd, yyyy etc). A logical field is denoted by a single 'Y' character.

When a field is selected, the name of the field appears on the status line. The field width is initially set to the default value. Once a field is inserted in the report template, you are free to adjust its location by selecting the item and dragging the mouse. The field width can be changed by simply pulling the sizing tabs. A field can be deleted by simply selecting the field and then pressing the 'del' key.

Field Value Types

A field is used to print a value. ReportEase Plus allows 6 types for field values.

Text Field:

The text field holds data that consists of characters and digits. The examples of the text fields would be a name, description or comments. The formatting options that are available for a text field include printing in capital letters, printing in small letters, capitalizing the first letter of each word in the field, and word wrapping. The word wrapping option allows a long text field to be printed in multiple lines.

Technical Note: Within the field structure, a text field has a type of TYPE_TEXT. During the report execution session, the application provides the text data using the CharData pointer (LPSTR) in the field structure.

Numeric and Float Fields:

These fields hold numeric values. The numeric fields hold whole numbers, whereas the Float fields hold floating point numbers. Numeric and Float fields are used to print numeric values such as dollar amount, quantity, measurements, etc. The formatting options that are available with these fields include number of decimal places, currency symbol, prefix and suffix for positive and negative numbers, zero padding or suppression, and comma formatting.

The decimal placement is treated differently for the numeric and float fields. For a float field, the digits on the right of the decimal point is given by the value of the field. However, the Report Designer allows you to print as many or as few digits to the right of the decimal point as you wish. As a result, the decimal place option simply performs truncation of decimal digits. For example, a float field with a value of 123.45678 can be printed as 123.4567, 123.456, 123.45 or simply 123. The number of decimal digits that are printed in these cases are 4, 3, 2, and 0 respectively.

A non float numeric field, on the other hand, is a whole number. The decimal field placement option in this case simply decides the number of digits to be printed to the right of the decimal point. The remaining digits are printed to the left of the decimal point. For example, a numeric field with a value of 1234567 can be printed as 123.4567 or 1234.567 or 1234567. The number of decimal digits that are printed in these cases are 4,3 and 0 respectively. Many applications prefer a numeric field over a float field for dollar values, as the numeric fields do not suffer report template rounding adjustments. However, the maximum value that can be represented using the numeric type may not exceed +-2,147,483,647. You must use the float field to represent a larger value.

Technical Note: Within the field structure, a numeric field is specified using the TYPE_NUM type, whereas a float field is specified using the TYPE_DBL type. During the report execution session, the application provides the numeric data using the NumData (long) variable and the float data using the DblData (double) variable.

Logical Field:

This type is used to represent a boolean value that can have only one of two value, such as yes/no, true/false, black/white. The formatting options available with this type allows you to specify the text to be printed for a true and false value.

Technical Note: Within the field structure, a logical field is specified using the

TYPE_LOGICAL type. During the report execution session, the application provides the data for this field using the NumData (long) field. The field value must be either 1 or 0.

Date Field:

This type is used to represent a date field. Various date formats are available including mm/dd/yy and dd/mm/yy.

Technical Note: Within the field structure, a date field is specified using the TYPE_DATE type. During the report execution session, the application provides the data for this field using the NumData (long) field. The long value for this field should be either YYMMDD or YYYYMMDD. If only 2 digits are provided for the year field, the report executor adds 1900 to the year value.

Picture Field:

This type of field denotes a picture id.

Technical Note: Within the field structure, a picture field is specified using the TYPE_PICT type. During the report execution session, the application provides the data (picture id) for this field using the NumData (long) field. The report executor actually calls a picture drawing routine in your application to draw the picture . This routine passes the current picture id as an argument.

Source of Field Data

A field may represent a data value, calculated value, system value or a user entered value.

Data Field

A data field is associated with the application file data. Your application provides a list of fields to choose from. Your application can choose to organize the data fields by data files. In the demo program, the customer and sales files provide the data fields. The customer file fields are indicated by the CUSTOMER-> prefix, whereas the sales file fields are indicated by the SALES-> prefix.

Technical Note: When the user wishes to insert a data field into a report template, the Report Designer calls a field selection routine provided by your application. This routine should allow the user to select a data field. The Report Designer allows your application to organize the fields in any way you wish. Therefore, your application is free to use a data set with any number of files in any relationship. The demo program first allows the user to select the file, and then displays the fields for the selected file. The demo program inserts the proper file prefix into the field name. The file name prefix must not be one of these reserved names: SYS, CALC and DLG. Your application passes the field name along with certain other information in a field structure (see Report Designer Interfaces).

Calculated Field

The calculated fields allow you to print a value which is derived using other fields, operators and functions. A calculated field is specified using a calculation expression. Refer to the *Calculation Expression* chapter for a complete description on this topic.

System Field

The system fields are used to print information such as the calendar date, time, page, and record number.

The system field list also includes a field called SECTION_ITEM_COUNT. This field can be used within the innermost section footer to print the number of detail records printed for the section.

Dialog Field

The dialog fields are used to get data from the user before executing the report. For example, you can prompt the user for the report dates. The Report Designer allows you to create a list of dialog fields. Like other fields, you can place a dialog field anywhere in the report. Typically, a dialog field for the report date will appear on the report header. You can also use the dialog fields in the report selection criteria. Thus, the report executor can filter out the records which don't meet a specific criteria. Refer to the SALES.FP report template (Detail Sales Report) for an example of the dialog fields.

Summary Fields:

The Report Designer allows you to summarize a numeric or float field. The summarized value of a field can be printed in any footer section (see Section Concepts). The following types of Summary Fields are available:

Totals:	The field values for all records within a section are accumulated.
Average:	The average field value for all records within a section is accumulated and then divided by the number of records within that section.
Minimum:	This Summary Field computes the minimum value of all records within a section.
Maximum:	This Summary Field computes the maximum value of all records within a section.
count:	This Summary Field prints the number of records with a section.

Section Concepts

In This Chapter

Section Types

ReportEase Plus organizes a report or a mail merge report template by sections. A report can have one or more of these sections.

Report Header and Report Footer:

The report header section is printed only once in the beginning of the report. Among other things, this section can be used to print a detail description for the report. The report footer section is printed at the end of the report. This section can be used to print the report summary.

Page Header and Page Footer:

The page header can be used to print the report name, current date, page number, column headers, etc. The text and data for this section is automatically printed after the top margin on every page. The page footer can be used to print the page totals or other pertinent text. The report footer is printed before the bottom margin on every page.

Section Headers and Section Footers:

ReportEase Plus allows up to 9 section headers and 9 section footers. The section headers and footers are numbered from 1 to 9. The section number 1 is the highest level section, where as the section number 9 is the lowest level section. A lower level section header is allowed only if all higher level section headers are already chosen. For example, you can create section number 2 only if section number 1 is already selected. Similarly, a section footer is allowed only if the corresponding section header is already chosen. However, you can select a lower level section footer without having to select all higher level section footers. For example, your report can contain section headers number 1 and 2, and section footer number 2.

A section header is always associated with a sort field. When a new section is created, a callback routine in your application is called to provide the user with a list of *sort* fields to choose from. Your application must have a capability to provide records sorted by one or more fields in the list. A section header is also associated with another field called *break* field. The value of this field is compared by the report executor to determine a sort break. Most reports will use the same field for both the sort field and the break field. However, the Report Designer allows you to specify a different field for comparison. The break field, unlike a sort field, can also be a calculated field. You can generate complex sort breaks using a calculated break field.

The footer sections are used to print summarized values for all records within the section (see Field Concepts). A summarized field is like an ordinary field but with the summarization attribute turned on. The following types of summarization is available:

Totals:	The field values for all records within a section are accumulated.
Average:	The average field value for all records within a section is accumulated and then divided by the number of records within that section.
Minimum:	This Summary Field computes the minimum value of all records within a section.
Maximum:	This Summary Field computes the maximum value of all records within a

section.

count: This Summary Field prints the number of records with a section.

For example, consider a customer order report with two sort breaks. Further, assume that the first sort break field is the state location of the customer, and the second sort field is the customer id. This report will print orders for each customer, with the customers grouped by the state location. You can insert an order summarization field in each section footer. The first section footer (higher level) will report the total orders by all customers within a state. The second section footer (lower level) will print the total orders for each customer. Refer to the *User Command* chapter for a description of inserting a summarization field into a footer section.

Detail Sections:

A report can have up to 9 detail sections. Typically, a report has only one detail section. Every detail section is printed for each record. The lower numbered detail sections are printed before the higher numbered detail sections. The detail sections print the most detail level data for each record. ReportEase Plus supports a parameter for the detail section that allows you to print multiple records across the page. This parameter can be used to print labels, such that two or more addresses can be printed in one row. This option is available for the reports having one detail section only.

There are two uses for multiple detail sections. The first is to report different fields for different record type. This can be accomplished by setting a report filter for each detail section such that only the desired detail section is printed for each record (refer to multdetl.fp report report template).

The second utility for the detail section is print two or more record types. For example, if your data set includes the 'customer', 'order' and 'location' data such that multiple order and location data is associated with each customer. In this situation, you can develop a report with two detail sections, one for the order record and one for the location record. Your program is, however, responsible for providing the data in the sorted fashion such that the order data is followed by the location data for each customer.

Section Selection Criteria

Once a section is created, by default it will print in its proper execution sequence. However, you can define an expression to print the section selectively. If a selection expression is provided, the section will print only if the expression evaluates to a TRUE value. A selection expression can use data fields, system fields, dialog fields, operators and functions. This feature is useful when designing complex reports. For example, you can suppress the detail section for selected records, yet accumulate the record fields for subtotals.

Section Parameters

These parameters can be selected for any section:

- Advance to the next page before printing the section.
- Advance to the next page after printing the section.
- Compress space between the beginning of the section and the topmost item in the section.
- Discard space after the bottom most item in the section. This attribute can be used to allow large memo (word-wrapped) fields. This attribute will automatically suppress the space after the smaller word-wrapped text data.
- Print sort header and detail records side-by-side. This flag is valid for only the innermost sort header section. The detail section field must be placed toward the right so that they don't overlap the sort header fields on the left side.

Calculation Expression

The calculation expressions can be used to perform the following:

- Define the calculated fields.
- Define the report selection criteria. The expression must evaluate to a TRUE or FALSE value.
- Define the section selection criteria. The expression must evaluate to a TRUE of FALSE value.

A calculation expression consists of operands and operators. The operands can be fields, functions, result of an if/then/else statement or another subexpression. Example of expressions:

1.amount * qty
2."abc" + "efg"
3.amount * (1 + profit_percentage)
4..if.state = "CA"
5.weekday("10/12/82")
6.profit_percentage*.TOTAL-OF.sales->amount

The first expression calculates the product of the amount and qty fields. The second expression will evaluate to "abcefg". The third expression is a product of the amount field and the result of another subexpression. The fourth expression evaluates to a TRUE value if the state field is equal to "CA", otherwise it evaluates to a FALSE value. The fifth expression returns the value of the 'weekday' function for 10/12/82. The sixth expression gives the profit amount for all records within a section.

Operator Precedence:

In an expression with multiple operators, the execution priority of an operator is determined by its precedence. The operator with the highest precedence gets executed first. The lower precedence operators use the result of the higher precedence operators as operands. You can override the default precedence by using parentheses. For example, 1 + 2 * 3 evaluates to 7. However, (1 + 2) * 3 will evaluate to 9. When an expression consists of two operators of the same precedence level, the operator on the left is executed before the operator on the right.

Result of an Expression:

The result of an expression provides a value of a specific type. For example, 100 + 200 results in 300, which is a number of a numeric type. Also, "cat" <> "dog" will result in a TRUE value which is an entity of the LOGICAL kind.

The following section describes ReportEase Plus operators in terms of its operands, precedence and result type. The precedence rank is indicated by a number. The higher precedence operators have higher value for the rank than a lower precedence operator.

In This Chapter

Operators

Logical OR

Operator Symbol:	.OR.
First Operand Type:	logical
Second Operand Type:	logical
Result Type:	logical
Precedence Rank:	100

Description: The logical OR operator returns a TRUE value if either the first operand or the second operand is TRUE. Otherwise, it returns a FALSE value. Examples:

```
10=(20-2).OR.10=(20-10)  -> TRUE

10=(20-2).OR.10=(20-8)   -> FALSE
```

Logical AND

Operator Symbol:	.AND.
First Operand Type:	logical
Second Operand Type:	logical
Result Type:	logical
Precedence Rank:	200

Description: The logical AND operator returns a TRUE value if both the first operand and the second operand are TRUE. Otherwise, it returns a FALSE value. Examples:

```
10=(30-20).AND.10=(20-10)       -> TRUE

10=(30-20).AND.10=(20-8)        -> FALSE
```

Equal

Operator Symbol:	=
First Operand Type:	Numeric,float,text,date,logical
Second Operand Type:	Same as the first operand type
Result Type:	logical
Precedence Rank:	300

Description: This operator returns a TRUE value if the first operand is equal to the second operand. Otherwise, it returns a FALSE value. Examples:

```
10=(30-20)      -> TRUE

10=(30-10)      -> FALSE
```

Not Equal

Operator Symbol:	<>
First Operand Type:	Numeric,float,text,date,logical
Second Operand Type:	Same as the first operand type
Result Type:	logical
Precedence Rank:	300

Description: This operator returns a TRUE value if the first operand is NOT equal to the second operand. Otherwise, it returns a FALSE value. Examples

```
10<>(30-20)     -> FALSE

10<>(30-10)     -> TRUE
```

Greater than

Operator Symbol:	>
First Operand Type:	Numeric,float,text,date,logical
Second Operand Type:	Same as the first operand type
Result Type:	logical
Precedence Rank:	400

Description: This operator returns a TRUE value if the first operand is greater than the second operand. Otherwise, it returns a FALSE value. Examples:

```
10>(30-22)      -> TRUE

10>(30-10)      -> FALSE

"ABC">"ACC"     -> FALSE
```

Less than

Operator Symbol:	<
First Operand Type:	Numeric,float,text,date,logical
Second Operand Type:	Same as the first operand type
Result Type:	logical
Precedence Rank:	400

Description: This operator returns a TRUE value if the first operand is less than the second operand. Otherwise, it returns a FALSE value. Examples:

```
10<(30-22)      -> FALSE

10<(30-10)      -> TRUE

"ABC"<"ACC"     -> TRUE
```

Greater than or Equal to

Operator Symbol:	>=
First Operand Type:	Numeric,float,text,date,logical
Second Operand Type:	Same as the first operand type
Result Type:	logical
Precedence Rank:	400

Description: This operator returns a TRUE value if the first operand is either greater or equal to the second operand. Otherwise, it returns a FALSE value. Examples:

```
10>=(30-22)      -> TRUE

10>=(30-10)      -> FALSE

"ABC">="AB"      -> TRUE
```

Less than or Equal to

```
Operator Symbol:        <=

First Operand Type:     Numeric,float,text,date,logical

Second Operand Type:    Same as the first operand type

Result Type:            logical

Precedence Rank:        400
```

Description: This operator returns a TRUE value if the first operand is either smaller or equal to the second operand. Otherwise, it returns a FALSE value.

```
10<=(30-22)      -> FALSE

10<=(30-10)      -> TRUE

"ABC"<="ABCD"    -> TRUE
```

Part of

Operator Symbol:	$
First Operand Type:	text
Second Operand Type:	text
Result Type:	logical
Precedence Rank:	500

Description: This operator returns a TRUE value if the first operand is contained within the second operand. Otherwise, it returns a FALSE value. Examples:

```
"KEEP"$"HOUSE KEEPER"   ->TRUE

"KEEPING"$"HOUSE KEEPER"        ->FALSE.
```

Addition

```
Operator Symbol:        +

First Operand Type:     numeric,float,text, date

Second Operand Type:    same as the first operand type when

                        the first operand type is numeric,

                        float or text.

                        When one of the operands is a 'date',

                        the other operand must be numeric.  Two

                        dates can not be added.

Result Type:            same as the first operand type when

                        both operands are numeric, float, or

                        text.  When one of the operand is a

                        'date', the result is also a 'date.

Precedence Rank:        600
```

Description: This operator adds the second operand to the first operand. If one of the operands is numeric and the other is float, the result will be of the float type. If the operand type is text, the second string is appended to the first string. Examples:

```
10 + 20 -> 30

10 + 20.5       -> 30.5

"Good " + "Day" -> "Good Day"

"5/9/99" + 1    -> "5/10/99"
```

Subtraction

Operator Symbol:	-
First Operand Type:	numeric,float,text,date
Second Operand Type:	same as the first operand type when the first operand is numeric, float or text. When the first operand is a 'date', the second operand can be a 'date' or numeric. Similarly, when the second operand is a 'date', the first operand can be a 'date' or numeric.
Result Type:	same as the first operand type when both operands are numeric, float, or text.

When both operands are 'date', the result is a number of days calculated by subtracting the second date from the first date. When the first operand is a date and the second operand is numeric, then the result is a date calculated by subtracting the number of days (second argument) from the date (first argument).

Precedence Rank:	600

Description: This operator subtracts the second operand from the first operand. If one of the operands is numeric and the other float, the result will be of float type. If the operand type is text, the second string is appended to the first string. However, any spaces after the first string are truncated and transferred at the end of the output string. Examples:

```
10 - 20 -> -10

10 - 20.5       -> -10.5

"Good " - "Day" -> "GoodDay "

"5/9/99" - "5/8/99"     -> 1

"5/9/99"  1    -> "5/8/99"
```

Multiplication

```
Operator Symbol:        *

First Operand Type:     numeric,float

Second Operand Type:    numeric, float

Result Type:            numeric,float

Precedence Rank:        700
```

Description: This operator multiplies both operands. If one of the operands is numeric and the other float, the result will be of float type. Examples:

```
10 * 20 -> 200

10 * 20.5       -> 205.
```

Division

```
Operator Symbol:         /

First Operand Type:      numeric,float

Second Operand Type:     numeric,float

Result Type:             numeric,float

Precedence Rank:         700
```

Description: This operator divides the first operand by the second operand. If one of the operand is numeric and the other float, the result type will be float. Examples:

```
10 / 2  -> 5

10 * 20 -> 0

10 * 20.0       -> .5
```

NOT

Operator Symbol:	.NOT.
First Operand Type:	logical
Second Operand Type:	N/A
Result Type:	logical
Precedence Rank:	800

Description: This operator negates the logical value of the first operator. Being a unary operator, it accepts only one operand. Examples:

```
.NOT.(10=(20-10))         -> FALSE

.NOT.(10=(20-8))          -> TRUE

.NOT.("KEEP"$"KEEPING")  -> FALSE
```

TOTAL OF

Operator Symbol:	.TOTAL-OF.
First Operand Type:	numeric,float type field
Second Operand Type:	N/A
Result Type:	Same as the first operand type
Precedence Rank:	900

Description: The operand for this operator must be a field. Being a unary operator, it accepts only one operand. This operator is allowed only in the calculation fields that are placed in a footer section. The operator will calculate the subtotal for the field indicated by the first operand. Example:

```
.TOTAL-OF.sales->amount calculates the total sales amount for

                 the footer section field.
```

AVERAGE OF

Operator Symbol:	.AVE-OF.
First Operand Type:	numeric,float type field
Second Operand Type:	N/A
Result Type:	Same as the first operand type
Precedence Rank:	900

Description: The operand for this operator must be a field. Being a unary operator, it accepts only one operand. This operator is allowed only in the calculation fields that are placed in a footer section. The operator will calculate the average value for the field indicated by the first operand. Example:

.AVE-OF.sales->amount	calculates the average sales amount for the footer section field.

MAXIMUM OF

Operator Symbol:	.MAX-OF.
First Operand Type:	numeric,float type field
Second Operand Type:	N/A
Result Type:	Same as the first operand type
Precedence Rank:	900

Description: The operand for this operator must be a field. Being a unary operator, it accepts only one operand. This operator is allowed only in the calculation fields that are placed in a footer section. The operator provides the largest value of the field indicated by the first operand. Example:

```
.MAX-OF.sales->amount  returns the largest sales amount for

                       the footer section field.
```

MINIMUM OF

Operator Symbol:	.MIN-OF.
First Operand Type:	numeric,float type field
Second Operand Type:	N/A
Result Type:	Same as the first operand type
Precedence Rank:	900

Description: The operand for this operator must be a field. Being a unary operator, it accepts only one operand. This operator is allowed only in the calculation fields that are placed in a footer section. The operator provides the smallest value of the field indicated by the first operand. Example:

```
.MIN-OF.sales->amount   returns the smallest sales amount for

                        the footer section field.
```

COUNT OF

Operator Symbol:	.COUNT-OF.
First Operand Type:	numeric,float type field
Second Operand Type:	N/A
Result Type:	Same as the first operand type
Precedence Rank:	900

Description: The operand for this operator must be a field. Being a unary operator, it accepts only one operand. This operator is allowed only in the calculation fields that are placed in a footer section. The operator provides the record count for a section. Example:

```
.COUNT-OF.sales->amount returns the number of records processed

                        within the current section.
```

Condition Statement

The ReportEase Plus calculation expressions allow an if/then/else statement. This statement evaluates the *if* condition for a TRUE or FALSE value. If the value is TRUE, then the entire expression evaluates to the subexpression following the *then* statement. Otherwise the entire expression evaluates to the subexpression following the *else* statement.

Examples:

```
.IF.sales->amount>100.THEN."GOOD SALE".ELSE."NOT SO GOOD SALE"
```

This example compares the sales amount and returns a text string. The resulting text string is equal to `"GOOD SALE"` when the sales->amount is greater than $100. Otherwise it is equal to `"NOT SO GOOD SALE"`.

It is important that the subexpression following the *then* and the *else* statement must return the same type result.

Examples of invalid statements:

```
.IF.sales->amount>100.THEN."GOOD SALE".ELSE.50

.IF.customer->state="CA".THEN.(100).ELSE.(5.0)
```

The second statement is not valid because the *then* statement evaluates to a numeric value, where as the *else* statement evaluates to a float value. Correct the second statement as following:

```
.IF.customer->state="CA".THEN.(100.0).ELSE.(5.0)

or

.IF.customer->state="CA".THEN.(100).ELSE.(5
```

)

Functions

The ReportEase Plus calculation expressions can use functions. A function accepts a predefined number of arguments and returns a value of a predefined type.

Add text on next line

```
Function Name:       AddLine

First Argument Type: text

Second Operand Type: N/A

Result Type:         text
```

Description: This function is used to add text to the next line. A new blank line is not created if the text is blank. This function can be used to print address using a calculation field. Example:

```
Name+AddLine(company)+AddLine(address1)+AddLine(address2)
```

Length of a Text String

```
Function Name:       LEN

First Argument Type: text

Second Operand Type: N/A

Result Type:         numeric
```

Description: This function returns the length of a text string. Examples:

```
LEN("ABCD") -> 4

LEN("GOOD DAY") -> 8
```

Position of a string within another string

```
Function Name:       InStr

First Argument Type: text

Second Operand Type: text

Result Type:         numeric
```

Description: This function returns the position of the second string within the first string. It returns 0 if the second string is not found within the first string. The string search is case-sensitive. Examples:

```
InStr("catdog","cat") -> 1

InStr("catdog","dog") -> 4

InStr("catdog","mouse") -> 0
```

Convert text to date field

```
Function Name:       ToDate

First Argument Type: text

Second Operand Type: N/A

Result Type:         Date
```

Description: This function converts a text type argument to date type. Examples:

```
ToDate("12/31/2002") -> 12/31/2002
```

Convert to Upper Case

```
Function Name:       UPPER

First Argument Type: text

Second Operand Type: N/A

Result Type:         text
```

Description: This function converts the given string to the upper case. Examples:

```
UPPER("abcd") -> "ABCD"

UPPER("Good Day") -> "GOOD DAY"
```

Convert to Lower Case

```
Function Name:       LOWER

First Argument Type: text

Second Operand Type: N/A

Result Type:         text
```

Description: This function converts the given string to the lower case. Examples:

```
LOWER("ABCD") -> "abcd"

LOWER("Good Day") -> "good day"
```

Trim Spaces

```
Function Name:       TRIM

First Argument Type: text

Second Operand Type: N/A

Result Type:         text
```

Description: This function returns a string by removing spaces from the beginning and ending of given string. Examples:

```
TRIM(" ABCD ") -> "ABCD"

TRIM("Good Day  ") -> "Good Day"
```

Extract a Word

```
Function Name:       WORD

First Argument Type: text

Second Operand Type: numeric

Result Type:         text
```

Description: This function extracts a word from the input string. The second argument specifies the word position to be extracted. Examples:

```
WORD("It is a Good Day",1) -> "It"

WORD("It is a Good Day",2) -> "is"
```

Extract a Character

```
Function Name:       CHAR

First Argument Type: text

Second Operand Type: numeric

Result Type:         text
```

Description: This function extracts a character from the input string. The second argument specifies the character position to be extracted. Examples:

```
CHAR("It is a Good Day",1) -> "I"

CHAR("It is a Good Day",2) -> "t"
```

Extract First Specified Number of Characters

```
Function Name:       FIRST

First Argument Type: text

Second Operand Type: numeric

Result Type:         text
```

Description: This function extracts the specified number of characters (argument #2) from the beginning of the specified (argument #1) text string. Examples:

```
FIRST("It is a Good Day",5) -> "It is"

FIRST("It is a Good Day",2) -> "It"
```

Extract Last Specified Number of Characters

```
Function Name:       LAST

First Argument Type: text

Second Operand Type: numeric

Result Type:         text
```

Description: This function extracts the specified number of characters (argument #2) from the end of the specified (argument #1) text string. Examples:

```
LAST("It is a Good Day",8) -> "Good Day"

LAST("It is a Good Day",3) -> "Day"
```

Convert to Text Type

```
Function Name:       TEXT

First Argument Type: numeric, float, date, logical

Second Operand Type: N/A

Result Type:         text
```

Description: This function converts any other type argument to the text type data using the default format specifications. Examples:

```
TEXT("3/4/92") -> "3/4/92" (text)

TEXT(123) -> "123"
```

Smaller Number

```
Function Name:       MIN

First Argument Type: numeric,float

Second Operand Type: numeric,float

Result Type:         numeric,float
```

Description: This function returns the smaller of the first and second arguments. If one of the arguments is numeric and the other is float, then the return type will be float.
Examples:

```
MIN(10,20) -> 10

MIN(10,20.0) -> 10.0
```

Larger Number

```
Function Name:       MAX

First Argument Type: numeric,float

Second Operand Type: numeric,float

Result Type:         numeric,float
```

Description: This function returns the larger of the first and second arguments. If one of the arguments is numeric and the other is float, then the return type will be float. Examples:

```
MAX(10,20) -> 20

MAX(10,20.0) -> 20.0
```

Round

```
Function Name:       ROUND

First Argument Type: float

Second Operand Type: numeric

Result Type:         float
```

Description: This function rounds the first argument to the number of decimal places specified by the second Argument. Examples:

```
ROUND(10.153,2) -> 10.15

ROUND(10.153,1) -> 10.2
```

Integer

```
Function Name:       INT

First Argument Type: float, text, date, logical

Second Operand Type: N/A

Result Type:         numeric
```

Description: This function converts any other type argument to the numeric type. For a 'float' type of argument, this operation discards any decimal digits from the first argument. The date type argument is converted to YYYYMMDD formatted numeric value. The logical type is converted either 1 or 0. Examples:

```
INT(10.153) -> 10

INT("123") -> 123

INT("3/4/92") -> 19920304

INT(1<>2) -> 1
```

Number (decimal value)

```
Function Name:       ToNumber

First Argument Type: text

Second Operand Type: N/A

Result Type:         float
```

Description: This function converts a text type argument to float type. Examples:

```
ToNumber("10.153") -> 10.153

ToNumber("123") -> 123

ToNumber("-123.456") -> -123.456
```

Absolute

```
Function Name:       ABS

First Argument Type: numeric,float

Second Operand Type: N/A

Result Type:         Same as the Argument
```

Description: This function returns the absolute value of the given argument. Examples:

```
ABS(-10.153) -> 10.153

ABS(10.153) -> 10.153

ABS(-12) -> 12
```

Day of the Week

```
Function Name:       WEEKDAY

First Argument Type: date

Second Operand Type: N/A

Result Type:         text
```

Description: This function returns the weekday for given date. Examples:

```
WEEKDAY("4/13/92") -> "Monday"

WEEKDAY("4/14/92") -> "Tuesday"
```

Extract Day

```
Function Name:       DAY

First Argument Type: date

Second Operand Type: N/A

Result Type:         numeric
```

Description: This function extracts the day (1 to 31) from the given date. Examples:

```
DAY("4/13/92") -> 13

DAY("4/14/92") -> 14
```

Extract Month

```
Function Name:       MONTH

First Argument Type: date

Second Operand Type: N/A

Result Type:         numeric
```

Description: This function extracts the month (1 to 12) from the given date. Examples:

```
DAY("4/13/92") -> 4

DAY("5/14/92") -> 5
```

Extract Year

```
Function Name:       YEAR

First Argument Type: date

Second Operand Type: N/A

Result Type:         numeric
```

Description: This function extracts the year from the given date. The year is returned using 4 digits. Examples:

```
DAY("4/13/92") -> 1992

DAY("5/14/93") -> 1993
```

Extract the number of immediate sort breaks or the number of detail records

```
Function Name:       BREAKS

First Argument Type: numeric

Second Operand Type: N/A

Result Type:         numeric
```

Description: When the argument value is a value from 1 to 9, this function returns the number of sort breaks encountered thus far for the given sort level. Consider a report which lists invoice items for each invoice for each customers. If you wish to print the number of invoices for a customer, create the following calculation expression in the footer section of the customer:

```
BREAKS(2)
```

If you wish to print the number of customers in the entire report, create following expression in the report footer:

```
BREAKS(1)
```

You can also set the first argument to 0 to retrieve the number of detail records in the last sort section. For example, if you wish to report the number of items for an invoice, create the following calculation expression in the footer section of the invoice:

```
BREAKS(0)
```

In the above examples the summarization type should be reset to 'value', since by default the calculation expressions entered in a footer section is assigned the summarization type of 'Total'.

Extract the number of any sort breaks or the number of detail records

```
Function Name:       TotalBreaks

First Argument Type: numeric

Second Operand Type: N/A

Result Type:         numeric
```

Description: When the argument value is a value from 1 to 9, this function returns the number of sort breaks encountered thus far for the given sort level. The difference between this function and the 'Breaks' functions is that this function allows you to access the sort-break count (or detail record count) of any level from any footer.

Consider a report which lists invoice items for each invoice for each customers for each department:

```
Department  (sort level 1)

Customer (sort level 2)

Invoice (sort level 3)

Invoice items (detail records)
```

If you wish to print the number of invoices for a department, create the following calculation expression in the footer section for the department:

```
TotalBreaks(3)
```

(The argument value 3 indicates the sort level of the 'invoice' sort)

If you wish to print the number of departments in the entire report, create following expression in the report footer:

```
TotalBreaks(1)
```

(The argument value 1 indicates the sort level of the 'department' sort)

You can also set the first argument to 0 to retrieve the number of detail records encountered thus far at the current sort footer. For example, if you wish to report the number of invoice-items for a customer, create the following calculation expression in the footer section of for customer:

```
TotalBreaks(0)
```

In the above examples the summarization type should be reset to 'value', since by default the calculation expressions entered in a footer section is assigned the summarization type of 'Total'.

ActiveX Interace

This chapter describes ActiveX properties, methods and events. Most of the methods and events described in this topic are only useful when interfacing the product using Low-level Interface.

Please refer to the Getting Started topic instead when using the product in the *Simple Mode.*

In This Chapter

Methods

RepMenuEnable

Retrieve the 'enable' status a menu command.

BOOL RepMenuEnable(CmdId)

int CmdId;	// Command id for the menu command. Please refer to the RepCommand function for a list of menu commands.

Return Value: This function returns TRUE if the given menu item should be enabled. The FALSE value indicates that the menu item should be grayed.

RepMenuSelect

Retrieve the 'check' status a menu command.

BOOL RepMenuSelect(CmdId)

int CmdId;	// Command id for the menu command. Please refer to the RepCommand function for a list of menu commands.

Return Value: This function returns TRUE if the given menu item should be 'checked'.

RvbDrawBitmap

Draw a bitmap to the output device context

int RvbDrawBitmap(hWnd as Integer, hImageWnd as Integer, image as Integer, x as Integer, y as Integer, width as Integer, height as Integer)

(for Win32 applications, replace the first parameter by pCtl, which is a pointer property exposed by the OCX control)

Description: This function is used within a 'DrawPicture' event handler to draw a bitmap (or part of) to the current report output device context. The 'hWnd' parameter is the window handle of the reporter control. The 'hImageWnd' is the window handle of the image control which contains the bitmap to draw. The 'image' parameter specifies a handle to a bitmap. The last four parameters specify the part of the bitmap to display. These parameters are specified in the percentage values. For example, to display the entire bitmap, the parameter values should be as following: x=0, y=0, width=100, height=100. To display the bottom half of the bitmap, the parameter values should be as following: x=0, y = 50, width = 100, height = 50.

Return Value: This function return TRUE if successful.

See Also

RvbGetPictureInfo
DrawPicture

RvbEnableformWindow

Enable or disable the Report Designer window

int RvbEnableformWindow(hWnd as Integer, enable as Integer)

(for Win32 applications, replace the first parameter by pCtl, which is a pointer property exposed by the OCX control)

Description: This function is used by the data file/field selection routines in their load/unload event handlers. The 'load' event handler should call this function with a 0 value for the 'enable' argument, thus disabling the Report Designer window for the duration of the user file/field selection. The 'unload' event handler should re-enable the Report Designer with a call to this function with a value of 1 for the 'enable' argument.

Return Value: This function always returns 1.

RvbExit

Close the Report Executor

int RvbExit(hWnd as Integer)

(for Win32 applications, replace the first parameter by pCtl, which is a pointer property exposed by the OCX control)

Description: This function frees up the resources used by the ReportExecutor. The hWnd parameter is the window handle of the control.

Return Value: This function returns 0 upon the successful execution, otherwise it returns an error code (see ERR_ constants in the REP.BAS file)

See Also
RvbRec
RvbInit

Rvbform

Launch the Report Designer

int Rvbform(formParam as Typeform)

(The Win32 applications need additional (pCtl) parameter before the 'Typeform' parameter, which is a pointer property exposed by the OCX control)

Description: This function is used to launch the Report Designer. Your application provides the report template name and other relevant parameters using the 'Typeform'. Please refer to the REP.BAS file for the description of the individual member variables for this structure.

Once the Report Designer is launched, it communicates with your program by firing the events.

Return Value: This function returns 0 upon the successful execution, otherwise it returns an error code (see ERR_ constants in the REP.BAS file)

See Also

SelectField
VerifyField

RvbGetDataField

Get the specified ReportExecutor data field.

int RvbGetDataField(hWnd as Integer, FieldNo as Integer, field as TypeField)

(for Win32 applications, replace the first parameter by pCtl, which is a pointer property exposed by the OCX control)

Description: This function is used to retrieve the field structure (TypeField) for the specified field. The 'hWnd' parameter is the window handle of the control. The 'FieldNo' parameter specifies the field to retrieve. This parameter must be between 0 and TotalFields - 1 (see RvbInit).

Return Value: This function returns a True value upon the successful execution.

See Also
RvbSetTextField
RvbSetNumField
RvbSetDoubleField

RvbGetformField

Get the current Report Designer field

int RvbGetformField(hWnd as Integer, field as TypeField, SortLevel as Integer)

(for Win32 applications, replace the first parameter by pCtl, which is a pointer property exposed by the OCX control)

Description: This function is used in pair with RvbSetformField function within the SelectField and VerifyField event handlers. The 'hWnd' parameter is the window handle of the control. The 'field' parameter returns the current field being selected or verified. The 'SortLevel' parameter indicates whether the field is being used for a sort break. The 'SortLevel' parameter is 0 if the field is not a sort break field. Otherwise, the value of the field (1, 2, 3...) indicates the sort section level to which this field belongs.

Return Value: This function returns a True value upon the successful execution.

See Also

RvbSetformField

RvbGetPictureInfo

Get the picture parameters

int RvbGetPictureInfo(hWnd As Integer, PictInfo as TypePict)

(for Win32 applications, replace the first parameter by pCtl, which is a pointer property exposed by the OCX control)

Description: This function is used to get the parameters to draw a 'picture' type field. This function is typically used within a 'DrawPicture' event handler. The 'hWnd' parameter is the window handle of the control. The information about the picture is returned by the 'TypePict' variable. The following parameters are available in the 'TypePict' structure:

```
Type TypePict

  hDC as Integer              ' device context of the reporting

                                device

  PictId as Integer           ' the value of the current picture

                                field

  FileId as Integer           ' the file id that contains the

                                current picture field

  FieldId as Integer          ' the field id that correspond to

                                the current picture field

  x as Integer                ' X location of the picture
rectangle

  y as Integer                ' Y location of the picture
rectangle

  width as Integer            ' width of the picture rectangle

  height as Integer           ' height of the picture rectangle

End Type
```

Return Value: This function return TRUE if successful.

See Also
RvbDrawBitmap
DrawPicture

RvbGetSortField

Get the specified ReportExecutor sort field.

int RvbGetSortField(hWnd as Integer, FieldNo as Integer, field as TypeField)

(for Win32 applications, replace the first parameter by pCtl, which is a pointer property exposed by the OCX control)

Description: This function is used to retrieve the field structure (TypeField) for the specified sort field. The 'hWnd' parameter is the window handle of the control. The 'FieldNo' parameter specifies the sort field to retrieve. This parameter must be between 0 and TotalSortFields - 1 (see RvbInit).

Return Value: This function returns a True value upon the successful execution.

See Also
RvbInit

RvbInit

Initialize the Report Executor

int RvbInit(hWnd as Integer, RepParm As TypeRep)

(for Win32 applications, replace the first parameter by pCtl, which is a pointer property exposed by the OCX control)

Description: This function is used to initialize the Report Executor. The hWnd parameter is the window handle of the control. The application provides the report template name and other relevant information using the TypeRep parameter. Please refer to the REP.BAS file for the description of the individual members of this structure.

This function updates in the values for two of the member variables: TotalFields, and TotalSortFields. These variables indicate a total number of data fields and total number of sort fields used by the report.

Return Value: This function returns 0 upon the successful execution, otherwise it returns an error code (see ERR_ constants in the REP.BAS file)

See Also

RvbRec
RvbExit
RvbGetDataField
RvbGetSortField

RvbRec

Print a data record

int RvbRec(hWnd as Integer)

(for Win32 applications, replace the first parameter by pCtl, which is a pointer property exposed by the OCX control)

Description: This function instructs the Report Executor to print the current record. The hWnd parameter is the window handle of the control.

Return Value: This function returns 0 upon the successful execution, otherwise it returns an error code (see ERR_ constants in the REP.BAS file)

See Also

RvbInit
RvbExit
RvbGetDataField
RvbGetSortField

RvbSetDoubleField

Set the value of the specified double type field.

int RvbSetDoubleField(hWnd as Integer, FieldNo as Integer, DataValue as double)

(for Win32 applications, replace the first parameter by pCtl, which is a pointer property exposed by the OCX control)

Description: This function is used to supply the data for a double type field. The 'hWnd' parameter is the window handle of the control. The 'FieldNo' parameter specifies the field number to supply data for. This parameter must be between 0 and TotalFields - 1 (see RvbInit). The 'DataValue' field should contain the value for the field.

Return Value: This function returns a True value upon the successful execution.

See Also
RvbGetDataField
RvbSetNumField
RvbSetTextField

RvbSetformField

Set the current Report Designer field

int RvbSetformField(hWnd as Integer, field as TypeField, valid as Integer)

(for Win32 applications, replace the first parameter by pCtl, which is a pointer property exposed by the OCX control)

Description: This function is used in pair with RvbGetformField function within the SelectField and VerifyField event handlers. The 'hWnd' parameter is the window handle of the control. The 'field' parameter contains the updated data for the current field being selected or verified. The 'valid' should be set to TRUE to indicate a valid field.

Return Value: This function returns a True value upon the successful execution.

See Also
RvbGetformField

RvbSetNumField

Set the value of the specified numeric, date or logical field

int RvbSetNumField(hWnd as Integer, FieldNo as Integer, DataValue as long)

(for Win32 applications, replace the first parameter by pCtl, which is a pointer property exposed by the OCX control)

Description: This function is used to supply the data for a numeric field. The 'hWnd' parameter is the window handle of the control. The 'FieldNo' parameter specifies the field number to supply data for. This parameter must be between 0 and TotalFields - 1 (see RvbInit).

The 'DataValue' field should contain the value for the field. For a 'date' field, the value should be in the yyyymmdd format (example: 19941231 for 12/3194). For a 'Logical' field, this value should be either 1 (True) or 0 (False).

Return Value: This function returns a True value upon the successful execution.

See Also

RvbGetDataField
RvbSetTextField
RvbSetDoubleField

RvbSetTextField

Set the value of the specified text field

int RvbSetTextField(hWnd as Integer, FieldNo as Integer, TextData as String, TextLen as Integer)

(for Win32 applications, replace the first parameter by pCtl, which is a pointer property exposed by the OCX control)

Description: This function is used to supply the data for a text field. The 'hWnd' parameter is the window handle of the control. The 'FieldNo' parameter specifies the field number to supply data for. This parameter must be between 0 and TotalFields - 1 (see RvbInit).

Return Value: This function returns a True value upon the successful execution.

See Also
RvbGetDataField
RvbSetNumField
RvbSetDoubleField

ActiveX Events

DrawPicture

Draw a picture field

Description: This event is fired by the Report Executor when it needs your application to draw a 'picture' type field. Your application will typically call the RvbGetPictureInfo function to retrieve the picture parameters and the RvbDrawBitmap function to draw your bitmap to the report output device context.

Postprocess

This event is sent after a user initiated action is *completed*. This message uses the following parameters:

wParam (or ActionType): This parameter can be one of the following:

ACTION_COMMAND:	This action indicates any of the menu or the accelerator key generated commands. The actual command id is given by the lParam (or ActionId) argument. For a list of command ids, please refer to the RepCommand API function description.
ACTION_VSCROLL:	This action message is sent when the vertical scroll bar is clicked. The 'lParam' argument for this message identifies the actual scrollbar operation and is given by the SB_xxxxx SDK constants.
ACTION_HSCROLL:	This action message is sent when the horizontal scroll bar is clicked. The 'lParam' argument for this message identifies the actual scrollbar operation and is given by the SB_xxxxx SDK constants.

lParam (or ActionId): This value is specific to the action type as described above.

Preprocess

This event is fired *before* a message is processed by the Report Designer. This message uses the following parameters:

wParam (or ActionType): This parameter can be one of the following:

ACTION_COMMAND:	See *Postprocess* event for description.
ACTION_VSCROLL:	See *Postprocess* event for description.
ACTION_HSCROLL:	See *Postprocess* event for description.
ACTION_LBUTTONDOWN:	Left mouse button down. The ActionId contains the x and y mouse position in the pixel units. The x position is given by the low 16 bits and the y position is given by the high 16 bits.
ACTION_RBUTTONDOWN:	Right mouse button down. The ActionId parameter holds the mouse position as described for the ACTION_LBUTTONDOWN message.
ACTION_LBUTTONUP:	Left mouse button up. The ActionId parameter holds the mouse position as described for the ACTION_LBUTTONDOWN message.
ACTION_RBUTTONUP:	Right mouse button up. The ActionId parameter holds the mouse position as described for the ACTION_LBUTTONDOWN message.
ACTION_LBUTTONDBLCLICK:	Left mouse double click. The ActionId parameter holds the mouse position as described for the ACTION_LBUTTONDOWN message.
ACTION_RBUTTONDBLCLICK:	Right mouse double click. The ActionId parameter holds the mouse position as described for the ACTION_LBUTTONDOWN message.
ACTION_MOUSEMOVE:	Mouse move. The ActionId contains the x and y mouse position in the pixel units. The x position is given by the low 16 bits and the y position is given by the high 16 bits.
ACTION_SIZE:	Window being resized.
ACTION_SETFOCUS:	The control window receiving focus.
ACTION_KILLFOCUS:	The control window loosing focus.

lParam (or ActionId): This value is specific to the action type as described above.

After your application processes this messages, you can call the RepIgnoreCommand function to instruct the Report Designer to ignore this message.

SaveAs

This event is fired when the user uses the 'Save' option to save a new report or when the user uses the 'SaveAs' option to save an existing report to another file name. The event contains the new name of the report template file.

SelectField

User field selection handler

Description: This event is fired by the Report Designer when your user wishes to paste a data field to the report template. This event allows your application to prompt the user for the field (and file) selection and return the information about the selected field.

Typically, this event handler should be structured as following:

▪Retrieve the current field structure using the RvbGetformField function. This function also returns the SortLevel for the current field. The sort level is zero if the current field is not used for a sort break, otherwise it contains the level of the sort section (1, 2, 3...).
▪Prompt the user for the field and file selection using a list box or some other GUI function. Some applications may need to restrict the number of fields available for selection when the SortLevel is non-zero.
▪Update in the field structure with the basic field information. The following field must be provided:

```
name:  Field name. Use -> to separate the file name, ex:

       SALE->DATE

type:  Field Type. See TYPE_ constants in the REP.BAS file

width: Maximum number of characters in the field.

DecPlace: Number of decimal places for a numeric or double

       type field
```

Optional fields:

```
FileId:   A sequential id for the selected file

FieldId:  A sequential id for the selected field.

ParaChar: Paragraph break character for a word-wrapped text

          field.
```

▪Return the updated field structure to the formEditor using the RvbSetformField function.

1.

Unload

This event is fired when the Report Designer or the report executor window is being closed. In response to this event your application should reset the 'open' flag in the 'formParm' structure.

UpdateToolbar

This event is fired when an external toolbar needs to be repainted (not available in the VBX interface). Your application can use the RepGetItemInfo function (Report Designer) or the RepGetPageInfo function (report executor) to retrieve the information to be displayed on your application's external toolbar.

VerifyField

User field verification handler

Description: This event is fired by the Report Designer when your user types in a data field name within an expression. This event allows your application to verify the field (and file) name and return the information about the current field.

Typically, this event handler should be structured as following:

- ▪Retrieve the current field structure using the RvbGetformField function. This function also returns the SortLevel for the current field. The sort level is zero if the current field is not used for a sort break, otherwise it contains the level of the sort section (1, 2, 3...).
- ▪Verify the specified field for validity. Skip the next two steps if the field is not valid.
- ▪Update in the field structure with the basic field information. The following field must be provided:

```
type:     Field Type. See TYPE_ constants in the REP.BAS file

width:    Maximum number of characters in the field.

DecPlace: Number of decimal places for a numeric or double

          type field
```

Optional fields:

```
FileId:  A sequential id for the selected file

FieldId: A sequential id for the selected field.

ParaChar:Paragraph break character for a word-wrapped text

         field.
```

- ▪Return the updated field structure to the formEditor using the RvbSetformField function.

ActiveX Properties

Here is a list of properties supported by the ActiveX control.

AutoRun	Set this property to TRUE to run the report automatically using the definition and data files specified here (default value=TRUE). You would set this property to FALSE to use the Low-level Interface functions to pass data to the report executor.
Command	This property is used to implement a menu command. Typically you would create a menu on the parent form containing the ReportEase control. When the user clicks on a menu option, you would use the Command property to invoke the selected option. For example, when the user clicks on your 'File Open' menu option, you would execute the following statement: Roc1.Command = ID_OPEN Please refer to the RepCommand function description for a list of menu command ids. Please refer to the design.vbp Visual Basic sample program for an example.
DataMapPictFile	This picture file (jpeg or bmp) is used to schematically display the data-file relationship to the user. This picture is displayed when the user wishes to insert a data field in the report template.
DataSetName	An *optional* data-set name. It can be used within your application to indicate a particular configuration of data files available for report template editing and execution. For example, your application might support two different type of reporting, such as sales, and inventory. You would set the DataSetName property to, say 'sales' when providing the data files containing sales data. Similarly, you would set the data set name to, say 'inventory' when providing the data files containing inventory data. The control will then display an error message if the user attempts to open an incorrect template for the current DataSetName. This data set name provided here is stored within any newly created report template. It is also used to verify if a requested report template is valid to run with the provided data files.
DesignMode	Set this property to TRUE to create a report designer control, set to FALSE to create a report executor control. *This property should be set at the application design-time only.* Setting it at the run-time has no effect.
DetailDataFile	Please refer to the Creating Report Executor Control for a description about this property.
DetailDefFile	Please refer to the Creating Report Executor Control for a

	description about this property.
hFrWnd	Window handle of the current ReportEase control instance.
HorzScrollBar	Display the horizontal scroll bar.
MasterDataFile	Please refer to the Creating Report Executor Control for a description about this property.
MasterDefFile	Please refer to the Creating Report Executor Control for a description about this property
OtherDataFiles	Please refer to the Creating Report Executor Control for a description about this property.
OtherDefFiles	Please refer to the Creating Report Executor Control for a description about this property.
PictureNameFile	This file is used to associate picture file names with picture ids. The data records passed to report executor may include a picture field. The picture field specifies a numeric picture id. This file contains the picture file names associated with the picture id. The picture file must be in BMP or JPEG format.
RepKey	Property to set your product license key. Your product license key is available in the key.txt file included in the distribution zip file. You do not need to set the property when using the product in the evaluation mode.
Standalone	Set this property to FALSE to embed the report control into your form (default value = FALSE). A TRUE value would be useful when printing or exporting the report without any user interface. *This property should be set at the application design-time only.* Setting it at the run-time has no effect.
SuppressPrintMessage	Set to TRUE to suppress the print progress message.
TemplateFile	Use the property to set any initial report template file to edit or to execute. This is an optional property since the user can select a report using the toolbar. *However, if this property is set, it should be set after setting all data and definition related properties.*
TextPitch	The text pitch to use when exporting the report to a comma or tab delimited text format.
Toolbar	Display internal toolbar and status bar ribbons.
Typeface	The default type face for report editing.

UseCurrentPrinter	Set to TRUE to print the report to the current system printer. Set to FASLE to print the report to the printer specified in the report template.
VerScrollBar	Display vertical scrollbar.

Recompiling the DLL

The ReportEase Plus DLL can be used with a Visual C++ application without any change. Follow these general steps to call the REP routines from a Visual C++ application:

1.Include the REP.H file into your application module that calls the REP functions. Use the REP API functions as necessary..

Recompiling REP32 DLL files

If you need to modify the DLL source code and recompile within the Visual C++ environment, follow these steps to create a Visual C++ project:

```
Files:           REP*.C (no rep*.cpp files), REP.DEF and REP.RC

Executable Type: Win32 DLL

Compiler Option: 1 Byte Alignment
```

If you are creating the project in the Release mode, then set the optimization to 'Default Optimization'.

Remaining parameters should be left at their default values.