USER MANUAL

ReportEase Plus

Report Writer Engine

(for Windows)

SUB SYSTEMS, INC.

1995

USER MANUAL

ReportEase Plus

Report Writer Engine

(for WIN32)

SUB SYSTEMS, INC.

1995

ReportEase Plus

Version 2.0
1995

11 Tiger Row
Georgetown, MA 01833

Software License Agreement

This license agreement allows the purchaser the right to modify the source code to incorporate it into an application. Such a target application may be distributed royalty free with these conditions:

a. The target application must not be a STAND ALONE report writer product.
b. The target application should be 'larger' than the ReportEase Plus routine itself.
c. The source code of this software must not be distributed in any form.
d. ReportEase Plus must not be ported to other operating system platforms.

Multideveloper Licenses: Each copy of the product is licensed to one developer. Multiple licenses are discounted as following:

2 to 4 Licenses:	20 percent discount
5 or more Licenses:	40 percent discount

Sub Systems, Inc. reserves the right to prosecute anybody found to be making illegal use of this software.

Sub Systems, Inc. offers a 30 day money back guarantee with the product. The source code diskette envelope must remain sealed. Must call for an RMA number before returning the product.

Disclaimer

Sub Systems, Inc. has made every effort to make this product reliable and error free. However, the manufacturer makes no warranties against any damage, direct or indirect, resulting from the use of the software or the manual and can not be held responsible for the same.

IBM PC,XT AND AT are the trademarks of International Business Machine.

MSDOS, Windows, Windows 95, Windows NT, WIN32s, WIN32, Visual Basic and Visual C++ are the trademarks of Microsoft Corp.

Delphi is the trademark of Borland International.

TABLE OF CONTENTS

General Overview.....

Getting Started.....

PART I: USER'S MANUAL.....

User Commands.....

File Menu.....

Edit Menu.....

Field Menu.....

Section Menu.....

Line, Label and Picture Commands.....

Object Arrangement Commands.....

Object Selection.....

Field Concepts.....

Field Placement and Field Width.....

Field Value Types.....

Source of Field Data.....

Summary Fields:.....

Section Concepts.....

Section Types.....

Section Selection Criteria.....

Section Parameters.....

Calculation Expression.....

Operators.....

Condition Statement.....

Functions.....

PART II: DEVELOPER'S GUIDE.....

Form Editor Interface

Report Executer Interface

Major Data Structures

Memory Considerations

Source Level Customization

Analysis of the Demo Program

ReportEase Plus File Format

Sort and Join Utilities

Visual Basic Support

VBX function calls
- RvbDrawBitmap
- RvbExit
- RvbForm
- RvbGetDataField
- RvbGetFormField
- RvbGetPictureInfo
- RvbGetSortField
- RvbInit
- RvbRec
- RvbSetDoubleField
- RvbSetFormField
- RvbSetNumField
- RvbSetTextField

VBX Events
- DrawPicture
- SelectField
- VerifyField
- Unload

Delphi Interface

Visual C++ Interface

General Overview

ReportEase Plus consists of two components. The first component is the form editor. The form editor allows you to develop report layouts. The second component is the report executer. The report executer is used to print a report using a specified form.

ReportEase Plus provides a comprehensive set of features. The intuitive graphic form editor 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 form editor 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 form editor 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 form editor 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 form editor 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 form editor 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 executer 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:

Form Editor: Your application calls the form editor with the name of the form to be edited. Your application also supplies a routine that allows the user to select the data fields.

Report Executer: Your application initializes the report executer by calling the initialization routine with the name of the form to run. Then, your application calls the print routine for each data record in the sorted data set. The Report Executer performs the record selection, sort breaks, calculations and printing functions. At the end, your application calls the exit routine to print the footers and release the resources.

Requirements

The total memory requirement for all ReportEase Plus modules is approximately 400 K bytes.

ReportEase Plus source code is compatible with Microsoft and Borland 'C' compilers.

Getting Started

ReportEase Plus Files:

The ReportEase Plus diskettes contain these files.

Interface files:

REP.H	Header file to be included into your application. It contains the function prototypes and parameter structures needed to interface with the form editor and report executer routines.
REP.LIB	(REP32.LIB for WIN32) ReportEase Plus import library. This file must be linked with your application.
REP.DLL	(REP32.DLL for WIN32) ReportEase Plus dynamic link library contains the APIs.
REP.RC	Contains the ReportEase Plus resources.
REP_DLG.DLG	Contains the dialog box templates. This file is included in the REP.RC file.
REP_DLG.H	Contains the identifiers for the dialog boxes. This file is included in the REP.RC file.
REP_CMD.H	Contains the ids for Form Editor commands. This file is included in the REP.RC and REP1.H file.
REP.DEF	The definition file to create the DLL.

Major Source Files:

REP.C	Contains the screen painting functions for the form editor.
REP1.C	Contains the screen item management functions.
REP_INI.C	Contains the initialization routines.
REP_BLK.C	Contains the picture import routines.
REP_FILE.C	Contains the file i/o and print control routines.
REP_FLD.C	Contains the field manipulation routines.
REP_SEC.C	Contains the section manipulation routines.
REP_FMT.C	Contains character format and font routines.
REP_DLG.C	Contains the call back routines for every dialog box used by ReportEase Plus.
REP_EXP.C	Contains the expression compilation and execution routines.
REP_REP.C	Contains the report initialization, execution, and exit routines.
REP_REP1.C	Contains the report output routines.
REP_MISC.C	Contains miscellaneous routines.

Other Include Files:

REP1.H	Header file needed by the source files. It defines the internal global variables. You do not need to include this file into your application.
REP_DEF.H	Contains various program constants.
REP_PROT.H	Contains ReportEase Plus prototypes.

Help Files:

REP.RTF	Help text in the RTF format.
REP.HPJ	Help Definition File.
REP.HLP	Compiled help file.

Demo Files:

DEMO.EXE	The executable demonstration program
DEMO.C	Demo Source File. This file demonstrates the ReportEase Plus function calls.
DEMO.RC	Demo resource file.
DEMO.DEF	Linker definition file.
DEMO.H	Demo include file.
DEMO.RES	Compiled resource file.

Sample Data Files:

CUSTOMER.DF	Customer field definition and customer
CUSTOMER.DB	data file used by the demo program.
SALES.DF	Sales field definition and sales
CUSTOMER.DB	data file used by the demo program.

Sample Report and Mail Merge Form Files:

16 Bit DLL:

CUST.FP	Customer summary report
SALES.FP	Detail sales report
SUMDATE.FP	Sales transactions by date
SALEQRT.FP	Quarterly Sales Report
LABEL.FP	Address labels.

32 Bit DLL:

CUST.FPC	Customer summary report
SALES.FPC	Detail sales report
SUMDATE.FPC	Sales transactions by date
SALEQRT.FPC	Quarterly Sales Report
LABEL.FPC	Address labels.

Auxiliary Utilities

UTIL.DLL	(UTIL32.DLL for WIN32) Demo file sort utility program.
UTIL.C	Source code for UTIL DLL
UTIL.DEF	linker definition file for UTIL DLL
UTIL.RC	Blank resource file for UTIL DLL

Make files:

MAKES-BC.BAT MAKES-BC	Compiles and links the DEMO with the ReportEase Plus routines using the Borland 'C' compiler
MAKES-MC.BAT MAKES-MC	Compiles and links the DEMO with the ReportEase Plus routines using the Microsoft 'C' compiler

PART I: USER'S MANUAL

User Commands

The form editor 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 form editor commands. We will discuss the commands by the order in which they appear in the menu.

File Menu

This submenu contains the following selections:

Save

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

Report Parameters

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

Page Margins: You can specify top, bottom, left and right margins in inches. The form editor 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 form such as address labels. When this option is enabled, the report executer will print trial records to allow you to adjust the form on the printer.

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 executer. 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 form. 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 form. 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 form 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 form.

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.

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 form editor 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.

Change Fonts and Text Color

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

When you select this option, the form editor shows the font and color selection dialog box. The current font and colors are preselected in the dialog box. Use this dialog box to specify your selections.

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 form. This option will display a list of fields to choose from. When you select a field, the form editor 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, 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).

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 form. 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 form editor automatically assigns a 'total' attribute to the field. This attribute instructs the report executer 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.

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

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 form. 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 form editor 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.

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 form editor 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 form. A dialog field is used to prompt the user for data before running the report. The field can also be inserted in the form 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 form by using the *Insert New Field* option from the field menu. A dialog field can appear in more than one place within a form. 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 form, and removed from any calculation expression, before it can be deleted from the dialog table.

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 form editor 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 form editor 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 form editor inserts the new section in the proper order within the form.

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 executer 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 executer 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 form editor 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 form editor 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.

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.

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 form editor 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 form, 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 executer 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 form editor 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 form editor 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 form editor inserts the text 'label' in the label item. The label text can be edited in the editing window.

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 form editor 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 form editor 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 form editor 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.

This menu also contains an undo function to reverse an unintended arrangement command.

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.

Undo Previous Arrangement Command

Use this function to undo the previous arrangement command.

Object Selection

Most form editor 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 are selected by drawing 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, hod 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.

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.

Field Placement and Field Width

When a field is inserted using a menu option or the field button, the form editor 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 form, 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 form editor 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 form 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 executer 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 executer 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 SALSE-> prefix.

Technical Note: When the user wishes to insert a data field into a form, the form editor calls a field selection routine provided by your application. This routine should allow the user to select a data field. The form editor 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 *Form Editor 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.

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 form editor 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 executer can filter out the records which don't meet a specific criteria. Refer to the SALES.FP form (Detail Sales Report) for an example of the dialog fields.

Summary Fields:

The form editor 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

Section Types

ReportEase Plus organizes a report or a mail merge form 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 executer to determine a sort break. Most reports will use the same field for both the sort field and the break field. However, the form editor 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 form).

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:

a. Advance to the next page before printing the section.

b. Advance to the next page after printing the section.

c. Compress space between the beginning of the section and the topmost item in the section.

d. 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.

Calculation Expression

The calculation expressions can be used to perform the following:

a. Define the calculated fields.

b. Define the report selection criteria. The expression must evaluate to a TRUE or FALSE value.

c. 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.

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
Second Operand Type:	same as the first operand type
Result Type:	same as the first operand type
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"

Subtraction

Operator Symbol:	-
First Operand Type:	numeric,float,text
Second Operand Type:	same as the first operand type
Result Type:	same as the first operand type
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 "

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
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
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
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
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
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:

.MIN-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. This section describes available functions:

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

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

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

PART II: DEVELOPER'S GUIDE

Form Editor Interface

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

1. Include the REP.H file into your application module which will interface with the form editor.

2. Create a list of fields used in your data base. The user will select the fields from this list to insert in the form. 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 form 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 form editing session.

3. Write a field selection routine. This routine will be called by the form editor whenever the user wishes to insert a data field in the form. 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 Form Editor 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 form editor 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 executer 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 form editor 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(struct StrField far *field,int SortFieldNo)
```

The first 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 second Argument indicates whether the form editor 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 *StrForm* (defined in the REP.H file). This structure is used to pass the initial form parameters to the *form* function. Example:

```
struct StrForm FormParm;
```

The *StrForm* 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;
}
```

6. Fill the *StrForm* structure variables as following:

x: Specify the initial X position (in device units) of the form editor window. You may specify CW_USEDEFAULT to use the default value.

y: Specify the initial Y position (in device units) of the form editor 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 form editor 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 form editor window is closed**)

file: Specify the name of the form template file. The full path name is allowed in the file name.

DataSetName: This field is used only when creating a new form. Using this field, you can specify a name for the data set needed for this form. The data set name for the form is stored in the disk file. Note, that this field is not needed for form editing and is never used internally by ReportEase Plus. When you initialize a report for execution, the report executer 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 form editor 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 form editor 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 form editor will place into this field the handle of the form editor window when it is created.

style: Use this field to specify the style word for the form editor window.
FontTypeFace: Specify the typeface for the default font. Set this field to NULL if you wish the form editor to use the preset default typeface.

open: Set this field to FALSE. The form editor will set this field to a TRUE value after opening the form editor window.

modified: This flag is used internally and it should be set to FALSE.

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 form editor routine as following:

```
form(&FormParm);
```

This routine displays the selected form in an editor window and allows the user to edit the form. 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

10. Edit the link statement in your make file to include the REP.LIB (REP32.LIB for WIN32 applications) import library file.

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

Report Executer Interface

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

RepInit(struct StrRep far *): Your application calls this routine to initialize the report executer for a form. The name of the form is provided using a variable in the StrRep structure.

RepRec(): Your application calls this routine for each record in the sorted data set.

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

Follow these steps to interface with the report executer:

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

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

   int FAR PASCAL DrawPicture(HDC hDC, int PictId, int FileId, int FieldId, RECT far *rect)

   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.

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 form 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
'A' = Ask User.

If the device is specified as 'A', the report executer prompts the user to select a printer or screen 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 executer 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 executer'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 executer. 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 executer. 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 executer. 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 executer. 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 form (see previous chapter, step #8). This variable does not have any significance for the internal use of the report executer.

SwapDir: Specify the directory path to store swapped screen pages. Set this variable to NULL, if you wish the report executer 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, 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 form editor window is closed**)

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 form. It also returns the data set used by the current form. 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 form editing session (see the previous chapter, step 5). These fields are not of any internal significance to the report executer.

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 executer 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()!=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();

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

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 form editor. 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 *Form Editor 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 Executer Interface* chapter for a complete description of the structure members.

StrFormHdr

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

FormSign: (unsigned) A valid form file will have a 2 byte code in the beginning of the file. The value of this code should be 0xDEBC.

name: (char [52]) Name of the form. The form 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 form.

FieldCount: (int) Total number of fields used in the form.

BreakFieldCount: (int) Total number of sort/break fields used in the form.

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 form 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 form 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 form: 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 form. The form editor 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 executer 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 executer to accumulate summary data for the numeric field.

HoldDbl: (double) Used by the report executer 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 executer 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 form. 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 form.

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 executer.

FirstItem: (int) The position of the first screen item belonging to this section. This field is used by the report executer.

ItemCount: (int) Number of screen items which belong to this section. This field is used by the report executer.

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 form. 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.

Memory Considerations

The memory requirement for ReportEase Plus can be roughly categorized as following:

Code	125K
DATA Segment	36K
Dynamic Data	250K

Additional 35K bytes of memory may be needed for 'C' run time library routines (Borland).

Although, ReportEase Plus routines leave an ample amount of memory of other tasks, you make take certain steps to release even more memory if necessary.

Reduce the MAX_FIELDS constant from 500 to 200. This will save approximately 85K bytes of dynamic memory. However, this change will allow only up to 200 fields per form.

Reduce the MAX_ITEMS constant from 150 to 75. This will save approximately 28K bytes of dynamic memory. However, this change will allow only up to 75 screen objects per form.

Reduce the MAX_DLGS constant from 12 to 4. This will save approximately 2K bytes in the data segment. However, this change will allow only up to 4 dialog fields.

To save memory from the code segment, compile the form editor and report executer as separate executables. The form editor does not require most of the routines from the REP_REP.C and REP_REP1.C modules. Whereas, the report executer does not need most of the routines from the REP1.C, REP_SEC.C, REP_FLD.C, and REP_BLK.C modules. In this arrangement, each executable file will save approximately 20K bytes.

Source Level Customization

The startup parameter structure for the form editor provides some customization. Additional customization is possible by altering the values of some global variables or making the minor changes to the source code.

Command Keystrokes:

You can change the keystroke for a command by simply editing the accelerator resource statement in the REP.RC file. First, select a new unique keystroke for the command. Then, edit the corresponding resource statement to reflect this change.

For example, if you wish to change the keystroke for the 'Delete Item' command, locate the original resource statement:

VK_DELETE, ID_DEL_ITEM, VIRTKEY, NOINVERT

If you wish to change the command key from the 'Del' key to the Alt G key combination, change the resource statement as following:

"G", ID_DEL_ITEM, VIRTKEY, NOINVERT, ALT

Screen Colors:

These global variables (defined in the REP1.H file) control the screen colors:

BackColor:	Background screen color.
DrawColor:	Color used to draw the rulers, section banners, dashed item boundary, etc.
InputAreaColor:	Color of the area between the form area and the menu area.
SelectionColor:	Color of the multiple selection rectangle, and the color of the selected option on the option bar.
OptRectColor:	Default color of the option bar.

The editor defines the color variable using the COLORREF declaration. A color variable has 3 components, each a byte long. The first byte represents the RED color, the second the GREEN, and the third byte represents the BLUE color. Each color byte can have a value from 0x00 to 0xFF. The 0x00 value represents the absence of this color component, whereas the 0xFF value represents full contribution of this color toward the final color. For example, a black color has all its components at 0 value, whereas a white color has all its components equal to 0xFF. All intermediate colors can be derived by varying the value of these 3 components.

Analysis of the Demo Program

This chapter describes the demo program. The demo program provides an example of interfacing with ReportEase Plus. You may like to begin with the demo program code to create your own interface.

Data Files

The demo program uses two data files: CUSTOMER and SALES file. The customer file contains the client data such as customer id, name and address. The sales file contains the information about order transactions for each customer id. This file contains the customer id, order date, order amount, description of the order item, etc. The customer id is the common field in these two files. During the report execution session, the customer id field is used to join the two files to produce a combined data set.

Each data file has an extension of .DB (CUSTOMER.DB and SALES.DB). The data is stored in the file in the text format. The fields are separated by the comma character. The text fields are enclosed within quotation marks. You can type the CUSTOMER.DB and SALES.DB files at the DOS prompt to display the contents of the file.

Each data file also has a data definition file which has an extension of .DF (CUSTOMER.DF and SALES.DF). Each line of the data definition file contains the information about one field of the file. The line contains the field name, maximum width of the field, field type, and number of decimal digits for the numeric and float fields. The field type can be text (T), numeric (N), float (F), date (D), and logical (L). The *ReadFields* function in the demo program reads the field definition files and stores the data about individual fields in a structure array called DataField.

Include Files

The demo program includes the REP.H file. This file must be included into your application module which will interface with ReportEase Plus.

Definition File

The demo.def file contains the export functions for its dialog boxes and two additional functions used for field selection and verifications. Each export function in the export section must have a unique ordinal number. see the DEMO.DEF file for an example.

Global Constants

MAX_FORMS	Number of forms that can be displayed by the demo program menu.
MAX_FILES	Number of data files supported by the demo program (CUSTOMER AND SALES)
MAX_FIELDS	Number of fields that each data file can have.
ITEM_WIDTH	Display width of the menu items.

Global Variables:

struct StrForm FormParm; This structure is used to call the *form* function. For the definition of the StrForm structure, refer to the REP.H file.

struct StrRep RepParm; This structure is used to call the *RepInit* function. The *RepInit* function initializes the report executer. For the definition of the StrRep structure, refer to the REP.H file.

char FormName[MAX_FORMS+2][NAME_WIDTH+2]; This array stores the form names for the form files found in the current directory. The form name is read from a header structure within a form file.

char FormFile[MAX_FORMS+2][13]; This array stores the names of the form files found in the current directory. The form files have a .FP extension.

struct StrDataFile DataFile[MAX_FILES]; This array stores the data file names and total number of fields in each data file.

struct StrDataField DataField[MAX_FILES][MAX_FIELDS]; This structure stores the field definition for each field of both files. The field definition is read from the CUSTOMER.DF and SALES.DF files.

Program Flow

The following pseudo code gives an overview of the demo program flow.

```
InitInstance() {
        Create Demo Window.
        Initialize Form Editor calling parameters.
        Initialize Report Executer Initialization parameters.
        Read the field information for each data file.
}

-----------------------------------------------------------------------------------------------------
ReadFields() { // read fields from a definition file
    Open the definition file.
    Read lines from the file.  Each line describes one field of the file.
    For each line, use ExtractField() to extract field attributes {
        Extract the field name.  Store it in the DataField array.
        Extract the field maximum width.  Store it in the DataField array.
        Extract the field type.  T=Text, N=Numeric, F=Float, D=Date, L=Logical.  The field
        attributes are stored in the DataField array using the TYPE_ constants (REP.H)
        Extract the number of decimal digits for the numeric and float fields.
    }
    Close the definition file.
}
-----------------------------------------------------------------------------------------------------
ExtractField() {
    Extract the next field attribute from a text line.  After extracting the current field attribute,
        this routine advances the index in the text line, so that the next call can extract the
        next attribute.
```

}

CAllEditor() {
    GetFormSelection()   // get a form name to edit
    Set the typeface to NULL (use default).
    Set the point size to 0 (use default).
    Call the form editor function:
        form(&FormParam)
}

GetFormSelection() {
    Get the report form files, add them to the selection list.
    When called from the EditForm function, add an additional selection to allow the creation
        of new report form.
    Show the selection in the list box and let the user select one form.
}

GetFormFiles() {
    Read from the current directory the files with the .FP extension.
    Open each file to read the form header.
    Extract the form name from the form header.
    Return the form name.
}

UserFieldSelection() {
    This routine is called by the form editor when the user wishes to insert a data field in the
        form. The form editor passes the pointer to the field structure (second argument).
        This routine will pass the information about the selected field using the field pointer.
        The third argument indicates if the new field will be used for sorting. The zero value
        for the second field indicates that the field is not a sort field. A non-zero value
        indicates the sort level for which the field will be used. The demo program allows
        only the fields from the CUSTOMER file to be used for sorting. Therefore, if the
        third argument is non-zero, the demo program allows the user to select the fields only
        from the CUSTOMER file.
    If the third Argument is 0, show the file names (CUSTOMER and SALES to select).
        Otherwise assume the CUSTOMER file.
    Show the fields for the selected file, and let the user select one field.
    Fill in these values in the field structure:
        field->name
        field->width
        field->type
        field->DecPlaces.
        The above values are copied from the DataField structure which was created by the
        ReadFields function during initialization.
        Fill in two additional values (optional):
        field->FileId
        field->FieldId
        This information is used to easily identify the field to fill in data during the report
        execution session.
}

VerifyField {
    This routine is called by the form editor to verify a data field used in the form. The form
        editor passes the pointer to the field structure (first argument). This routine will
        provide the information about the selected field using the field pointer. The second
        argument indicates if the new field will be used for sorting. The zero value for the
        second field indicates that the field is not a sort field. A non-zero value indicates the

sort level for which the field will be used. The demo program allows only the fields from the CUSTOMER file to be used for sorting.
Verify that the field name exists in the CUSTOMER and SALES files.
Fill in the field values in the field structure:
}

RunReport() {
GetFormSelection() // select a form to execute
Fill the report initialization structure (StrRep) as following:
device='A' // ask user for the output device
specify the window width for screen output.
Call the report initialization routine:
RepInit(&RepParm) // a Report Executer function
This routine returns two pointers. The first pointer points to the report executer's output field array. This pointer is used by the *PrintRecord* function to supply the data. The second pointer points to another field array which contains the sort fields used by the form. The *PrepareFile* function extracts the sort field names from this array to sort the data records as needed.
PrepareFile() //Prepare data set
PrintRecord() //print each record
RepExit() //Close the report executer
}

PrepareFile() { // Sort and join the customer and sales files.
Extract the sort fields from the sort field array.
Sort the customer file by the given sort fields.
Determine if any of the fields in the SALES files have been used by the form.
If the SALES file is used, join the SALES file to the CUSTOMER file.
The data set is now ready for printing.
}

PrintRecords() {
Read each record from the sorted data set file.
For each record field in the data record:
Initialize the data values in the output field array.
For each field in the output array:
If the field in the output array is an application field,
and if the field id and file id are the same as the current record field:
Fill in the data value for the field.
}
RepRep() // Ask report executer to print the current record
}
}

ReportEase Plus File Format

The ReportFile consists of 7 sections in the following sequence:

Form Header Block
Screen Object Block
Field Block
Break Field Block
Section Block
Dialog Block
Font Table

Form 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 form. 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 form.

Section Block: This block stores the section attributes for all 23 sections allowed by the form editor. For a description of various sections allowed in the form, 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 form editor. 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 form 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 Height	WORD
Picture Width	WORD
Image Size	DWORD
Info Size	DWORD
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. The demo program uses 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 filename, int NumKeys, LPINT KeyTable)

The first argument specifies the name of the 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 specified the number of sort fields to be used for sorting. The third 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.

The output sort file name consists of the input file name prefix with a .SRT extension.

Return Value: This function returns TRUE when successful.

Examples:

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

   FileSort("myfile",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",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.

Visual Basic Support

(Not applicable to WIN32)

ReportEase Plus can now be used by a Visual Basic application. Your application should include the REP.BAS and RVB.VBX files in the project. You should also have REP.DLL and RVB.VBX files available in the project directory.

The VBX manifests itself on the Visual Basic toolbar with an icon. Your application should create a RE control by clicking on the icon. This control is visible only during the design time and is used to communicate with the Form Editor and Report Executor windows. The Form Editor and Report Executor use pop-up windows which are not confined to your application form window.

Your application is responsible for providing the data field names to the Form Editor and data field values to the Report Executor. Your application communicates with the VBX using the events and function calls. This section describes the function calls and events in alphabetic order. The DMO_VBX program included in the diskette provides an example of using these function calls and events. Please refer to this demo program source as needed.

VBX function calls

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)

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 Event

RvbExit

Close the Report Executor

int RvbExit(hWnd as Integer)

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: RvbInit, RvbRep

RvbForm

Launch the Form Editor

int RvbForm(FormParam as TypeForm)

Description: This function is used to launch the Form Editor. Your application provides the form 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 Form Editor 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 event, VerifyField Event.

RvbGetDataField

Get the specified ReportExecutor data field.

int RvbGetDataField(hWnd as Integer, FieldNo as Integer, field as TypeField)

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: RvbInit, RvbSetTextField, RvbSetNumField, RvbSetDoubleField

RvbGetFormField

Get the current Form Editor field

int RvbGetFormField(hWnd as Integer, field as TypeField, SortLevel as Integer)

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)

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, DrawBitmap event

RvbGetSortField

Get the specified ReportExecutor sort field.

int RvbGetSortField(hWnd as Integer, FieldNo as Integer, field as TypeField)

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)

Description: This function is used to initialize the Report Executor. The hWnd parameter is the window handle of the control. The application provides the form 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)

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)

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, RvbSetTextField, RvbSetNumField

RvbSetFormField

Set the current Form Editor field

int RvbSetFormField(hWnd as Integer, field as TypeField, valid as Integer)

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)

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)

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

VBX 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.

SelectField

User field selection handler

Description: This event is fired by the Form Editor 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:

1. 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...).

2. 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.

3. 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.

4. Return the updated field structure to the FormEditor using the RvbSetFormField function.

VerifyField

User field verification handler

Description: This event is fired by the Form Editor 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:

1. 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...).

2. Verify the specified field for validity. Skip the next two steps if the field is not valid.

3. 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.

4. Return the updated field structure to the FormEditor using the RvbSetFormField function.

Unload

This event is fired when the form editor or the report executor window is being closed. In response to this event your application should reset the 'open' flag in the 'FormParm' structure.

Delphi Interface

A Delphi application (not applicable to WIN32) interfaces directly with the REP.DLL. The VBX wrapper (RVB.VBX) is not needed to interface with the DLL. Please refer to the DMO_DLP.DPR demo program as an example of interfacing a Delphi application to ReportEase Plus.

Interface Units: The package contains the following files to interface with a Delphi application:

REP.PAS: This file contains the constant and type definition to interface with the DLL. This file is to be included in the Interface section of your application unit which interfaces with ReportEase Plus. Example:

```
Interface
uses ....
{$I REP.PAS}
```

REP_PROT.PAS: This file contains the function declarations for the DLL. This file is to be included in the Implementation section of your application unit which interfaces with ReportEase Plus. Example:

```
Implementation
uses ....
{$I REP_PROT.PAS}
```

The REP.PAS and REP_PROT.PAS files together provide the same functionality as the REP.H file does for a 'C' language application. The constants, types and function names in these units are the same as they are in the REP.H file.

Message Callback Function: The ReportEase Plus form editor and report executer send a message to the parent window (your application form window) before closing the form editor or the report executer windows. Because, Delphi does not include a provision to intercept custom messages, ReportEase plus provides an alternative method of sending the message to a Delphi application. Your application defines a callback function to receive the message.

The following example sets a flag when the form editor or the report executer session is being ended:

```
function MsgCallback(hWnd: THandle; msg: integer): LongInt;
begin
    if (msg=REP_CLOSE) then ReportClose:=True;  {set a global flag}
end;
```

For a callback function to start receiving messages, it must be registered with the DLL immediately after the form editor or the report executer is initialized. Example:

```
form(FormParm);        {launch form editor}
RepSetMsgCallback(@MsgCallback);   {register the callback function}
```

Visual C++ Interface

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.

2. Modify the *alignment* compiler option for your application to specify the alignment at **1 byte**.

Recompiling REP 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, REP.DEF and REP.RC

Executable Type: Windows DLL

Compiler Option: 1 Byte Alignment

Remaining parameters should be left at their default values.