Reports Authoring Visual Environment

Developer Guide

MANUAL
for Reference & Learning

Rave revision 7.0.5C

Single User License Agreement

This is a legal Agreement between you, as the end user, and Nevrona Designs. By opening the enclosed sealed disk package, or by using the disk, you are agreeing to be bound by the terms of this Agreement. If you do not agree with the terms of this Agreement, promptly return the unopened disk package and accompanying items, (including written materials), to the place you obtained them for a full refund.

1. Grant of License:

Nevrona Designs grants to you the right to use one copy of the enclosed Nevrona Designs program, (the Software), on a single terminal connected to a single computer (i.e. CPU). You may make one copy of the Software for back-up purposes for use on your own computer. You must reproduce and include the copyright notice on the back-up copy. You may not network the Software or use it on more than a single computer or computer terminal at any time, unless a copy is purchased for each computer or terminal on the network that will use the Software. You may transfer this Software from one computer to another, provided that the Software is used on only one computer at a time. You may not rent or lease the Software, but you may transfer the Software and accompanying written material and this license to another person on a permanent basis provided you retain no copies and the other person agrees to accept the terms and conditions of this Agreement. THIS SOFTWARE MAY NOT BE DISTRIBUTED, IN MODIFIED OR UNMODIFIED FORM, AS PART OF ANY APPLICATION PROGRAM OR OTHER SOFTWARE THAT IS A LIBRARY-TYPE PRODUCT, DEVELOPMENT TOOL OR OPERATING SYSTEM, OR THAT MAY BE COMPETITIVE WITH, OR USED IN LIEU OF, THE PROGRAM PRODUCT, WITHOUT THE EXPRESS WRITTEN PERMISSION OF NEVRONA DESIGNS. This license does include the right to distribute applications using the enclosed software provided the above requirements are met.

2.Term:

This Agreement is effective until you terminate it by destroying the Software, together with all copies. It will also terminate if you fail to follow this agreement. You agree upon termination to destroy the Software, together with all copies thereof.

3. Copyright:

The software is owned by Nevrona Designs and is protected by United States laws and international treaty provisions. Therefore, you must treat the Software like any other copyrighted material (e.g. a book or musical recording) EXCEPT that you may either (a) make one copy of the Software solely for back-up or archival purposes, or (b) transfer the Software to a single hard disk provided you keep the original solely for back-up or archival purposes. You may not copy the written materials accompanying the Software.

Prepared: December 2008 in Arizona

Limited Warranty

1. Limited Warranty:

Nevrona Designs warrants that the disks on which the Software is furnished to be free from defects in material and workmanship, under normal use, for a period of 90 days after the date of the original purchase. If, during this 90-day period, a defect in the disk should occur, the disk may be returned with proof of purchase to Nevrona Designs, which will replace the disk without charge. Nevrona Designs warrants that the Software will perform substantially in accordance with the accompanying written materials. Nevrona Designs does not warrant that the functions contained in the Software will meet your requirements, or any operation of the Software will be uninterrupted or error-free. However, Nevrona Designs will, after being notified of significant errors during the 90-day period, correct demonstrable and significant Software or documentation errors within a reasonable period of time, or refund all or a fair portion of the price you have paid for the Software at Nevrona Designs' option.

2. Disclaimer of Warranties:

Nevrona Designs disclaims all other warranties, either expressed or implied, including but not limited to implied warranties of merchantability of fitness from particular purpose, with respect to the Software and accompanying written materials. This limited warranty gives you specific legal rights, you may have others, varying from state to state. Nevrona Designs will have no consequential damages. In no event, shall Nevrona Designs or its suppliers be liable for damages whatsoever, (including without limitation, damages for loss of business profits, business interruption, loss of business information, or any pecuniary loss), arising out of the use or the inability to this Nevrona Designs product, even if Nevrona Designs has been advised of the possibility of such damages. Some states do not allow the exclusion of limitation of liability for consequential or incidental damages, and this limitation may not apply to you.

3. Sole Remedy:

Nevrona Designs' entire liability in your inclusive remedy shall be, at Nevrona Designs' option, either: (1) The return of the purchase price paid; or (2) Repair or replacement of the Software that does not meet Nevrona Designs' limited warranty, which is returned to Nevrona Designs with a copy of your receipt.

4. Governing Law:

This Agreement will be construed and governed in accordance with laws of the State of Arizona.

5. U.S. Government Restricted Rights:

This Software and documentation are provided with restrictive rights. Use, duplication or disclosure by the Government is subject to restrictions set forth in Section c(1)(ii) of the Rights and Technical Data in Computer Software clause at 52.227-7013.

Nevrona Support

Technical Support

Technical support is provided to registered users for questions or problems with Rave. For fastest service contact us by e-mail or fax. Please include both the Rave version and product serial number (found on the Help About screen) along with any information related to the problem.

Internet:	tech@nevrona.com
Web page:	http://www.nevrona.com
Fax Phone:	602.296-0189
Mailing Address:	Nevrona Designs 5301 S Superstition Mountain Dr Ste 104-345 Gold Canyon AZ 85218-1917

News Groups

Several newsgroups are provided free of charge to assist you in getting help with our products. When you visit our newsgroups you will be connected to other users with similar interests. If you have a question, just post it to the newsgroups and others reading the newsgroups will see the message and be able to respond to it. You'll also see questions and solutions from other users as they are posted.

The Nevrona Designs newsgroups are available at news://news.nevrona.com.

To access the Nevrona Designs newsgroups, create a new server entry in your newsgroup reader with the Host address of news.nevrona.com and open that server. You should then see several newsgroups that you can subscribe to.

Sales Support

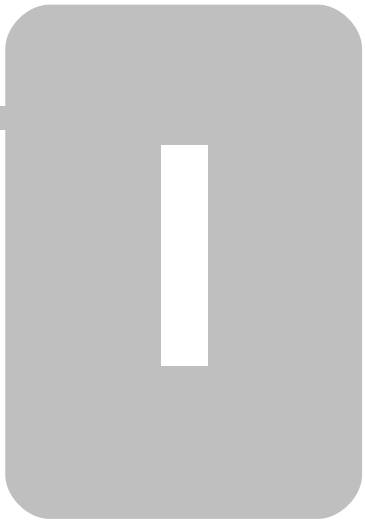
Internet:	sales@nevrona.com
Phone:	480 . 491 - 5492

Prepared: December 2008 in Arizona

Table of Contents

Introduction

Chapter



1 Introduction

Congratulations! You've made an excellent choice. Rave was designed to give the power of reporting back to the programmer where it belongs. Rave does this by offering a powerful suite of printing components that simplify the task of creating professional reports. Rave does not need any extra .DLL's, .VBX's or .EXE's. Reports can be written entirely with code and compiled in your application for easier distribution and faster execution. Or you can use the visual designer and its components for creating your reports. The visually designed reports are normally stored in one or more file(s) that are saved as an external RAV file. If desired, these report definitions can be saved in the application EXE. Read through this manual and the examples on the accompanying disk and you'll soon be turning nightmare printing jobs into dream reports.

Contacting Nevrona Designs

Upgrading from bundled version of Rave Reports to BEX

1.1 License Agreement

1. **Grant of License** - Nevrona Designs grants to you the right to use one copy of the enclosed Nevrona Designs program, (the Software), on a single terminal connected to a single computer (i.e. CPU). You may make one copy of the Software for back-up purposes for use on your own computer. You must reproduce and include the copyright notice on the back-up copy. You may not network the Software or use it on more than a single computer or computer terminal at any time, unless a license is purchased for each computer or terminal on the network that will use the Software. You may transfer this Software from one computer to another, provided that the Software is used on only one computer at a time. You may not rent or lease the Software, but you may transfer the Software and accompanying written material and this license to another person on a permanent basis provided you retain no copies and the other person agrees to accept the terms and conditions of this Agreement.

THIS SOFTWARE MAY NOT BE DISTRIBUTED, IN MODIFIED OR UNMODIFIED FORM, AS PART OF ANY APPLICATION PROGRAM OR OTHER SOFTWARE THAT IS A LIBRARY-TYPE PRODUCT, DEVELOPMENT TOOL OR OPERATING SYSTEM, OR THAT MAY BE COMPETITIVE WITH, OR USED IN LIEU OF, THE PROGRAM PRODUCT, WITHOUT THE EXPRESS WRITTEN PERMISSION OF NEVRONA DESIGNS. This license includes the right to distribute applications using the enclosed software provided the above requirements are met. Certain files are redistributable with your applications, see the enclosed Redist.txt for more information.

2. **Term** - This Agreement is effective until you terminate it by destroying the Software, together with all copies. It will also terminate if you fail to follow this agreement. You agree upon termination to destroy the Software, together with all copies thereof.

3. **Copyright** - The software is owned by Nevrona Designs and is protected by United States laws and international treaty provisions. Therefore, you must treat the Software like any other copyrighted material (e.g. a book or musical recording) **EXCEPT** that you may either (a) make one copy of the Software solely for back-up or archival purposes, or (b) transfer the Software to a single hard disk provided you keep the original solely for back-up or archival purposes. You may not copy the written materials accompanying the Software.

1.2 Limited Warranty

1. **Limited Warranty** - Nevrona Designs warrants that the disks on which the Software is furnished to be free from defects in material and workmanship, under normal use, for a period of 90 days after the date of the original purchase. If, during this 90-day period, a defect in the disk should occur, the disk may be returned with proof of purchase to Nevrona Designs, which will replace the disk without charge. Nevrona Designs warrants that the Software will perform substantially in accordance with the accompanying written materials. Nevrona Designs does not warrant that the functions contained in the Software will meet your requirements, or any operation of the Software will be uninterrupted or error-free. However, Nevrona Designs will, after being notified of significant errors during the 90-day period, correct demonstrable and significant Software or documentation errors within a reasonable period of time, or refund all or a fair portion of the price you have paid for the Software at Nevrona Designs' option.

2. **Disclaimer of Warranties - Nevrona Designs disclaims all other warranties, either expressed or implied, including but not limited to implied warranties of merchantability of fitness from particular purpose, with respect to the Software and accompanying written materials. This limited warranty gives you specific legal rights, you may have others, varying from state to state. Nevrona Designs will have no consequential damages. In no event, shall Nevrona Designs or its suppliers be liable for damages whatsoever, (including without limitation, damages for loss of business profits, business interruption, loss of business information, or any pecuniary loss), arising out of the use or the inability to this Nevrona Designs product, even if Nevrona Designs has been advised of the possibility of such damages. Some states do not allow the exclusion of limitation of liability for consequential or incidental damages, and this limitation may not apply to you.**

3. **Sole Remedy** - Nevrona Designs' entire liability in your inclusive remedy shall be, at Nevrona Designs' option, either: (1) The return of the purchase price paid; or (2) Repair or replacement of the Software that does not meet Nevrona Designs' limited warranty, which is returned to Nevrona Designs with a copy of your receipt.

4. **Governing Law** - This Agreement will be construed and governed in accordance with laws of the State of Arizona, United Stated of America.

5. **U.S. Government Restricted Rights** - This Software and documentation are provided with restrictive rights. Use, duplication or disclosure by the Government is subject to restrictions set forth in Section c(1)(ii) of the Rights and Technical Data in Computer Software clause at 52.227- 7013.

1.3 Upgrade Rave BE

Rave Reports BE (Bundled Edition)

from

Delphi 7 or 8, BDS 2005 or 2006 or RAD Studio 2007 or 2009

to

Rave Reports (Developer or Architect)

There are several Bundled Editions (BE) of Rave Reports which are distributed with CodeGear (Borland) products. These bundled versions of Rave Reports are fully functional professional level reporting tool. However, if you upgrade to either the Developer or Architect versions, you will receive these additional features:

Rave Reports (Developer or Architect) includes .NET

Supports Versions	VCL (Win32)	WinForm	VCL (.NET)	CLX
C++Builder 4 and 5	X			
C++Builder 6	X			X
C++Builder 2009	X			
Delphi 4, 5 and 6	X			
Delphi 7	X			X
Delphi 8	no	no	no	
Borland Developer Studio 2005	X	code only	X	
Borland Developer Studio 2006	X	code only	X	
CodeGear RAD Studio 2007	X	code only	X	
CodeGear Delphi 2009	X			

All versions of Rave Reports (Developer or Architect) include the following:

- Reports are cross-platform compatible
- Full component source code
- Additional rendering components (output reports to BMP, HTML, JPG, PDF, RTF, TXT and WMF)
- Direct access to Rave's code based printing APIs
- Access to scripting engine
- Section component OnPrint event (access to many code base methods)
- Additional code base components - TablePrinter and DbTablePrinter
- Additional code base components - DetailShell, MasterShell, ReportShell and LabelShell
- Ability to create and install add-in components and design-time customization
- debugger for scripts in visual reports
- native charting components (both code based and visual)
- Report Summary expert (print contents of visual report design)
- Mirrored components can be overridden for more reuse flexibility
- Free technical support via email
- End User Designer license included in Architect ONLY
- Rave Reporting Server available

For more information about what Rave Reports can do for you, visit us on the web at http://www.nevrona.com/rave/be.html or contact us via one of the methods listed in the Support section.

1.4 TechSupport

Contacting Nevrona Designs

Web page:	http://www.nevrona.com
Sales:	Please email sales@nevrona.com or call (480) 491-5492.
Technical Support:	Please see http://www.nevrona.com/support for full information.
News Groups:	news://news.nevrona.com
Mailing Address:	5301 S Superstition Mountain Dr Ste 104-345 Gold Canyon AZ 85218-1917
Voice Phone:	(480) 491-5492
Fax:	(602) 296-0189

Please include both your Rave version and serial number in all messages to Nevrona. If it is a technical support request then it would also help if you included your operating system and language you are using.

Operating Systems including SP (service pack level)

Windows Linux .NET other

Language including version and service pack

Delphi C++Builder VB C# other

1.5 Installation

IMPORTANT: If you have installed an older or demo copy of Rave Reports, first remove all instances of those files from the component library and your hard disk before continuing with the installation process. If you do not remove all older versions of Rave Reports you may get a type mismatch or other compiler / runtime error.

Step 1: Backup your component library. If you encounter any problems during the installation process, you can restore your component library by restoring this file.

Step 2: Run the SETUP.EXE program found on the product disk and answer the prompts as they are presented. Make sure to read the installation notes for any additional steps that may be necessary or changes that may have occurred since the documentation was prepared.

Connecting the Help File To Delphi / C++Builder's IDE: By connecting Rave Report's Help file to the IDE, you can get context sensitive help for any Rave Reports event, method or property just by placing the cursor on the keyword and pressing F1. To enable this feature, you must first follow these steps:

Delphi 4:

1 Copy the RaveDevRef.HLP and RaveDevRef.CNT files into your Delphi ..\Help directory.
2 Edit the DELPHI4.CFG and remove any reference to Rave*.HLP
3 Edit the DELPHI4.CNT and add two lines (at the end of Include section):
4 :Include RaveDevRef.CNT
5 :Link RaveDevRef.HLP
6 See note below

OpenHelp Utility (Delphi 5, 6 and 7):

1. Copy the Rave*.CNT and Rave*.HLP files into your Delphi??\Rave??.\Help directory.
2. Start Delphi -Select the "Customize" option in the "Help" menu (OpenHelp utility)
3. Do the following for each of the "Contents", "Index" and "Link" tabs
4. Select the "Add Files" option under the "Edit" menu.
5. Add Rave.CNT to "Contents" tab and Rave.HLP to both the "Index" and "Link" tabs
6. Add RaveDevRef.CNT to "Contents" tab and RaveDevRef.HLP to both the "Index" and "Link" tabs
7. See note below

If you do not have access to the OpenHelp utility, then do the following "manual" steps.

1 Add a Link statement referencing the Rave.HLP file at the end of the ..\Delphi??\Help\DEL??XTRA.OHL file.
2 :Link RaveDevRef.HLP

Delphi 8, 2005 and 2006

1. Uses MS Help version 2 (not supported by Microsoft)
2. Requires VS.NET to register
3. There are no manual steps to add help

C++Builder:

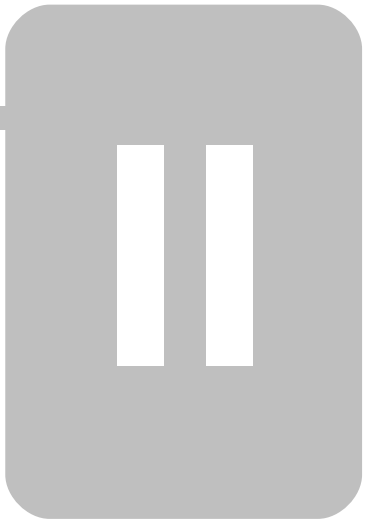
1. Run the ..Borland\Common files\OH.EXE (OpenHelp utility).
2. Click the Search Ranges tab in OpenHelp.
3. Choose Add below the Available Help Files list box.
4. Browse to the ..\Rave??\RaveDevRef.HLP file then choose Open.
5. Click the OK button to save the changes and close the OpenHelp utility.

NOTE

After making any changes to the help system, be sure to delete any existing GID file in the Help directory. (...\Help\DELPHI#.GID). This will force the Help system to rebuild this file on the next usage. This is a hidden file, so you will not see it unless you have your Windows File Explorer options set to display hidden files.

FAQ

Chapter



2 FAQ

Frequently Asked Questions

There are many Rave Tips and Tricks on the Nevrona web site at: http://www.nevrona.com/rave/tips.shtml .

Also, there are many Rave Add-Ons available at: http://www.nevrona.com/rave/addons.shtml . When downloading Add-Ons, be sure to get the one that matches your version of Rave Reports.

2.1 Design / Layout

Questions about Report Design or Layout.

2.1.1 Footers

QUESTION:
I need to design a report so that the footer is at the physical bottom of each page, regardless of how much or how little is in the detail section.

ANSWER:
You need to set the Band's "PositionMode" property to pmAbsolute. Then set the "PositionValue" to desired starting location. With pmAbsolute the starting position is measured from the top of the controlling region for this band.

2.1.2 Headers

QUESTION:
My report MUST have the same header and footer on EACH page, regardless of how many detail records there are for the master record.

ANSWER:
The solution is put the items you want on every page directly ON THE PAGE not in a band. Pardon the caps. Just wanting to emphasis that you do not have to do anything or everything with bands in a region. The main thing to get across is that Rave is a Page-Based designer and exactly what that means.

If you are doing a form style report, like one of the IRS forms, etc, you don't need to use a region at all. We have seen some RAV files that were ALL form style and none of the reports had a region. Of course, if you are doing a "form" report, you may need to use the "DataCycle" component to move through the data tables. Remember, the "DataCycle" should be the first component in the tree list.

If you are doing a band style report then you use the region component as a "holder" for band components.

If a component has to be in a region, it will not let you put it outside of a region. Try and put a band directly on the page - can not do it. Try other components, Text, DataText, DataMemo etc. and they will go where you like. Of course, you have to assign the DataView and DataField properties on the data driven components as you normally would.

2.1.3 Page Numbers

QUESTION:
How can I put Page Number and Date Information on my Reports?

ANSWER:
Drop a DataText component directly on the Page where you want one of your desired informational items. Start the DataField-Editor by clicking on the ellipse (3 dots) on the right side of the DataField Property. There you can define several things other than the normal "Database" type components. What we want to focus on is the "Report Variables" section. The drop box in that section is where you can select from many of the commonly needed report variables like "CurrentPage", "TotalPage" and a variety of date strings. Down at the bottom of the "Data Text Editor" dialog is a "Data Text" section. Once you have selected a "Report Variable", you have to transfer your selection to the "Data Text" area. This is done with the "Insert Report Var" button. As you transfer your selections, you will note that they appear in the "Data Text" area. If you know the exact format, you can actually type or modify the information directly in the Data Text box. You should end up with something like the following:

For a "Page 1 of 99 Pages" format use:

```
"Page" & Report.CurrentPage & "of" & Report.TotalPages & "Pages"
```

For a "Report printed on: mm/dd/yy" use:

```
"Report printed on:" & Report.DateShort"
```

"+" and "&" signs are for concatenation. The "+" sign is for no spaces and the "&" sign will give you a space between.

2.2 Error

Some of the possible errors, warnings or problems you might encounter when using Rave Reports.

2.2.1 Duplicate License

QUESTION:
I am getting a message "Duplicate License Key Found", what should I do?

ANSWER:
The error that you are getting indicates that multiple copies of Rave are being run on the network at the same time with the same serial number. This is not allowed. If you have more than one developer that needs to use Rave then you will need to purchase additional copies of Rave and reinstall each copy of Rave with the individual serial numbers. That would allow multiple copies of Rave to be run at the same time. Having the demo installed first would not cause this, as the BEX install would over write all of the demo files.

2.2.2 Treeview Missing

QUESTION:
I can not see the Project Treeview on the right side of the Rave designer anymore. How can I get it back?

ANSWER:
Rename or delete your RAVE.INI. The next time you start RAVE, it will create a new INI file with default-settings. You might want to look at your preference settings before you do this. This has been fixed on the latest version of Rave. Earlier versions, allowed a "0" width of the side panels which made it impossible to resize it if it every got set to zero width.

2.2.3 Unable to Gain Control

QUESTION:
I am getting a "Unable to gain control of Rave Data Communication System" message. What should I do?

OR

I need to run multiple instances of my application, each of which uses Rave. When I try and do this I get errors. What do I need to do to enable this to work?

ANSWER:
Direct DataViews are not designed to be thread safe. Therefore, if you are running multiple instances of an application using Direct Data Views when using Rave you are very likely to get error messages about gaining control of Rave. *Your best solution for threading applications is to use [Driver DataViews], which are designed to be [threadsafe].* Driver DataViews require Rave BEX.

Solution to try if you must use Direct DataViews
The most likely reason errors are being generated is because the separate applications' Direct DataView groups are colliding. If you need to run multiple programs with separate reporting direct data view groups, set RPDefine.DataID to a unique value and the separate apps will not collide. Setting the DataID value needs to be done prior to executing any reports. It is recommended that you do this in the OnCreate event of your main form and use something that will be unique to the application.

Delphi Example:
```
procedure TForm1.FormCreate(Sender: TObject);
begin
  RPDefine.DataID := IntToStr(HInstance);
end;
```

C++Builder Example:
```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Rpdefine::DataID = IntToStr(HInstance);
}
```

Multiple Application Instances, Update

There have been a number of reports of problems when using the code provided above. First, it is critical that the DataID variable be set *before* any Rave connections have been created. Also, tests indicate that, for reasons not clear at this time, using the main form HInstance variable does not necessarily provide a value different for each instance of the application.

The currently recommended way of setting the DataID variable is to place the code in the project's .DPR file and to use the application handle:

Delphi Example:
```
begin
  RPDefine.DataID := IntToStr(Application.Handle);
  Application.Initialize;
  {etc...}
end;
```

You will need to add RPDefine and Sysutils to the DPR file's USES clause.

2.2.4 Windows Vista

QUESTION:
I get an error message the Rave.EXE has stopped when I try and run it on a Windows Vista system. What should I do?

OR

I get a message when I try to launch Rave from a RvProject Component that says "Cannot find Rave.INI". What do I need to do to enable this to work?

ANSWER:
Windows Vista security rules have changed and now are more restrictive on what programs can and cannot do in the "Program Files" directory tree. Basically, older versions of Rave did a read and write to a Rave.INI file. The bundled editions (BE) versions were located in the same directory structure as Delphi which defaulted to "Program Files". Those reads and writes are not allowed under the Windows Vista operating system.

SOLUTION

for Rave Reports version 8 (Developer or Architect)
You should not have a problem with Windows Vista if you are using version 8 or later of Rave Reports.

for Rave BE (Bundled Editions) versions 7 or earlier
If you are using one of the Borland / Bundled Editions (BE) of Rave, then Rave Reports will be installed in the same directory structure as the version of Delphi was. In general, this will be the "Program Files" area. To attempt to resolve the Windows Vista security restrictions, you need to make sure that your "Group" or "User Name" has the "Full Control" permissions set to "Allow".

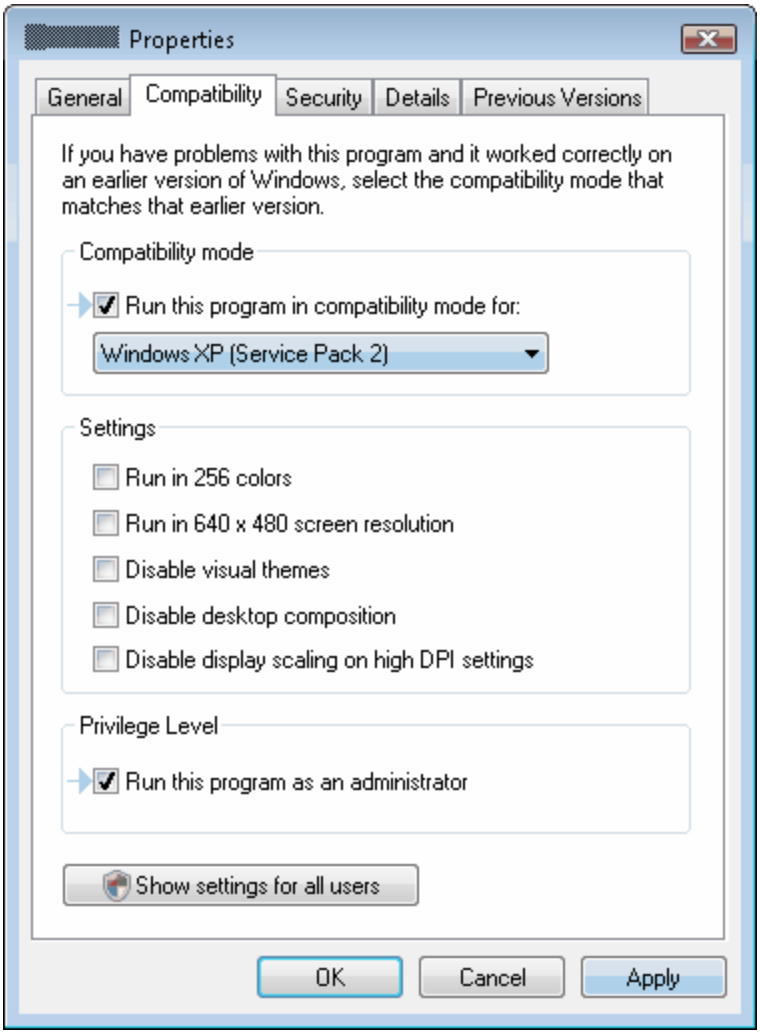
1. Go to Program Files directory where you install the CodeGear / Borland product
   Program File\CodeGear\RAD Studio\5.0
2. Highlight RaveReports
3. Right Click on RaveReports and Open properties
4. Click the Security Tab
5. Select your own user name from the list displayed
6. Click the Edit button
7. Select your own uses name again
8. Go to the "Permissions for Users" dialogue box
9. Click the 'Full Control' Allow box
10. Click Apply
11. accept all the prompts

for Rave Reports BEX (Borland Edition eXtended) versions 7 or earlier
If you are using one of the earlier eXtended editions (BEX) of Rave Reports, then we would suggest that you do NOT install Rave Reports in the "Program Files" area. Instead, we would suggest that you install it off the root directory, something like C:\Rave7.

If you are still getting the User Account Control (UAC) security screen and want to ***try*** and prevent it from popping up each time you start the program - then read the following. After reading, if you are comfortable with making those kind of changes, then proceed with caution.

In a single sentence, you need to get to the property screen of the Rave Reports program and set the compatibility mode options to something that the Windows Vista security module can tolerate. An example of the Windows Vista program property screen is shown after this part. The steps to accomplish this are:

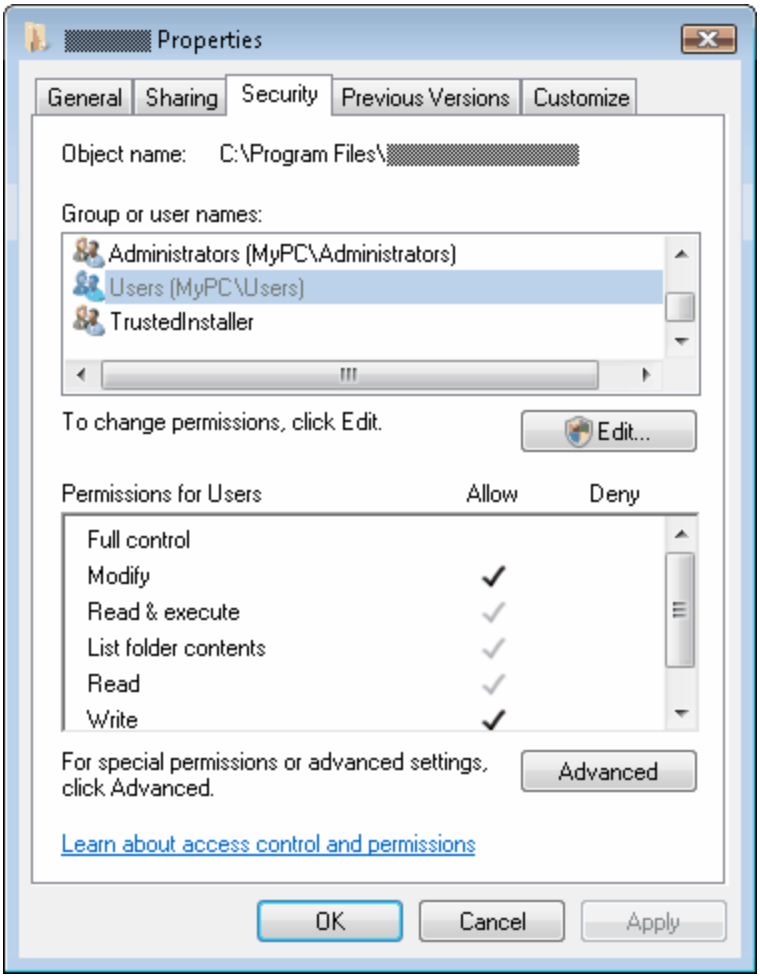
1) Go to the folder where the program is located. This is the folder that you used when you installed this program (for example: "C:\RaveBEX705" ".
2) Find the program file (for example: RAVE.EXE)
3) Right Click on that program EXE and select the properties option (toward the bottom of the dialog screen)
4) When the Properties Dialog opens, go to the "Compatibility" tab
   a) Set the Compatibility Mode
      1) Check the box for "Run this program in compatibility mode for"
      2) Use the drop box to set the mode to "Windows XP"
   b) In the "Privilege Level" area check the box for "Run this program as Administrator"
5) Remember to click on the "Apply" button when you are done.
6) Close the property screen.



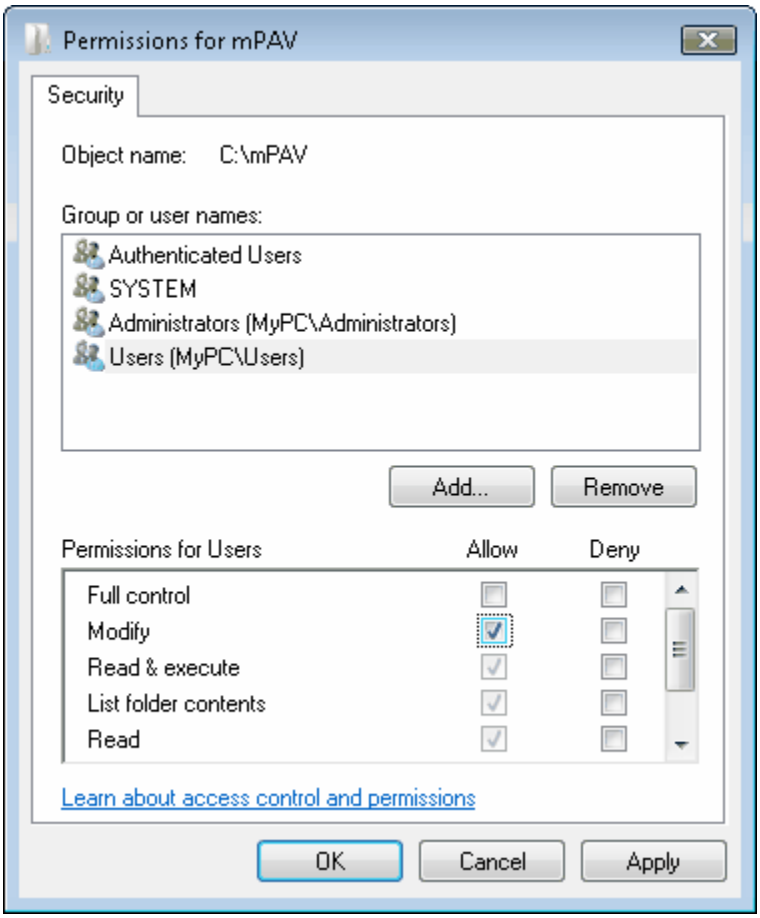
If you are still having UAC issues or must install the program in the "Program Files" folder, then another suggestion is to change the "Modify" permissions for that program folder. The steps are similar to above:

1) Go to the folder where the program is located. This is the folder that you used when you installed this program (for example: "C:\RaveBEX705").
2) Right Click on the folder that contains the program executable file (for example: C:\RaveBEX705)
3) Select the properties option (toward the bottom of the dialog screen)

4) When the Properties Dialog opens, go to the "Security" tab and find the "Group or user Names" box
   a) Click on the "Edit" button to change permissions



   b) Find and Select the "Users" group
   c) Go to the "Permissions for Users" box
   d) On the "Full Control" permission line - click the check box for "Allow"
   e) On the "Modify" permission line - click the check box for "Allow"
   f) Click on the "Apply" button when you are done.
   g) Close the edit permissions dialog screen
5) Click on the "OK" button to close the property screen.



2.3 Fonts

Questions about selecting or using Fonts.

2.3.1 Dynamically Changing Fonts

QUESTION
How can I dynamically change a Font color, name, size or style?

SOLUTION
Some report designs can have one or more fields that would be enhanced if the you could control the font attributes of those fields to emphasis them when a certain condition occurred. There are a couple of ways to do this in Rave Reports.

1) One choice would be to "mirror" different sections. Each section would probably look very similar and would contain Text or DataText components with the font attributes you want. These sections to be mirrored should be on a Global Page. One section could contain all your components with font attributes set as "Arial" "10 points" "Normal" "Black". Another section would also contain the same components with the font attributes set as "Arial" "10 points" "Bold" "Red".

2) Another choice is to use the FontMaster component and a Rave event . to control which FontMaster is used. First, drop several FontMaster components on a Global Page. Set one FontMaster name to "FontArial10NormalBlack" with the font attributes set as "Arial" "10 points" "Normal" "Black". Then set the other FontMaster name to "FontArial10BoldRed" with the font attributes set as "Arial" "10 points" "Bold" "Red". The following is an example of an event that will change the FontMirror property of a DataText component to point to the FontMaster you want based upon the contents of a DataField. This could be used in an OnBeforePrint or OnGetText event of your Text or DataText component(s).

```
{ Event for DataTextFishName.OnGetText }
function DataTextFishName_OnGetText(Self: TRaveDataText; var Value: string);
begin
   if UpperCase(BioLifeDVCategory.AsString) = 'SHARK' then
      Self.FontMirror := FontArial10BoldRed;
   else
      Self.FontMirror := FontArial10NormalBlack;
   end if;
end OnGetText;
```

2.3.2 How to Bold text

QUESTION:
I want output like **bold[Field1]** + " " + *italic[Field2]* side by side so that Field2 is moved based on the length of Field1.

ANSWER:
With the latest version of Rave BEX, you can accomplish this by dropping down a section component and then switch over to the event editor. There you define an OnPrint event and writing the following code:

```
Report.Bold := true;
Report.Print(DriverDataView1Field1.AsString);
Report.Bold := false;
Report.Print(' ');
Report.Italic := true;
Report.Print(DriverDataView1Field2.AsString);
Report.Italic := false;
```

2.3.3 Missing Text

QUESTION:
When I save a report in RTF format, some of the text appears to be chopped off. What should I do?

OR

When I preview my report on screen it does not look the same as the printed output.

ANSWER:
This problem is usually caused by using a font which is not true-type. For example, "MS Sans Serif" is not a True-Type font and often causes a problem. Fonts which are not true-type do not scale very well and will look quite different when viewed on screen as compared with how they will look when printed. For this reason, we do not recommend that you use any fonts that are not true-type fonts. Usually, switching your font(s) to a true-type will solve the problem that you are having.

2.3.4 Overlapping Words

QUESTION:
During preview, I have a problem where the spaces are not correct between words. The words often overlap each other.

ANSWER:
Rave Reports main focus is to output reports as accurately as possible to paper. A preview problem occurs in that most printers have very high resolutions when compared to your display screen. Printers are 600 dpi or better while screens are 96 dpi. Please note that 96 does NOT divide into 600 nicely. That means that the preview scaling factors are often factional values and can cause problems when trying to simulate the printed output on your screen. This can result in the overlapping text problems that you have noticed during the report preview process. To minimize this preview problem on your display, try one or both of the following:

1) Make sure you only use True-Type fonts as they scale better. "MS Sans Serif" is NOT a True-Type font.

2) Another method to improve the preview is to drop down a TRvSystem component and set it as the engine property of your TRvProject component. Then set the RvSystem property SystemFiler.AccuracyMethod to amAppearance and see if that works better for your display.

2.3.5 Unicode Support

QUESTION:
I can not make any of my fonts work. How do I solve that?

ANSWER:
The Windows version of Rave Reports does not support composite or Unicode fonts due to some core issues with ALL Win32 versions of Delphi. Rave does have limited support for MBCS (Multi-Byte Character Sets) basically because of the nature of the way the characters are stored in Delphi Win32. There is limited supported for Japanese and Chinese. We have worked (and will continue to do so) with each of our Render components (HTML, PDF and RTF) to be as flexible as possible with the Windows versions.

The .NET versions of Rave Reports does support composite and Unicode fonts. Rave Reports for .NET support for "symbolic languages" like Japanese, Chinese and Korean is much better since all strings are stored internally as two byte characters.

2.4 Miscellaneous

Miscellaneous or general use questions about Rave Reports.

2.4.1 How to Deploy

QUESTION:
How can I deploy my Rave-Reports?

ANSWER:
There are several ways to store / deliver your reports:

1. Use the external RAV-file. Requires that you define it in the TRaveProject-component.
```
RvProject1.ProjectFile := ExtractFilePath(Application.ExeName) +
'Project1.rav';
RvProject1.Open;
```

Be alert that if you select the RAV-file in the Delphi-IDE, then Delphi will store the complete path information including the developer folder. Normally, the User will not have the same folder structure and that will result in an error that your application can not find the RAV file.

2. Store the RAV file directly in your EXE application (only one file to distribute).

Go to your RvProject component and select the "StoreRav" property. Click on the ellipse (3dots) and point to your RAV file. Remember to use the "Load" and "Save" buttons whenever you have made changes to the RAV file.

NOTE: with Rave 5.0.4 or earlier, the form that contains the RvProject must NOT be set to "text" format. To change it, go to the form that contains the RvProject-component. Right-click on the form and be sure that the Delphi DFM format is not set as "Text Format".

3. Store the Rave-file inside a Resource-file, BLOB-Field or in a DLL like the following:

```
Var
  TP_res : TResourceStream;  (or all the other ways)

Begin
  TP_res := TResourceStream.create(Hinstance,'mein_report','RAVE');
  If (res = NIL) Then Begin
    MessageDlg("There is a problem, please call the developer g>…..',
mtError, [mbok],0);
    exit;
  End;
  RvProject1.New;
  RvProject1.LoadFromStream(TP_res);
  RvProject1.SelectReport('Invoice',false);
  RvProject1.Execute;      // or ExecuteReport….
  RvProject1.Close;
  TP_res.Free;              // the pre-.net Garbage Collector <g>
End;
```

2.4.2 More Documentation

QUESTION:
Do you have more documentation or tutorial type information?

ANSWER:
The following is a list of available documentation or tutorial type information.

1) Be sure to look at the samples installed on your system, usually C:\Rave\Demos. If you are using the original bundled version of Rave included with Delphi 7, be sure that you upgrade to the 5.0.8 version (available on CodeGear's / Borland Delphi 7 registered users page) which fixes several bugs including a missing file which Borland left out of the first release.

2) There are several PDF documents installed on your system, usually C:\Rave\Docs. Again, if you are using the bundled version of Rave with Delphi 7, then these docs were included on a separate CD since they ran out of space.

3) Documentation (current and older versions) can be found at:
http://www.nevrona.com/rave/download.html.

4) There are many tips and tricks on the Nevrona website at:
http://www.nevrona.com/rave/tips.shtml

5) Scripting demos for BEX users are in the \Rave\Demos\Scripting folder. Scripting documentation is included in the RANT section of this document or can be downloaded from:
http://www.nevrona.com/files/rant.txt . You can see what methods and properties are available for the Rave classes by looking in the Rave\Source*.RVS (Rave Scripting source libraries).

6) There are several tutorial type articles on our Add-Ons / Lessons page at
http://www.nevrona.com/rave/addons.shtml . Look for:

"Reporting Solutions in Delphi" webinar replay
"First Steps for Rave with Delphi" PDF article
"Introduction to Rave" I through IV articles
"First Steps for Rave Reporting Server" PDF article
"Rave goes .NET" PDF article
"Rave Overview Presentation" from ADUG symposium

2.4.3 Shell Components

QUESTION:
I have heard about tLabelShell but I can not find it. Where are the shell components?

ANSWER:
The shell components are only provided in the BEX version of Rave Reports. The BEX version also includes source code and is backwards compatible with prior versions of Rave (ReportPrinter Pro).

The shell components are not included in the BE version of Rave Reports which shipped with Delphi. The BE version is NOT backwards compatible with prior versions of Rave (ReportPrinter Pro).

2.4.4 SQL parameter

QUESTION:
How do I pass a parameter to a SQL query?

ANSWER:
To pass a parameter to a SQL statement you need to do the following::

1. Set the parameter you want to pass in your Delphi code.

   ```
   RvProject1.SetParam('ParamName','YourValue');
   ```

2. Set the QueryParam property of the Query Designer with something like:

   ```
   SQLParam1=Param.ParamName
   ```

   please note NO SPACES around equal sign

3. now build your SQL statement like the following:

   if it is a numeric compare
   ```
   Select * from table where Field=:SQLParam1
   ```

   if it is a string compare
   ```
   Select * from table where Field=':SQLParam1'
   ```

   please note NO SPACES around equal sign

2.4.5 ThreadSafe application

QUESTION:
How can I create a Threadsafe application using Rave?

ANSWER:
Driver Dataviews (DVR data connection icon) were designed specifically to be ThreadSafe and can be used in threading type applications. There are several of these compiled drivers installed with the latest version of Rave Reports in the DataLinks folders.

Direct Dataviews (VIEW data connection icon) are not ThreadSafe, and should not be used in threading type applications.

You can download a variety of Driver Dataviews from the Rave Add-Ons page. Please make sure to check your Rave version and only download a driver if it matches your system. The add-ons page is located at:
http://www.nevrona.com/rave/addons.shtml

If you do not find the specific driver you need, you can create your own. Download one of other drivers and look at the source code and readme.txt. That can be used as a pattern to create your own Driver DataView.

DataLink Driver

2.5 Printer

Questions about Printing or Printer issues.

2.5.1 Report Destination

QUESTION:
How do I determine whether the user previewed or printed a report?

ANSWER:
First, we have to caution you that selecting "print" does not mean the user actually got a complete print job. The printer could have had a paper jam, gone off-line etc. It is also important to note that Rave simply communicates with the Windows Printer API and NOT with the printer. So when you select "Print" in Rave - ALL that means is that the print job was sent to the Printer API. The spooler probably got the job but it is unknown if the printer or user got it.

However, you can get a clue on what the user selected by using the OnAfterPrint and OnAfterPreviewPrint events on the TRvSystem component. In those events, put some code to look at the return value of the ReportDest property. The return value can be used to determine which selection the user has chosen - File, Preview or Printer. This ReportDest return value is set after the user exits the setup form.

2.5.2 Printer Bypass Setup

QUESTION:
How can I totally bypass the setup dialog and go directly to the printer.

ANSWER:
With the RAVE-IDE you can set your desires with Edit-> Options-> Preferences menu option. Then on the "Preferences" dialog - select the "Printing" option on the left side. Set the "Output Options" and the "Print Definition" sections to your output requirements.

For your application's, you must set the RvProject component engine property to a RvSystem component. If you do not already have a RvSystem available, then drop one on the same form as your RvProject. Once you have set the engine property, then go to the RvSystem-component and you have a lot of properties that you can set, as needed. Some of the more popular ones are:

To disable the Output Dialog from showing:

```
RvSystem1.SystemSetups := RvSystem1.SystemSetups - [ssAllowSetup];
```

To change the default destination of the output (File, Preview or Printer):

```
RvSystem1.DefaultDest := rdPrinter; (sets default as printer)
```

To set the Preview Windows for maximized size

```
RvSystem1.SystemPreview.FormState := wsMaximized
```

2.5.3 Printer Changing

QUESTION:

How do I change the active printer when I am doing a report?

ANSWER:

To programmatically change the printer in a code based report do the following:

1) Place "RpDevice" in your Uses clause

2) Get the desired printer name, for example:

RpDev.PrinterSetupDialog;

sPrinterName := RpDev.Device;

save this in INI etc for later use in your program

3) Set the printer with

RpDev.SelectPrinter(sPrinterName, False);

The second parameter indicates whether you need an exact match or not on the printer name.

2.5.4 Printer Problems

QUESTION:
MS Word works with my printer but Rave Reports does not. What is the problem with Rave?

ANSWER:
"It works with MS Word" is a very common user statement <g>. However, all that really has been validated with Word is that the computer to printer connections are working and that the printer driver is functioning at the level needed by MS Word. Most reporting programs can and will stress a printer driver in ways that you would not normally do with Word.

For example, there was a color HP DeskJet printer driver that would not print Bold pale yellow "Arial" text at an angle. Make the text any other color or different font or turn off the Bold or make the text horizontal, then all would be fine. Once the exact problem was identified, even Word had a problem. After working with HP, they acknowledged a problem and updated their driver which fixed the issue. It took about 8 months. In the mean time, the user had changed the color to silver.

2.5.5 Printer Model Issues

QUESTION:
I am having a problem printing to a particular printer. The preview looks OK.

ANSWER:
The first thing to do is make sure you are using the latest version of Rave Reports. We could have printer problems, but it should be with all printers not just one model. For example, at one time we had a problem reading / setting the collate property in the latest version of Windows. If you do not use the collate feature or your printer did not have a collate setting, then it would not have been an issue.

The next response to printer issues is that we would suggest that you make sure you have the latest printer driver from the vendor's web site. Rave Reports communicates to the Printer by using the Windows API. The Windows API communicates with the Printer Driver which communicates with the printer. What this means is that Rave is pretty much insulated from printer specific issues.

2.6 Tips and Tricks

Some of the more popular "Tips and Tricks" are listed here. Be sure to visit the Nevrona web site at: http://www.nevrona.com/rave/tips.shtml for many more.

2.6.1 Getting & Setting Parameters

QUESTION
How would I set / get parameters while using the Rave Language?

SOLUTION
You must first define your parameter(s) in Rave to be able to set or get parameter(s) while in the Rave Event Editor. Using the parameter "MyParam" as an example, the following code would then set and then use the value from the specified parameter. Remember that parameters must always be a "string" value. This code was tested from the OnGetText event of a text component.

```
// To SET a parameter in Delphi (before running a report)
RaveProject.SetParam('MyParam', 'My Param Value');

// To GET your parameter within an event
Value := RaveProject.GetParam('MyParam');

// To SET your parameter within an event
RaveProject.SetParam('MyParam', 'NewValueAsString');

// Sample event that "increments" a parameter named "LineCount"
{ Event for DataTextCount1.OnGetText }
function DataTextCount1_OnGetText(Self: TRaveDataText; var Value: string);
var
   iCount: Integer;
   sCount: String;
begin
   iCount := StrToInt(RaveProject.GetParam('LineCount'));
   sCount := IntToStr(iCount + 1);
   Value := sCount;
   RaveProject.SetParam('LineCount', sCount);
end OnGetText;
```

2.6.2 Hiding Bands or Components

QUESTION:
How do I hide a band based upon a DataText value?

SOLUTION:
The solution depends upon what version of Rave Reports you are using.

1) The recommended method is to use the "Visible" property which was added to Rave Reports (5.11 or later) on almost all of the components, including the band components. The description of this property is that it "*determines whether the component will be printed or not. Typically, set through scripting in the OnBeforePrint or OnBeforeReport events*". You do this by putting a script that sets the "visible" property in a parent of the component you want to control (or in another component). Do not put a script that changes visibility in the same component you want want to control as it will run until it is "false" but will not run after that as both the component and event "are no longer visible". This example shows a header band event that is controlling the visibility of a row footer band.

```
{ Event for HeaderBand.OnBeforePrint }
function Header_OnBeforePrint(Self: TRaveBand);
begin
  if DvInventoryOnHand.AsInteger > 0 then
    BandRowFooter.visible := True;
  elseif DvInventoryOnHand.AsInteger = 0 then
    BandRowFooter.visible := False;
  else
    BandRowFooter.visible := True;
  end if;
end OnBeforePrint;
```

2) For versions of Rave Reports that do not have the “Visible“ property, conditional hiding of bands can be accomplished through the use of data mirrors. To do this, drop down a couple of sections on a global page or second report page and place all of your components that need to be printed in the first section. The second section is left blank and its height is set to zero. Then on your main report page you drop down a DataMirrorSection component inside your band and set it up so that it either mirrors the empty section or the section with your components on it. Set the DataMirrorSection's height to zero and then based on a parameter or field in your database you mirror the appropriate section.

However, be aware that mirroring regions and bands can lead to “Page State” conflicts with unpredictable results.

2.6.3 Render NDR to HTML

QUESTION
I have previously saved NDR files and I want to save them to HTML files. How would I do that?

SOLUTION
The following code starts with having a TRvRenderHTML component on your form named RenderHTML.

NOTE:
The starting file name given for the resulting html is 'test.html'. The actual files generated will reflect the name plus the page number. So for this example, the resulting file name will be 'test1.html', 'test2.html' and so forth.

Delphi:
```
procedure TForm1.Button1Click(Sender: TObject);
var
  NDRStream: TMemoryStream;
begin
  NDRStream := TMemoryStream.Create;
  try
    NDRStream.LoadFromFile('test.ndr');
    RenderHTML.PrintRender(NDRStream, 'test.html');
  finally
    NDRStream.Free;
  end; { tryf }
  ShowMessage('NDR Converted');
end;
```

C++Builder:
```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemoryStream* NDRStream;
  NDRStream = new TMemoryStream;
  try {
    NDRStream->LoadFromFile("test.ndr");
    RenderPDF->PrintRender(NDRStream, "test.html");
  }// try
  __finally {
    delete NDRStream;
  }// __finally
  ShowMessage("NDR Converted");
}
```

2.6.4 Render NDR to PDF

QUESTION:
I have previously saved NDR files and I want to save them to PDF files. How would I do that?

SOLUTION:
The following code starts with having a TRvRenderPDF component on your form named RenderPDF.

Delphi:
```
procedure TForm1.Button1Click(Sender: TObject);
var
  NDRStream: TMemoryStream;
begin
  NDRStream := TMemoryStream.Create;
  try
    NDRStream.LoadFromFile('test.ndr');
    RenderPDF.PrintRender(NDRStream, 'test.pdf');
  finally
    NDRStream.Free;
  end; { tryf }
  ShowMessage('NDR Converted');
end;
```

C++Builder:
```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemoryStream* NDRStream;
  NDRStream = new TMemoryStream;
  try {
    NDRStream->LoadFromFile("test.ndr");
    RenderPDF->PrintRender(NDRStream, "test.pdf");
  }// try
  __finally {
    delete NDRStream;
  }// __finally
  ShowMessage("NDR Converted");
}
```

2.6.5 Render NDR with NO preview

QUESTION:
I want to save my report to a NDR file without any setup dialog or preview screen. How would I do that?

SOLUTION:
The following code requires that you have a RvNDRWriter, RvProject and RvSystem components on your form.

Delphi:
```
Procedure TForm1.Button1Click(Sender: TObject);
Begin
  // Set your RvProject engine property to NDRWriter
  RvProject1.Engine := RvNDRWriter1;
  RvNDRWriter1.FileName := 'Test1.NDR';
  RvProject1.ExecuteReport('YourReportName');
  // re-set the RvProject engine property here
  RvProject1.Engine := RvSystem1;
```

```
  ShowMessage('NDR file created');
End;
```

C++Builder:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Set your RvProject engine property to NDRWriter
  RvProject1->Engine = RvNDRWriter1;
  RvNDRWriter1->FileName = "Test1.NDR";
  RvProject1->ExecuteReport("YourReportName");
  // re-set the RvProject engine property here
  RvProject1->Engine = RvSystem1;
  ShowMessage("NDR file created");
}
```

2.6.6 Render NDR to Preview

QUESTION:

I have previously saved NDR files and I want to preview them again. How would I do that?

SOLUTION:

The following code starts with having a RvSystem component on your form named RvSystem1.

Be advised that both "DoNativeOutput" and "RenderObject" are not required and need not be set.

Delphi:

```
Procedure TForm1.Button1Click(Sender: TObject);
Begin
  RvSystem1.DefaultDest := rdPreview;
  RvSystem1.SystemFiler.FileName := 'Test.NDR';
  // RvSystem1.SystemSetups := RvSystem1.SystemSetups - [ssAllowSetup];
  RvSystem1.Execute;
  ShowMessage('Preview created');
End;
```

C++Builder:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RvSystem1->DefaultDest = rdPreview;
  RvSystem1->SystemFiler->FileName = "Test.NDR";
  // RvSystem1->SystemSetups = RvSystem1->SystemSetups - [ssAllowSetup];
  RvSystem1->Execute;
  ShowMessage("Preview created");
}
```

2.6.7 Render NDR to PRN

QUESTION:
I have previously saved NDR files and I want to save them to printer (PRN) files. How would I do that?

SOLUTION:
This solution applies to Rave Reports version 5.1.2 or later. The following code starts with having a RvSystem component on your form named RvSystem1.

It is important to note that the current printer (default) will be used to generate the printer file. This means that the generated file can ONLY be sent to that printer since the contents of printer files are specific to the printer they were generated for.

Also, the copy binary command on some operating systems may have problems if you use the PRN extension.

COPY filename.ext PRN: /B

Be advised that both "DoNativeOutput" and "RenderObject" are not required for this NDR file conversion and need not be set.

Delphi:

```
Procedure TForm1.Button1Click(Sender: TObject);
Begin
  RvSystem1.DefaultDest := rdPrinter;
  RvSystem1.SystemFiler.FileName := 'Test.NDR';
  RvSystem1.SystemOptions := RvSystem1.SystemOptions + [soNoGenerate];
  RvSystem1.SystemSetups := RvSystem1.SystemSetups - [ssAllowSetup];
  RvSystem1.SystemFiler.StreamMode := smFile;
  RvSystem1.OutputFileName := 'test1.ptr';
  RvSystem1.Execute;
  ShowMessage('Printer File created');
End;
```

C++Builder:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RvSystem1->DefaultDest = rdPrinter;
  RvSystem1->SystemFiler->FileName = "Test.NDR";
  RvSystem1->SystemOptions = RvSystem1->SystemOptions + [soNoGenerate];
  RvSystem1->SystemSetups = RvSystem1->SystemSetups - [ssAllowSetup];
  RvSystem1->SystemFiler->StreamMode = smFile;
  RvSystem1->OutputFileName = "test1.ptr";
  RvSystem1->Execute;
  ShowMessage("Printer File created");
}
```

2.6.8 Render to PDF with NO Setup

QUESTION:
How do I go directly to a PDF file without displaying the setup dialog box?

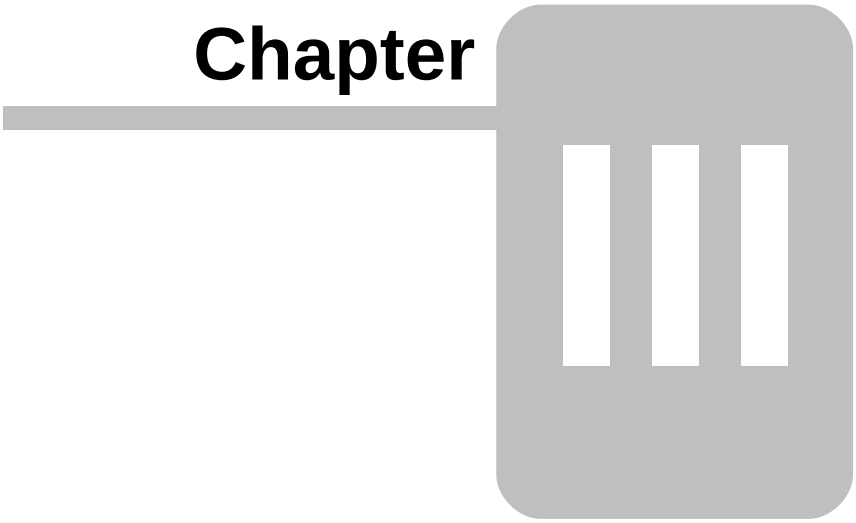
SOLUTION:
Starting with Rave 4, you have the ability to save reports in a PDF file. To accomplish this without any user intervention via the setup dialog, drop a RvRenderPDF component on your form along with your RvSystem component. Make sure the RvRenderPDF "Active" property is set to true. Then execute the following code in your Button / Menu OnClick event:

Delphi Example:
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  RvSystem1.DefaultDest := rdFile;
  RvSystem1.DoNativeOutput := false;
  RvSystem1.RenderObject := RvRenderPDF1;
  RvSystem1.OutputFileName := 'test1.pdf';
  RvSystem1.SystemSetups := RvSystem1.SystemSetups - [ssAllowSetup];
  RvSystem1.Execute;
end;
```

C++Builder Example
```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RvSystem1->DefaultDest = rdFile;
  RvSystem1->DoNativeOutput = false;
  RvSystem1->RenderObject = RvRenderPDF1;
  RvSystem1->OutputFileName = "test.pdf";
  RvSystem1->SystemSetups = RvSystem1->SystemSetups >> ssAllowSetup;
  RvSystem1->Execute();
}
```

DataLink Driver .

Chapter



3 DataLink Driver .

DataLink Drivers are DLL files with an RVD extension. By using Rave's DataLink architecture, drivers entail very little code and are quite quick to put together. There are several DataLink drivers already done and are available on the Nevrona web site Add-Ons page at: http://www.nevrona.com/rave/addons.shtml . Please be sure to download the one that applies to your version of Rave Reports.

3.1 Creating

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

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

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

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

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

3.2 Example DPR

DataLinkSample.dpr

```
library DataLinkSample;

uses
  RvDLBase, RvDLSample, RvDLSampleCfg;

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

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

begin
end.
```

3.3 Example PAS

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

RvDLSample.pas

```
unit RvDLSample;

interface

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

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

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

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

implementation
```

{ TDLSampleDriver }

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

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

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

```
{ TDLSampleConnection }

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

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

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

{ TDLSampleResultSet }

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

initialization
  RegisterDriverClass(TDLSampleDriver);
end.
```

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

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

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

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

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

3.4 RvDLSampleCfg

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

```
unit RvDLSampleCfg;

interface

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

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

implementation

{$R *.dfm}

uses
  RvDLCommon, RvDLSample;

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

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

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

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

Format Codes

Chapter

4 Format Codes

The DisplayFormat property allows you to format the value given using a format string. The following format specifiers are supported in the format string:

4.1 Alphanumeric Items

The following is a list of the different alphanumeric format codes and what they accomplish for each output type.

Examples:

Format String	123456.78	-123.0	0.5	-0.0
#,##0.00	123,456.78	-123.00	0.50	0.00
#.#	123456.8	-123	0.5	0
$,0.00	$123,456.78	$-123.00	$0.50	$0.00
0.00;(0.00);'-'	123456.78	(123.00)	0.50	-----

Specifier	Represents
0	Digit place holder. If value being formatted has a digit where the '0' appears, then the digit is copied to the output string. Otherwise, a '0' is in the output string.
#	Digit place holder. If value being formatted has a digit where the '#' appears, then the digit is copied to the output string. Otherwise, nothing appears in that position.
.	Decimal point. The first '.' character in the format string determines the location of the decimal separator in the formatted value. The actual character used as a the decimal separator in the output string is determined by the Number Format of the International section in the Windows Control Panel.
,	Thousand separator. If format string contains a ',' characters, the output will have thousand separators inserted between each group of three digits to left of decimal point. The actual character used as a thousand separator in the output is determined by the Number Format of the International section in the Windows Control Panel.
E+	Scientific notation. If any of the strings 'E+', 'E-', 'e+', or 'e-' are contained in the format string, the number is formatted using scientific notation. A group of up to four '0' characters can immediately follow the 'E+', 'E-', 'e+', or 'e-' to determine the minimum number of digits in the exponent. The 'E+' and 'e+' formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The 'E-' and 'e-' formats output a sign character only for negative exponents.
'**xx'/"xx"**	Characters enclosed in single or double quotes are output as-is, and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost '0' before the decimal point in the format string and the rightmost '0' after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholder's ('0' or '#') to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholder's to the left of the '.' character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative, and zero values, the format string can contain between one and three sections separated by semicolons.

One section:	The format string applies to all values.
Two sections:	The first section applies to positive values and zeros, and the second section applies to negative values.
Three sections:	The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If the section for negative values or the section for zero values is empty, that is if there is nothing between the semicolons that delimit the section, the section for positive values is used instead.

If the section for positive values is empty, or if the entire format string is empty, the value is formatted using general floating-point formatting with 15 significant digits.

4.2 Date Time Items

Items that are either a date or time field can use the following format codes. The format specifiers are not case sensitive. If the format parameter is blank then the value is formatted as if a 'c' specifier had been given. The following format specifiers are supported:

Examples:

dddd, mmmm d, yyyy => Monday, September 21 2004
d mmm yy => 21 Sep 04

Specifier	Displays
c	Displays date using format given by ShortDateFormat global variable, followed by time using format given by LongTimeFormat global variable. The time is not displayed if fractional part of the DateTime value is zero.
d	Displays the day as a number without a leading zero (1-31).
dd	Displays the day as a number with a leading zero (01-31).
ddd	Displays the day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames global variable.
dddd	Displays the day as a full name (Sunday-Saturday) using the strings given by the LongDayNames global variable.
ddddd	Displays date using format given by the ShortDateFormat global variable.
dddddd	Displays date using format given by the LongDateFormat global variable.
m	Displays month as number without leading zero (1-12). If m specifier immediately follows h or hh specifier, the minute rather than month is displayed.
mm	Displays month as number with leading zero (01-12). If mm specifier immediately follows h or hh specifier, the minute rather than month is displayed.
mmm	Displays the month as an abbreviation (Jan-Dec) using the strings given by the ShortMonthNames global variable.
mmmm	Displays the month as a full name (January-December) using the strings given by the LongMonthNames global variable.
yy	Displays the year as a two-digit number (00-99).
yyyy	Displays the year as a four-digit number (0000-9999).

Specifier	Displays
h	Displays the hour without a leading zero (0-23).
hh	Displays the hour with a leading zero (00-23).

n	Displays the minute without a leading zero (0-59).
nn	Displays the minute with a leading zero (00-59).
s	Displays the second without a leading zero (0-59).
ss	Displays the second with a leading zero (00-59).
t	Displays time using format given by the ShortTimeFormat global variable.
tt	Displays time using format given by the LongTimeFormat global variable.
am/pm	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'am' for any hour before noon, and 'pm' for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'a' for any hour before noon, and 'p' for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	Uses the 12-hour clock for the preceding h or hh specifier, and displays the contents of the TimeAMString global variable for any hour before noon, and the contents of the TimePMString global variable for any hour after noon.
/	Displays date separator character given by DateSeparator global variable.
:	Displays time separator character given by TimeSeparator global variable.
'xx'/"xx"	Characters enclosed in single or double quotes are displayed as-is, and do not affect formatting.

RANT

Chapter

5 RANT

Rave Application iNterface Technology (RANT). This document is provided to enable programmers to interface with the RAVE (Report Authoring Visual Environment) Report Designer at a very low level.

5.1 Introduction

This document assumes that you are familiar with normal Delphi component development. Many of the concepts and principles for interfacing with Rave are very similar to the code interface of Delphi. If you are already familiar with Delphi component development, you should still pay close attention to this document as there are subtle yet important differences that will allow you to take full advantage of RANT. After reading this document, you should be able to create custom Rave components, property editors, component editors, project editors (also know as wizards) and control the Rave environment in a wide variety of ways.

5.1.1 Technical Support

Technical support for items contained within this specification will be limited to Rave specific features. We cannot provide free technical support for programming questions which are common to all Delphi components and classes. We recommend that if you encounter problems with your RANT components or classes, you first refer to the Delphi Component Writer's Guide manual or help file, books on Delphi programming or post a message on a public Delphi forum. An excellent location to post these type of questions is on the Nevrona Forum where you can communicate directly with other Rave users who may have already encountered and solved the same problems you are having. For more instructions on how to join the Nevrona Forum go to http://www.nevrona.com.

5.2 Creating a Rave Component

One of the most powerful capabilities of Rave is the ability to create and dynamically install custom reporting components. Every component shipped with RAVE was created and registered in the same manner as what is available to the RANT developer. The first step to creating a Rave component is to understand the basic architecture of Rave's class library.

5.2.1 Basic Classes

Rave includes the following three basic component classes, TRaveComponent, TRaveControl and TRaveContainerControl are the basic Rave classes that most Rave components descend from. If you are creating a custom Rave component you should either descend from an existing rave component or one of these classes. The relationship of these classes is as follows:

```
TComponent
.|
.+-> TRaveComponent
.     |
.     +-> TRaveControl
.           |
.           +-> TRaveContainerControl
```

TRaveComponent (RVClass.pas) - This is the base class for all Rave components and descends directly from Delphi's TComponent. TRaveComponent classes are non-visual in nature and thus should only be used for items which will not be displayed on the page designer or the report. However, the Rave user will still be able to select a TRaveComponent class and modify its properties through the Project Tree.

TRaveControl (RVClass.pas) - This class descends from TRaveComponent and is the base class for all visual components. All TRaveControl classes have Left, Top, Width, Height, Anchor, Mirror and DisplayOn properties (although they may not be visible on all components). TRaveControl classes display themselves through both the Paint method (for the visual designer) and the Print method (for the printed output).

TRaveContainerControl (RVClass.pas) - This class descends from TRaveControl and adds basic parent-child functionality for components that will contain other components. Examples of TRaveContainerControl classes are TRaveBand and TRaveSection. Note that the parent-child methods and properties are defined at the TRaveComponent level and may be overridden from any class level (TRaveContainerComponent for example), however, if you are creating a visual container component you normally will want to descend from this class.

5.2.2 Other Classes

There are several other important classes used by many of Rave's editor and project management components. You will not normally need to descend your components from these classes, however, you will often interface with components of these class types to perform the tasks that you need to.

```
TComponent
.|
.+-> TRaveComponent
.    |
.    +-> TRaveDataField
.    |
.    +-> TRaveContainerComponent
.        |
.        +-> TRaveProjectItem
.            |
.            +-> TRavePage
.            |
.            +-> TRaveDataView
.            |
.            +-> TRaveReport
.            |
.            +-> TRaveProjectManager
```

TRaveContainerComponent (RVClass.pas) - This class descends from TRaveComponent and adds basic parent-child functionality similar to TRaveContainerControl. This class is used by Rave's TRaveProjectItem class. If you are creating a visual component that will contain other components, you should descend from the TRaveContainerControl class instead.

TRaveProjectItem (RVClass.pas) - This is the base class for all of Rave's project oriented classes including TRavePage, TRaveDataView, TRaveReport and TRaveProjectManager. This class descends from TRaveContainerComponent and adds a basic streaming and active interface.

TRavePage (RVClass.pas) - This class descends from TRaveProjectItem and is the class used to define each page of a report. All components on a page are owned by the page and can be accessed through the pages Components and ComponentCount properties. There are two types of TRavePage components in a Rave project, Report and Global. Report pages belong to a specific report while Global pages are available to the whole system. The Global Boolean property of TRavePage can be used to determine which category a TRavePage class instance falls into.

TRaveDataView (RVData.pas) - This class descends from TRaveProjectItem and is the class used to define a data view. The TRaveDataView class is the owner and parent for all TRaveDataField components belonging to that data view. This class is discussed in further detail in the section "Interfacing with TRaveDataSystem".

TRaveDataField (RVData.pas) - This class descends from TRaveComponent and is the base class for all data field classes (TRaveIntegerField, TRaveStringField, ...). A TRaveDataField component will always be owned and parented by the TRaveDataView class that defines the data view that it is a part of. This class and it's descendent's is where your components will interface with the data that's available through the data views. This class is discussed in further detail in the section "Interfacing with TRaveDataSystem".

TRaveReport (RVProj.pas) - This class descends from TRaveProjectItem and is the class used to define all Rave reports. The TRaveReport class is the owner and parent of all report pages that belong to that report but does not own or parent any global pages even though they may be visible in the page designer for that report. The TRaveReport class is also the main starting point to print reports through the Execute method.

TRaveProjectManager (RVProj.pas) - This class descends from TRaveProjectItem and is the class that manages all components in a report project. TRaveProjectManager is the owner and parent of all TRaveReport, global TRavePage and TRaveDataView classes. This is the main interface to load, save, import to and export from a report project. This is also the main interface that you will use to find a specific report or other component. This class is discussed in further detail in the section "Interfacing with TRaveProjectManager".

5.2.3 Example Component

For this sample we will create a simple text component that presets the font color to blue when it is created. First you'll want to create a unit containing your component class definition. This is very similar to the steps you would follow for creating a normal Delphi component. Here is the source for our text component that overrides the default color:

```
unit NDCsBlue;
interface
uses
  Classes, Graphics, RVClass, RVCsStd;

type
  TNDBlueText = class(TRaveText)
  public
    constructor Create(AOwner: TComponent); override;
  end; { TNDBlueText }

implementation
  constructor TNDBlueText.Create(AOwner: TComponent);
  begin { Create }
    inherited Create(AOwner);
    FFont.Color := clBlue;
  end;  { Create }
end.
```

5.2.3.1 Designer Only Library Files

To save size in compiled applications, certain properties and methods are only available when compiling for the visual designer. This is accomplished with the DESIGNER compiler directive and is used throughout the Rave classes. For example, the Paint method is used by the visual designer to display a component to the screen and so it must be defined within {$IFDEF DESIGNER}...{$ENDIF} directives. The Paint method, however, is not needed by a normal Delphi application and so it's code is not included in the executable.

These designer-only files are located in the C:\Rave\RV directory and can be recompiled with the C:\Rave\Source\Rave??.BAT file. When compiling a package for inclusion in the Rave designer, the file in the C:\Rave\RV directory should be referenced as the Rave library file.

5.2.4 Register Rave Component

After you've created your Rave component, it's time to register it with the Rave system. The first step is to create a global procedure called RaveRegister. The spelling and capitalization must match exactly or it will not work. The RaveRegister procedure will be called when your component package is added to the Rave system to register your components or other editors. Inside of the RaveRegister procedure you will want to call RegisterRaveGroup, RegisterRaveComponents and RegisterRaveProperties.

5.2.4.1 Register Component Group

The RegisterRaveGroup procedure creates a component group for your component and must be called before you register the component class. The first parameter is the name of the group and the second parameter is the full name. Currently in the Rave designer, each group is stored in a dockable toolbar and the full name is used as the caption for the toolbar. The following source is an example of RegisterRaveGroup being called to create a group called ABC with a caption of "ABC Components".

```
RegisterRaveGroup('ND','ND Components');
```

5.2.4.2 Register Component Class

The RegisterRaveComponents procedure actually registers your component class with the Rave system, loads the component icons and creates a component button on the group toolbar. The first parameter is the name (not the full name) of the group that you want to add this component to and the second is an array of component classes that you want to register. All classes that are registered must descend from TRaveComponent or one of it's descendent's. The following source is an example of RegisterRaveComponents being called to create two components, TNDHeaderText and TNDFooterText.

```
RegisterRaveComponents('ND',[TNDHeaderText,TNDFooterText]);
```

5.2.4.3 Register Component Properties

The RegisterRaveProperties procedure registers the published properties for your components and must be called after the component class is registered. With this procedure you can determine which properties are visible to Beginner or Intermediate level users, which properties are Developer only, and which properties are hidden. The first parameter is the component class you are registering the properties for. The second and third parameters are a list of properties that are available to Beginner and Intermediate level properties. By default, all published properties are available to Advanced level users. Any properties registered in the Intermediate properties are available to Intermediate and Advanced level users while any properties registered in the Beginner properties are available to all level users. Developer level properties are set with the fourth parameter and will be visible in the programmer version of the Rave designer and not visible in the end-user version of the Rave designer. The last parameter defines properties that are to be hidden to all users and can be used to hide properties that were visible in a ancestor class. The following source is an example of RegisterRaveProperties being called for the TRaveDataText component.

```
RegisterRaveProperties(TRaveDataText,
 {Beginner}     'DataField;DataView',
 {Intermediate} 'LookupDataView;LookupField;LookupDisplay',
 {Developer}    '',
 {Hidden}       'Text');
```

5.2.4.4 Create Component Icons

Each component has a 24x24 pixel icon that is used to represent it. In the visual designer, this icon is placed on a button that is used to create the component. There is also a 16x16 pixel icon that is used to represent the component in the Project Tree. Lastly, there is a 24x24 pixel icon that each component group (see section Register Component Group) uses to represent itself. In the visual designer, this icon is used on a button on the Toolbar toolbar to show or hide the component group toolbar. The name of the large component icon must match the name of the component class. The small component icon must match the name of the component class plus the characters '16' following it. The component group icon must match the name (not the full name) of the component group. So for our example with TNDBlueText in the ND component group, we would create bitmap resources named TABCBlueText (24x24 pixel component icon), TNDBlueText16 (16x16 pixel component icon) and ND (24x24 pixel component group icon). These bitmap resources will normally be included in a resource file (*.res) with the same name as the unit containing the component class. To register the icons with the unit, include the following source immediately after the implementation keyword in the unit containing the component class.

```
{$IFDEF DESIGNER}
{$R *.RES}
{$ENDIF}
```

5.2.4.5 Putting It Together

Certainly there are plenty of examples of components in the Rave source. There is also a special web page available on the Nevrona Designs web site that contains other useful components (available shortly after release). The following is a complete example of a component and the RaveRegister procedure.

```
unit NDCsBlue;
interface
uses
  RVClass, RVCsStd;

type
  TNDBlueText = class(TRaveText)
  public
    constructor Create(AOwner: TComponent); override;
  end; { TNDBlueText }

  procedure RaveRegister;

implementation

{$IFDEF DESIGNER}
{$R *.RES}
{$ENDIF}

  procedure RaveRegister;

  begin { RaveRegister }
    RegisterRaveGroup('ND','ND Components');

    RegisterRaveComponents('ND',[TNDBlueText]);

    RegisterRaveProperties(TNDBlueText,
     {Beginner}     '',
     {Intermediate} '',
     {Developer}    '',
     {Hidden}       '');
  end;  { RaveRegister }

  constructor TNDBlueText.Create(AOwner: TComponent);

  begin { Create }
    inherited Create(AOwner);
    FFont.Color := clBlue;
  end;  { Create }
end.
```

5.2.5 Distribution

How do you distribute your components and editors? Now that you've created the source for your component, you will need to compile it into a package and other formats depending upon how it is to be used.

5.2.5.1 Unique Names

As with any library file, you must uniquely name your component, editor and files to avoid conflicts with items distributed from other authors. Any Rave components or editors distributed with Rave will be located in files beginning with "RP" or "RV". It is recommended that you select a unique prefix of at least 2 characters that will uniquely identify any files that are yours.

You should also apply this uniqueness when creating component and editor classes. All Rave components or editors distributed with Rave will be named with the prefix "TRave". It is recommended that you select a unique prefix that will uniquely identify any classes that are yours.

The following are guidelines that we have found to be a successful:

- Avoid long file names. Even though Rave is currently only 32-bit, Delphi 1.0 support is planned for the near future and this will cause problems if you are creating a Rave component. Also, certain DOS based compression software and even some network operating systems do not handle long file names very well.

- Select a 2 character unique identifier (e.g. your initials) and use the following naming scheme for the type of item contained in each file (assume ND is the identifier you have chosen and Xxxx is a unique identifier of the item):

* Components: NDCsXxxx.pas
* Wizards (Project Editors): NDWzXxxx.pas
* Property Editor: NDPEXxxx.pas
* Component Editors: NDCEXxxx.pas
* Packages: ND_Xxxxx.dpk

These naming conventions keep file names at or under 8 characters and will help users of your Rave add-on's to understand what is in each file.

5.2.5.2 Library Files

The 3.0 version of the Rave designer requires that all packages to be installed within it, are compiled with Delphi 4.0. For designer only items such as property editors or report wizards, this is typically the only form that will need to be compiled.

For components, however, this is only one of the forms that the source needs to be compiled for. When an application prints a report that contains a custom component, the component code itself needs to be compiled in the application. So, for example, if a Delphi 2.0 user uses a component that you have written, you will need to provide either .PAS or .DCU files to allow them to compile the component code into their application.

5.2.5.3 Creating Packages

Creating Packages for Custom Components or Editors

To compile a package containing custom Rave components or editors you will either need to use Delphi IDE to create a new package (Component| Install Component...) or the recommended method of compiling the package with the command line compiler. If you decide to use the Delphi IDE, you need to make sure that you only include your own custom units.

The recommended method to compile packages for the Rave designer is to create a .DPK file and use the command line compiler. This is a cleaner method than using the IDE package tools since Delphi does several things that are geared more for it's own packages. The following an example of the .DPK file for our example component. Notice that we include any packages containing units required by our components in the Requires clause. The Contains clause lists the units containing the custom components or editors and the Description compiler directive defines the name of the package that will be displayed in the designer's Edit|Preferences...|Packages dialog.

```
package ND_Tools;
{$DESCRIPTION 'ND Tools'}
requires
  VCL40, RVCL30, RVCLS30;
contains
  NDCsBlue;
end.
```

It is also useful to create a batch file to compile your custom package. The following is a sample of the batch file that can be used to compile our example component. This batch file is designed to be used from a directory under C:\Rave such as C:\Rave\Custom. Notice the DESIGNER compiler directive that is added to compile any code that is for the visual designer. Also notice the inclusion of the designer-only units in the RV directory (/U..\RV compiler option).

```
------------------------
Contents of RANTCOMP.BAT
------------------------
  @echo off
  if "%NDD4%"=="" goto endsetup
  REM ********************
  REM Compile Rave Package
  REM ********************
  REM
  REM Usage: RANTCOMP packagename
  REM
  REM Note: The packagename parameter should not include the .dpk extension
  REM
  REM Desc: This batch file will compile a package containing a RAVE add-on
  REM       and place the .BPL in the parent directory (normally C:\Rave).
  REM       It is meant to be run from a directory under Rave such as
  REM       C:\Rave\RANT. The Rave library files must reside in C:\Rave\RV.
  REM
  REM ************************************************
  %NDD4%\bin\dcc32 %1.dpk /b /h /w /LE.. /U..\RV -$d-l-n+p+r-s-t-w- /DDESIGNER
  if errorlevel 1 goto enderror
  del *.dcu >nul
  del %1.dcp >nul
  goto endok
  :enderror
  echo Error!
  goto endok
  :endsetup
  echo You will need to define a variable in your AUTOEXEC.BAT file
  echo called NDD4 that points to your Delphi 4.0 directory and reboot
  echo before using this batch file. Example:
  echo   SET NDD4=C:\Program Files\Borland\Delphi4
  :endok
```

5.2.5.4 Compiled Binaries Directory

There are several binary directories under the C:\Rave main directory. The following is a list of these directories and how to rebuild them.

Directory	Description
C:\Rave\D4	Delphi 4.0 binary files. Run C:\Rave\Source\FullD4.BAT to rebuild.
C:\Rave\D5	Delphi 5.0 binary files. Run C:\Rave\Source\FullD5.BAT to rebuild.
C:\Rave\D6	Delphi 6.0 binary files. Run C:\Rave\Source\FullD6.BAT to rebuild.
C:\Rave\D7	Delphi 7.0 binary files. Run C:\Rave\Source\FullD7.BAT to rebuild.
C:\Rave\C4	C++Builder 4.0 binary files. Run C:\Rave\Source\FullC4.BAT to rebuild.
C:\Rave\C5	C++Builder 5.0 binary files. Run C:\Rave\Source\FullC5.BAT to rebuild.
C:\Rave\C6	C++Builder 6.0 binary files. Run C:\Rave\Source\FullC6.BAT to rebuild.
C:\Rave\RV	Designer-only Rave files. Run C:\Rave\Source\RaveD?.BAT to rebuild. These files should be used instead of the other binary files when compiling a package for

```
inclusion into the Rave designer.
```

5.2.5.5 OverrideProperties

Using TRaveComponent.OverrideProperties to handle changes in properties
---

The OverrideProperties method is an easy way to intercept the normal reading and writing of your components properties. This is normally done to maintain backwards compatibility with existing project files that contain property data in a older format or under a different name. It may also be used to write data that does not fit into the normal types such as graphics files or record structures.

The first thing you will want to do is override the OverrideProperties method in your component class and call the Filer.OverrideProperty method for each property that you want to override.

```
  procedure TMyRaveComponent.OverrideProperties(Filer: TRaveFiler);

  begin { OverrideProperties }
    Filer.OverrideProperty('MyProperty',ReadMyProperty,WriteMyProperty);
  end;  { OverrideProperties }
```

```
You will also need to define a read and write method that will be called when
a property of name, 'MyProperty', is encountered for your component type.
```

```
  procedure TMyRaveComponent.ReadMyProperty(Reader: TRaveReader);

  begin { ReadMyProperty }
  // code to read MyProperty from Reader.StreamHandler
  end;  { ReadMyProperty }

  procedure TMyRaveComponent.WriteMyProperty(Writer: TRaveWriter);

  begin { WriteMyProperty }
  // code to write MyProperty to Writer.StreamHandler
  end;  { WriteMyProperty }
```

There are many different methods of the TRaveWrite and TRaveReader classes (defined in RVCLASS.PAS) and you should become familiar with their operation before continuing.

Here's an example of an override that was needed because a property changed from a TStrings type to a string type:

```
procedure TMyRaveComponent.ReadDescription(Reader: TRaveReader);

var
  TempStrings: TStrings;
  ValueKind: TValueKind;

begin { ReadDescription }
  With Reader do begin
    ValueKind := TValueKind(StreamHelper.ReadByte);
    Case ValueKind of
      vkPropList: begin { Read a TStrings object }
        TempStrings := TStrings.Create;
        try
          ReadProperties(TempStrings);
          Description := TempStrings.Text;
        finally
          TempStrings.Free;
        end; { tryf }
      vkString: begin { Read a normal string }
        Description := StreamHelper.ReadString;
      end;
    end; { case }
  end; { with }
end;  { ReadDescription }

procedure TMyRaveComponent.OverrideProperties(Filer: TRaveFiler);

begin { OverrideProperties }
  Filer.OverrideProperty('Description',ReadDescription,nil);
end;  { OverrideProperties }
```

Notice that we didn't define a write method for this property override. That is because we want the normal output format to be written so there is no need.

Rave also supports Delphi's TPersistent.DefineProperties interface for writing custom data to the stream but this is more limiting than Rave's OverrideProperties interface and should be used if you only want to read and write a custom format for your data.

5.2.5.6 Listener Interface

Using the Rave Listener Interface

A special interface has been built into all Rave components to allow them to communicate with each other. You do this by defining methods in a component class to make it a speaker of a specific conversation and then registering another class as the listener of that class. Note that it is the listener object's responsibility to add itself to the conversation of a speaker object at runtime. An example of where this is used in Rave is the TRaveCalcText component (the listener class) and the TRaveDataBand component (the speaker class). The conversation name is 'CalcNewData' and allows DataBand components to notify CalcText components when to perform calculations. The CalcText components adds itself as a listener to a specific DataBand defined by it's Controller property. Now that this conversation has been defined, it would be possible to create other speakers (as is the case for the component, TRaveCalcController). You could also create other listeners (as is the case for the component, TRaveCalcTotal).

5.2.5.6.1 Create Speaker Class

Creating a Rave speaker class

For this example we'll create a conversation type called ABC and create a simple speaker class. At a minimum, the following items must be defined in the speaker class:

```
TRaveSpeakerClass = class(TRaveControl)
protected
  ABCListenList: TRaveMethodList;

  procedure Changing(OldItem: TRaveComponent;
                     NewItem: TRaveComponent); override;
public
  procedure AddListener(Conversation: string;
                        ListenMethod: TRaveListenEvent); override;
  procedure RemoveListener(Conversation: string;
                           ListenMethod: TRaveListenEvent); override;
  function Habla(Conversation: string): Boolean; override;
end; { TRaveSpeakerClass }
```

1. ABCListenList field: A listener list, of type TRaveMethodList, to keep track of all listeners. There needs to be a separate listener list for each conversation that the speaker class can speak.

2. AddListener method: Override this method to add a listener to a conversation (listener list). Since the speaker class can be a speaker of several conversations, the Conversation parameter should be checked to add it to the proper listener list. Even if the speaker class only speaks one conversation, the Conversation parameter should still be checked since there is nothing stopping a listener class from attempting it add itself to a conversation that doesn't exist.

```
procedure TRaveSpeakerClass.AddListener(Conversation: string;
                                        ListenMethod: TRaveListenEvent);

begin { AddListener }
  inherited AddListener(Conversation,ListenMethod);
  If CompareText(Conversation,'ABC') = 0 then begin
    If not Assigned(ABCListenList) then begin
      ABCListenList := TRaveMethodList.Create;
    end; { if }
    ABCListenList.AddMethod(TMethod(ListenMethod));
  end; { if }
end;  { AddListener }
```

3. RemoveListener method: Override this method to remove a listener from a conversation (listener list). The same rules specified above for checking the Conversation parameter apply to RemoveListener as well.

```
procedure TRaveSpeakerClass.RemoveListener(Conversation: string;
                                           ListenMethod: TRaveListenEvent);

begin { RemoveListener }
  inherited RemoveListener(Conversation,ListenMethod);
  If (CompareText(Conversation,'ABC') = 0) and
   Assigned(ABCListenList) then begin
    ABCListenList.RemoveMethod(TMethod(ListenMethod));
  end; { if }
end;  { RemoveListener }
```

4. Habla method: Override this method to notify listener classes whether the speaker class can speak a particular conversation. Conversation names should not be case-sensitive so it is best to use the CompareText function when determining a match.

```
function TRaveSpeakerClass.Habla(Conversation: string): Boolean;

begin { Habla }
  Result := CompareText(Conversation,'ABC') = 0;
end;  { Habla }
```

5. Changing method: Override this method to track when a listener component has been deleted or replaced. See section 2.8 for more information in how to use the Changing method.

```
procedure TRaveSpeakerClass.Changing(OldItem: TRaveComponent;
                                     NewItem: TRaveComponent);

begin { Changing }
  inherited Changing(OldItem,NewItem);
  If Assigned(ABCListenList) and Assigned(OldItem) then begin
    If Assigned(NewItem) then begin
      ABCListenList.ReplaceObject(OldItem,NewItem);
    end else begin
      ABCListenList.RemoveObject(OldItem);
    end; { else }
  end; { if }
end;  { Changing }
```

6. Speak method: Call this method whenever the speaker class wants to send a message to the listener objects. The first parameter is the listener list and the second parameter is the message. The structure of the Msg parameter is defined by the type of conversation being spoken. For the CalcNewData conversation, no data is needed so the Msg parameter is nil instead of an actual object. For more complicated conversations, actual objects could be sent to transfer more information.

```
procedure Speak(List: TRaveMethodList; Msg: TObject);
```

5.2.5.6.2 Create Listener Class

Creating a Rave listener class

Continuing with the previous example of the ABC conversation, we will now create a simple listener class. At a minimum, a listener method must be defined in the listener class. The format for this method must match as follows (although the name of the method can and should be changed).

```
TRaveListenerClass = class(TRaveControl)
protected
  FSpeakerObj: TRaveComponent;

  procedure ABCListen(Speaker: TRaveComponent;
                      Msg: TObject);
  procedure SetSpeakerObj(Value: TRaveComponent);
published
  property SpeakerObj: TRaveComponent read FSpeakerObj write SetSpeakerObj;
end; { TRaveListenerClass }
```

1. The listener object must add itself to a specific conversation by calling the AddListener() method of a speaker class object:

```
SpeakerObj.AddListener('ABC',ABCListen);
```

2. The listener object should make sure to remove itself from a conversation if it no longer wants

to receive messages from a specific speaker:

```
SpeakerObj.RemoveListener('ABC',ABCListen);
```

3. Typically, both of these calls can be combined into a single method for setting the property containing the reference to the speaker object.

```
procedure TRaveListenerClass.SetSpeakerObj(Value: TRaveComponent);

begin { SetSpeakerObj }
  If Value = FSpeakerObj then Exit;
  If Assigned(FSpeakerObj) then begin
    FSpeakerObj.RemoveListener('ABC',ABCListen);
  end; { if }
  FSpeakerObj := Value;
  If Assigned(FSpeakerObj) then begin
    FSpeakerObj.AddListener('ABC',ABCListen);
  end; { if }
end;  { SetSpeakerObj }
```

4. The code that is inside of the ABCListen method will be executed whenever the speaker object calls the Speak() method for the 'ABC' conversation. The speaker object will be passed as the first parameter and the message will be passed as the second parameter. The structure of the Msg parameter is defined by the type of conversation being spoken. For the CalcNewData conversation, no data is needed so the Msg parameter is nil instead of an actual object. For more complicated conversations, actual objects could be sent to transfer more information.

5.2.5.6.3 Speaker Class Editor

Property editors for speaker classes

Since several different classes can qualify as speakers for a conversation, the listener component should use the Habla method to determine if a component is a valid speaker rather than checking for specific class types. The following is a sample property editor for our sample speaker and listener classes. By using the Include function of the TRaveComponentPropertyEditor class, only component that speak the ABC conversation will be listed in the drop-down list for the TRaveListenerClass.SpeakerObj property.

```
interface
  TRaveABCSpeakerPropertyEditor = class(TRaveComponentPropertyEditor)
  protected
    function Include(Value: TComponent;
                     Data: longint): Boolean; override;
  end; { TRaveABCSpeakerPropertyEditor }

implementation
  procedure RaveRegister;

  begin { RaveRegister }
    RegisterRavePropertyEditor(TypeInfo(TRaveComponent),TRaveListenerClass,
    'SpeakerObj',TRaveABCSpeakerPropertyEditor);
  end;  { RaveRegister }

  function TRaveABCSpeakerPropertyEditor.Include(Value: TComponent;
                                       Data: longint): Boolean;

  begin { Include }
    Result := (Value is TRaveComponent) and TRaveComponent(Value).Habla('ABC');
  end;  { Include }
```

5.2.5.7 TRaveComponent Changing

The TRaveComponent.Changing method

The Changing method allows components to respond to components that are being added, deleted or replaced during design-time. This is similar to Delphi's Notification method but is done a little differently. For component additions, OldItem will be nil and NewItem will be the component being added. For component deletions, OldItem will be the component being deleted and NewItem will be nil. For component replacements (usually done during imports), OldItem will be the component being replaced and NewItem will be the component that should replace OldItem. If the component class keeps references to other components, it should check to see if those references match deleted or replaced components through the Changing method and take appropriate actions so that invalid component references are not called.

```
procedure TSampleClass.Changing(OldItem: TRaveComponent;
                                NewItem: TRaveComponent);

begin { Changing }
  inherited Changing(OldItem,NewItem);
  If FComponentReference = OldItem then begin
    FComponentReference := NewItem; { Handles deletes and replaces }
  end; { if }
end;  { Changing }
```

5.2.5.7.1 NotifyChanging Procedure

Using NotifyChanging procedure at runtime

If you are deleting a Rave component at runtime, you will need to notify the other components in the same report project of this change or they attempt to reference an object that no longer exists. This is done through the NotifyChanging procedure.

```
procedure NotifyChanging(OldItem: TRaveComponent;
                         NewItem: TRaveComponent);
```

For example, if you are going to delete a dataview you would call:

```
DataView1 := RaveProject1.ProjMan.FindRaveComponent('ObseleteDataView',nil);
NotifyChanging(DataView1,nil);
RaveProject1.ProjMan.DeleteItem(DataView1); // Delete after notifying
```

If you are going to delete a normal component you would call:

```
Page1 := RaveProject1.ProjMan.FindRaveComponent('CustReport.Page1',nil);
Text1 := RaveProject1.ProjMan.FindRaveComponent('Text1',Page1);
NotifyChanging(Text1,nil);
Text1.Free; // Free after notifying
```

What this is doing above is telling all components that the component referenced in OldItem is going to be changed to NewItem (which is nil in these examples). Not that within Rave at design-time, this notification is normally handled automatically.

If you want to replace a component with another, you can also use NotifyChanging. This will automatically change all references from the old to the new dataview and keep the links intact. If you simply deleted the old dataview and then added a new one, any components such as databands or datatexts would no longer be pointing to a dataview. Here's an example of replacing one dataview with another:

```
OldDataView := RaveProject1.ProjMan.FindRaveComponent('ObseleteDataView',nil);
NewDataView := CreateDataView(CreateDataCon(RPDataSetConnection1));
NotifyChanging(OldDataView,NewDataView);
RaveProject1.ProjMan.DeleteItem(OldDataView); // Delete after notifying
```

5.2.5.8 Hiding and Moving Components

Hiding and Moving Rave Components in the Visual Designer

If you want to hide or move certain Rave components to another toolbar there are several ways to do this. The most difficult is to actually modify the RaveRegister procedure to either remove the component reference or assign it to a different component group (toolbar). However, a simpler method exists that will allow you to customize the components contained in the Rave designer for yourself or your end-users. This is done through the Windows registry. To hide a component, set the following string value to '1':

HKEY_CURRENT_USER\Software\Nevrona Designs\Rave\Components\TRaveBitmap\Hidden

Obviously this string is for the TRaveBitmap component. If you want to hide another component, simply replace TRaveBitmap with the full class name of the component. Setting Hidden to any value other than '1' or not having a Hidden string value will cause the component to appear in the visual designer. Note that reports containing hidden components can still be loaded, edited and printed. The resourceful user could create additional, 'hidden' components simply by copying and pasting any that already exist in the report.

To move a component to another group (toolbar), set the following string value to the name of the destination group (i.e. Report, Standard,...).

HKEY_CURRENT_USER\Software\Nevrona Designs\Rave\Components\TRaveBitmap\Group

Once again, this is an example of the string value for TRaveBitmap. If you want to move another component, simply replace TRaveBitmap with the full class name of the component.

5.3 TRaveProjectManager Interface

The Rave project manager class is the owner of all report, global page and data view components in a reporting project. It provides many methods and properties to interface with the reporting project.

If you need to interface with the project manager from within a component or editor, the global variable, ProjectManager of type TRaveProjectManager, provides what you need. To access the ProjectManager variable you must include the unit RVProj in your uses clause.

If you are interfacing with the project manager from within a Delphi or C++Builder application you will want to access the TRaveProject.ProjMan property instead of the global variable, ProjectManager. This is because multiple projects may be open at the same time.

5.3.1 TRaveProjectManager Events

This is a list of TRaveProjectManager Events.

TImportConflictEvent = procedure(CurrentItem: TRaveProjectItem; var ImportName: string) of object;

property **OnImportConflict**: TImportConflictEvent;
- This event will be called whenever a naming conflict is encountered during an import. This will allow you to change the name of the imported component to a unique name.

5.3.2 TRaveProjectManager Methods

The following is a list of the TRaveProjectManager Methods.

procedure **ActivateReport**(Report: TRaveReport);
- Activates Report as the current report.

procedure **ClearChanged**;

- The procedure should be called to set the DataChanged property to false. This procedure should be called with caution since it may cause changes to the report project to not be saved.procedure **Load**;
- Load will load the report project from the file specified by the FileName property.

procedure **DeactivateReport**;

- Deactivates the current report.

procedure **DeleteItem**(Item: TRaveProjectItem; Notify: Boolean);

- This method is used to delete TRaveProjectItem components such as reports, global pages and dataviews from the report project. In most cases, Notify should be set to true to notify the designer of the deleted item.

procedure **ExportProject**(ExportFileName: string; Items: TList);

- Creates an export report project containing the items listed in Items. Only Reports, Global Page and Data Views should be referenced in Items.

function **FindRaveComponent**(Name: string; DefRoot: TRaveComponent): TRaveComponent;

- This function can be used to find a component of a given name. The Name parameter can include the owner name as well as the component name (e.g. "Report1.Page2") with the DefRoot parameter set to nil. The other method of calling FindRaveComponent is to pass the component name in the Name parameter and the owner component in the DefRoot parameter.

  NOTE: The ProjectManager component owns all reports, global pages and DataViews. Report components own their report pages and all page components are the owner for all components contained within.

function **GetParam**(Param: string): string;

- This function will return the value of the project parameter, Param.

function **GetUniqueName**(BaseName: string; NameOwner: TRaveComponent; UseCurrent: Boolean): string;

- This function will calculate a unique name for a component. This can be useful when creating components within a report wizard. The BaseName property is the starting point used for the component name. NameOwner is the owner component that will be used to search for naming conflicts. UseCurrent tells the function whether to attempt to use BaseName first before appending unique suffixes. When UseCurrent is true, the BaseName parameter should consist of the class base name appended to the desired name separated by a '|' character. So for a text component that you want to name 'MyText' you would pass the string 'MyText|Text' as the BaseName parameter and UserCurrent set to true.

Examples:
TxtComp.Name := ProjectManager.GetUniqueName('Text',PageComp,false);
TxtComp.Name :=
ProjectManager.GetUniqueName('ReportTitle|Text',PageComp,true);

function **ImportProject**(ImportFileName: string; AutoReplace: Boolean): Boolean;
- Imports an exported report project into the current project. AutoReplace will determine if all duplicate items are automatically replaced instead of using the OnImportConflict event.

procedure **LoadFromStream**(Stream: TStream);
- Loads a report project from Stream. The Unload method should be called before LoadFromStream is called.

procedure **New**;
- New will create a new report project.

function **NewReport**: TRaveReport;
function **NewGlobalPage**: TRavePage;
function **NewDataView**: TRaveDataView;
- These methods can be used to create a new report, global page or data view to the report project. The new item will be returned as the result of the function. For creating DataView dynamically at runtime you should use the CreateDataView function which is covered in section on TRaveDataSystem Global Functions.

procedure **Save**;
- Save will save the report project to the file specified by the FileName property.

procedure **SaveToStream**(Stream: TStream);
- Save the report project out to Stream.

procedure **SetParam**(Param: string; Value: string);
- This procedure set the project parameter, Param, to the text specified in Value.

procedure **Unload**;
- Unloads all items from current project.

5.3.3 TRaveProjectManager Properties

Public properties of TRaveProjectManager
---

property **ActiveReport**: TRaveReport (read only)
- This property returns the currently active report. To activate a new report use the ActivateReport method.

property **Categories**: TStrings (read/write/pub)
- This property defines the available report categories for the report project. Each line defines a name for a separate report category.

property **DataChanged**: Boolean (read/write*)
- DataChanged should be set to true whenever a change is made to the report project. Note that once DataChanged is set to true, it will only be set to false again when the project is saved or by calling the ClearChanged method.

property **DataViewList**: TList (read only)
property **GlobalPageList**: TList (read only)
property **ReportList**: TList (read only)
- These properties provide access to the reports, global pages and dataviews of the report project. Since ProjectManager owns all of these components, you can also access these components through the Components property just like any other Delphi component.

property **FileName**: string (read/write)
- This property is the name of the file used by the Save and Load methods.

property **Parameters**: TStrings (read/write/pub)
- This property defines the project parameters for the report project. Each line defines a parameter name and values can be set or retrieved using the SetParam and GetParam methods.

property **Printing**: Boolean (read only)
- Printing will be true if a report is currently being printed.

property **Saved**: Boolean (read/write)
- This property can be used to tell if the report project has been saved to the disk or not. When a project is loaded from disk or after is has been saved to disk, Saved will be true.

property **StreamParamValues**: Boolean (read/write)
- This property should not normally be used. It is used internally to transfer parameter values to the end-user Rave designer.

property **Units**: TPrintUnits (read/write/pub)
- This property defines the units that will be used across the reporting project. Valid values include unInch, unMM, unCM, unPoint and unUser. When changing Units to values other than unUser, UnitsFactor will be set to the appropriate value. unUser allows custom values of the UnitsFactor property to handle any kind of units conversion.

property **UnitsFactor**: TRaveFloat (read/write/pub)
- This property defines the relationship between the units used across the report project and inches. For example, when using a Units value of unMM (millimeters), UnitsFactor would be 25.4 (25.4 mm per inch). Custom unit conversions can be set by setting this property to any value desired.

property **Version**: integer (read only)

- This property is the version of the currently loaded project. Once the project is saved it will be automatically upgraded to the current version.

5.4 TRaveDataSystem

The Rave data system is controlled by the TRaveDataSystem class and the RaveDataSystem global variable. Through the RaveDataSystem object you can interface with data connection components to request actions or retrieve data. The RaveDataSystem interface provides more flexibility but is also more complicated. For most cases, the functionality provided by the TRaveDataView component is sufficient and is much simpler.

5.4.1 TRaveDataSystem Events

Events of TRaveDataSystem

```
TRaveDataResult = (drContinue,drAbort,drPause);
TTimeoutEvent = procedure(    DataSystem: TRaveDataSystem;
                              Counter: integer;
                              Timeout: integer;
                              EventType: integer;
                              Connection: string;
                              First: Boolean;
                          var DataResult: TRaveDataResult) of object;

property OnSmallTimeout: TTimeoutEvent
property OnLargeTimeout: TTimeoutEvent
```

- These events are called when a timeout occurs while communicating with a data connection. The small timeout event will be encountered first and should serve as a warning that communication has stalled. This may occur because of the data connection performing some type of action (i.e. opening a table). The large timeout event will be called after the delay is longer than the configured maximum and the user should then be prompted for action. The DataSystem parameter is the current TRaveDataSystem object that is being used. Counter is the current timing count (in 1/100ths of a second). Timeout is the maximum time that is allotted for this event to execute. EventType is the type of event that is being executed. Connection is the name of the data connection and First is set to true if this is the first time the small or large timeout has occurred for this timeout. The timeout events will be continually called every 1/100th of a second so processing should be brief. The DataResult parameter allows the event to control the timeout processing. The default is drContinue which allows continued attempts to perform the data event. drAbort will abort the data event while drPause will suspend attempted retries until a drContinue is passed back.

TRaveDataAction = (daOpen,daClose);
TDataActionEvent = procedure(DataSystem: TRaveDataSystem;
DataAction: TRaveDataAction) of object;
property OnDataAction: TDataActionEvent
- This event can be used to keep track of when data connections are opened and closed by the Rave data system. The Rave designer uses this event to update the Data connection status LED during report execution.

5.4.2 TRaveDataSystem Methods

Methods of TRaveDataSystem
---

function **CallEvent**(EventType: integer; DataCon: TRaveDataConnection): boolean;
- This function will execute a specific data event for the given data connection. The data connection must already have been opened with a call to OpenDataEvent. EventType must be one of the valid data event constants: DATAFIRST, DATANEXT, DATAEOF, DATAGETCOLS, DATAGETROW, DATASETFILTER, DATAGETSORTS, DATASETSORT, DATAOPEN, DATARESTORE, DATAACKNOWLEDGE, DATAFREEALTBUF. The format of the data in the communication buffer differs depending upon the type of data event. This function is normally called by the data view object itself and should not be called directly.

procedure **ClearBuffer**;
- This procedure will clear the contents of the communication buffer and reset the buffer pointer to the beginning. As with the ReadXxxx and WriteXxxx methods, ClearBuffer should not normally be called as TRaveDataView or the CallEvent method handles when the buffer needs to be cleared.

procedure **CloseDataEvent**(DataCon: TRaveDataConnection);
- This function will close a data connection that was previously opened by a call to OpenDataEvent. This function is normally called by the data view object itself and should not be called directly.

procedure **CreateAltFileMap**(BufIdx: integer);
- This method will create an alternate communication buffer. This method is used internally by the data connection components and should not be called directly.

procedure **FreeAltFileMap**(DataCon: TRaveDataConnection);
- This method will free an alternate communication buffer. This method is used internally by the data connection components and should not be called directly.function **GainControl**: boolean;
- This function should be called at the beginning of a Rave data session. Only one application can have control of the Rave data system at one time. If control cannot be gained, the result of the function will be false. Otherwise if control is successfully gained, the result will be true.

function **IsUnique**(Name: string): boolean;
- This function will check for duplicate data connections of the same connection name. This function will not normally need to be called.

function **OpenDataEvent**(AName: string; DataCon: TRaveDataConnection): boolean;
- This function will open a data connection for a given connection name. If the connection is opened successfully, the result of the function will be true. This function is normally called by the data view object itself and should not be called directly.

procedure **PrepareEvent**(EventType: integer);
- This method will prepare the communication buffer for the given data event. Any additional data values that are required must still be written using the WriteXxxx methods.

function **ReadBool**: boolean;
procedure **ReadBuf**(var Buffer; Len: integer);
function **ReadCurr**: currency;
function **ReadDateTime**: TDateTime;
function **ReadFloat**: extended;
function **ReadInt**: integer;
function **ReadPtr**(Len: integer): pointer;
function **ReadStr**: string;
- These functions will read data from the Rave communication buffers. These functions are not normally needed as the TRaveDataView class handles the reading of the communication buffers.

procedure **ReleaseControl**;
- This procedure must be called at the end of a Rave data session to release control of the Rave data system.

procedure **UpdateConnections**;
- This procedure will query the Windows environment for active data connections and update the RTConnectList and DTConnectList properties.

procedure **WriteBool**(Value: boolean);
procedure **WriteBuf**(var Buffer; Len: integer);
procedure **WriteCurr**(Value: currency);
procedure **WriteDateTime**(Value: TDateTime);
procedure **WriteFloat**(Value: extended);
procedure **WriteInt**(Value: integer);
procedure **WriteStr**(Value: string);
- These functions will write data to the Rave communication buffers. These functions are not normally needed as the TRaveDataView class or the CallEvent method handles writing to the communication buffers.

5.4.3 TRaveDataSystem Properties

Properties of TRaveDataSystem
---

property **DTConnectList**: TStringList (read only)

- This is a list of all design-time data connection names. A design-time data connection is one that exists on a form currently loaded in Delphi or C++Builder. The Object array property of DTConnectList contains the TRaveDataConnection object that contains additional information about the design-time data connections.

property **RTConnectList**: TStringList (read only)

- This is a list of all runtime data connection names. A runtime data connection is one that exists in a running application. The Object array property of RTConnectList contains the TRaveDataConnection object that contains additional information about the runtime data connections.

5.4.4 Global Functions

Global functions related to the Rave data system

There are several global functions defined in the RVData unit that are useful for dealing with data related the Rave data system.

function **CreateDataCon**(RPConnection: TRPCustomConnection): TRaveDataConnection;

- This function will create a TRaveDataConnection object for a given data connection component (TRPCustomConnection or one of its descendent's). The result of this function will normally be used in conjunction with the CreateDataView function.

function **CreateDataView**(DataCon: TRaveDataConnection): TRaveDataView;

- This function will create a DataView object for the given TRaveDataConnection object (normally created through a call to the CreateDataCon function). Do not free the DataCon object that is passed into this function since it will belong to the DataView and will be freed by the DataView itself. The following code is how a dataview would normally be created dynamically at runtime:

```
MyDataView := CreateDataView(CreateDataCon(TableConnection1));
```

function **CreateFieldName**(DataViewName: string; FieldName: string): string;

- This function will create a valid field name for a given data view. This function does not normally need to be called since it is called from within the CreateFields procedure.

procedure **CreateFields**(DataView: TRaveDataView; DeletedFields: TStringList; ReplacedFields: TStringList; DoCreate: boolean);

- This function will create field components for a specific data view. If non-nil values are passed in for the DeletedFields and ReplacedFields parameters, the CreateFields procedure will return existing field components that will be deleted or replaced if the DoCreate parameter is set to true. In the Rave designer, these parameters are used to warn the user that fields are about to be deleted or replaced. This function does not normally need to be called since it is called from within the CreateDataView function.

procedure **DataViewFirst**(DataView: TRaveDataView; DetailKey: TRaveFieldName;
MasterDataView: TRaveDataView; MasterKey: TRaveFieldName; SortKey: string);
- This function will initialize a data view for the given filter values (DetailKey, MasterDataView and MasterKey) and sort order (SortKey).

function **PerformLookup**(LookupDataView: TRaveDataView; LookupValue: string;
LookupValueField: TRaveDataField; LookupField: TRaveFieldName;
LookupDisplay: TRaveFieldName; LookupInvalid: string): string;
- This function will perform a lookup for the given parameters and return the text that it evaluates to.

function **ProcessDataStr**(DefaultDataView: TRaveDataView; Value: string): string;
- This function will process a DataView and DataText combination and return the text that it evaluates to.

5.5 TRaveDesigner

Interfacing with TRaveDesigner

The TRaveDesigner class allows you to view and control the settings of the Rave design environment. Access to the designer object is done through the TRavePage.Designer property. Note that only projects that are loaded in the Rave visual designer will have designers assigned to the TRavePage.Designer property so it is not possible to interface with the TRaveDesigner class from within a Delphi or C++Builder application. If generic access to the currently displayed designer is needed, a global variable, CurrentDesigner, is provided in the RVClass unit.

5.5.1 TRaveDesignerMethods

Methods of TRaveDesigner

procedure **AddPip**(Index: byte;
Control: TRaveControl;
Cursor: TCursor;
X: TRaveUnits;
Y: TRaveUnits);
- This method will create a selection/sizing pip at the current position X,Y. When creating most components, the default pip locations will be sufficient, however the pip methods will allow for custom pip placement and functionality. Each pip for a specific component must have a unique Index value. A component can respond to pip movements by overriding the TRaveControl.PipSize method.

procedure **Modified**;
- This method should be called whenever modifications have been made to the components in the current designer. This will signal the project manager that the changes need to be saved and will also allow the designer to update it's various displays.

procedure **RemovePips**(Control: TRaveControl);
- This method removes all pips for the component defined by Control.

procedure **SwitchPips**(RavePip: TRavePip;
SwitchIdx: byte);
- This method will switch the pips defined by RavePip and the pip for the same component at index SwitchIdx. This is normally done when a pip of a component is dragged past another pip during resizing.

procedure **UpdatePip**(Index: byte;
Control: TRaveControl;
X: TRaveUnits;
Y: TRaveUnits);
- This method will update the position of the pip defined by Index.

{ Find methods }

---

function **FindContainerAt**(X,Y: TRaveUnits;
NewChild: TClass): TRaveControl;
- This method returns the name of a valid parent component at location X,Y for a component of type NewChild. If no valid parent is found, the page will be the default container.

function **FindControl**(Name: string): TRaveComponent;
- This method will search the current page for the component of the given Name.

function **FindControlAt**(X,Y: TRaveUnits): TRaveControl;
- This method returns the name of the Control at X,Y. If no Control is found at that location, then this will return the current Page component.

{ Position methods }

---

function **SnapX**(Value: TRaveUnits): TRaveUnits;
function **SnapY**(Value: TRaveUnits): TRaveUnits;
- These methods will take a Value which could be the result of some calculation and converts it to the nearest grid value. For example, if grid spacing is set for 0.1 inches and you pass the function a value of 1.12, then it will return a value of 1.1.

function **XD2I**(Value: longint): TRaveUnits;
function **YD2I**(Value: longint): TRaveUnits;
- These methods convert the printer canvas measurement (dots) to unit measurements (inches).

function **XI2D**(Value: TRaveUnits): longint;
function **YI2D**(Value: TRaveUnits): longint;
- These methods convert units measurements (inches) to printer canvas

measurements (dots).

{ Selection methods }

---

procedure **ClearSelection**;
- This method removes all components from the current selection.

procedure **CopySelection**;
- This method will copy the selected components to the Windows clipboard.

procedure **DeleteSelection**;
- This method will delete all components in the current selection.

procedure **DeselectControl**(Control: TRaveComponent);
- This method removes the component defined by Control from the current selection.

function **IsSelected**(Control: TRaveComponent): boolean;
- This method will return whether the component defined by Control is in the current selection.

procedure **MoveSelection**(X,Y: TRaveUnits);
- This method will move the top, left corner of the selected component(s) to the position defined by X,Y.

procedure **PasteSelection**;
- This method will paste the components previously copied to the Windows clipboard into the current page.

procedure **SelectChildren**(Control: TRaveComponent);
- This method will add all children components of the component defined by Control to the current selection.

procedure **SelectControl**(Control: TRaveComponent);
- This method adds the component defined by Control to the current selection.

procedure **SelectType**(ProjectItem: TRaveProjectItem;
RaveClass: TClass);
- This method will add all components on the current page that are the same type as RaveClass to the current selection.

procedure **ToggleControl**(Control: TRaveComponent);
- This method toggles the selected status of the component defined by Control.

{ Zooming methods }

---

procedure **AlignSelection**(AlignType: integer);

- This method will perform the alignment action defined by AlignType. Valid values for AlignType include: RaveAlignLeft, RaveAlignHCenter, RaveAlignRight, RaveAlignHCenterInParent, RaveAlignHSpace, RaveAlignEquateWidths, RaveAlignTop, RaveAlignVCenter, RaveAlignBottom, RaveAlignVCenterInParent, RaveAlignVSpace, RaveAlignEquateHeights, RaveAlignMoveForward, RaveAlignMoveBehind, RaveAlignBringToFront, RaveAlignSendToBack, RaveAlignTapLeft, RaveAlignTapRight, RaveAlignTapUp, RaveAlignTapDown, RaveAlignTapHSizeDown, RaveAlignTapHSizeUp, RaveAlignTapVSizeDown and RaveAlignTapVSizeUp.

procedure **CenterWindow**(X,Y: TRaveUnits);
- This method changes the page layout to center at the point indicated by the position, X,Y. This procedure does NOT change the zoom factor.

procedure **ZoomIn**(X,Y: TRaveUnits);
- This method increases the zoom factor by the zoom increment amount set by the preferences.

procedure **ZoomOut**;
- This method decreases the zoom factor by the zoom increment amount set by the preferences.

procedure **ZoomPage**;
- This method changes the zoom factor so that the full page plus the Minimum border fills the page designer screen.

procedure **ZoomPageWidth**;
- This method changes the zoom factor so that the form width plus the minimum border fills the page designer screen.

procedure **ZoomSelected**;
- This method changes the zoom factor so that all selected items will fill the page designer screen. If only one item is selected, then the zoom factor will be changed so that single item is full screen.

function **ZoomToRect**(X1,Y1,X2,Y2: TRaveUnits): TRaveFloat;
- This method changes both the page position and zoom factor so that the area indicated by X1,Y1,X2,Y2 will occupy the page design. This function examines both the vertical and horizontal sizes when determining what zoom factor should be used so that the full rectangle is shown.

5.5.2 TRaveDesignerProperties

Properties of TRaveDesigner

property **GridPen**: TPen (read/write)
- This property defines the pen used to draw the grid of the current page.

property **MinimumBorder**: TRaveFloat (read/write)

- This property defines the border around the page that will be used by the ZoomPage and ZoomPageWidth methods. The default is 1.0 inches.

property **Page**: TRavePage (read only)
- This property defines the page component that this designer is currently being used to edit.

property **Selection**[Index: integer]: TRaveComponent (read only)
- This property will allow access to specific selected component. The Index property is 0 based.

property **Selections**: integer (read only)
- This property will return the number of components that are in the current selection.

property **ZoomFactor**: TRaveFloat (read/write)
- This property sets or returns the current zoom factor.

5.6 Property Editor

The property editor interface of Rave allows you to create and respond to property editors through the TRavePropertyEditor class (RVTool.pas). The property editor can apply to a specific property of a specific component type, a specific property across all components or all properties of a specific type for a specific component or across all components.

5.6.1 Types

The following is a list of Types of property editors

- Simple - This property editor only displays an edit box allowing the user to type in a string of data (e.g. TRavePage.Name).

- Listing - Like a Simple property editor, but also displays a drop down list of values to select from (e.g. TRectangle.Color).

- Editor - Like a Simple property editor, but also displays an editor button that brings up a custom dialog to edit the property (e.g. TRaveText.Font).

- Editor/Listing - This is a combination of the Listing and Editor property editors. It allows both a drop down list of values to select from and an editor button that brings up a custom dialog to edit the property (e.g. TRaveDataText.DataField)

5.6.2 Steps to Create

The following is a list of steps to create a property editor

1: Create a class descending from TRavePropertyEditor (or one of its descendent's). Below is a list of the methods that will normally need to be overridden for the different types of property editors (these methods are described in detail below).

Simple Property Editor - GetOptions, GetValue and SetValue
Listing Property Editor - GetOptions, GetValue, SetValue and GetValues
Editor Property Editor - GetOptions, GetValue and Edit
Editor/Listing Property Editor - GetOptions, GetValue, SetValue, GetValues and Edit

```
type
  TMyPropertyEditor = class(TRavePropertyEditor)
  public
    function GetOptions: TPropertyOptionsSet; override;
    function GetValue: string; override;
    procedure SetValue(Value: string); override;
  end; { TMyPropertyEditor }
```

2: Create methods for GetOptions, GetValue, SetValue, GetValues and/or Edit as necessary. For examples, there are many property editors defined in this unit and other units in the Rave library.

3: Register the property editor by calling RegisterRavePropertyEditor from within a global scope procedure named RaveRegister (exact case required). The PropType parameter should be the type info structure for the property type you are registering. The function TypeInfo() will return the value you need. The ControlClass parameter defines the class of components that this property editor applies to. If this parameter is nil, then the property editor applies to all component classes. The PropName parameter allows you to limit the property editor to only properties of a specific name. If this field is blank then the property editor applies to all properties with a type info of PropType. The EditorClass property should be the class type of the property editor you are registering. See the RaveRegister procedure in this unit for examples.

```
procedure RaveRegister;

begin { RaveRegister }
  RegisterRavePropertyEditor(TypeInfo(integer),TMyComponent,'MyValue',
    TMyPropertyEditor);
end;  { RaveRegister }
```

4: Include the unit containing the property editor in a Delphi package and register that with RAVE through Edit|Preferences|Components.

5.6.3 Methods

The following is a list of the TRavePropertyEditor Methods.

procedure **Modified**;
- This method should be called whenever a property value is modified so that the Property Panel can refresh its display.

function **GetOrdValue**(Index: integer): integer;
function **GetFloatValue**(Index: integer): extended;
function **GetStrValue**(Index: integer): string;
function **GetVariantValue**(Index: integer): variant;
- These methods should be called to get the property value of the instance specified by Index. If the property is a set, class or pointer type, use GetOrdValue and typecast it appropriately.

procedure **SetOrdValue**(Value: integer);
procedure **SetFloatValue**(Value: extended);
procedure **SetStrValue**(Value: string);
procedure **SetVariantValue**(Value: variant);
- These methods should be called to set the property value of all instances connected to this property editor. If the property is a set, class or pointer type, use SetOrdValue and typecast it appropriately. Note however, if the property is a class type and the object is owned by the component (i.e. Font), you should iterate through the Instance property assigning the data instead of overwriting the object pointer which is what SetOrdValue will do.

function **SameValue**(TestComp: TRaveComponent;
TestValue: string): boolean;
- This method is used internally by the property panel to determine whether the current value is different than the default (Highlight Changes option). You will should not need to call this method directly.

5.6.4 Properties

The following is a list of the TRavePropertyEditor properties.

property **Instance**[Index: integer]: TRaveComponent (read only)
- This property returns the Index'th component being edited. See also InstCount.

property **InstCount**: integer (read only)
- This property returns the numbers of components that are currently being edited by this property editor (i.e. multiple components are selected). If poMultiSelect is not in the property editor options, this value will always be 1. See also Instance property.

property **Name**: string (read only)
- This property returns the name of the property field being edited.

property **Options**: TPropertyOptionsSet (read only)
- This property returns the current options for this property editor. See also TPropertyOptions and GetOptions.

property **PropInfo**[Index: integer]: PPropInfo (read only)
- This property will return the PPropInfo structure for the instance specified by Index. See Delphi's TypInfo unit for a description of PPropInfo.

property **Value**: string (read/write)
- This property sets or returns the value of the property as a string. See also GetValue, SetValue.

5.6.5 Virtual Methods

The following is a list of the TRavePropertyEditor Virtual methods

procedure **Edit**;

- This virtual method will be called whenever the user presses the editor button on the property panel. Normally you will want to display a dialog allowing the user to edit a property that is too complicated for a simple text string. This method will only be called if poEditor is in the property editors options.

function **GetOptions**: TPropertyOptionsSet;

- This virtual method should be overridden to return the set of options for this property editor. See also Options property and TPropertyOption.

poEditor - This property has a dialog editor
poListing - This property returns a listing of values
poNoSort - The listing returned by this property should not be sorted
poMultiSelect - This property can be edited across a multiple selection
poLiveUpdate - This property is updated whenever any change is made
poReadOnly - This property can only be edited through the list or editor
poPassword - Edit/Display this property with *'s
poRefreshAll - When this property is changed, all windows will be refreshed. Used with properties such as Locked and Mirror that drastically change the contents of the entire Property Panel.

function **GetValue**: string;
procedure **SetValue**(Value: string);

- These two virtual methods should be overridden to set and return the value of the property as a string. If poLiveUpdate is in the property editors options, SetValue will be called for each keypress the user makes in the properly editor's edit box. See also Value property.

procedure **GetValues**(ValueList: TStrings);

- This virtual method should be overridden to return the list of available options in the TStrings object, ValueList. If poNoSort is not in the Options property, the items will be sorted alphabetically. This method will only be called if poListing is in the property editors options.

procedure **PaintValue**(Canvas: TCanvas;
Rect: TRect;
DefaultValue: string);

- This virtual method allows a property editor to customize the contents of the property panel. You will not normally need to override this method but it can be used to provide more visual information on a property value. The color, line style and fill style property editors are examples of where this can be useful.

5.7 Component Editor

Creating a Rave Component Editor

The component editor interface of Rave allows you to create and respond to menu items that you create on a components popup menu (from a right click). This is done through the TRaveComponentEditor class (RVTool.pas). This will allow you to perform actions such as opening a dialog to allow the user to modify the component's properties. Creating a component editor requires the following steps:

5.7.1 Steps to Create

The following are the steps for creating a component editor.

1. Create a class descending from TRaveComponentEditor and override the AddToMenu and RunFromMenu procedures.

```
type
  TMyComponentEditor = class(TRaveComponentEditor)
  public
    procedure AddToMenu(AddMenuItem: TAddMenuItemProc); override;
    procedure RunFromMenu(ID: integer); override;
  end; { TMyComponentEditor }
```

2. Create a method for AddToMenu and call AddMenuItem for each line you want to create in the component's popup menu (see TAddMenuItemProc definition below). ID should be unique for this component editor and greater than 0 for each menu item. ParentID should be 0 to place the new menu item on the first level. To make a child menu from an existing menu item, pass the parent's ID value as ParentID.

```
procedure TMyComponentEditor.AddToMenu(AddMenuItem: TAddMenuItemProc);

begin { AddToMenu }
  AddMenuItem('MenuItem 1',1,0);
  AddMenuItem('MenuItem 2',2,0);
  AddMenuItem('SubMenuItem 2a',21,2);
  AddMenuItem('SubMenuItem 2b',22,2);
  AddMenuItem('-',0,0); // Create a divider
  AddMenuItem('MenuItem 3',3,0);
  AddMenuItem('MenuItem 4',4,0);
end;  { AddToMenu }
```

3. Create a method for RunFromMenu which will be called whenever a user selects a menu item from a components popup menu. The ID value will be equal to a value given in the previous calls to AddMenuItem from the AddToMenu procedure. If an ID of -1 is given, the default action should be taken (i.e. User double-clicked the component. You can get access to the component currently being edited through the Instance property.

NOTE: Menu Items which are parents to other menu items will not cause a RunFromMenu call.

```
procedure TMyComponentEditor.RunFromMenu(ID: integer);

begin { RunFromMenu }
  Case ID of
    -1,1: begin
    // MenuItem1 action (double-click default)
    end;
    21: begin
    // MenuItem 2a action
    end;
    22: begin
    // MenuItem 2b action
    end;
    3: begin
    // MenuItem 3 action
    end;
    4: begin
    // MenuItem 4 action
    end;
  end; { case }
end;  { RunFromMenu }
```

4. Register the component editor by calling RegisterRaveComponentEditor from within a global scope procedure named RaveRegister (exact case required). The ControlClass parameter should be the class type of the component you are defining an editor for and the EditorClass parameter should be the class type of the component editor you are registering.

```
procedure RaveRegister;

begin { RaveRegister }
  RegisterRaveComponentEditor(TMyComponent,TMyComponentEditor);
end;  { RaveRegister }
```

5. Include the unit containing the component editor in a Delphi package and register that with RAVE through Edit|Preferences|Components.

5.8 Project Editor

Enter topic text here.

5.8.1 Create Menu Project Editor

Use the following steps to create a menu-based project editor

1. Create a class descending from TRaveProjectEditor and override the AddToMenu and RunFromMenu procedures.

```
type
```

```
  TMyProjectEditor = class(TRaveProjectEditor)
  public
    procedure AddToMenu(AddMenuItem: TAddMenuItemProc); override;
    procedure RunFromMenu(ID: integer); override;
  end; { TMyProjectEditor }
```

2. Create a method for AddToMenu and call AddMenuItem for each line you want to create in the Tools menu (see TAddMenuItemProc definition above). ID should be unique for this project editor and greater than 0 for each menu item. ParentID should be 0 to place the new menu item on the first level. To make a child menu from an existing menu item, pass the parent's ID value as ParentID.

procedure TMyProjectEditor.AddToMenu(AddMenuItem: TAddMenuItemProc);

```
begin { AddToMenu }
  AddMenuItem('MenuItem 1',1,0);
  AddMenuItem('MenuItem 2',2,0);
  AddMenuItem('SubMenuItem 2a',21,2);
  AddMenuItem('SubMenuItem 2b',22,2);
  AddMenuItem('-',0,0); // Create a divider
  AddMenuItem('MenuItem 3',3,0);
  AddMenuItem('MenuItem 4',4,0);
end;  { AddToMenu }
```

3. Create a method for RunFromMenu which will be called whenever a user selects a menu item from the Tools menu. The ID value will be equal to a value given in the previous calls to AddMenuItem from the AddToMenu procedure. NOTE: Menu Items which are parents to other menu items will not cause a RunFromMenu call.

```
procedure TMyProjectEditor.RunFromMenu(ID: integer);

begin { RunFromMenu }
  Case ID of
    -1,1: begin
    // MenuItem1 action (double-click default)
    end;
    21: begin
    // MenuItem 2a action
    end;
    22: begin
    // MenuItem 2b action
    end;
    3: begin
    // MenuItem 3 action
    end;
    4: begin
    // MenuItem 4 action
    end;
  end; { case }
end;  { RunFromMenu }
```

4. Register the project editor by calling RegisterRaveProjectEditor from within a global scope procedure named RaveRegister (exact case required). The EditorClass parameter should be the class type of the project editor you are registering.

```
procedure RaveRegister;

begin { RaveRegister }
  RegisterRaveProjectEditor(TMyProjectEditor);
end;  { RaveRegister }
```

5. Include the unit containing the project editor in a Delphi package and register that with RAVE through Edit|Preferences|Components.

5.8.2 Create Design Time Components

The following are special notes about creating components at design-time

1. Most of the interface to the components of a project are through the ProjectManager component. ProjectManager is of type TRaveProjectManager and is located in the unit RVProj.pas. To create a new report call ProjectManager.NewReport. To create a new GlobalPage call ProjectManager.NewGlobalPage and to create a new data view call ProjectManager.NewDataView. To create a new report page, call ProjectManager.ActiveReport.NewPage. To create other types of components follow the directions below.

2. You will need to assign the owner for the component using the AOwner parameter of the Create constructor. Any TRaveField components should have the appropriate TRaveDataView component as their owner. Any other components that are placed on a Global Page or Report Page should use the TRavePage object as their owner.

3. You will need to assign the Parent property for all components. For components being placed on a Global Page or Report Page, the parent should be the graphical owner of the component (e.g. If you want to place a component in a TRaveSection, you need to set the Parent property to the TRaveSection component). For all other components, the Parent property should be set to the same as the Owner property.

4. You will need to set the Name property for all components. Call ProjectManager.GetUniqueName to make sure you are using a unique name. The BaseName parameter is the base part of the name that you want to use (i.e. "Section"). The NameOwner parameter is the Owner component whose namespace you will be checking against (normally the TRavePage object). The UseCurrent parameter tells whether you want to try the current BaseName before appending digits on the end (i.e. Try "CustomerName" before trying "CustomerName1", "CustomerName2",..).

5. For visual components (anything deriving from TRaveControl) you will need to initialize the Top, Left, Width and Height properties. These should be the location relative to the upper left corner of the component's Parent.

6. For all components, you will need to call AddComponent(Component) to signal the visual designer that the component has been added to the visual designer. If your wizard is deleting component that have already been added to the report project, your will need to call DeleteComponent(Component) before calling the components Free method.

5.8.3 Create Project Editor

The following is the steps for creating an event-based project editor

There are several project events that can be handled and a single project editor can handle as many project events that it needs to. The following is a list of project events currently supported (Item and Param values list the type of data that is passed in for each project event type). Before events such as peBeforeAppClose will use the return value of HandleEvent to determine whether to continue with (true) or cancel (false) the specific action.

peAfterAppOpen - Called after the Rave visual designer is opened
(Item = ProjectManager, Param = nil)

peBeforeAppClose - Called before the Rave visual designer is closed
(Item = ProjectManager, Param = nil)

peAfterProjectOpen - Called after a project is opened
(Item = ProjectManager, Param = nil)

peBeforeProjectClose - Called before a project is closed
(Item = ProjectManager, Param = nil)

peBeforeNewProject - Called before a new project is created
(Item = ProjectManager, Param = nil)

peAfterNewProject - Called after a new project is created
(Item = ProjectManager, Param = nil)

peBeforeNewReport - Called before a new report is created
(Item = nil, Param = nil)

peAfterNewReport - Called after a new report is created
(Item = the new TRaveReport object, Param = nil)

peBeforeNewReportPage - Called before a new report page is created
(Item = nil, Param = nil)

peAfterNewReportPage - Called after a new report page is created
(Item = the new TRavePage object, Param = nil)

peBeforeNewGlobalPage - Called before a new global page is created
(Item = nil, Param = nil)

peAfterNewGlobalPage - Called after a new global page is created
(Item = the new TRavePage object, Param = nil)

peBeforeNewDataView - Called before a new data view is created
(Item = nil, Param = nil)

peAfterNewDataView - Called after a new data view is created
(Item = the new TRaveDataView object, Param = nil)

peBeforeProjectSave - Called before a project is saved
(Item = ProjectManager, Param = nil)

peAfterProjectSave - Called after a project is saved
(Item = ProjectManager, Param = nil)

peBeforeReportPrint - Called before a report is printed
(Item = ProjectManager, Param = nil)

peAfterReportPrint - Called after a report is printed
(Item = ProjectManager, Param = nil)

peDataConOpen - Called when a data connection is opened
(Item = nil, Param = the TRaveDataConnection object)

peDataConClose - Called when a data connection is closed
(Item = nil, Param = the TRaveDataConnection object)

peDataConConnect - Called when a data connection is made
(Item = nil, Param = the TRaveDataConnection object)

peDataSystemOpen - Called when the Rave data system is opened
(Item = nil, Param = the TRaveDataSystem object)

peDataSystemClose - Called when the Rave data system is closed
(Item = nil, Param = the TRaveDataSystem object)

peDataSystemCallEvent - Called before a Rave data event is executed
(Item = nil, Param = pointer to a TDataSystemEventData record)

peDataSystemEventCalled - Called after a Rave data event is executed
(Item = nil, Param = pointer to a TDataSystemEventData record)

peShowAboutDialog - Called before the About dialog is shown. This will allow an alternate About Box dialog to be displayed. If the result of HandleEvent is false, the normal Rave about box will not be shown. NOTE: Any replacement About Box must clearly display the Nevrona Designs copyright as follows "Copyright © 1995-2005, Nevrona Designs" Failure to do so will be a violation of the software license.
(Item = nil, Param = nil).

peGetAppTitle - Called when the designer is initializing to retrieve an alternate title bar text. The RaveTitle string variable (defined in RVDefine.pas) should be set to the desired text.
(Item = nil, Param = nil)

peBeforePageChange - Called before the page designer is changed from one page to another. Refer to the CurrentDesigner variable (defined in RVClass.pas) for information about the designer that is being moved deselected.
(Item = nil, Param = nil)

peAfterPageChange - Called after the page designer is changed from one page to another or when the Page Designer tab is selected. Refer to the CurrentDesigner variable (defined in RVClass.pas) for information about the newly selected designer.
(Item = nil, Param = nil)

peAddShortCuts - This event is called when the designer is building the list of shortcuts that it will support. By default, all project editors that create tool menu items will have short cut entries created, but this project event allows you to create additional shortcuts. Call RaveCreateShortCut (defined in RVClass.pas) to add a short cut to the list as follows:

```
procedure RaveCreateShortCut(Desc: string; // Description used in IDE
                             Name: string = ''; // Registry Key
                             Item: TComponent = nil; // Action component
                             Key1: TShortCut = 0; // Short-cut key
                             Key2: TShortCut) = 0; // Second key (if any)
```

If Name is blank, the value of ('Rave@' + Desc) will be used. Name is the value used to identify the shortcut in the Windows registry. Once Name is defined, you should not change it or the link will be broken for existing settings. The action component defined in Item is the component that will be "executed" when the short cut is performed. TAction.OnExecute, TMenuItem.OnClick and

TControl.OnClick are the only recognized actions. If Item is nil, a title bar will be created within the ShortCut Editor. Rave supports 2 key short cuts (i.e. ^K-H). To define one, define the first key in Key1 and the second key in Key2.
(Item = nil, Param = nil)

peAfterZoomChange - This event is called whenever the zoom factor is changed. You can access the current zoom factor through the CurrentDesigner.ZoomFact property.
(Item = nil, Param = nil)

peAfterZoomToolChange - This event is called when the Zoom Tool state of the current designer is changed. You can access the current state of the zoom tool with the Zooming boolean variable (defined in RVDefine.pas).
(Item = nil, Param = nil)

peAfterSelectionChange - This event is called whenever the selection of components is changed. You can access information about the currently selected components through the CurrentDesigner.Selections and Selection properties.
(Item = nil, Param = nil)

peLoadState - This event is called when the current state of the designer is being loaded from the registry.
(Item = nil, Param = nil)

peSaveState - This event is called when the current state of the designer is being saved to the registry.
(Item = nil, Param = nil)

peAfterPropertiesModified - This event is called whenever a property is changed in the Property Panel.
(Item = nil, Param = nil)

peInitialize - This event is called when the IDE is being initialized. Use this event instead of FormCreate or a similar event.
(Item = nil, Param = nil)

peSelectComponent

peShowPage - This event is called whenever a new page needs to be shown in the Page Designer. The Item can contain either the page to be shown or a component (in which case, that's component's page will be shown).
(Item = Page or Component, Param = nil)

peBeforeReportActivate -

peAfterReportActivate -

pePrepareViewChange

peAfterViewChange

peShowControlPopup

The following steps will show how to create an event-based project editor.

1: Create a class descending from TRaveProjectEditor and override the HandleEvent function.

```
type
  TMyProjectEditor = class(TRaveProjectEditor)
  public
    function HandleEvent(ProjectEvent: TProjectEvent;
                         Item: TRaveComponent;
                         Param: pointer): boolean; override;
  end; { TMyProjectEditor }
```

2: Create a method for HandleEvent and check the ProjectEvent parameter for the specific project event(s) that you want to handle.

```
function TMyProjectEditor.HandleEvent(ProjectEvent: TProjectEvent;
                                      Item: TRaveComponent;
                                      Param: pointer): boolean;
var
  Report: TRaveReport;

begin { HandleEvent }
  Result := true;
  Case ProjectEvent of
    peAfterNewReport: begin
      Report := Item as TRaveReport;
    { Insert code to get a description and assign to Report.Description }
    end;
  end; { case }
end;  { HandleEvent }
```

3: Register the project editor by calling RegisterRaveProjectEditor from within a global scope procedure named RaveRegister (exact case required). The EditorClass parameter should be the class type of the project editor you are registering.

```
procedure RaveRegister;

begin { RaveRegister }
  RegisterRaveProjectEditor(TMyProjectEditor);
end;  { RaveRegister }
```

4: Include the unit containing the project editor in a Delphi package and register that with RAVE through Edit|Preferences|Components.

5.8.3.1 Create Event System

The following steps will show how to create an event-based project editor.

1: Create a class descending from TRaveProjectEditor and override the Handles function. This function defines which events this project editor defines methods for. If there are more than one event type that are defined, separate with semicolons (e.g. 'DataEvents;ToolMenuEvents').

```
TMyProjectEditor = class(TRaveProjectEditor)
public
  function Handles: string; override;
end; { TMyProjectEditor }

function TMyProjectEditor.Handles: string;
```

```
begin
  Result := 'DataEvents';
end;
```

2: Define the event methods that you want to handle. Use the same name and structure as the methods defined in the event handler interface structure. It is not necessary to define all methods of the interface.

Data event handler interface structure in RVData.pas (this is already created, no need to define it yourself):

```
IRaveDataEventHandler = interface
  ['{E89848DC-28E3-4BBB-A8C8-ADC36904EDA4}']
  procedure DataConOpen(DataCon: TRaveDataConnection);
  procedure DataConClose(DataCon: TRaveDataConnection);
  procedure DataConConnect(DataCon: TRaveDataConnection);
  procedure DataSystemOpen(DataSystem: TRaveDataSystem);
  procedure DataSystemClose(DataSystem: TRaveDataSystem);
  procedure DataSystemCallEvent(DataSystem: TRaveDataSystem;
                                EventData: TDataSystemEventData);
  procedure DataSystemEventCalled(DataSystem: TRaveDataSystem;
                                  EventData: TDataSystemEventData);
end; { IRaveDataEventHandler }
```

New project editor source:

```
TMyProjectEditor = class(TRaveProjectEditor)
public
  function Handles: string; override;
  procedure DataConOpen(DataCon: TRaveDataConnection);
end; { TMyProjectEditor }

procedure TMyProjectEditor.DataConOpen(DataCon: TRaveDataConnection);
begin
// Do whatever you want to do when a data connection is opened
end;
```

3: Register the project editor by calling RegisterRaveProjectEditor from within a global scope procedure named RaveRegister (exact case required). The EditorClass parameter should be the class type of the project editor you are registering.

```
procedure RaveRegister;

begin { RaveRegister }
  RegisterRaveProjectEditor(TMyProjectEditor);
end;  { RaveRegister }
```

5.8.3.2 Create Event Handler

The following shows how to create and call an event handler

In the last section, we showed how to create a project editor to handle existing event handler system. This section will show how to create and call your own event handler systems.

1. Define an event handler interface structure. This is the methods or events that a project editor

can define. The GUID that is on the second line can be generated by pressing Shift-Ctrl-G within the Delphi IDE.

```
  IMyEventHandler = interface
    ['{36E1A0F0-F7EE-487E-AE66-F291400A1052}']
    procedure MyProcA(ID: integer);
    procedure MyProcB(Value: string);
  end; { IMyEventHandler }
```

2. Define an event handler class.

```
// Event methods type
  TMyEventMethod = (meMyProcA,meMyProcB);

// Event handler class
  TMyEventHandler = class(TRaveEventHandler, IMyEventHandler)
  protected
  // Parameter fields
    FID: integer;
    FValue: string;
    FMyEventMethod: TMyEventMethod;
  // Overrides
    procedure Process; override;
    function Handles: string; override;
  // Interface property
    property ProjectEditor: TRaveProjectEditor read FProjectEditor implements
     IMyEventHandler;
  public
  // Shell Interface methods
    procedure MyProcA(ID: integer);
    procedure MyProcB(Value: string);
  // Broadcast methods
    procedure DoMyProcA(ID: integer);
    procedure DoMyProcB(Value: string);
  end; { TMyEventHandler }

// Overrides
  procedure TMyEventHandler.Process;

  var
    IItem: IMyEventHandler;

  begin { Process }
    IItem := self;
    Case FMyEventMethod of
      meMyProcA: IItem.MyProcA(FID);
      meMyProcB: IItem.MyProcB(FValue);
    end; { case }
  end;  { Process }

  function TMyEventHandler.Handles: string;

  begin { Handles }
    Result := 'MyEvents';
  end;  { Handles }

// Shell Interface methods
  procedure TMyEventHandler.MyProcA(ID: integer); begin end;
  procedure TMyEventHandler.MyProcB(Value: string); begin end;

// Broadcast methods
  procedure TMyEventHandler.DoMyProcA(ID: integer);
```

```
begin { DoMyProcA }
  FID := ID; // Assign parameter(s) to field variables
  FMyEventMethod := meMyProcA; // Determine which method to execute
  Broadcast; // Broadcast event to all project editors
end;  { DoMyProcA }

procedure TMyEventHandler.DoMyProcB(Value: string);

begin { DoMyProcB }
  FValue := Value; // Assign parameter(s) to field variables
  FMyEventMethod := meMyProcB; // Determine which method to execute
  Broadcast; // Broadcast event to all project editors
end;  { DoMyProcB }
```

3. Define a variable of the same type as your event handler.

```
var
  MyEventHander: TMyEventHandler;
```

4. Register the event handler by calling RegisterRaveEventHandler from within a global scope procedure named RaveRegister (exact case required). The EventHandlerClass parameter should be the class type of the project editor you are registering. The function will return an instance of your event handler class that you should store.

```
procedure RaveRegister;

begin { RaveRegister }
  MyEventHandler := RegisterRaveProjectEditor(TMyEventHandler);
end;  { RaveRegister }
```

5. To execute an event for all project editors simply call the broadcast method for the event handler such as:

MyEventHandler.DoMyProcB('This is a test');

This would call the MyProcB method for all project editors that handle the MyEvents event type and define a method called MyProcB.

6. To execute an event for a single project editor instance (e.g. AProjectEditor) assign the project editor using the SetProjectEditor method, define an interface variable and assign your event handler to it then call the interface method (not the DoXxxxx broadcast method) of the interface variable (not the event handler).

```
var
  IEventHandler: IMyEventHandler;

  MyEventHandler.SetProjectEditor(AProjectEditor);
  IEventHandler := MyEventHandler;
  IEventHandler.MyProcB('This is a test');
```

---

1. Create a directory under C:\Rave that is the same name as the component package that you will be creating (for this example: C:\Rave\ND_AbtBx).

2. In C:\Rave\ND_AbtBx create a form (.PAS and .DFM) called NDCsAbt and place the following code in there (place whatever controls on the form that you want for the about box):

```
unit NDCsAbt;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, RVClass, RVTool;

type
  TNDAboutBoxForm = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Button1: TButton;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TNDAboutBoxProjectEditor = class(TRaveProjectEditor)
  public
    function HandleEvent(ProjectEvent: TProjectEvent;
                         Item: TRaveComponent;
                         Param: pointer): boolean; override;
  end; { TNDAboutBoxProjectEditor }

  procedure RaveRegister;

var
  NDAboutBoxForm: TNDAboutBoxForm;

implementation

{$R *.DFM}

  procedure RaveRegister;

  begin { RaveRegister }
    RegisterRaveProjectEditor(TNDAboutBoxProjectEditor);
  end; { RaveRegister }


(*****************************************************************************}
    ( class TNDAboutBoxProjectEditor

(*****************************************************************************)

    function TNDAboutBoxProjectEditor.HandleEvent(ProjectEvent: TProjectEvent;
                                                  Item: TRaveComponent;
                                                  Param: pointer): boolean;

  begin { HandleEvent }
    Result := true;
    Case ProjectEvent of
      peShowAboutDialog: begin
        With TNDAboutBoxForm.Create(Application) do try
          ShowModal;
          Result := false; // Signal not to show original about box
        finally
          Free;
        end; { with }
      end;
    end; { case }
  end;  { HandleEvent }
```

```
end.
```

3. Also in C:\Rave\ND_AbtBx create a package unit called ND_AbtBx.dpk and place the following code in there:

```
package ND_AbtBx;

{$DESCRIPTION 'About Box Replacement Property Editor (ver 1.0)'}
{$IMPLICITBUILD OFF}

requires
  VCL40,
  RVCL30;

contains
  NDCsAbt;

end.
```

4. Create a batch file in C:\Rave called RANTCOMP.BAT and place the following commands in there:

```
@echo off
if exist setenv.bat call setenv.bat
..\source\computil SetupD4
if exist setenv.bat call setenv.bat
if "%1%"=="" goto showhelp
goto docomp
:showhelp
echo ************************************************
echo Compile RANT Package (for the Rave Visual Designer)
echo ************************************************
echo :
echo Usage: RANTCOMP packagename
echo :
echo Note: The packagename parameter should not include the .dpk extension
echo :
echo Desc: This batch file will compile a package containing a RAVE add-on
echo       and place the .BPL in the parent directory (normally C:\Rave).
echo       It is meant to be run from a directory under Rave containing
echo       the source for the add-on. The Rave library files must reside
echo       in C:\Rave\RV.
echo ************************************************
goto endok
:docomp
%NDD4%\bin\dcc32 %1.dpk /b /h /w /z /LE.. /U..\RV -$d-l-n+p+r-s-t-w-
/DDESIGNER %2 %3 %4
if errorlevel 1 goto enderror
goto endok
:enderror
echo Error!
:endok
```

5. Change to the C:\Rave\ND_AbtBx directory and run "..\RANTCOMP ND_AbtBx"

6. Start Rave and install the new package file. The BPL will be located in C:\Rave.

7. If the RANT package defines any components, place the .PAS and any other relevant files in the library C:\Rave?\D4 (or C4 for C++Builder 4.0, D5 for Delphi 5.0, ...)

Server

Chapter

6 Server

.

6.1 What Is

Rave Server can also serve applications and act as a middle tier. This allows Reports to be run on a centralized managed high power server that has a fast network connection to the database. Since only the actual Reports are returned to the application and the Reports are executed on the Server, the application does not require a high-speed connection to the Report Server. The application does not require any database connectivity either.

The application passes all parameters and Report information to the Server, and retrieves the generated Report via HTTP and displays it within the application, exactly as if the Report had been generated from the application.

Note that to make use of this alternative, the client application would be required to support HTTP calls and responses.

6.2 System Requirements

Rave Server can be used in two different ways for Windows platforms. Rave Server can either be used as a standalone application or as a service. The service can only run on Windows NT or 2000 (Server and Workstation).

6.3 Installing

6.4 Running as Application

By default Rave Server is installed as an application. Note that in the current version, only when using Rave as an application, may the console be accessible.

6.5 Running as Service

To run Rave Server as a service, first install it as such. To do so, in a Command Prompt window, change to the directory where Rave was installed and then type:

> RaveReportServer –install

To uninstall type:

> RaveReportServer –uninstall

Once it has been installed as a service, it is managed the same way as other services in Windows NT or 2000.

6.6 Data Directory

A data subdirectory is created underneath the folder where Rave Server is installed. When the server starts, all the report projects found in the data directory will be loaded. Projects may be added and removed as necessary from this folder. If any of the project files change, they will be reloaded the next time they are requested, which will cause a slight delay in response time on the first re-access to the project.

6.7 DataLinks

Rave Server will load its data driver links from the "DataLinks" subdirectory.

6.8 Configuring Server

Independently of whether the server is installed as a service or as an application, runtime parameters can still be configured using the configuration tool, which is installed with the server. The Rave Server Configurator consists of three tabs:

1. General
2. License Keys
3. Packages

In the General tab, the port on which Rave Server is running can be changed. By default the server runs on port 4330. Generally there is no need to change this value unless there is some special reason (such as firewall restrictions) that requires use of a more standard port. If the server is run on port 80 (which is the default HTTP port), please note that any web servers currently running will have to be shutdown in order to avoid conflicts.

In the License Keys tab, additional license keys may be added or excess license keys may be removed. Please make sure that when entering a license key that it is typed exactly as it has been provided. A mistyped key will result in the Configurator not validating the key and thus, no error will be returned if incorrect values are entered.

In the Packages tab, packages can be added or removed from the server package list. Any entries that are added here will be launched when the server is started.

Changes made in any of the tabs will not take effect until the server is restarted.

6.9 Report Types

With Rave Server only non-database reports or reports containing SQLDataView components can be produced. Direct DataViews are not supported.

When generating a report using Rave Solo, there are two properties associated with the SQLDataView: AuthDesign and AuthRun. These contain parameters needed to connect to an RDBMS (relational database management system). When running the report using Rave Server, the parameters in AuthRun are used to actually connect to the database and not the values in AuthDesign. This allows development and deployment on different machines.

For debugging purposes Rave Server can be started with the **/AuthDesign** parameter and it will use the AuthDesign information instead of the AuthRun info.

Executing the server with the **/notray** option will cause the server to run without appearing in the system tray.

When running a report, there are three different formats of output to choose from: HTML, PDF (Adobe Acrobat) and NDR (Rave's native format).

HTML outputs require browsers that support both HTML version 4 or higher and JavaScript. Currently, the two most used browsers, Internet Explorer and Netscape Navigator (both as of version 4), have the capabilities to correctly view reports generated by Rave.

PDF reports can be viewed using Adobe's Acrobat Reader, which is available as a free download from www.adobe.com.

A last alternative is to use Rave's native format (NDR). These can be viewed using Rave Viewer, also available without charge from www.nevrona.com. Although it requires more effort from the users end (or from the system administrator), by using this format more functionality can be provided, such as zooming, and ultimately converting to HTML or PDF directly from the viewer.

If either Adobe Acrobat Reader or Rave Viewer is installed on the end-user system, the browser will automatically launch them when the corresponding report is returned from a request. If the message "Save To Disk" appears when a report is selected and clicked, this means that the application hasn't been installed on the system or somehow has not been associated to the file extensions used.

6.10 Web Browser

Rave Server can serve Reports directly to a web browser. Reports can be integrated into an existing web site by using specialized URLs or HTML forms.

6.11 Link Web Reports with URL's

A report can be linked to any web site by just including a link that will call the report server with the appropriate parameters.

A report URL consists of 4 parameters:

1. Report action
2. Project name
3. Report Name
4. Output format (Optional)

Report action

The action can either be GenerateReport or ExecuteReport. The first action generates a report and displays it to the user. The second does much the same, except for the fact that it previously checks to see if the report has been generated. If so, it then displays a cached copy. The latter is much more efficient and only effects HTML formatted reports.

Project name

Indicates the project file the report is contained in. Rave can have more than one report per project (rav) file.

Report name

Specifies which report of the ones contained in the project file should be displayed.

Output format (Optional)

Choose the display format desired for the report. The options are HTML, PDF and NDR. By default it uses HTML.

There are two permitted syntaxes for specifying reports:
**http:// {ip_address} : {port_number} / {[ER|GR]} / {project } /
{report} [/HTML] [/PDF] [/NDR]**

?ND_Project={project}&ND_Report={report}[&ND_Output={[HTML|PDF|NDR]}]

Both syntaxes have the same capabilities and work in the same way. The only difference is that when using a form (which we will cover below), the URL is generated using the second syntax.

6.12 Reports with HTML Forms

If the reports that need to be shown depend on user-input, you can use HTML forms to perform this. Instead of using links, you can place an HTML form with fields indicating the options of the report.

To do so, place HTML form fields in your form that allow the user to select at least 3 of the 4 (note that output format is optional) parameters explained previously. The ACTION of the form should be:

**<FORM ACTION= "http:// {ip_address} : {port_number} /
{[ExecuteReport|GenerateReport]}" METHOD="[GET] | [POST]">**

6.13 Application Clients

Rave Server can also serve applications and act as a middle tier. This allows reports to be run on a centralized managed high power server that has a fast network connection to the database. Since only the actual reports are returned to the application and the reports are executed on the server, the application does not require a high-speed connection to the report server. The application does not require any database connectivity either.

The application passes all parameters and report information to the server, and retrieves the generated report via HTTP and displays it within the application, exactly as if the report had been generated from the application.

Note that to make use of this alternative, the client application would be required to support HTTP calls and responses.

Units

Chapter

7 Units

The following is a list of the source files that make up Rave along with a short description of what is contained within.

WARNING

The RvINFO.OBJ and RpRave.DCU files contain your ID and other pertinent information about the Rave system you purchased. As such, they are not to be distributed, released or shared by any other users.

7.1 Component Sets

Rave Component Sets – RvCsXxxx

Filename	Classes / Components	Description
RvCsStd		Standard Components
RvCsDraw		Graphics Components
RvCsRpt		Report Components
RvCsBars		Barcode Components

7.2 Compression Methods

Compression Methods – RVCmXxxx

Filename	Classes / Components	Description
RvCmHuff		Compression functions

7.3 Language Engine

Rave Language Engine – RvLE####

Filename	Classes / Components	Description
RvLECode		Rave Language Engine - Code Generators
RvLEDefine		Rave Language Engine - Constants and Types
RvLEExpr		Rave Language Engine - Expression Classes
RvLEID		Rave Language Engine - ID Classes
RvLEModule		Rave Language Engine - Module Classes
RvLERun		Rave Language Engine - Execution Engine
RvLESystem		Rave Language Engine - System Functions
RvLEType		Rave Language Engine - Type Classes
RvLEUtil		Rave Language Engine - Utility Functions

7.4 Language Compiler

Rave Language Compiler – RvL#Comp

Filename	Classes / Components	Description
RvLDComp		Delphi Syntax Compiler

7.5 Project Editors

Project Editors – RvWz####

Filename	Classes / Components	Description
RvWzSimp		Simple Table Listing Report Wizard
RvWzMD		Master-Detail Report Wizard
RvWzExpert		Report Expert Wizard

7.6 Property Editors

Property Editors – RvPE####

Filename	Classes / Components	Description
RvPEStrL		String list property editor
RvPEBand		Band property editor
RvPEImg		Image (Bitmap / Metafile) property editor
RvPERLoc		ReprintLoc property editor

7.7 Units - Archived

Units – Archived .

Filename	Classes / Components	Description
RpShell	TDetailShell TMasterShell TReportShell	Banded style report framework components
RpLabel	TLabelShell	Label style report framework component
RpTable	TTablePrinter	Table printer framework component
RpTabItm	TTableItem	Base class for TTableColumn and TTableSection
RpTabCol	TTableColumn	Table column component
RpTabSec	TTableSection	Table header / footer component
RpDbTabl	TDbTablePrinter	Database table printer framework component
RpSecFrm		Property editor for TReportSection
RpCedFrm		Component editor for TTablePrinter and TDbTablePrinter
RpAddFrm		Add Fields dialog for above component editor
RpSect	TReportSection	Report section class

7.8 Units - Rave

Rave Units

Filename	Classes / Components	Description
RvDefine		Types and constants for Rave Class Library
RvUtil		Utility functions for Rave Class Library
RvClass	TRaveObject TRaveComponent TRaveControl TRaveProjectItem TRavePage TRaveDesigner TRavePip	Base and designer classes for all Rave components
RvData	TRaveDataField TRaveDataView	DataView classes and property editors
RvProj	TRaveReport TRaveProjectManager	Report and Project components
RvTool	TRavePropertyEditor TRaveComponentEditor TRaveProjectEditor	Editor classes for designer experts
RpDefine	TRpComponent	Base class to all components, types and constants
RpBase	TBaseReport	Base class for TCanvasReport and TReportFiler
RpCanvas	TCanvasReport	Base class for TReportPrinter and TFilePrinter
RpFiler	TReportFiler	Report file output component
RpRTFilr	TRTFFiler	RTF (Rich Text Format) filer component
RpTxFilr	TTextFiler	Text filer component
Rprinter	TReportPrinter	Printer output component
RpFPrint	TFilePrinter	Report file to printer output component
Rpreview	TFilePreview	Report file to screen output component
RpSystem	TRvSystem	ReportSystem component
RpSetFrm	TRpSetupForm	Setup form for TRvSystem
RpStaFrm	TRpStatusForm	Status form for TRvSystem
RpPreFrm	TRpPreviewForm	Preview form for TRvSystem
RpMemo	TMemoBuf	Memo buffer class
RpDbUtil	TDbMemoBuf	Database memo buffer class and other DB utils
RpDevice	TRpDevice	Printer device manager class
RpReg		Component registration code for Rave

Rave Component Events (scripting)

RAVE components	On After Print	On After Report	On Before Print	On Before Report	On Calc Value	On Get Group	On Get Text	On Get Value	On Mirror Value	On Print	Uses
Project Level	X	X	X	X							
DataField	X	X	X	X							RvData
DataView	X	X	X	X							RvData
Page	X	X	X	X							RvClass
ProjectManager	X	X	X	X							RvProj
Report	X	X	X	X							RvProj
All BAR CODE components	X	X	X	X							RvCsBars
All DRAWING components	X	X	X	X							RvCsDraw
REPORT components											RvCsRpt
Band	X	X	X	X		X					
CalcController	X	X	X	X							
CalcOp	X	X	X	X				X			
CalcText	X	X	X	X	X		X				
CalcTotal	X	X	X	X	X			X			
DataBand	X	X	X	X		X					
DataCycle	X	X	X	X							
DataMemo	X	X	X	X							
DataMirror	X	X	X	X					X	X	
DataText	X	X	X	X			X				
Region	X	X	X	X							
STANDARD components											RvCsStd
Bitmap	X	X	X	X							
FontMaster	X	X	X	X							
Memo	X	X	X	X							
MetaFile	X	X	X	X							
PageNumInit	X	X	X	X							
Section	X	X	X	X						**X**	
Text	X	X	X	X							

Index

- C -

- D -

- E -

- F -

- I -

- P -

- R -

- S -

- T -

- U -

- W -