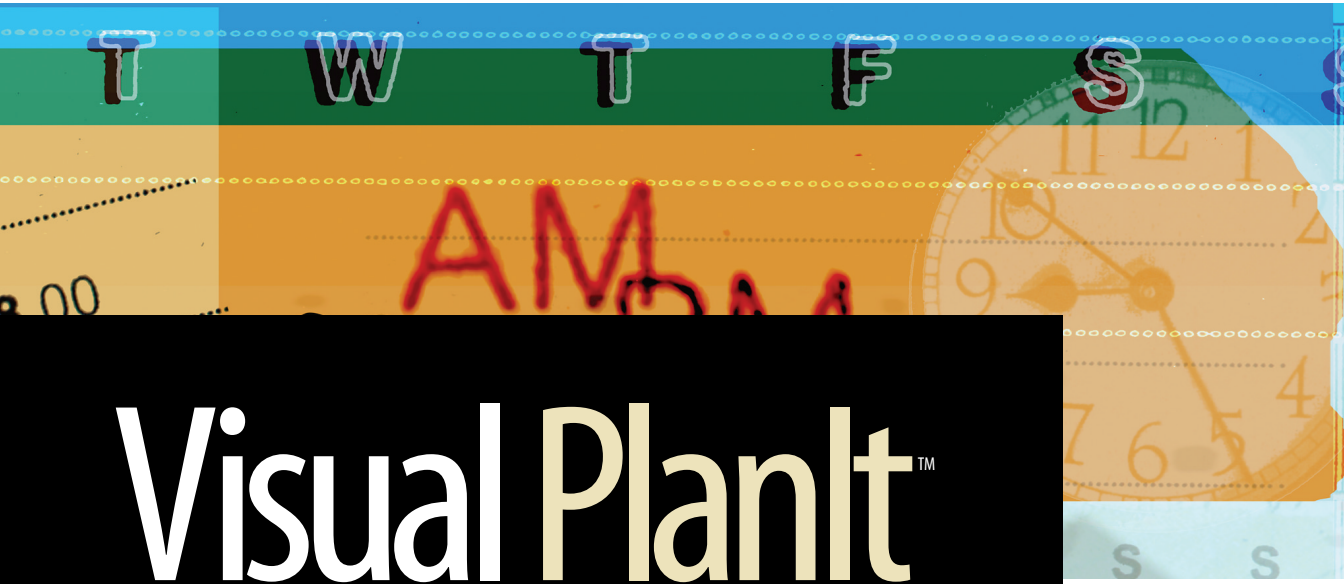
**Personal information management VCL toolkit for Delphi and C++Builder**




# Visual PlanIt™

Add powerful scheduling, contact and task management capabilities to your applications

TURBOPOWER®
Software Company

# Visual PlanIt™

TurboPower Software Company
Colorado Springs, CO

www.turbopower.com

© 2002 TurboPower Software Company. All rights reserved.

# Table of Contents


## Chapter 1: Introduction and Tutorial ..... 1

System Requirements ..... 6
Installation ..... 7
Organization of this Manual ..... 10
Technical Support ..... 13
Visual PlanIt Quick-Start Tutorial ..... 14
Simple Visual PlanIt PIM Application ..... 15
Overview of Printing with Visual PlanIt ..... 19
Printing Tutorial ..... 24

## Chapter 2: Visual PlanIt Building Blocks ..... 27

TVpComponent Class ..... 28
TVpCustomControl Class ..... 30
TVpLinkableControl Class ..... 32
TVpCategoryColorMap ..... 35
TVpCategoryInfo ..... 37
TVpFont Class ..... 39
TVpCollectionItem ..... 41
TVpCollection ..... 43
TVpTimeSlotColor ..... 46
TVpDialogPlacement Class ..... 48
TVpAttributeItem ..... 50
TVpAttributes ..... 52
TVpPrintShape ..... 54
TVpPrintCaption Class ..... 57
TVpPrintFormatElementItem Class ..... 61
TVpPrintFormatElement Class ..... 68
TVpPrintFormatItem Class ..... 70
TVpPrintFormat Class ..... 73
TVpPrinter Class ..... 75

## Chapter 3: Visual PlanIt Data Storage Building Blocks ..... 89

Overview ..... 90
Visual PlanIt DataStore Components ..... 91
TVpCustomDataStore Class ..... 92
TVpCustomDBDataStore ..... 102
TVpBDEDataStore Class ..... 103
TVpDependentInfo Class ..... 109

Visual PlanIt Internal Data Storage Classes . . . . . 111
TVpResources Class . . . . . 112
TVpResource Class . . . . . 115
TVpSchedule Class . . . . . 120
TVpEvent Class . . . . . 125
TVpTasks Class . . . . . 132
TVpTask Class . . . . . 136
TVpContacts Class . . . . . 140
TVpContact Class . . . . . 144
TVpTimeRange Class . . . . . 154

Chapter 4: Visual PlanIt Database Design . . . . . 157
Resources Table . . . . . 158
Events Table . . . . . 159
Contacts Table . . . . . 160
Tasks Table . . . . . 162

Chapter 5: Visual PlanIt Components . . . . . 163
TVpControlLink Component . . . . . 165
TVpCustomLEDLabel Class . . . . . 169
TVpLEDLabel Component . . . . . 170
TVpCustomClock Class . . . . . 175
TVpClock Component . . . . . 176
TVpHandOptions Class . . . . . 183
TVpClockDigitalOptions Class . . . . . 188
TVpCalendar Component . . . . . 190
NavBar Component . . . . . 198
TVpCustomNavBar Class . . . . . 200
TVpNavBar Component . . . . . 201
TVpNavFolder Class . . . . . 214
TVpNavBtnItem Class . . . . . 217
TVpPrintPreview Component . . . . . 220
TVpPrintFormatComboBox Component . . . . . 226
TVpCustomDateEdit Class . . . . . 228
TVpDateEdit Component . . . . . 229
TVpResourceCombo Component . . . . . 234
TVpDayView Component . . . . . 236
TVpCHAttributes Class . . . . . 248
TVpRHAttributes Class . . . . . 250
TVpTimeSlotColor Class . . . . . 252
TVpWeekView Component . . . . . 254
TVpMonthView Component . . . . . 261

TVpContactGrid Component . . . . . 268
TVpContactHeadAttr Class . . . . . 276
TVpTaskList Component . . . . . 278
TVpTaskDisplayOptions Class . . . . . 285
TVpBaseDialog Class . . . . . 288
TVpResourceEditDialog Component . . . . . 290
TVpEventEditDialog Component . . . . . 291
TVpContactEditDialog Component . . . . . 292
TVpTaskEditDialog Component . . . . . 293
TVpNotificationDialog Component . . . . . 294
TVpPrintPreviewDialog Component . . . . . 295

Chapter 6: Visual PlanIt Example Applications . . . . . 297
Scheduler . . . . . 298
DayView Demo . . . . . 298
ContactList Demo . . . . . 299
TaskList Demo . . . . . 299
Printer Demo . . . . . 299

Identifier Index . . . . . i

Subject Index . . . . . ix

# Chapter 1: Introduction and Tutorial

Visual PlanIt is a collection of components that radically simplify the creation of Personal Information Management (P.I.M.) style applications in Borland Delphi and C++Builder.

Visual PlanIt's controls are optimized and highly integrated with the Delphi and C++Builder development environments. Since the VCL components compile directly into your applications there's nothing extra to distribute and install with your programs unless you decide to use run-time packages. Best of all, Visual PlanIt's complete source code is included so that you can see exactly how it works. A high level overview of the components follows.

*Figure 1.1: Component Palette.*

## TVpCustomDataStore class

The basic Visual PlanIt database connectivity is built in to the TVpCustomDataStore class. It is the ultimate ancestor of all of the DataStore classes, but is not a complete implementation.

The DataStore component is responsible for managing the connection to the external database, creating the tables when needed, and providing a data pipeline to the various TVpLinkableControl descendants (DayView, WeekView, MonthView, etc.). The most common DataStore components are the TVpBDEDataStore and the TVpFlexDataStore. They are installed on the component palette by default.

Other DataStore descendants are included with the library, but are database specific and therefore cannot be deployed as compiled packages. For a complete, up to date listing of the DataStore components, see the ReadMe.hlp file.

The other DataStore components will have to be compiled and installed onto the palette before you can use them. You will also find detailed instructions for compiling each individual DataStore package in the ReadMe.hlp file.

In the interest of brevity, the BDEDataStore is the one that will be used in these tutorials. The other DataStore components implement the same interface and will function with the Visual PlanIt components in an identical manner. For more information see the complete documentation on page 92.

## TVpControlLink component

The ControlLink component is responsible for maintaining a list of all of the related components and synchronizing them with each other. It will automatically identify and connect to all TVpLinkableControl descendants when it is dropped on the form. Additionally, when a TVpLinkableControl is dropped on a form it will automatically connect to the first ControlLink it finds. For more information see the complete documentation on page 165.

## TVpDayView component

The TVpDayView is a TVpLinkableControl descendant that displays the events of one or more days, for one resource. The days can be broken up into increments of 60, 30, 20, 15, 10, 6, or 5 minutes. It allows in-place editing and provides an event editing dialog box. It connects to a DataStore component for data access and a ControlLink for synchronization. For more information see the complete documentation on page 236.

## TVpWeekView component

The TVpWeekView is a TVpLinkableControl descendant that displays the events of one week for one resource. It allows in-place editing and provides an event editing dialog box. It connects to a DataStore component for data access and a ControlLink for synchronization. For more information see the complete documentation on page 254.

## TVpMonthView component

The TVpMonthView is a TVpLinkableControl descendant that displays the events of one month for one resource. It does not allow for in-place editing nor does it provide an event editing dialog box. It connects to a DataStore component for data access and a ControlLink for synchronization. Use a MonthView along with an associated WeekView or DayView component for editing events. For more information see the complete documentation on page 261.

## TVpContactGrid component

The TVpContactGrid is a TVpLinkableControl descendant, grid-style control designed for displaying a list of contacts. It also has in-place editing and a contact editing dialog box, as well as draggable column resizing. It connects to a DataStore component for data access and a ControlLink for synchronization. For more information see the complete documentation on page 268.

## TVpTaskList component

The TVpTaskList is a TVpLinkableControl descendant designed for displaying a list of tasks. It connects to a DataStore component for data access and a ControlLink for synchronization. It offers support for in-place editing as well as an edit dialog box. For more information see the complete documentation on page 278.

## TVpResourceCombo component

The TVpResourceCombo component provides a combo box list of available resources. When the DataStore loads, the combo boxes' items list populates itself with the description of each resource. When the user selects a resource from the list, the combo box notifies the ControlLink which forces the other components to refresh themselves. For more information see the complete documentation on page 234.

## TVpLEDLabel component

The LEDLabel is the component that is used by the TVpClock to create the digital display. It is surfaced here as a bonus for your use in your own applications. For more information see the complete documentation on page 170.

## TVpClock component

TVpClock is an analog clock that can display the system's current time, updated every second or a time as determined by the Time property to display schedules, meeting times, and so on. Both the clock face and the clock hands are customizable. It also has an option for a digital display. For more information see the complete documentation on page 176.

## TVpCalendar component

Visual PlanIt provides a flexible monthly calendar that is specially designed to synchronize with other Visual PlanIt components via Visual PlanIt's ControlLink component. A properly configured TVpCalendar component will reflect date changes from, and pass new date selections to all other Visual PlanIt components, which share the same ControlLink and DataStore. For more information see the complete documentation on page 190.

## TVpNavBar component

The TVpNavBar component emulates the controls you see on the left side of Microsoft Outlook™. The contents of the component are arranged by folders and items. Each TVpNavBar contains one or more folders. Within each folder are one or more folder items. By default, folders are depicted as buttons on the TVpNavBar. Clicking on a folder's button or tab makes that folder the active folder and displays its contents. Clicking on a folder item generates an event that can be used to perform some action in the program. A typical use for a TVpNavBar is to place the bar in the left side of a form with different item views displayed to the right of the bar.

The NavBar also has the ability for folders to contain other components. Set any folder's FolderType property to ftContainer and it becomes the equivalent of a TScrollBox that accepts other components dropped on it at design time. For more information see the complete documentation on page 201.

## TVpBDEDataStore component

The BDEDataStore component descends from the TVpCustomDataStore class and implements the functionality required to connect to a standard BDE database engine. For more information see the complete documentation on page 103.

## TVpDateEdit component

The DateEdit component is a simple edit control with an attached popup calendar display. It simplifies the task of entering dates into a field. For more information see the complete documentation on page 229.

## TVpResourceEditDialog component

The ResourceEditDialog is a dialog box used for adding and editing resources. A resource is the entity that owns events, contacts, and tasks. Events, contacts, and tasks cannot exist without a resource.

Unlike all of the other dialog boxes, the ResourceEditDialog is not built in to any of the other components. If you want to edit a resource, you need to either use this dialog component, or create your own. For more information see the complete documentation on page 290.

## TVpEventEditDialog component

The EventEditDialog is the dialog box that is automatically displayed by the DayView and MonthView components. It is surfaced as a component so that you may use it in other circumstances. For more information see the complete documentation on page 291.

## TVpContactEditDialog component

The ContactEditDialog is the dialog box that is automatically displayed by the ContactGrid component. It is surfaced as a component so that you may use it in other circumstances. For more information see the complete documentation on page 292.

## TVpTaskEditDialog component

The TaskEditDialog is the dialog box that is automatically displayed by the TaskList component. It is surfaced as a component so that you may use it as you see fit. For more information see the complete documentation on page 293.

## TVpNotificationDialog component

The NotificationDialog is the alarm, or alert dialog box, that is displayed by the DataStore components in response to an event's alarm. It is surfaced as a component so that you may use it as you see fit. For more information see the complete documentation on page 294.

## TVpPrintPreviewDialog component

The PrintPreviewDialog component is based on the PrintPreview and PrintFormatCombo components. It allows the user to select a pre-defined print format and see how that format will appear on the printed page. For more information see the complete documentation on page 295.

## TVpPrintFormatComboBox component

TVpPrintFormatComboBox provides a combo box for selecting pre-defined print formats. For more information see the complete documentation on page 226.

## TVpPrintPreview component

TVpPrintPreview provide an image of how a print format will appear on the printed page. For more information see the complete documentation on page 220.

# System Requirements

Visual PlanIt is a pure, 32-bit library. Some decisions were made early in the development cycle that force the library to only support Delphi 4 and C++Builder 4 and above. For optimal performance, you must have the following hardware and software:

1. A computer capable of running Microsoft Windows XP, Windows 2000, Windows NT, Windows ME, Windows 98, or Windows 95. At least 128MB of RAM is recommended.
2. Borland Delphi Version 4 or later or C++Builder 4 or later.
3. An installation of all Visual PlanIt files and example programs for one compiler requires about 5MB of disk space.

# Installation

Visual PlanIt can be installed directly from the CD-ROM.

## The setup program

Insert the TurboPower Product Suite CD-ROM and follow the instructions presented by the setup program.

SETUP installs Visual PlanIt in C:\TurboPower\Visual PlanIt by default. You can specify a different directory if desired. You can choose a full or partial installation. Full installation is recommended, but if you need to conserve disk space you can install only selected portions of Visual PlanIt.

SETUP installs to your start menu under Programs\TurboPower\Visual PlanIt by default (you can specify a different name, if desired).

### Installing for multiple compiler versions

Visual PlanIt supports all Delphi compilers version 4 and above and all versions of C++Builder 4 and above. However, the compiled package file format is different for each version. The Visual PlanIt setup program allows you to select compiler support for each of the different, supported versions of Delphi or C++Builder.

By default, only the installed compilers will be selected when Visual PlanIt is installed for the first time. If Visual PlanIt is being re-installed or upgraded, only support for the previously supported compilers will be automatically selected. If you have installed a new compiler and wish to install Visual PlanIt support for that compiler, you must explicitly select support for that compiler.

### Installing into Delphi

If you choose to install Visual PlanIt into a supported Delphi IDE and you choose to install the help files, example applications, and on-line documentation, you will find the files located in the following directories:

Help: VisualPlanIt\HELP\DELPHI

Examples: VisualPlanIt\EXAMPLES\DELPHI

Documentation: VisualPlanIt\PDF

### Installing into C++Builder

Visual PlanIt can be used with either Delphi or C++Builder. If you choose to install Visual PlanIt into a supported C++Builder IDE and you choose to install the help files, example applications, and on-line documentation, you will find the files in the following directories:

Help: VisualPlanIt\HELP\CBUILDER

Examples: VisualPlanIt\EXAMPLES \CBUILDER.

Documentation: VisualPlanIt\PDF

Header and object files for each version of C++Builder are installed into the \HPP* (Where '*' stands for your compiler version) directory located in the Visual PlanIt root directory. ' For example header files for C++Builder 5 will be located in the VisualPlanIt\HPP5 directory. If you need to generate header and object files to support other C++Builder compilers, you can either re-install Visual PlanIt and select support for the new compiler, or use the DCC32.EXE program to compile the VPREG.PAS file using -jphn as the command line options.

In the event that you need to modify the Visual PlanIt source code and rebuild the packages, you must move the newly generated *.hpp files from Visual PlanIt's root directory into the proper \HPP* directory.

## Component installation

### Delphi and C++Builder packages

To avoid version conflicts with applications using different versions of the Visual PlanIt packages, each version of Visual PlanIt comes with packages using slightly different names. This allows you to use the same Visual PlanIt installation with multiple versions of Delphi and C++Builder.

The Visual PlanIt packages have the following form:

VNnXxVv.BPL

Nn is the version number of Visual PlanIt.

Xx are product-specific characters.

Vv is the version of VCL supported.

For example:

V100_D50.BPL is the Visual PlanIt version 1.00, design time package for Delphi 5.

Visual PlanIt packages are automatically installed into Delphi and C++Builder if Visual PlanIt is installed using the setup program.

Note: The IDE's MUST be shut down prior to running the installer, or the packages and help files will fail to install properly.

If you need to install the packages manually, you can install them using the IDE's Install Packages menu item on the IDE's Component menu.

Be sure to alter the library path so that it includes the path to the Visual PlanIt source files (Tools | Environment Options | Library Page) or add the Visual PlanIt path to your system Path environment. This allows the compiler to find the Visual PlanIt source files when required.

For C++Builder you must also make sure that the proper hpp* path appears in the library search path.

The component palette will be updated with a new Visual PlanIt tab.

### Run-time packages

Visual PlanIt does not ship with pre-compiled run-time packages. However, there is a basic run-time package project that you can use to create your own run-time packages.

Open the VpRuntime.dpk package project in the IDE, and click the Compile button to create a standard run-time package that contains all of the Visual PlanIt components and classes. You may also add or remove files from the package project to create a package that only contains the functionality used by your application.

## Installing integrated help

The Visual PlanIt help system is typically installed into Delphi and C++Builder by the SETUP program, but steps for manual installation are provided if the need arises.

### Installing integrated help for Delphi 4 - 6 and C++Builder.

To install the help into Delphi or C++Builder, use the Help | Customize option from the IDE. Select the Contents page and browse to the desired Visual PlanIt \Help directory and load the *.CNT file. In the Index and Link pages, do the same thing with the *.HLP files. When you are finished, select File | Save Project.

## Demonstration and example programs

Look for examples in the Visual PlanIt\EXAMPLES directory. Visual PlanIt examples are provided for both Delphi and C++Builder.

The demonstration programs are documented at the end of this book on page 297. The example programs are provided so you can see how to use the various Visual PlanIt components. See the source code of these programs for more information.

# Organization of this Manual

This manual is organized as follows:

1. Chapter 1 is an introduction and a how-to of sorts, describing how to create basic Visual PlanIt applications.
2. Chapter 2 describes the base classes that are used by all of the Visual PlanIt components.
3. Chapter 3 describes the Visual PlanIt Data-Storage mechanism.
4. Chapter 4 describes a generalized back-end database layout for the Visual PlanIt DataStore.
5. Chapter 5 describes the Visual PlanIt Components.
6. Chapter 6 describes the example applications.
7. A separate identifier index and subject index are provided.

Each chapter starts with an overview of the classes and components discussed in that chapter. The overview also includes a hierarchy for those classes and components. Each class and component is then documented individually, in the following format:

## Overview

A description of the class or component.

## Hierarchy

Shows the ancestors of the class being described, generally stopping at a VCL class. The hierarchy also lists the unit in which each class is declared and the number of the first page of the documentation of each ancestor. Some classes in the hierarchy are identified with a number in a bullet: ❶. This indicates that some of the properties, methods, or events listed for the class being described are inherited from this ancestor and documented in the ancestor class.

## Properties, methods, and events lists

The properties, methods, and events for the class or component are listed. Some of these may be identified with a number in a bullet: ❶. In these cases, they are documented in the ancestor class from which they are inherited.

## Reference section

Details the properties, methods, and events of the class or component. These descriptions are in alphabetical order. They have the following format:

- Declaration of the property, method, or event.
- Default value for properties, if appropriate.
- A short, one-sentence purpose. A symbol is used to mark the purpose to make it easy to skim through these descriptions.
- Description of the property, method, or event. Parameters are also described here.
- Examples are provided in many cases.
- The “See also” section lists other properties, methods, or events that are pertinent to this item.

Throughout the manual, the symbol is used to mark a caution. Please pay special attention to these items.

## On-line help

Although this manual provides a complete discussion of each component, keep in mind that there is an alternative source of information available. Help is available when you press <F1> with the caret positioned on the property, method, event, or component.

## Naming conventions

To avoid class name conflicts with components and classes included with Delphi or from other third party suppliers, all Visual PlanIt class names begin with “TVp.” The “Vp” stands for Visual PlanIt. Source files also start with “Vp”, making it easier to distinguish Visual PlanIt source files from other files.

“Custom” in a component name means that the component is a base class for descendent components. Components with “Custom” as part of the class name do not publish any properties; instead descendants will publish those properties that are applicable to the derived component. If you create descendent components, use these custom classes and publish only the properties that you need instead of descending from the component class itself.

## Design conventions

Within a class, Visual PlanIt chooses between dynamic and virtual methods on the basis of performance requirements. Virtual methods are chosen for time-critical operations that might need to be overridden in descendants. Dynamic methods, which consume less code space, are used when performance is less important. Static methods are used when there is no reason ever to override the method.

All methods and variables in Visual PlanIt are either public or protected (as opposed to private). If you look through the source code, you may come across sections of class declarations similar to this:

```
TVpFoo = class(TBaseFoo)
protected {private}
    FAutoIndent      : Boolean;
    FBorderStyle     : TBorderStyle;
    FByteLimit       : LongInt;
    FFont            : TVpFont;
    FHideSelection   : Boolean;
    FHighlightColors : TI1Colors;
    FInsertMode      : Boolean;
```

These should be treated as private sections. In other words, you should not routinely access these variables or methods from descendant classes. The section is defined as protected rather than private to give you access in cases that we may not have anticipated. If you access a method or variable in one of these sections, you should do so as a last resort and only after investigating and understanding the potential effects on other areas of the component's code.

Each Visual PlanIt class may include additional fields and methods that are protected and are documented only in the source code. It is unlikely that you will ever need to use or understand these, but the interested programmer should study the source code for additional ideas.

# Technical Support

The best way to get an answer to your technical support question is to post it in the Visual PlanIt newsgroup on our news server (news.turbopower.com). Many of our customers find the newsgroups a valuable resource where they can learn from other's experiences and share ideas in addition to getting answers to questions. In addition, the newsgroups are monitored by an elite task force of volunteer experts, called TPX members. These people are uniquely qualified to help with our products.

To get the most from the newsgroups, it is recommended that you use dedicated newsreader software. More newsgroup information can be found at www.turbopower.com/tpslive.

Note: Newsgroups are public so please do not post your product serial number, 16-character product unlocking code or any other private numbers (such as credit card numbers) in your messages.

If you don't have access to a newsreader, or are on a restricted Internet line, you may send a support request to mailsupport@turbopower.com.

# Visual PlanIt Quick-Start Tutorial

Visual PlanIt is, among other things, a library of visual controls for creating PIM style applications. It consists of a number of visual controls as well as some specialized data-storage components, which work in conjunction to create powerful applications with a minimum amount of coding. Each section of this chapter provides one exercise that will help you to master the skills and understanding that you will need to effectively master the library, and all of its components.

The components, which make up the Visual PlanIt library are all documented in detail in other chapters of this manual. They are presented here in an abbreviated format, in order that you will be more familiar with them as you go through the tutorials.

# Simple Visual PlanIt PIM Application

This is a simple (one line of source code) tutorial that will help you get up to speed with Visual PlanIt as quickly as possible. It is highly recommended that you read it completely before exploring the library, so that you will understand the basics of how to use the library. If you have any questions, please don't hesitate to visit the public Visual PlanIt newsgroup at news.turbopower.com.

Here's how to create a basic PIM style application:

Follow the steps completely before compiling and running the application for the first time.

1. Open your IDE and create a new application.

2. Drop a TPanel on the form and set its properties as follows:

```
Align            alTop
Caption          ''
```

3. Drop a TVpBDEDataStore component on the form and set its properties as follows:

```
AliasName        "Visual PlanIt"
AutoConnect      True
AutoCreate       True
AutoCreateAlias  True
ResourceID       0
```

4. Drop a ControlLink component on the form.

5. Drop a DayView component on the form and set its properties as follows:

```
Align             alLeft
ShowResourceName  False
```

Note: The DayView, WeekView, MonthView, ContactGrid, and TaskList are TVpLinkableControl descendants, so they will automatically connect to the DataStore and ControlLink when they are dropped on the form.

6. Drop a MonthView component on the form. Place it on the form, just to the right of the DayView, Even with the top of the DayView. Set its properties as follows:

```
DayNameStyle     dsShort
ShowEvents       False
```

7. Drop a ContactGrid component on the form, Place it just beneath the MonthView. There are no properties to set.

8. Drop a TaskList component on the form and set its properties as follows:

```
Align          alRight
```

9. Drop a TVpResourceCombo on the panel. (You may want to place a label next to it that says, “Resource”)

10. Drop a TVpResourceEditDialog component on the panel.

11. Drop a TButton on the panel, just to the right of the ResourceCombo, and set its properties as follows:

```
Caption          “Create a new resource”
Width            130
```

12. Define the TButton’s OnClick event handler as follows (Note: This is the only source code in the tutorial):

Delphi:

```
procedure TForm1.Button1Click(Sender : TObject);
begin
  VpResourceEditDialog1.AddNewResource;
end;
```

C++Builder:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  VpResourceEditDialog1->AddNewResource();
}
```

Your application's only form will look something like Figure 1.2.

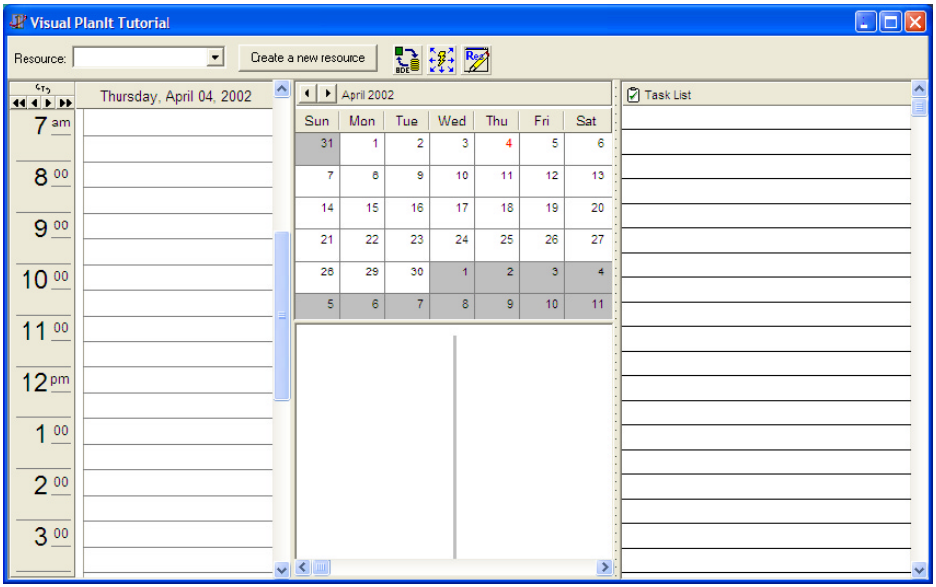



*Figure 1.2: Tutorial Application's Main Form.*

13. Make sure to save the project.

14. If you have the debugger set to stop on Delphi Exceptions (In Delphi 5 click Tools | Debugger Options click the "Language Exceptions" tab and it is the "Stop on Delphi Exceptions" check box at the bottom of the form.), you will see exceptions as the DataStore attempts to open each table. Click OK on each one and press F9. Each table will be created. There are 4 tables to create, so you will get 4 errors. These errors are caught at normal run time, you will only see them in the debugger.

    If you de-select the "Stop on Delphi Exceptions" check box, you won't get those exceptions in the debugger.

15. Compile and run the application.

The new database is empty so you will not be able to create any events, tasks, or contacts until you create a resource.

To explore how you can interact with the application, try the following:

- Double-click on the DayView to add a new event.
- Double-click on the TaskList to add a task.
- Double-click on the ContactGrid to add a contact
- Click around (and move around with the arrow keys) on the MonthView and notice how the DayView is in-sync (made possible by the work of the ControlLink.)
- Shut down the application and add a WeekView.
- Select a day in the past. Add a new event. Set its alarm. Save it. It should trigger a reminder dialog box as soon as the next minute passes. It will trigger the alert immediately if you shut down the application and re-start it.

# Overview of Printing with Visual PlanIt

The Printer property of the TVpControlLink component maintains the print formats used. Visual PlanIt uses a sophisticated printing and page rendering mechanism that supports multiple print formats, each containing multiple print elements.

The print formats are associated with a TVpControlLink component. Each control link can have its own set of print formats. Print formats can either be created at design time, or created in an XML file and loaded at run time.

The basic structure is the TVpPrinter class (the Printer property of the TVpControlLink component) contains extra information needed for printing in addition to a PrintFormats property. The print format property contains a list of print formats. Each print format contains a list of components that are needed to print the page.

To access the print formats at design time, open up the Printer property on a TVpControlLink component. This part of the print format stores information that is needed, but not part of the print format. This information includes things like the start and end of days for TVpDayView components as well as the granularity of that component. Since the start and end times is more of a matter of personal preference as opposed to an integral part of the print format, the information is stored here. These fields are not saved or loaded in the XML file.

The PrintFormats property of the TVpPrinter class is where the actual definition of the print format starts. This is a TCollection in which every item in the TCollection is a print format. Each print format has several properties. The FormatName and Description properties give the format a name and an optional detailed description. The FormatName is used by other components to lookup the print format and is a required property. The DayInc and DayIncUnits properties are used to describe how many days are represented on a single page. Depending on the elements that are used in the print format, the number of days represented on a single page can vary greatly.

The actual print elements are accessed via the Elements property. The Elements property is also a TCollection in which each item represents an individual print element for this print format. The type of element to print is controlled by the ItemType property. Most of the item types match directly up to Visual PlanIt components. However, the itCaption and itShape item types are pseudo-elements used to add simple shapes and text to the print format. These properties are defined in the Shape and Caption properties of the element. The itCaption element also supports the insertion of variables. Several variables are automatically created that represent useful things about the page that is being printed. Examples of these are page numbers and dates (in various formats). Additional variables

can be added at run time and variables can even be modified while they are being looked up. The documentation on the TVpPrintCaption and TVpPrinter classes contains more information on the print caption variables.

The elements are positioned on the page using the Left, Top, Width, Height, and Measurement properties. Measurement defines how the Left, Top, Width, and Height properties will be used. By default, items are positioned using a percentage of the page. Elements can also be positioned using inches or pixels directly. As a rule of thumb, all elements in a print format should use the same value setting for the Measurement attribute (see Table 1.2 on page 21 for possible values).

If the element uses date information (like DayViews and WeekViews), the starting date of the element on the page is modifiable by using the DayOffset and DayOffsetUnits properties. This is useful in cases where multiple instances of the same element occur on the same page. For example, a print format consisting of two day views can use these properties so the second day view shows the day after the first day. When doing this, it is important to go back to the print format and adjust the DayInc and DayIncUnits properties to correctly reflect the number of days that is actually rendered.

The rotation property allows the element to be rotated in 90-degree increments.

When printing, the properties of components on the form will be used to determine how the component will be printed. If no component can be found, a default one will be created and the default values used.

### Print format XML layout

The LoadFromFile and SaveToFile methods of the TVpPrinter class maintain the print formats as an XML file.

The root node of the XML file is the Visual PlanIt PrintFormats element. This element consists of a single attribute, Version, which contains the version level of the print formats. The version stamp consists of three digits separated by dots. The TVpPrintFormat class contains zero or more of the PrintFormat element.

The PrintFormat element has several attributes. These are listed in Table 1.1.

Table 1.1: *PrintFormat attributes*

| Attribute | Meaning |
|---|---|
| DayIncrement | Numeric value of the number of days represented in a page. This value is modified by the DayIncrementUnits attribute. |
| DayIncrementUnits | The units that DayIncrement is measured in. Legal values for this attribute are Day, Week, Month or Year. |
| Description | Description of the print format. |
| Name | Name of the print format. This is required and must be unique. |

The PrintFormat element can contain one or more Element elements. The attributes of the Element are listed in Table 1.2.

Table 1.2: *Element attributes*

| Attribute | Meaning |
|---|---|
| DayOffset | The date offset used to render this element. This value is an integer number of days that will be modified by the DayOffsetUnits attribute. |
| DayOffsetUnits | The units that the DayOffset attribute will be measured in. Legal values for this attribute are Day, Week, Month or Year. |
| Height | Floating point value indicating the height of the rectangle the element will be rendered in. This value will be measured using the units specified in the Measurement attribute. |
| Item | The type of element to use. Legal values for this attribute are DayView, WeekView, MonthView, Calendar, Shape or Caption. |
| Left | Floating point value indicating the left side of the rectangle the element will be rendered in. This value will be measured using the units specified in the Measurement attribute. |
| Measurement | How the Left, Top, Width and Height elements will be measured. Legal values for this attribute are AbsolutePixel, Percent or Inches. |

Table 1.2: *Element attributes (continued)*

| Attribute | Meaning |
|---|---|
| Rotation | Amount to rotate the element. Legal values for this attribute are 0, 90, 180 or 270. |
| Top | Floating point value indicating the top of the rectangle the element will be rendered in. This value will be measured using the units specified in the Measurement attribute. |
| Width | Floating point value indicating the width of the rectangle the element will be rendered in. This value will be measured using the units specified in the Measurement attribute. |

If the Item attribute is “Shape”, Element should contain a single Shape element. The Shape element has the only the Type outline. Type indicates the type of shape to render. Legal values are Rectangle, TopLine, BottomLine, LeftLine, RightLine, TLToBRLine, and BLToTRLine.

Shape requires a single Brush element. The Brush element contains two attributes. The first of these is Color, which stores the brush color. This is a hexadecimal number laid out in the same format as a TColor property. The second attribute is the Style attribute. This stores the brush style that will be used in drawing the shape. Legal values for this attribute are Solid, Clear, Horizontal, Vertical, FDiagonal, BDiagonal, Cross, and DiagCross. Shape contains no elements.

Shape requires a single Pen element. The pen element has the attributes shown in Table 1.3.

Table 1.3: *Pen attributes*

| Attribute | Meaning |
|---|---|
| Color | The pen color. This is a hexadecimal number laid out in the same format as a TColor property. |
| Style | The style of the pen. Legal values are Solid, Dash, Dot, DashDot, DashDotDot, Clear and InsideFrame. |
| Width | Width of the pen. This is a number indicating the number of pixels wide the pen is. |

If the Item attribute is “Caption”, a single Caption sub-element is required. The Caption element contains a single attribute, Caption. Caption is the caption that will be printed.

The Caption element requires a single Font sub-element. The attributes for the Font element are shown in Table 1.4.

Table 1.4: *Font attributes*

| Attribute | Meaning |
|---|---|
| Bold | Indicates if the font should be bold. Legal values are True or False. |
| CharSet | The character set to use. Legal values for this attribute are ANSI, Default, Symbol, Mac, ShiftJIS, Hangeul, Johab, GB2312, ChineseBig5, Greek, Turkish, Vietnamese, Hebrew, Arabic, Baltic, Russian, Thai, EastEurope or OEM. |
| Color | The color of the text. This is a hexadecimal number laid out in the same format as a TColor property. |
| Height | The integer value representing the height of the font. |
| Italic | Indicates if the font should be italicized. Legal values are True or False. |
| Name | The name of the font. |
| Pitch | The font pitch. Legal values for this are Default, Variable or Fixed. |
| Strikeout | Indicates if the font should be struck out. Legal values are True or False. |
| Underline | Indicates if the font should be underlined. Legal values are True or False. |

# Printing Tutorial

Print formats are fairly complicated. This tutorial shows how to create print formats at design time. These formats can be saved and loaded by using the SaveToFile and LoadFromFile methods of the TVpPrinter class.

## Creating a single element print format

The simplest print format to create is a DayView that fills the entire page.

First, start a new project and drop a TVpControlLink and TVpPrintPreview component on the form. Double-click the TVpPrintPreview's ControlLink property to set it to ControlLink1.

Click on the TVpControlLink component and open up the Printer property. Double-click on the PrintFormats property to open up the PrintFormats editor. This is a TCollection editor. Each item that is created in this list is a new print format. Click Insert to add a new print format.

Each print format requires a name. Set the FormatName to "DayView." Since this print format will consist of a single day, set the DayInc value to 1 and the DayIncUnits to duDay. Double-click the Elements property to open the element editor. Like the Print formats, this is a TCollection editor. Press Insert to add a new element.

By default, a DayView component is created that fills the entire page.

Run the program. You should see a somewhat shrunken day view in the print preview.

## Creating a more complex print format

Exit the program and go back to the PrintFormats TCollection editor. Press Insert to add another print format.

Set the name of this new print format to “Two DayViews.” This new print format is going to consist of two side by side rotated DayView components. Since this will reflect two days, change the DayInc property to 2 and leave DayIncUnits at duDay. Double-click the Elements property to access the Elements property editor.

Inside the Elements editor, click Insert to add an element. Change the Height property to 50. By default, the measurements are in percentage of the page, so this will result in an element that is positioned on the top half of the page. Change the Rotation property to ra90.

Go back to the Elements editor and insert a second element. On this one, change the Height and Top properties to 50. This will put the element on the bottom half of the page. Again, change the Rotation property to ra90. To get the second day view to show the next day, change the DayOffset to one and leave the DayOffsetUnits as duDay.

Lastly, drop a TVpPrintFormatComboBox on the form and connect its ControlLink property to the TVpControlLink component. Run the program. The print format combo box will switch between two print formats.

# Chapter 2: Visual PlanIt Building Blocks



This chapter provides information on the core classes and components used by Visual PlanIt. It describes the following components and classes:

- TVpComponent
- TVpCustomControl
- TVpLinkableControl
- TVpCategoryColorMap
- TVpCategoryInfo
- TVpFont
- TVpCollection
- TVpCollectionItem
- TVpTimeSlotColor
- TVpDialogPlacement
- TVpBaseDialog
- TVpAttributeItem
- TVpAttributes
- TVpPrintShape
- TVpPrintCaption
- TVpPrintFormatElement
- TVpPrintFormatElementItem
- TVpPrintFormat
- TVpPrintFormatItem
- TVpPrinter

# TVpComponent Class

The TVpComponent class implements the Version property, along with a design-time about box dialog.

## Hierarchy

TComponent (VCL)

    TVpComponent (VPBase)

## Properties

Version

## Version — read-only property

```
property Version : string
```

Displays the library version in the Object Inspector.

The Version property also has a specialized design-time property editor that displays the Visual PlanIt about box when the ellipsis button is clicked.

# TVpCustomControl Class

This is the base class for all Visual PlanIt visual controls. It implements the Version property along with a design-time about box dialog. It also implements some internal mouse-handling logic.

## Hierarchy

TCustomControl (VCL)

TVpCustomControl (VPBase)

## Properties

Version

## Events

AfterEnter

AfterExit

OnMouseWheel

### AfterEnter — protected event

```
property AfterEnter : TNotifyEvent
```

Defines an event handler that is called when the control gets focus.

### AfterExit — protected event

```
property AfterExit : TNotifyEvent
```

Defines an event handler that is called when the control loses focus.

### OnMouseWheel — protected event

```
property OnMouseWheel : TVpMouseWheelEvent
```

```
TVpMouseWheelEvent = procedure(Sender : TObject;
  Shift : TShiftState; Delta, XPos, YPos : Word) of object;
```

Defines an event handler that is called when the mouse wheel is scrolled over the control.

### Version — read-only property

```
property Version : string
```

Displays the library version in the Object Inspector.

The Version property also has a specialized design-time property editor that displays the Visual PlanIt about box when the ellipses button is clicked.

# TVpLinkableControl Class

TVpLinkableControl is the base class for all Visual PlanIt data-bound controls. It implements the DataStore and ControlLink properties as well as the abstract LinkHandler procedure, which is where the notification messages from the ControlLink are processed.

## Hierarchy

TCustomControl (VCL)

❶ TVpCustomControl (VPBase) . . . . . 30

TVpLinkableControl (VPBaseDS)

## Properties

ControlLink
DataStore
ReadOnly
❶ Version

## Methods

GetControlType
GetLastPrintLine
LinkHandler
RenderToCanvas

## Events

❶ AfterEnter
❶ AfterExit
❶ OnMouseWheel

### ControlLink — property

```
property ControlLink : TVpControlLink
```

Synchronizes other TVpLinkableControls with each other.

The ControlLink is responsible for notifying other dependent components when important events have taken place. Other TVpLinkableControls on the form, that share the same ControlLink component, will synchronize with each other.

See also: TVpControlLink

### DataStore — property

```
property DataStore : TVpCustomDataStore
```

Provides access to the Visual PlanIt data.

The DataStore is the component that manages all of the Visual PlanIt data.

See also: TVpCustomDataStore

### GetControlType — virtual abstract method

```
function GetControlType : TVpItemType; virtual; abstract;
```

Used by the printing subsystem.

There should not be any need to call GetControlType directly.

### GetLastPrintLine — method

```
function GetLastPrintLine : Integer;
```

Used by the printing subsystem's rendering engine to determine when to stop rendering the control to the canvas.

There should not be any need to call GetLastPrintLine directly.

### LinkHandler — virtual abstract method

```
procedure LinkHandler(
  Sender : TComponent; NotificationType : TVpNotificationType;
  const Value : Variant); virtual; abstract;
```

Used by the ControlLink component to synchronize components with each other.

Do not call LinkHandler directly. Doing so could cause the control to get out of sync with its peers.

### ReadOnly — property

```
property ReadOnly : Boolean
```

Defines whether the data in the control can be edited.

### RenderToCanvas — virtual abstract method

```
procedure RenderToCanvas(
  RenderCanvas : TCanvas; RenderIn : TRect;
  Angle : TVpRotationAngle; Scale : Extended;
  RenderDate : TDateTime; StartLine : Integer;
  StopLine : Integer; UseGran : TVpGranularity;
  DisplayOnly : Boolean); virtual; abstract;
```

Used by the printing subsystem to render the control.

RenderToCanvas handles the actual rendering of the control. When the control is painted at run time, RenderToCanvas is called with the control's canvas. When the control is printed, RenderToCanvas is called with the printer's canvas. Each TVpLinkableControl descendant must define its own RenderToCanvas method.

# TVpCategoryColorMap

The CategoryColorMap maps event category index values with the appropriate color-code, and category name. Events store their category as an integer. When they are painted in the TimeGrid, the color-coding, and category name is determined via the values contained in the DataStore's CategoryColorMap.

Categories with an empty description are not displayed in the EventEditDialog.

## Hierarchy

TPersistent (VCL)

TVpCategoryColorMap (VPBase)

## Properties

Category (0 - 9)

## Methods

GetColor

GetName

### Category (0 - 9) — property

```
property Category0 : TVpCategoryInfo
property Category1 : TVpCategoryInfo
property Category2 : TVpCategoryInfo
property Category3 : TVpCategoryInfo
property Category4 : TVpCategoryInfo
property Category5 : TVpCategoryInfo
property Category6 : TVpCategoryInfo
property Category7 : TVpCategoryInfo
property Category8 : TVpCategoryInfo
property Category9 : TVpCategoryInfo
```

Groups the category name and color together in the Object Inspector.

The TCategoryInfo is comprised of the category's color and descriptive name.

### GetColor — method

```
function GetColor(Index : Integer) : TColor;
```

Returns the color value of the category specified by the value of Index.

The color may also be accessed via the desired property.

Both of the following code snippets return the color of category 0:

```
CatColor := CategoryColorMap.Category0.Color;
CatColor := CategoryColorMap.GetColor(0);
```

### GetName — method

```
function GetName(Index : Integer) : string;
```

Returns the name of the category specified by the value of Index.

The category's name may also be accessed via the desired property.

Both of the following code snippets return the description of category 0:

```
CatDescription := CategoryColorMap.Category0.Description;
CatDescription := CategoryColorMap.GetName(0);
```

# TVpCategoryInfo

Used by the TVpCategoryColorMap class to maintain the color-coding and name of one event category.

## Hierarchy

TPersistent (VCL)

TVpCategoryInfo (VPBase)

## Properties

CategoryIndex Color Description

### CategoryIndex — property

```
property CategoryIndex : Integer
```

Contains the index if this particular category.

Index is hard coded at construction. It will be between 0 and 9.

### Color — property

```
property Color : TColor
```

Contains the color-code of this particular category.

The following code snippet returns the color assigned to category 0:

```
CatColor := CategoryColorMap.Category0.Color;
```

### Description — property

```
property Description : string
```

Contains the name of this particular category.

The following code snippet returns the description of category 0:

```
CatDescription := CategoryColorMap.Category0.Description;
```

# TVpFont Class

The TVpFont class is a very simple, owned TFont with the added ability to invalidate its parent when it has been changed.

## Hierarchy

TFont (VCL)

TVpFont (VPBase)

## Methods

Changed

### Changed — protected method

```
procedure Changed; override;
```

Notifies its owner when the font has changed by calling the owner's Invalidate method.

# TVpCollectionItem

Models one of the items contained in a TVpCollection. The TVpCollectionItem is a direct descendant of the TCollectionItem class with the added Version, DisplayText, and Name properties.

## Hierarchy

TCollectionItem (VCL)

TVpCollectionItem (VPBase)

## Properties

DisplayText Name Version

## Reference Section

### DisplayText — property

```
property DisplayText : string
```

Contains the text that is displayed in relation to this item.

Not all collection items may use the DisplayText property, but those items that are displayed in controls may use this property to store a string that can be used for the item's display.

### Name — property

```
property Name : string
```

Contains the name of the particular item.

### Version — read-only property

```
property Version : string
```

Displays the library version in the Object Inspector.

The Version property also has a specialized design-time property editor that displays the Visual PlanIt about box when the ellipses button is clicked.

# TVpCollection

TVpCollection is a direct descendant of the TCollection class. It is used by some of the Visual PlanIt components for maintaining a collection of objects.

## Hierarchy

TCollection (VCL)

TVpCollection (VPBase)

## Properties

| | | |
|---|---|---|
| Count | Item | ReadOnly |

## Methods

| | | |
|---|---|---|
| Add | GetOwner | ParentForm |
| GetEditorCaption | ItemByName | |

## Events

| | | |
|---|---|---|
| OnChanged | OnGetEditorCaption | OnItemSelected |

## Reference Section

### Add — method

```
function Add : TVpCollectionItem;
```

Creates a new item in the collection and returns a reference to it.

### Count — property

```
property Count : Integer
```

Returns the number of items in the collection.

### GetEditorCaption — method

```
function GetEditorCaption : string;
```

Returns a caption, for use by a design-time editor.

### GetOwner — method

```
function GetOwner : TPersistent;
```

Returns a reference to the component that owns the collection.

### Item — property

```
property Item[Index : Integer] : TVpCollectionItem
```

Returns a particular item in the collection.

### ItemByName — method

```
function ItemByName(Name : String) : TVpCollectionItem;
```

Returns a reference to the item specified by Name. If Name is an empty string, ItemByName returns nil.

### OnChanged — event

```
property OnChanged : TNotifyEvent
```

Defines an event handler that is called when a published property for an item in the collection changes.

### OnGetEditorCaption — event

```
property OnGetEditorCaption : TVpGetEditorCaption
```

```
TVpGetEditorCaption = procedure(var Caption : string) of object;
```

Defines an event handler that is called when a call is made to GetEditorCaption.

If the event is defined, the editor caption is passed in to the event as a parameter.

### OnItemSelected — event

```
property OnItemSelected : TVpItemSelectedEvent
```

```
TVpItemSelectedEvent = procedure(
  Sender : TObject; Index : Integer) of object;
```

Defines an event handler that is called an item in the list is selected.

If the event is defined, the index of the selected item is passed in as a parameter.

### ParentForm — method

```
function ParentForm : TForm;
```

Returns a reference to the parent form of the control that owns the collection.

### ReadOnly — property

```
property ReadOnly : Boolean
```

Determines whether changes to the collection are allowed.

# TVpTimeSlotColor

Used by the DayView component to group the time specific, color coding into one place in the object inspector.

## Hierarchy

TPersistent (VCL)

TVpTimeSlotColor (VPBase)

## Properties

| | | |
|---|---|---|
| Active | Inactive | Weekend |
| Holiday | Weekday | |

## Reference Section

### Active — property

```
property Active : TColor
```

Defines the color of an active time block.

In normal business applications the active time block would be between 8:00 AM and 5:00 PM.

### Holiday — property

```
property Holiday : TColor
```

Defines the color of a holiday.

### Inactive — property

```
property Inactive : TColor
```

Defines the color of an inactive block of time.

In normal business applications the inactive time block would be between 5:00 PM and 8:00 AM the next day.

### Weekday — property

```
property Weekday : TColor
```

Defines the color of a normal weekday.

### Weekend — property

```
property Weekend : TColor
```

Defines the color of a normal weekend day.

# TVpDialogPlacement Class

Used by the TVpBaseDialog class to control run-time placement and size of the dialog box.

## Hierarchy

TPersistent (VCL)

TVpDialogPlacement (VPDlg)

## Properties

| | | |
|---|---|---|
| Height | Position | Width |
| Left | Top | |

### Height — property

```
property Height : Integer
```

Defines the height of the dialog form.

The Height property is ignored unless Position is set to mpCustom.

### Left — property

```
property Left : Integer
```

Defines the Left side placement of the dialog form.

The Left property is ignored unless Position is set to mpCustom.

### Position — property

```
property Position : TVpDialogPosition
```

```
TVpDialogPosition = (mpCenter, mpCenterTop, mpCustom);
```

Defines the placement of the dialog form.

The form's size is left as designed when Position is set to mpCenter and mpCenterTop. If Position is set to mpCustom, the dialog form is resized and placed according to the values of Left, Top, Height, and Width.

### Top — property

```
property Top : Integer
```

Defines the Top placement of the dialog form.

The Top property is ignored unless Position is set to mpCustom.

### Width — property

```
property Width : Integer
```

Defines the Width placement of the dialog form.

The Width property is ignored unless Position is set to mpCustom.

# TVpAttributeItem

TVpAttributeItem is a TCollectionItem descendent that stores a single Name/Value pair. This class is used in conjunction with TVpAttributes to store a series of variables.

Visual PlanIt uses these classes to accumulate attributes when loading the print format attributes and for storing variables when printing captions.

## Hierarchy

TCollectionItem (VCL)

❶ TVpCollectionItem (VPBase) . . . . . 41

TVpAttributeItem (VPPrtFmt)

## Properties

| | | | | | |
|---|---|---|---|---|---|
| | Collection | | Name | ❶ | Version |
| ❶ | DisplayText | | Value | | |

## Reference Section

### Collection — property

```
property Collection : TVpAttributes
```

Contains the TVpAttributes collections that this Name/Value pair belongs to.

### Name — property

```
property Name : string
```

The Name portion of the Name/Value pair.

See also: Value

### Value — property

```
property Value : string
```

The Value portion of the Name/Value pair.

See also: Name

# TVpAttributes

TVpAttributes maintains a list of Name/Value pairs. This class is used in conjunction with TVpAttributeItems to cache attributes when loading print formats and for storing variables used when printing captions.

## Hierarchy

TCollection (VCL)

    TVpAttributes (VPPrtFmt)

## Properties

Items

## Items — property

```
property Items[Index : Integer] : TVpAttributeItem
```

Contains an array of Name/Value pairs stored in this collection.

# TVpPrintShape

TVpPrintShape is a pseudo ItemType used by the Visual PlanIt print formats to allow for the printing of lines and rectangles. Setting a TVpPrintFormatElementItem's ItemType property to itShape enables this functionality. The type of shape, and the pen and brush used to draw the shape is defined in this class. The TVpPrintFormatElementItem class contains the positioning and rotation of the shape.

## Hierarchy

TPersistent (VCL)

TVpPrintShape (VPPrtFmt)

## Properties

Brush Pen Shape

## Methods

PaintToCanvas

### Brush — property

```
property Brush : TBrush
```

Defines the brush used when drawing the shape.

See also: Pen

### PaintToCanvas — method

```
procedure PaintToCanvas(ACanvas : TCanvas; ARect : TRect;
  Angle : TVpRotationAngle; Viewport : TRect);
```

```
TVpRotationAngle = (ra0, ra90, ra180, ra270);
```

Paints the shape to the specified canvas.

This method is normally called from the TVpPrinter class. In most cases, this method does not need to be called manually.

The shape is rendered to fill the rectangle specified by ARect.

The Angle parameter specifies how the shape will be rotated within the viewport specified by Viewport.

### Pen — property

```
property Pen : TPen
```

Defines the Pen used when drawing the shape.

See also: Brush

```
property Shape : TVpShapeType

TVpShapeType = (ustRectangle, ustTopLine, ustBottomLine,
  ustLeftLine, ustRightLine, ustTLToBRLine, ustBLToTRLine);
```

Defines the shape.

The shapes are defined in the following table:

| ShapeType | Description |
|---|---|
| ustRectangle | Draws a rectangle. |
| ustTopLine | Draws a horizontal line across the top of the shape. |
| ustBottomLine | Draws a horizontal line across the bottom of the shape. |
| ustLeftLine | Draws a vertical line on the left side of the shape. |
| ustRightLine | Draws a vertical line on the right side of the shape. |
| ustTLToBRLine | Draws a diagonal line from the top-left corner to the bottom-right corner. |
| ustBLToTRLine | Draws a diagonal line from the bottom-left corner to the top-right corner. |

# TVpPrintCaption Class

The TVpPrintCaption class is a pseudo ItemType used by the Visual PlanIt print formats to allow for the printing of captions. Setting a TVpPrintFormatElementItem's ItemType property to itCaption enables this functionality. The text and font used to draw the caption is defined in this class. The TVpPrintFormatElementItem class contains the positioning and rotation of the caption.

## Print caption variables

In addition to containing static text, print captions can also contain variables. When the page is either printed or rendered to a canvas, any variables found in the print caption are automatically substituted with their values. The TVpControlLink's OnGetVariable event will fire every time a variable is looked up while printing. In this event, the value substituted for the variable can be modified.

Variables are indicated by a leading '$' character followed by a series of alphanumeric characters or '_'. Any character that is not alphanumeric or a '_' indicates the end of the variable name. A variable name may be terminated by a ';'. This allows for variables to be placed with text immediately following their names.

Caution: Variables are case sensitive.

By default, several variables are added. These are automatically added as needed when a page is rendered. Default variables can also be added via the TVpPrinter's AddDefaultVariables and UpdateDateVariable methods.

The automatically created variables are listed in Table 2.1. Several of the default variables correspond to date-time format strings used in the VCL FormatDateTime function.

Table 2.1: *Print caption variables*

| Variable | Description |
|---|---|
| DayNumber | The day as a number without a leading zero. |
| DayNumber0 | The day as a number with a leading zero. |
| DayAbbrev | The day as an abbreviation. This uses the values in the VCL ShortDayNames global variable. |
| DayName | The full name of the day. This uses the values in the VCL LongDayNames global variable. |
| ShortDate | The date in the format given by the VCL ShortDateFormat global variable. |

Table 2.1: *Print caption variables (continued)*

| Variable | Description |
|---|---|
| DayNumber | The day as a number without a leading zero. |
| LongDate | The date in the format given by the VCL LongDateFormat global variable. |
| Month | The month as a number with without a leading zero. |
| Month0 | The month as a number with a leading zero. |
| MonthAbbv | The month as an abbreviation. This will use the value of the VCL ShortMonthNames global variable. |
| MonthName | The month name. This will use the value of the VLC LongMonthNames global variable. |
| ShortYear | Two digit year. |
| LongYear | Four digit year. |
| DateSep | Date separator as defined by the VLC DateSeparator global variable. |
| d | The day as a number without a leading zero. |
| dd | The day as a number with a leading zero. |
| ddd | The day as an abbreviation. This uses the values in the VCL ShortDayNames global variable. |
| dddd | The full name of the day. This uses the values in the VCL LongDayNames global variable. |
| ddddd | The date in the format given by the VCL ShortDateFormat global variable. |
| dddddd | The date in the format given by the VCL LongDateFormat global variable. |
| m | The month as a number with without a leading zero. |
| mm | The month as a number with a leading zero. |
| mmm | The month as an abbreviation. This will use the value of the VCL ShortMonthNames global variable. |
| mmmm | The month name. This will use the value of the VLC LongMonthNames global variable. |
| yy | Two digit year. |
| yyyy | Four digit year. |
| / | Date separator as defined by the VLC DateSeparator global variable. |

Table 2.1: *Print caption variables (continued)*

| Variable | Description |
|---|---|
| DayNumber | The day as a number without a leading zero. |
| StartHour12 | The starting hour of TVpDayView components using a 12 hour format. |
| StartHour24 | The starting hour of TVpDayView components using a 24 hour clock. |
| StartHourAMPM | The AM/PM portion of the starting hour of TVpDayView components. |
| StopHour12 | The ending hour of TVpDayView components using a 12 hour format. |
| StopHour24 | The ending hour of TVpDayView components using a 24 hour clock. |
| StopHourAMPM | The AM/PM portion of the ending hour of TVpDayView components. |
| Granularity | The granularity of day view components as a number. |

If the page is being printed as a result of the Print method or the RenderPage of the TVpPrinter class, the "$Page" variable is defined as the current page number being printed.

Additional variables can be added or modified by using the AddVariable method or the ChangeVariable method of the TVpPrinter class.

## Hierarchy

TPersistent (VCL)

TVpPrintCaption (VPPrtFmt)

## Properties

Caption Font

## Methods

PaintToCanvas

## Reference Section

### Caption — property

```
property Caption : string
```

Contains the caption to be printed.

### Font — property

```
property Font : TFont
```

Defines the font used to render the caption.

The font will automatically scale when the print job is sent to a printer.

### PaintToCanvas — method

```
procedure PaintToCanvas(ACanvas : TCanvas; ARect : TRect;
  Angle : TVpRotationAngle; Viewport : TRect; RealString : string);
```

Paints the caption to the specified canvas.

This method is normally called from the TVpPrinter class. In most cases, this method does not need to be called manually.

The caption is rendered to fill the rectangle specified by ARect.

The Angle parameter specifies how the shape will be rotated within the viewport specified by Viewport.

# TVpPrintFormatElementItem Class

This class is used in conjunction with TVpPrintFormatElement to list all of the elements that make up an individual print format.

## Hierarchy

TCollection (VCL)

❶ TVpCollectionItem (VPBase) . . . . . 41

TVpPrintFormatElementItem (VPPrtFmt)

## Properties

| | | | | | |
|---|---|---|---|---|---|
| | Caption | | Height | | Shape |
| | Collection | | ItemType | | Top |
| | DayOffset | | Left | ❶ | Version |
| | DayOffsetUnits | | Measurement | | Width |
| ❶ | DisplayText | ❶ | Name | | |
| | ElementName | | Rotation | | |

### Caption — property

```
property Caption : TVpPrintCaption
```

Defines a caption to be rendered as part of the print format.

Caption is only used when the ItemType is itCaption.

See also: ItemType

### Collection — property

```
property Collection : TVpPrintFormatElement
```

Specifies the print format of which this element is a member.

The Collection property provides a means to get to the TVpPrintFormatElement class that contains all the elements used to define the print format.

### DayOffset — property

```
property DayOffset : Integer
```

Default: 0

Adjusts the date of print element.

For ItemTypes of itDayView, itWeekView, itMonthView, and itCalendar, the DayOffset adjusts the date of the individual element. This adjustment is calculated by multiplying the DayOffset property by the DayOffsetUnits property. This allows the print format to have multiple of the same item types reflecting different dates.

For example, a print format that contained two itDayViews, the first one should be set up with a DayOffset of zero. The second one would have a DayOffset value of 1 and a DayOffsetUnits value of duDay. When printed, the two DayViews should show two consecutive dates on the page.

The TVpPrintFormatItem's DayInc and DayIncUnits work together with this property. DayInc and DayIncUnits specify the number of days on a page.

Negative DayOffsets are allowed.

See also: DayOffsetUnits, ItemType

## DayOffsetUnits property

```
property DayOffsetUnits : TVpDayUnits
```

```
TVpDayUnits = (duDay, duWeek, duMonth, duYear);
```

Default: duDay

Adjusts the date of a print element.

For ItemTypes of itDayView, itWeekView, itMonthView and itCalendar, the DayOffset adjusts the date of the individual element. This adjustment is calculated by multiplying the DayOffset property by the DayOffsetUnits property. This allows the print format to have multiple of the same item types reflecting different dates.

For example, a print format that contained two itDayViews, the first one should be set up with a DayOffset of zero. The second one would have a DayOffset value of 1 and a DayOffsetUnits value of duDay. When printed, the two DayViews should show two consecutive dates on the page.

The TVpPrintFormatItem's DayInc and DayIncUnits work together with this property. DayInc and DayIncUnits specify the number of days on a page.

See also: DayOffset, ItemType

## ElementName property

```
property ElementName : string
```

Names the print format element.

The ElementName property is displayed in the TVpPrintFormatElement TCollection editor. This property is useful for keeping track of the individual pieces of a complicated page format.

## Height property

```
property Height : Extended
```

Default: 100

Specifies the height of the element.

The Left, Top, Width, and Height properties specify where on the page the element will be printed. The value of the property is either in pixels, inches or as a percentage of the page. Which of these three units is in use is determined by the Measurement property.

See also: Left, Measurement, Top, Width

```
property ItemType : TVpItemType
```

```
TVpItemType = (itDayView, itWeekView, itMonthView, itCalendar,
  itShape, itCaption, itTasks, itContacts);
```

Default: itDayView

Determines the type of element to print.

With the exception of itCaption and itShape, the values of ItemType refer to Visual PlanIt components. The relationship between the ItemTypes and the components is laid out in the following table:

| Item Type | Component Printed |
|---|---|
| itCalendar | TVpCalendar |
| itCaption | Caption pseudo-type |
| itContacts | TVpContactGrid |
| itDayView | TVpDayView |
| itMonthView | TVpMonthView |
| itShape | Shape pseudo-type |
| itTasks | TVpTaskList |
| itWeekView | TVpWeekView |

Item types of ItCaption and itShape define pseudo ItemTypes that are used to print shapes and text as a print format element.

The element will be printed in the rectangle defined by the Left, Top, Width, and Height properties. This rectangle will be rotated on the page as defined by the Rotation property.

Note: When rendering the item, the printing will seek out a matching component connected to the owning TVpControlLink. If a component is found, the properties from that component will be used to control the output. If a component is not found, it will be created and the default settings for the component will be used.

See also: Caption, Height, Left, Rotation, Shape, Top, Width

## Left — property

```
property Left : Extended
```

Default: 0

Positions the left side of the element.

The Left, Top, Width, and Height properties specify where on the page the element will be printed. The value of the property is either in pixels, inches or as a percentage of the page. Which of these three units is in use is determined by the Measurement property.

See also: Height, Left, Measurement, Top, Width

## Measurement — property

```
property Measurement : TVpItemMeasurement
```

```
TVpItemMeasurement = (imAbsolutePixel, imPercent, imInches);
```

Default: imPercent

Specifies the units used to position the element on the page.

The Left, Top, Width, and Height properties define a rectangle that controls where the element will be printed on the page. The units used for measuring this rectangle are defined by the Measurement property.

Three units are used for positioning elements. imAbsolutePixel positions the element precisely by pixels. The same number of pixels will be used regardless of the output medium used. imInches determines the pixels per inch of the output media and positions the element based on that sizing. imPercent positions the element as a percentage of the page.

imAbsolutePixel allows for precise positioning and sizing of elements, but is not suitable for scaling to differing sized outputs. imPercent is less precise in positioning and sizing, but is well suited for scaling to different sizes. imInches falls somewhere in the middle. It is more precise than imPercent, but less scalable.

As a general rule, all the elements of a print format should use the same element measurement.

See also: Height, Left, Top, Width

### Rotation — property

```
property Rotation : TVpRotationAngle
TVpRotationAngle = (ra0, ra90, ra180, ra270);
```

Default: ra0

Rotates the element on the page.

Rotation will rotate the element in terms of rectangle defined by the Left, Top, Width, and Height.

See also: Height, Left, Top, Width

### Shape — property

```
property Shape : TVpPrintShape
```

Defines a shape to be drawn as part of the print format.

Shape is only used when the ItemType is itShape.

See also: ItemType

### Top — property

```
property Top : Extended
```

Default: 0

Positions the top of the element.

The Left, Top, Width, and Height properties specify where on the page the element will be printed. The value of the property is either in pixels, inches or as a percentage of the page. Which of these three units is in use is determined by the Measurement property.

See also: Height, Left, Measurement, Width

## Width property

```
property Width : Extended
```

Specifies the width of the element.

Default: 100

The Left, Top, Width, and Height properties specify where on the page the element will be printed. The value of the property is either in pixels, inches or as a percentage of the page. Which of these three units is in use is determined by the Measurement property.

See also: Height, Left, Measurement, Top

# TVpPrintFormatElement Class

TVpPrintFormatElements maintains the list of all the elements used in an individual print format. The actual definition of an element is kept in the TVpPrintFormatElementItem class.

## Hierarchy

TCollection (VCL)

TVpPrintFormatElement (VPPrtFmt)

## Properties

Items

## Items — property

```
property Items[Index : Integer] : TVpPrintFormatElementItem
```

Contains the print format elements used to define the print format.

# TVpPrintFormatItem Class

TVpPrintFormatItem defines an individual print format. This class is used in conjunction with TVpPrintFormat to maintain a list of print formats associated with a TVpControlLink component.

## Hierarchy

TCollectionItem (VCL)

❶ TVpCollectionItem (VPBase) . . . . . 41

TVpPrintFormatItem (VPPrtFmt)

## Properties

| | | | | | |
|---|---|---|---|---|---|
| | Collection | | Description | | FormatName |
| | DayInc | ❶ | DisplayText | ❶ | Name |
| | DayIncUnits | | Elements | ❶ | Version |

## Reference Section

### Collection — property

```
property Collection : TVpPrintFormat
```

Specifies the collection of print formats of which this print format is a member.

The Collection property provides a means to get to the TVpPrintFormat class that contains all the print formats associated with a particular TVpControlLink.

### DayInc — property

```
property DayInc : Integer
```

Default: 0

Indicates the number days per page displayed in the print format item.

For print formats that include day elements like imDayView, imWeekView, imMonthView, or imCalendar, the DayInc and DayIncUnits properties of the print format are used to indicate the total number of days that are displayed per page. The total number of days is calculated as DayInc multiplied by DayIncUnits.

Individual elements in the print format can have their dates adjusted by adjusting the TVpPrintFormatElementItem's DayOffset and DayOffsetUnits properties. The values of those properties should be taken into consideration when calculating the values for the TVpPrintFormatItem's DayInc and DayIncUnits properties. For example, a print format that consists of two week views should have the DayInc property set to 2 and the DayIncUnits property set to duWeek.

See also: DayIncUnits

### DayIncUnits — property

```
property DayIncUnits : TVpDayUnits
```

```
TVpDayUnits = (duDay, duWeek, duMonth, duYear);
```

Default: duDay

Indicates the units of the DayInc property.

For print formats that include day elements like imDayView, imWeekView, imMonthView or imCalendar, the DayInc and DayIncUnits properties of the print format are used to indicate the total number of days that are displayed per page. The total number of days is calculated as DayInc multiplied by DayIncUnits.

See also: DayInc

## Description — property

```
property Description : string
```

Provides an optional description for the print format.

This property is used to keep a description of the print format. It is intended to store longer and more descriptive descriptions than is used for the FormatName.

See also: FormatName

## Elements — property

```
property Elements : TVpPrintFormatElement
```

Specifies all the elements that make up this print format.

A print format contains various elements like DayViews, Calendars and text that are assembled when the print format is rendered. The Elements property is a TCollection descendent that maintains all the elements that go together to form the print format.

## FormatName — property

```
property FormatName : string
```

Names the print format.

FormatName is used to specify a name for the print format. This name is displayed in the TVpPrintFormat TCollection editor and is used by other components to refer to the print format.

Longer descriptions of the format should be put in the Description property.

Caution: A unique FormatName is required for each print format.

See also: Description

# TVpPrintFormat Class

TVpPrintFormats maintains a list of print formats associated with a TVpControlLink component. Individual print formats are accessible through the Items property.

## Hierarchy

TCollection (VCL)

TVpPrintFormat (VPPrtFmt)

## Properties

Items

## Items — property

```
property Items[Index : Integer] : TVpPrintFormatItem
```

Contains the print formats associated with a TVpControlLink component.

# TVpPrinter Class

The TVpPrinter class contains all the printer formats that are associated with a TVpControlLink component as well as additional data that is needed for printing.

## Hierarchy

TPersistent

TVpPrinter

## Properties

BottomMargin
CurFormat
DayEnd
DayStart
Granularity
HaveContactGrid
HaveDate
HaveTaskList
LastContact
LastTask
LeftMargin
MarginUnits
PrintFormats
RightMargin
TopMargin

## Methods

AddDefaultVariables
AddVariable
ChangeVariable
ClearVariables
DeleteVariable
Find
HaveVariable
LoadFromFile
LookupVariable
NotifyLinked
PaintToCanvasRect
Print
RenderPage
SaveToFile
UpdateDateVariables
ValidFormat

### AddDefaultVariables — method

```
procedure AddDefaultVariables(Date : TDateTime);
```

Adds a series of default variables.

Variables are used in print captions to allow for dynamic data in the printing. See the documentation on TVpPrintCaption for details of what variables are added. Variables regarding the date of the page to render as well the granularity, and the start and end times of any TVpDayView components used in the print format.

PaintToCanvasRect, Print, and RenderPage call this method for each print element.

See also: PaintToCanvasRect, Print, RenderPage, UpdateDateVariables

### AddVariable — method

```
procedure AddVariable(VarName : string; Value : string);
```

Adds a variable that can be used in print captions.

Variables are used in print captions to allow for dynamic data in the printing.

Additional variables remain valid until they are deleted using DeleteVariable or ClearVariables, or the TVpControlLink component is destroyed.

See also: ChangeVariable, ClearVariables, DeleteVariable

## BottomMargin — property

```
property BottomMargin : Extended
```

Default: 0

Specifies the bottom margin of the page.

The LeftMargin, TopMargin, RightMargin, BottomMargin, and MarginUnits parameters specify margins on the page. As with the margins used when printing, the units used for measuring the margins are defined by the MarginUnits property.

Three units are used for measuring the margins. imAbsolutePixel sizes the margin precisely by pixels. The same number of pixels will be used regardless of the output medium used. imInches determines the pixels per inch of the output media and sizes the margin based on that sizing. imPercent sizes the margin as a percentage of the page.

imAbsolutePixel allows for precise sizing, but is not suitable for scaling to differing sized outputs. imPercent is less precise in positioning and sizing, but is well suited for scaling to different sizes. imInches falls somewhere in the middle. It is more precise than imPercent, but less scalable.

The BottomMargin used in the print preview should be the same bottom margin used when printing the component using either the Print method or the RenderPage method of the TVpPrinter class.

See also: LeftMargin, MarginUnits, RightMargin, TopMargin

## ChangeVariable — method

```
procedure ChangeVariable(VarName : string; NewValue : string);
```

Changes a print caption variable.

Variables are used in print captions to allow for dynamic data in the printing.

See also: AddVariable, ClearVariables, DeleteVariable, LookupVariable

## ClearVariables — method

```
procedure ClearVariables;
```

Removes all print caption variables.

Variables are used in print captions to allow for dynamic data in the printing.

This removes all the variables. Default variables will come back as they are needed.

See also: AddVariable, DeleteVariable

### CurFormat — property

```
property CurFormat : Integer
```

Specifies the current print format.

See also: ValidFormat

### DayEnd — property

```
property DayEnd : TVpHours
```

```
TVpHours = (h12AM, h_1AM, h_2AM, h_3AM, h_4AM, h_5AM, h_6AM, h_7AM,
  h_8AM, h_9AM, h10AM, h11AM, h12PM, h_1PM, h_2PM, h_3PM, h_4PM,
  h_5PM, h_6PM, h_7PM, h_8PM, h_9PM, h10PM, h11PM);
```

Specifies the end of the day for TVpDayView components in the print format.

See also: DayStart

### DayStart — property

```
property DayStart : TVpHours
```

```
TVpHours = (h12AM, h_1AM, h_2AM, h_3AM, h_4AM, h_5AM, h_6AM, h_7AM,
  h_8AM, h_9AM, h10AM, h11AM, h12PM, h_1PM, h_2PM, h_3PM, h_4PM,
  h_5PM, h_6PM, h_7PM, h_8PM, h_9PM, h10PM, h11PM);
```

Specifies the start of the day for TVpDayView components in the print format.

See also: DayEnd

### DeleteVariable — method

```
function DeleteVariable(VarName : string) : Boolean;
```

Deletes the specified print caption variable.

Variables are used in print captions to allow for dynamic data in the printing.

This will Return True is the variable was found and deleted. It will return False if the variable was not found.

### Find — method

```
function Find(const v : string) : Integer;
```

Finds a variable by the name specified by v.

Variables are used in print captions to allow for dynamic data in the printing.

If the variable is found, its index is returned. If the variable is not found, -1 is returned.

See also: HaveVariable, LookupVariable

### Granularity — property

```
property Granularity : TVpGranularity
```

```
TVpGranularity = (
  gr60Min, gr30Min, gr20Min, gr15Min, gr10Min, gr6Min, gr5Min);
```

Specifies the granularity for TVpDayView components in the print format.

Variables are used in print captions to allow for dynamic data in the printing.

### HaveContactGrid — read-only, run-time property

```
property HaveContactGrid : Boolean
```

Indicates if the print job being performed has contact grid components.

RenderPage and Print set this value to indicate that the active page format uses contact grid components. The value reflects the result of the last page that has been rendered.

See also: HaveDate, HaveTaskList, Print, RenderPage

### HaveDate — read-only, run-time property

```
property HaveDate : Boolean
```

Indicates if the print job being performed has date components.

RenderPage and Print set this value to indicate that the active page format uses date components. The value reflects the result of the last page that has been rendered.

See also: HaveContactGrid, HaveTaskList, LastContact, Print, RenderPage

### HaveTaskList — read-only, run-time property

```
property HaveTaskList : Boolean
```

Indicates if the print job being performed has task list components.

RenderPage and Print set this value to indicate that the active page format uses task list components. The value reflects the result of the last page that has been rendered.

See also: HaveContactGrid, HaveDate, LastTask, Print, RenderPage

### HaveVariable — method

```
function HaveVariable(VarName : string) : Boolean;
```

Returns True is the specified variable is defined.

Variables are used in print captions to allow for dynamic data in the printing.

See also: Find, LookupVariable

### LastContact — read-only, run-time property

```
property LastContact : Integer
```

Indicates the last contact that has been printed while a print job is running.

RenderPage and Print set this value while pages are being rendered. This value is only valid if HaveContactGrid is True. It reflects the value of the last page that has been rendered.

After calling RenderPage, the value of LastContact can be fed into the StartContact parameter in the next call to RenderPage to start the contact grids at the correct place.

See also: HaveContactGrid, Print, RenderPage

### LastTask — read-only, run-time property

```
property LastTask : Integer
```

Indicates the last task that has been printed in while a print job is running.

RenderPage and Print set this value while pages are being rendered. This value is only valid if HaveTaskList is True. It reflects the value of the last page that has been rendered.

After calling RenderPage, the value of LastTask can be fed into the StartTask parameter in the next call to RenderPage to start the task lists at the correct place.

See also: HaveTaskList, Print, RenderPage

## LeftMargin — property

```
property LeftMargin : Extended
```

Default: 0

Specifies the left margin of the page.

The LeftMargin, TopMargin, RightMargin, BottomMargin, and MarginUnits parameters specify margins on the page. As with the margins used when printing, the units used for measuring the margins are defined by the MarginUnits property.

Three units are used for measuring the margins. imAbsolutePixel sizes the margin precisely by pixels. The same number of pixels will be used regardless of the output medium used. imInches determines the pixels per inch of the output media and sizes the margin based on that sizing. imPercent sizes the margin as a percentage of the page.

imAbsolutePixel allows for precise sizing, but is not suitable for scaling to differing sized outputs. imPercent is less precise in positioning and sizing, but is well suited for scaling to different sizes. imInches falls somewhere in the middle. It is more precise than imPercent, but less scalable.

The LeftMargin used in the print preview should be the same left margin used when printing the component using either the Print method or the RenderPage method of the TVpPrinter class.

See also: BottomMargin, MarginUnits, RightMargin, TopMargin

## LoadFromFile — method

```
procedure LoadFromFile(const FileName : string; Append : Boolean);
```

Loads print formats from a file.

Print formats can be saved and loaded in an XML format. The details of that format can be found in the overview section on print formats.

When Append is True, the print formats in the file specified by FileName are added to the existing list of file formats. Keep in mind that format names must be unique. If a format by the same name already exists in the list of print formats, it will be overwritten by the version that is loaded from the file. When Append is False, the current formats are cleared before the new formats are loaded.

See also: SaveToFile

## LookupVariable — method

```
function LookupVariable(VarName : string) : string;
```

Returns the value of the specified variable.

If the variable is not found, an empty string is returned. Use HaveVariable or Find to determine if the variable exists.

See also: Find, HaveVariable

## MarginUnits — property

```
property MarginUnits : TVpItemMeasurement
```

```
TVpItemMeasurement = (imAbsolutePixel, imPercent, imInches);
```

Default: imInches;

Specifies the units used in calculating the margins.

The LeftMargin, TopMargin, RightMargin, BottomMargin, and MarginUnits parameters specify margins on the page. As with the margins used when printing, the units used for measuring the margins are defined by the MarginUnits property.

Three units are used for measuring the margins. imAbsolutePixel sizes the margin precisely by pixels. The same number of pixels will be used regardless of the output medium used. imInches determines the pixels per inch of the output media and sizes the margin based on that sizing. imPercent sizes the margin as a percentage of the page.

imAbsolutePixel allows for precise sizing, but is not suitable for scaling to differing sized outputs. imPercent is less precise in positioning and sizing, but is well suited for scaling to different sizes. imInches falls somewhere in the middle. It is more precise than imPercent, but less scalable.

The MarginUnits used in the print preview should be the same margin units used when printing the component using either the Print method or the RenderPage method of the TVpPrinter class.

See also: BottomMargin, LeftMargin, RightMargin, TopMargin

## NotifyLinked — method

```
procedure NotifyLinked;
```

Notifies components when print formats have changed.

When print formats are added or deleted, components that use the print format information, like the TVpPrintPreview and TVpPrintFormatComboBox are automatically notified. In some cases where print formats are modified, the components may not be notified of the change. NotifyLinked exists to allow manual notification of the linked components.

## PaintToCanvasRect — method

```
procedure PaintToCanvasRect(
  ACanvas : TCanvas; ARect : TRect; ADate : TDateTime);
```

Paints the print format to a canvas.

PaintToCanvasRect will paint the current print format, specified in CurFormat to the rectangle specified by ARect on ACanvas.

Warning: Although PaintToCanvasRect can use any date, task lists and contact grids will always start at the beginning. Use RenderPage to control the task lists and contact grids.

See also: CurFormat, RenderPage

## Print — method

```
procedure Print(APrinter : TPrinter; LeftMargin : Extended;
  StartDate : TDateTime; EndDate : TDateTime);
```

Prints a multi-page print format to the specified printer.

The print format specified by CurFormat is printed to the printer specified by APrinter.

The print job will run from StartDate to EndDate. If the print format does not use date components, the job will run until all the data has been printed. If the print format uses task lists, this will be when all the tasks have been printed. If it uses a contact grid, this will be when all contacts have been printed.

The pages will be printed within the margins specified by LeftMargin, TopMargin, RightMargin, BottomMargin, and MarginUnits.

The output can be sent to non-printer canvases by using the PaintToCanvasRect or RenderPage methods.

See also: BottomMargin, CurFormat, LeftMargin, MarginUnits, PaintToCanvasRect, RenderPage, RightMargin, TopMargin

## PrintFormats — property

```
property PrintFormats : TVpPrintFormat
```

Defines the print formats.

PrintFormats stores all the print formats and the elements that make up those formats. See the TVpPrintFormat documentation on page 73 for more information.

## RenderPage — method

```
procedure RenderPage(ACanvas : TCanvas; ARect : TRect;
  PageNum : Integer; var ADate : TDateTime; EndDate : TDateTime;
  var StartContact : Integer; var StartTask : Integer;
  var LastPage : Boolean);
```

Paints the current print format to an arbitrary canvas one page at a time.

RenderPage is intended to print the current print format to an arbitrary TCanvas specified by ACanvas. The pages will be rendered inside the rectangle specified by ARect, using margins specified by the LeftMargin, TopMargin, RightMargin, BottomMargin, and MarginUnits properties.

The rendering is performed one page at a time. The ADate, EndDate, StartContact and StartTask parameters determine the page rendered. For the first page, ADate should be set to the first date in to render. StartContact and StartTask should be set to zero to start rendering the first contacts and tasks. EndDate should be set to the last date to print. This is used primarily for last page determination. As each page is rendered, ADate, StartContact, and StartTask are updated to reflect the correct settings for the next page. When the final page of the report has been rendered, RenderDate will set the LastPage flag to True.

For sending the entire print job to a printer, use the Print method.

This example renders the first two pages of a print job to two TImage components:

```
var
  StartDate    : TDateTime;
  StartContact : Integer;
  StartTask    : Integer;
  LastPage     : Boolean;

begin
  { Select the first print format.
    At least one print format must be defined. }

  VpControlLink1.Printer.CurFormat := 0;

  { Initialize the rendering }

  StartDate    := Now;
  StartContact := 0;
  StartTask    := 0;

  { Render the first page - no margins are used }

  VpControlLink1.Printer.RenderPage(
    Image1.Canvas, Image1.ClientRect,
    0, 0, 0, 0, imPercent, 1, StartDate, Now + 7,
    StartContact, StartTask, LastPage);

  { Render the second page }

  if not LastPage then
    VpControlLink1.Printer.RenderPage(
      Image2.Canvas, Image2.ClientRect, 0, 0, 0, 0,
      imPercent, 1, StartDate, Now + 7, StartContact,
      StartTask, LastPage)
end;
```

See also: BottomMargin, CurFormat, LeftMargin, MarginUnits, PaintToCanvasRect, Print, RightMargin, TopMargin

## RightMargin — property

```
property RightMargin : Extended
```

Default: 0

Specifies how margins are measured.

The LeftMargin, TopMargin, RightMargin, BottomMargin, and MarginUnits parameters specify margins on the page. As with the margins used when printing, the units used for measuring the margins are defined by the MarginUnits property.

Three units are used for measuring the margins. imAbsolutePixel sizes the margin precisely by pixels. The same number of pixels will be used regardless of the output medium used. imInches determines the pixels per inch of the output media and sizes the margin based on that sizing. imPercent sizes the margin as a percentage of the page.

imAbsolutePixel allows for precise sizing, but is not suitable for scaling to differing sized outputs. imPercent is less precise in positioning and sizing, but is well suited for scaling to different sizes. imInches falls somewhere in the middle. It is more precise than imPercent, but less scalable.

The RightMargin used in the print preview should be the same right margin used when printing the component using either the Print method or the RenderPage method of the TVpPrinter class.

See also: Bottom, LeftMargin, MarginUnits, TopMargin

## SaveToFile — method

```
procedure SaveToFile(const FileName : string);
```

Saves the print formats to an XML base file.

Print formats can be saved and loaded in an XML format. The details of that format can be found in the overview section on print formats.

See also: LoadFromFile

## TopMargin — property

```
property TopMargin : Extended
```

Default: 0

Sets the top margin of the print preview page.

The LeftMargin, TopMargin, RightMargin, BottomMargin, and MarginUnits parameters specify margins on the page. As with the margins used when printing, the units used for measuring the margins are defined by the MarginUnits property.

Three units are used for measuring the margins. imAbsolutePixel sizes the margin precisely by pixels. The same number of pixels will be used regardless of the output medium used. imInches determines the pixels per inch of the output media and sizes the margin based on that sizing. imPercent sizes the margin as a percentage of the page.

imAbsolutePixel allows for precise sizing, but is not suitable for scaling to differing sized outputs. imPercent is less precise in positioning and sizing, but is well suited for scaling to different sizes. imInches falls somewhere in the middle. It is more precise than imPercent, but less scalable.

The TopMargin used in the print preview should be the same top margin used when printing the component using either the Print method or the RenderPage method of the TVpPrinter class.

See also: BottomMargin, LeftMargin, MarginUnits, RightMargin

## UpdateDateVariables — method

```
procedure UpdateDateVariables(Date : TDateTime);
```

Adds date related print caption variables.

UpdateDateVariables adds the date related print caption variables. See the documentation on TVpPrintCaption on page 57 for details on what variables are added by default.

See also: AddDefaultVariables

## ValidFormat — method

```
function ValidFormat(const v : Integer) : Boolean;
```

Determines if the requested format is valid.

Returns True if v is a legal value to use for the CurFormat property.

See also: CurFormat

# Chapter 3: Visual PlanIt Data Storage Building Blocks



The single most powerful feature of the Visual PlanIt component library is the way that it allows you to effortlessly create powerful PIM style applications with absolutely no database programming. All of the database complexity has been encapsulated into the various DataStore components.

Visual PlanIt ships with a number of pre-built DataStore components that support many of the popular database engines. Each DataStore component handles the connection to, and the transfer of data to and from the database as well as providing access to the underlying Visual PlanIt data structures for each of the TVpLinkableControl descendants.

The data used by the Visual PlanIt components, is encapsulated in the following classes:

- TVpCustomDataStore
- TVpCustomDBDataStore
- TVpBDEDataStore
- TVpDependentInfo
- TVpResources
- TVpResource
- TVpSchedule
- TVpEvent
- TVpTasks
- TVpTask
- TVpContacts
- TVpContact
- TVpTimeRange

# Overview

The TVpCustomDataStore class encapsulates all of the Visual PlanIt data storage classes and defines a standard interface for transferring data to and from the external database as well as dependent TVpLinkableControl descendants.

Each DataStore component defines the communication layer between Visual PlanIt and the external database engine, as well as providing database engine specific properties. For instance, the BDEDataStore provides properties like AliasName, AutoCreateAlias, DriverName, Params, SessionName, etc.

Data store components are provided for the BDE, TurboPower's FlashFiler, as well as many other database engines.

Before you can use any of the TVpLinkableControl descendants, you must first properly configure an appropriate DataStore component and attach it to the LinkableControl descendant's DataStore property. You are also responsible for properly configuring the external database engine for the new connection.

# Visual PlanIt DataStore Components

The TVpDataStore components are non-visual components that manage the shuffling of data to and from the TVpLinkableControls (TVpTimeGrid, TVpContactGrid, and TVpTaskList) and the external database. None of the TVpLinkableControls can function apart from a properly configured DataStore component.

# TVpCustomDataStore Class

The CustomDataStore class is the ultimate ancestor for all DataStore components. It provides a standard interface for the communication between the Visual PlanIt visual controls and the data.

Descendant classes define the way in which the data is transferred to and from the external database. New DataStore components descend directly from the TVpCustomDataStore, complete the interface definition, and implement the database layer.

## Hierarchy

TComponent (VCL)

❶ TVpComponent (VpBase) . . . . . 28

TVpCustomDataStore (VpBaseDS)

## Properties

| | | |
|---|---|---|
| AutoConnect | DayBuffer | ResourceID |
| AutoCreate | DefaultEventSound | Resources |
| CategoryColorMap | EnableEventTimer | TimeRange |
| Connected | PlayEventSounds | ❶ Version |
| Date | Resource | |

## Methods

| | | |
|---|---|---|
| Load | PostResources | RefreshResource |
| PostEvents | RefreshEvents | SetResourceByName |
| PostContacts | RefreshContacts | |
| PostTasks | RefreshTasks | |

## Events

| | | |
|---|---|---|
| OnAlert | OnConnect | OnDisconnect |

## Reference Section

### AutoConnect — property

```
property AutoConnect : Boolean
```

Determines whether the DataStore should automatically attempt a connection immediately after loading.

Setting AutoConnect to True will cause the DataStore to attempt a connection to the database when the application starts up. Set it to False if you need to manually perform administrative tasks prior to connection.

### AutoCreate — property

```
property AutoCreate : Boolean
```

Determines whether the DataStore will automatically create the tables on connection.

If AutoCreate is True and the tables don't exist when the DataStore attempts a connection, they will be created and then opened. If AutoCreate is True, the connection will fail due to no database tables being found.

Caution: If AutoCreate is True and your application is not saved prior to being run in the debugger, the tables will be created in the IDE's \bin directory.

### CategoryColorMap — property

```
property CategoryColorMap : TVpCategoryColorMap
```

Maps the category color-coding to the category names.

The CategoryColorMap maps event category index values with the appropriate color-code, and category name. Events store their category as an integer. When they are painted in the TimeGrid, the color-coding, and category name is determined via the values contained in the DataStore's CategoryColorMap. For more information, see the TVpCategoryColorMap class on page 35.

The following code snippets show two different ways to get the color and description of category 0:

```
CatColor := DataStore.CategoryColorMap.Category0.Color;
CatDescription
  := DataStore.CategoryColorMap.Category0.Description;
```

```
CatColor := DataStore.CategoryColorMap.GetColor(0);
CatDescription := DataStore.CategoryColorMap.GetName(0);
```

### Connected — property

```
property Connected : Boolean
```

Controls the database's connection state.

To connect to the database, set Connected to True. To disconnect, set Connected to False.

### Date — property

```
property Date : TDateTime
```

Specifies the target date for the dependent TVpLinkableControl descendants.

The value of Date is synchronized with the dependent TVpLinkableControls.

### DayBuffer — property

```
property DayBuffer : Integer
```

Determines how many days worth of events will be loaded into system memory from the database.

DayBuffer represents the number of days on each side of "Date" to load events for. If the value of DayBuffer is set to 31 (the default), 63 days worth of events will be loaded into the system's memory; 31 days on each side of Date, plus Date.

### DefaultEventSound — property

```
property DefaultEventSound : string
```

Contains the path and file name of the default *.wav file that will be played when an event's Alert is fired.

When an event comes due, an alert is fired that results in a sound being played and a notification dialog box being displayed.

Each event can have its own sound assigned. If an event doesn't have a sound associated with it, the default sound plays. If there is not a wav file associated with the DefaultEventSound, the system plays the standard MessageBeep.

If the value of PlayEventSounds is False, no sounds are played.

See also: PlayEventSounds

## EnableEventTimer — property

```
property EnableEventTimer : Boolean
```

Determines whether events fire the alert when they come due.

If EnableEventTimer is set to True, a sound will play and a dialog box will be displayed when an event's starting date and time elapses.

If an OnAlert event handler has been assigned, it will be fired when an event comes due.

Note: The sounds will only play if PlayEventSounds is set to True.

See also: OnAlert, PlayEventSounds, DefaultEventSound.

## Load — virtual method

```
procedure Load; virtual;
```

Loads data from the database into memory according to the values of Date and DayBuffer.

The CustomDataStore class's Load method only calls a protected method called NotifyDependents that invalidates all dependent controls. Descendant components must define this event to extract the proper data from the database and load it into memory.

## OnAlert — event

```
property OnAlert : TVpEventEvent
```

```
TVpEventEvent = procedure(
  Sender : TObject; Event : TVpEvent) of object;
```

Defines an event handler that is called when an event's start time passes.

If an event handler is defined for OnAlert, default behavior is skipped. The OnAlert event handler is responsible for playing the event's alarm sound and displaying the dialog box.

The following code snippet illustrates how to properly use the OnAlert event:

```
procedure TForm1.VpDataStore1Alert(
  Sender : TObject; Event : TVpEvent);
begin
  { If the alert for this event is already displayed }
  { then bail out. }
  if Event.AlertDisplayed then Exit;

  { Play the event sound }
  if VpDataStore1.PlayEventSounds then begin

    { if the event has a sound assigned, then play it. }
    if FileExists(Event.AlarmWavPath) then
      SndPlaySound(PChar(Event.AlarmWavPath),
        snd_Async)

    { otherwise, play the default sound. }
    else if FileExists(VpDataStore1.DefaultEventSound) then
      SndPlaySound(PChar(VpDataStore1.DefaultEventSound),
        snd_Async)

    { if neither sounds are assigned, just ding. }
    else
      MessageBeep(0);
  end;

  { Launch the notification dialog box }
  VpNotificationDialog1.Execute(Event);
end;
```

Note: This source code snippet assumes there is a properly configured TVpNotificationDialog component on the form. You must also add VpData, and mmSystem to the form's "uses" clause. VpData is where TVpEvent is defined, and mmSystem is where sndPlaySound is defined.

See also: DefaultEventSound, EnableEventTimer, PlayEventSounds

## OnConnect — event

```
property OnConnect : TNotifyEvent
```

Defines an event handler that is called when the database successfully connects.

When the DataStore component successfully connects to the database, the OnConnect event is fired.

The following code snippet illustrates how to use a TShape as a status indicator for the DataStore:

```
procedure TForm1.VpDataStore1Connect(Sender : TObject);
begin
  Shape1.Brush.color := clLime;
end;
```

See also: OnDisconnect

## OnDisconnect — event

```
property OnDisconnect : TNotifyEvent
```

Defines an event handler that is called when the database successfully disconnects.

When the DataStore component successfully disconnects from the database, the OnDisconnect event is fired.

The following code snippet illustrates how to use a TShape as a status indicator for the DataStore:

```
procedure TForm1.VpDataStore1Disconnect (Sender : TObject);
begin
  Shape1.Brush.color := clRed;
end;
```

See also: OnConnect

## PlayEventSounds — property

```
property PlayEventSounds : Boolean
```

Determines whether the system will play a sound when an event's start time elapses.

PlayEventSounds is ignored if EnableEventTimer is False, as no events will trigger an alert.

Note: If an OnAlert event handler is defined, the event handler is responsible for playing any sounds associated with the firing of an alert.

See also: DefaultEventSound, EnableEventTimer, OnAlert

### PostContacts — virtual abstract method

```
procedure PostContacts; virtual; abstract;
```

Forces the DataStore to iterate through the contacts and post their changes to the database.

This is where contacts marked as deleted are actually removed from the database and freed from memory. PostContacts is automatically called by PostResources.

PostContacts does not exist in the base class, it is implementation specific and must be implemented in each descendant DataStore component.

See also: PostResources

### PostEvents — virtual abstract method

```
procedure PostEvents; virtual; abstract;
```

Forces the DataStore to iterate through the events and post their changes to the database.

This is where events marked as deleted are actually removed from the database and freed from memory. PostEvents is automatically called by PostResources.

PostEvents does not exist in the base class, it is implementation specific and must be implemented in each descendant DataStore component.

See also: PostResources

### PostResources — virtual abstract method

```
procedure PostResources; virtual; abstract;
```

Forces the DataStore to iterate through the resources and post their changes to the database.

This is where resources marked as deleted are actually removed from the database and freed from memory. PostResources iterates through the resources and posts changes to the database along with any of their events, contacts, and task changes.

See also: PostContacts, PostEvents

### PostTasks — virtual abstract method

```
procedure PostTasks; virtual; abstract;
```

Forces the DataStore to iterate through the tasks and post their changes to the database.

This is where tasks marked as deleted are actually removed from the database and freed from memory. PostTasks is automatically called by PostResources.

PostTasks does not exist in the base class, it is implementation specific and must be implemented in each descendant DataStore component.

See also: PostResources

### RefreshContacts — virtual method

```
procedure RefreshContacts; virtual;
```

Calls NotifyDependents to refresh all dependent controls.

RefreshContacts only calls NotifyDependents in the base class. It is implementation specific and must be implemented in each descendant DataStore component.

### RefreshEvents — virtual method

```
procedure RefreshEvents; virtual;
```

Calls NotifyDependents to refresh all dependent controls.

RefreshEvents only calls NotifyDependents in the base class. It is implementation specific and must be implemented in each descendant DataStore component.

### RefreshResource — virtual method

```
procedure RefreshResource; virtual;
```

Calls NotifyDependents to refresh all dependent controls.

RefreshTasks only calls NotifyDependents in the base class. It is implementation specific and must be implemented in each descendant DataStore component.

In descendant classes, it should refresh the resources from the database, and then call each of the other Refresh methods.

See also: RefreshContacts, RefreshEvents, RefreshTasks

### RefreshTasks — virtual method

```
procedure RefreshTasks; virtual;
```

Calls NotifyDependents to refresh all dependent controls.

RefreshTasks only calls NotifyDependents in the base class. It is implementation specific and must be implemented in each descendant DataStore component.

### Resource — property

```
property Resource : TVpResource
```

Provides access to the currently selected Resource.

Change the currently active resource via the ResourceID property or the SetResourceByName method.

See also: ResourceID, SetResourceByName

### ResourceID — property

```
property ResourceID : Integer
```

Provides the ID of the currently selected resource.

Change the currently selected resource by setting ResourceID to a new value or calling the SetResourceByName method with the description of the desired resource.

See also: Resource, SetResourceByName

### Resources — property

```
property Resources : TVpResources
```

Provides access to the active list of resources.

All of the Visual PlanIt data manipulation is accomplished through a resource. Events, Contacts, and Tasks cannot exist outside of a resource. Resources maintains a list of all of the active resources. For more information, see the TVpResources class on page 112.

## SetResourceByName — virtual abstract method

```
procedure SetResourceByName(Value : string); virtual; abstract;
```

Allows you to switch to a new resource by passing in the name of the desired resource.

This is the method called by the TVpResourceEditDialog component.

The following code snippet illustrates how to use change resources via the resource name:

```
VpDataStore1.SetResourceByName('Andrew Denton');
```

See also: Resource, ResourceID

## TimeRange — property

```
property TimeRange : TVpTimeRange
```

Specifies a range of time, for which event data will be buffered into memory. The range is specified by Date and DayBuffer, beginning with DayBuffer days before Date and ending with DayBuffer days after Date. For more information, see the TVpTimeRange class on page 154

See also: Date, DayBuffer, TVpTimeRange

# TVpCustomDBDataStore

The TVpCustomDBDataStore defines the interface for TDataset compliant DataStore components. If you want to create your own TDataset compliant DataStore component, descend from this class and implement the abstract methods.

The TVpCustomDBDataStore doesn't implement any new properties, events, or methods. Rather, it defines a basic TDataset implementation.

## Hierarchy

TComponent (VCL)

- TVpCustomComponent (VpBase) . . . . . 28
  - TVpCustomDataStore (VpBaseDS) . . . . . 92
    - TVpCustomDBDataStore (VpDBDS)

# TVpBDEDataStore Class

The BDEDataStore component descends from the TVpCustomDataStore class and implements the functionality required to connect to a standard BDE database engine.



## Hierarchy

TComponent (VCL)

❶ TVpComponent (VpBase) . . . . . 28

TVpCustomDataStore (VpBaseDS) . . . . . 92

TVpCustomDBDataStore (VpDBDS) . . . . . 102

TVpBDEDataStore (VpBDEDS)

## Properties

| | | |
|---|---|---|
| AliasName | LoginPrompt | ResourceID |
| AutoCreateAlias | Params | ❶ Version |
| DayBuffer | ReadOnly | |

## Methods

| | | |
|---|---|---|
| CreateTable | PostContacts | PostResources |
| Load | PostEvents | PostTasks |

## Events

| | |
|---|---|
| OnConnect | OnDisconnect |

### AliasName — property

```
property AliasName : string
```

Sets the name of the BDE database alias that will be used by this component.

See also: AutoCreateAlias

### AutoCreateAlias — property

```
property AutoCreateAlias : Boolean
```

Flag that determines whether the DataStore should automatically create a BDE alias if the specified one doesn't exist, or there is none specified.

When the BDEDataStore attempts to connect to the database, it checks the session to make sure that the alias exists. If it doesn't, and AutoCreateAlias is set, the alias will be created with default settings.

If the AliasName is empty, an alias will be created, based on the name of the application, as shown in the following code snippet from the BDEDataStore's SetConnected procedure:

```
{ if there is no defined alias name, create one based }
{ on the application's name, and assign it to FDatabase }
if FAliasName = '' then begin
  FAliasName := ExtractFileName(ParamStr(0));
  FAliasName := Copy(FAliasName, 1,
   Pos('.', FAliasName) - 1);
 FDatabase.AliasName := FAliasName;
end;
```

### CreateTable — method

```
procedure CreateTable(TableName : string);
```

Creates the appropriate fields, and indexes specified by TableName.

The TVpBDEDataStore has the ability to automatically create the database tables for the user. When the CreateTable method is called, it examines the value of TableName, and calls the TVpCustomDBDataStore's CreateFieldDefs and CreateIndexDefs methods to generate the table's structure.

Following is the TVpBDEDataStore's CreateTable method:

```
procedure TVpBDEDataStore.CreateTable(TableName: string);
var
  Table: TTable;
begin
  Table := nil;
  if TableName = 'Resources' then begin
    { Create Resources Table }
    Table := FResourceTable;
    with Table do begin
      Active := False;
      Name := 'Resources';
      DatabaseName := FDatabase.DatabaseName;
      TableType := ttParadox;

      CreateFieldDefs(TableName, FieldDefs);
      CreateIndexDefs(TableName, IndexDefs);
    end;
  end

  else if TableName = 'Events' then begin
    { Create Events Table }
    Table := FEventsTable;
    with Table do begin
      Active := False;
      Name := 'Events';
      DatabaseName := FDatabase.DatabaseName;
      TableType := ttParadox;
      CreateFieldDefs(TableName, FieldDefs);
      CreateIndexDefs(TableName, IndexDefs);
    end;
  end

  else if TableName = 'Contacts' then begin
    { Create Contacts Table }
    Table := FContactsTable;
    with Table do begin
      Table.Active := False;
      Table.Name := 'Contacts';
      Table.DatabaseName := FDatabase.DatabaseName;
      Table.TableType := ttParadox;

      CreateFieldDefs(TableName, FieldDefs);
      CreateIndexDefs(TableName, IndexDefs);
    end;
  end
```

```
    else if TableName = 'Tasks' then begin
      { Create Tasks Table }
      Table := FTasksTable;
      with Table do begin
        Table.Active := False;
        Table.Name := 'Tasks';
        Table.DatabaseName := FDatabase.DatabaseName;
        Table.TableType := ttParadox;

        CreateFieldDefs(TableName, FieldDefs);
        CreateIndexDefs(TableName, IndexDefs);
      end;
    end;
    if Table <> nil then
      Table.CreateTable;
  end;
```



## DayBuffer — property

```
property DayBuffer : Integer
```

Default: 30

Specifies how many days worth of events will be read from the database and buffered in memory.

When the DataStore loads the resource's events into memory, it checks the value of Date and DayBuffer. It then creates a time range (via the TimeRange property) that encompasses a range of DayBuffer days before and DayBuffer days after the start date.

For instance if the value of DayBuffer is 30, and the value of Date is January 31, 2001, the DataStore will load all events that fall into the range between January 1, 2001 and March 2, 2001, for a range of 61 days (30 days on either side of Date, plus the day represented by Date).

## Load — method

```
procedure Load;
```

Loads all data, for all resources, from the database.

Load iterates through and creates all resource records, and all of their contacts, and tasks, as well as all of their events that fall within the TimeRange specified by the values of Date and DayBuffer.

See also: DayBuffer, TVpCustomDataStore.Date

### LoginPrompt — property

```
property LoginPrompt : Boolean
```

Specifies whether to prompt users for a login.

### OnConnect — event

```
property OnConnect : TNotifyEvent
```

Defines an event handler that is called when the database successfully connects.

### OnDisconnect — event

```
property OnDisconnect : TNotifyEvent
```

Defines an event handler that is called when the database successfully disconnects.

### Params — property

```
property Params : TStrings
```

Specifies database connection parameters for the BDE alias.

### PostContacts — method

```
procedure PostContacts(Resource : TVpResource); override;
```

Posts all contact changes for the specified resource.

### PostEvents — method

```
procedure PostEvents(Resource : TVpResource); override;
```

Posts all event changes for the specified resource.

### PostResources — method

```
procedure PostResources; override;
```

Iterates through all resources, posting their changes to the database.

PostResources also calls PostEvents, PostContacts and PostTasks for each resource.

See also: PostContacts, PostEvents, PostTasks

### PostTasks — method

```
procedure PostTasks(Resource : TVpResource); override;
```

Posts all task changes for the specified resource.

### ReadOnly — property

```
property ReadOnly : Boolean
```

Limits the DataStore component to read-only.

When the ReadOnly flag is set, all controls connected to ReadOnly DataStore components will have their ReadOnly attribute set as well, which will prevent them from having the ability to edit the events, contacts or tasks.

When the ReadOnly flag is cleared, the dependent controls will not have their ReadOnly flag modified.

### ResourceID — property

```
property ResourceID : Integer
```

Default: -1

Sets the active Resource by the resource's ID number.

Changing the value of ResourceID to that of a valid resource causes the DataStore component to re-populate itself with data from the new resource.

# TVpDependentInfo Class

The TVpDependentInfo class maintains information about one of the controls, which is dependent on the ControlLink. The ControlLink maintains a list of these objects, which is iterates through when it sends notification messages to its dependents.



## Hierarchy

TObject (VCL)

TVpDependentInfo (VPBaseDS)

## Properties

| | |
|---|---|
| Component | EventHandler |

### Component — property

```
property Component : Pointer
```

Contains a reference to the dependent control's instance.

### EventHandler — property

```
property EventHandler : TVpControlNotifyEvent
```

```
TVpControlNotifyEvent = procedure(Sender : TComponent;
  NotificationEvent : TVpNotificationType;
  const Value : Variant) of object;
```

Returns a reference to the dependent control's LinkHandler procedure.

When the ControlLink needs to notify any dependent controls of a change, it iterates its DependentList, firing each control's LinkHandler via the TVpDependentInfo's EventHandler property.

```
TVpDependentInfo = class { Used by the ControlLink component }
protected{private}
  FComponent : Pointer;
  FEventHandler : TVpControlNotifyEvent;
public
  property Component : Pointer read FComponent write FComponent;
  property EventHandler : TVpControlNotifyEvent
    read FEventHandler write FEventHandler;
end;
```

See also: TVpLinkableControl.LinkHandler

# Visual PlanIt Internal Data Storage Classes

To enhance performance, Visual PlanIt caches a certain amount of data in memory. The DataStore component manages how the data is loaded in to memory and how it is transferred to and from the visual controls. The following classes are used to store the Visual PlanIt data in memory:

- TVpResources class
- TVpResource class
- TVpSchedule class
- TVpEvent class
- TVpTasks class
- TVpTask class
- TVpContacts class
- TVpContact class
- TVpTimeRange class

# TVpResources Class

The TVpResources class maintains a collection of TVpResource instances.

A Resource is the ultimate owner of the events, contacts and tasks. No event, contact or task can exist without a resource.

For instance, a conference room would be a resource and the reservations would be the conference room's events. Another example would be a salesman. His appointments would be the events, his customers would be his contacts, and the tasks would be his to-do list.

## Hierarchy

TObject (VCL)

TVpResources (VpData)

## Properties

| | |
|---|---|
| Count | Owner |

## Methods

| | |
|---|---|
| AddResource | GetResource |
| ClearResources | RemoveResource |

### AddResource — method

```
function AddResource : TVpResource;
```

Adds a new resource to the list and returns a reference to it.

The following code snippet shows how to use AddResource:

```
function TForm1.AddNewResource : Boolean;
var
  Res : TVpResource;
begin
  Result := False;
  if VpDataStore1 <> nil then begin
    Res := VpDataStore1.Resources.AddResource;
    if Res <> nil then begin
      VpDataStore1.Resource := Res;
      Res.Description := 'Rick Brodzinsky';
      VpDataStore1.PostResources;
      { display the resource edit dialog box}
      Result := VpResourceEditDialog1.Execute;
      { if the user cancels the dialog box}
      { then delete the new resource }
      if not Result then
        Res.Deleted := True;
    end;
    VpDataStore1.PostResources;
    VpDataStore1.Load;
  end;
end;
```

### ClearResources — method

```
procedure ClearResources;
```

Removes all resources from the list and destroys them.

ClearResources does not permanently delete the resources from the database. Rather, it is used in the DataStore's destructor to de-allocate the memory used by the resources and clear the list.

See also: RemoveResource

### Count — property

```
property Count : Integer
```

Returns the number of Resources in the list.

### GetResource — method

```
function GetResource(Index : integer) : TVpResource;
```

Returns a reference to the resource located at the specified position in the list.

The following code snippet shows how to iterate through the resources, gathering their descriptions:

```
for I := 0 to pred(DataStore.Resources.Count) do begin
  if DataStore.Resources.GetResource(I).Description <> '' then
    Items.Add(DataStore.Resources.GetResource(I).Description);
end;
```

### Owner — property

```
property Owner : TObject
```

Returns a reference to the TVpCustomDataStore descendant (DataStore component) that owns this resource's object.

### RemoveResource — method

```
procedure RemoveResource(Resource : TVpResource);
```

Sets the resource's deleted and changed flags to True.

RemoveResource only marks the resource for deletion. The resource isn't physically removed from the list until the DataStore posts to the database.

To permanently delete the resource, call RemoveResource followed by PostResources.

See also: TVpCustomDataStore.PostResources, TVpResource.Deleted

# TVpResource Class

The TVpResource class models a TVpResource object. Each instance represents one record in the database's Resources table.

A resource is the parent of the schedule, task list, and contact list.



## Hierarchy

TObject (VCL)

    TVpResource (VpData)

## Properties

| | | |
|---|---|---|
| Active | Description | ResourceID |
| Changed | EventsDirty | Schedule |
| Contacts | ItemIndex | Tasks |
| ContactsDirty | Notes | TasksDirty |
| Deleted | Owner | |

## Reference Section

### Active — property

```
property Active : Boolean
```

Flag that determines if the resource should be made available to the user.

If Active is False, the resource won't show up in the ResourceCombo, and its events, contacts and tasks will not be displayed in the various Visual PlanIt controls.

Active is a persistent field and is stored in the external database.

### Changed — property

```
property Changed : Boolean
```

Flag that determines if this resource has been changed.

The value of Changed is checked as data is being saved to the database. If it is True, the resource's data is updated in the database. If it is False, this record is skipped.

### Contacts — property

```
property Contacts : TVpContacts
```

Provides access to this resource's contacts list.

The contacts object maintains a list of this resource's contacts.

### ContactsDirty — property

```
property ContactsDirty : Boolean
```

Flag that determines whether any of this resource's contacts have been modified or added.

When the external database is being updated, the value of ContactsDirty is checked to determine if any of the contacts have changes that need to be posted to the database. If the value of ContactsDirty is False, execution skips iterating through the contacts list.

## Deleted — property

```
property Deleted : Boolean
```

Flag that determines whether the record has been deleted and should be removed from the database.

The value of Deleted is checked as data is being written to the database. If it is True, the resource, and all of its events, contacts and tasks are removed from the database, and released from memory. If it is False, this record is updated in the database, depending on the value of Changed.

See also: Changed, TVpResources.RemoveResource

## Description — property

```
property Description : string
```

Provides access to the resource's description field.

Description contains the name of the resource.

Description is a persistent field and is stored in the external database.

## EventsDirty — property

```
property EventsDirty : Boolean
```

Flag that determines whether any of this resource's events have been modified or added.

When the external database is being updated, the value of EventsDirty is checked to determine if any of the events have changes that need to be posted to the database. If the value of EventsDirty is False, execution skips iterating through the events list.

## ItemIndex — property

```
property ItemIndex : Integer
```

Provides the index of this particular resource.

This value is assigned to the resource object when it is created and added to the Resources list. It is not a persistent, database field. It may change as resources are added or deleted.

### Notes — property

```
property Notes : string
```

Provides access to the resource's Notes field.

The Notes field contains a free-form text value where user's can enter notes specific to this resource.

Notes is a persistent field and is stored in the external database.

### Owner — property

```
property Owner : TVpResources
```

Returns a reference to the TVpResources instance that owns this resource item.

### ResourceID — property

```
property ResourceID : Integer
```

Provides access to the resource's ID.

The ResourceID is used by the external database as an index field.

ResourceID is a persistent field and is stored in the external database.

### Schedule — property

```
property Schedule : TVpSchedule
```

Provides access to this resource's event schedule.

The schedule object maintains a list of this resource's events.

## Tasks — property

```
property Tasks : TVpTasks
```

Provides access to this resource's tasks list.

The tasks object maintains a list of this resource's tasks.

## TasksDirty — property

```
property TasksDirty : Boolean
```

Flag that determines whether any of this resource's tasks have been modified or added.

When the external database is being updated, the value of TasksDirty is checked to determine if any of the tasks have changes that need to be posted to the database. If the value of TasksDirty is False, execution skips iterating through the tasks list.

# TVpSchedule Class

The Schedule class maintains the list of TVpEvents for a resource.

3

## Hierarchy

TObject (VCL)

TVpSchedule (VpData)

## Properties

| | |
|---|---|
| EventCount | Owner |

## Methods

| | | |
|---|---|---|
| AddEvent | DeleteEvent | GetEvent |
| BatchUpdate | EventCountByDay | Sort |
| ClearEvents | EventsByDate | |

### AddEvent — method

```
function AddEvent(StartTime, EndTime : TDateTime) : TVpEvent;
```

Adds a new event to the schedule and returns a reference to it.

AddEvent takes two TDateTime values, (StartTime and EndTime), creates a new TVpEvent object with those parameters, and returns a reference to the new event.

The following code snippet is from the DayView component's WMLButtonDblClick message handler. When the mouse is double-clicked over an empty area of the DayView Component, a new event is created.

```
StartTime := trunc(FDisplayDate)
  +  dvLineArray[ActiveRow].Time;
EndTime := StartTime + dvTimeIncSize;
dvActiveEvent := DataStore.Resource.Schedule.AddEvent(
  StartTime, EndTime);
```

### BatchUpdate — method

```
procedure BatchUpdate(Value : Boolean);
```

Sets a flag that prevents flicker while many events are updated at once.

When each of the event-based LinkableControl descendants paint themselves, they first check the value of the schedule's BatchUpdate. If it is True, they fall through without refreshing the control. Set BatchUpdate True when you are making lots of changes to the schedule at once, such as when you are loading a large amount of schedule data.

```
procedure LoadLotsOfData;
begin
  Schedule.BatchUpdate(True);
  {Load lots of data into the schedule}
  Schedule.BatchUpdate(False);
end;
```

Note: Calls to BatchUpdate are cumulative. Each call to BatchUpdate(True) must have a matching call to BatchUpdate(False) or the controls will not ever refresh the event data.

### ClearEvents — method

```
procedure ClearEvents;
```

Deletes all events from memory and removes them from the schedule.

ClearEvents does not permanently delete the events from the database. Rather, it is used in the schedule's destructor to de-allocate the memory used by the event objects, and clear the list.

### DeleteEvent — method

```
procedure DeleteEvent(Event : TVpEvent);
```

Marks the specified event's deleted flag True.

Calling DeleteEvent doesn't actually delete the event. It simply marks the event as deleted. The event isn't actually removed from memory until the next call to the DataStore's PostEvents method.

To permanently delete the event, call DeleteEvent followed by a call to the DataStore's PostEvents method.

See also: EventCountByDay

### EventCount — property

```
property EventCount : Integer
```

Returns the number of events loaded into memory.

Event count only returns the number of events loaded into memory. It does not return the total number of events for the resource. The number of events loaded into memory varies with the value of the DataStore component's DayBuffer property, as events are added and deleted, and as the DataStore component's Date property changes.

Use EventCount to iterate through the events that are currently loaded into memory.

## EventCountByDay — method

```
function EventCountByDay(Value : TDateTime) : Integer;
```

Returns the number of events scheduled for a specific day.

EventCount by Day iterates through the available events (events that have been loaded into memory) and counts the number of events that have a start date that matches the date portion of the passed in TDateTime value.

EventCount by day does not count events that are marked for deletion.

See also: DeleteEvent, EventsByDate

## EventsByDate — method

```
procedure EventsByDate(Date : TDateTime; EventList : TList);
```

Populates the supplied TList with a reference to each event that has a StartTime that falls on this day.

The following code is from the WeekView's RenderToCanvas method:

```
DataStore.Resource.Schedule.EventsByDate(wvStartDate
  + I, EventList);
```

See also: EventCountByDay

## GetEvent — method

```
function GetEvent(Index : Integer) : TVpEvent;
```

Returns a reference to the event specified by the value of Index.

## Owner — property

```
property Owner : TVpResource
```

Returns a reference to the TVpResource instance that owns this schedule.

## Sort — method

```
procedure Sort;
```

Sorts the events by StartTime and EndTime.

The Schedule is first sorted by StartTime.

If two or more events have the same start time, they are further sorted by EndTime.

If two or more events have matching StartTime and EndTime, they are further sorted in the order in which they were created.

# TVpEvent Class

The TVpEvent class models one of the resource's schedule events.

## Hierarchy

TObject (VCL)

TVpEvent (VpData)

## Properties

AlarmAdv
AlarmAdvType
AlarmSet
AlarmWavPath
AlertDisplayed
AllDayEvent
Category
Changed
CustInterval
Deleted
Description
EndTime
ItemIndex
Note
Owner
RecordID
RepeatCode
SnoozeTime
StartTime

## AlarmAdv — property

```
property AlarmAdv : Integer
```

Indicates the amount of time before the event's StartTime to sound the alarm.

AlarmAdv determines the number of Minutes, Hours, or Days (depending on the value of AlarmAdvType) before the event's StartTime that the alarm should be activated.

AlarmAdv is a persistent field and is stored in the external database.

The VpMisc unit has a GetAlarmAdvanceTime function that takes AlarmAdv and AlarmAdvType as parameters and returns a TDateTime, which represents the amount of time that the alarm is advanced.

The following code snippet is from the TVpCustomDataStore.dsOnTimer procedure:

```
if (Event <> nil) and Event.AlarmSet then begin
  AdvanceTime := GetAlarmAdvanceTime(Event.AlarmAdv,
    Event.AlarmAdvType);
  AlarmTime := Event.StartTime - AdvanceTime
    + Event.SnoozeTime;
  {...}
```

See also: AlarmAdvType, AlarmSet

## AlarmAdvType — property

```
property AlarmAdvType : TVpAlarmAdvType
```

```
TVpAlarmAdvType = (atMinutes, atHours, atDays);
```

Default: atMinutes

Determines the units of time represented by the AlarmAdv property.

AlarmAdvType is a persistent field and is stored in the external database.

See also: AlarmAdv, AlarmSet

## AlarmSet — property

```
property AlarmSet : Boolean
```

Flag that indicates whether the event should sound an alarm.

If the AlarmSet property is True at a time determined by the event's StartTime, AlarmAdv, and AlarmAdvType properties, the DataStore component will play a sound and display an alarm dialog box to notify the user of the impending event.

AlarmSet is a persistent field and is stored in the external database.

See also: AlarmAdv, AlarmAdvType

## AlarmWavPath — property

```
property AlarmWavPath : string
```

Contains the path and file name of the *.wav file that will be played when this event's Alert is fired.

When this event comes due, an alert is fired that results in a sound being played and a notification dialog box being displayed.

Each event can have its own sound assigned. If an event doesn't have a sound associated with it, the DataStore's default sound plays. If there is not a wav file associated with the DataStore's DefaultEventSound, the system plays the standard MessageBeep.

If the value of the DataStore's PlayEventSounds is False, no sounds are played.

AlarmWavPath is a persistent field and is stored in the external database.

In the CustomDBDataStore class, AlarmWavPath is defined as a 255-character string called "DingPath."

See Database Design on page 157.

See also: TVpCustomDataStore.DefaultEventSound,
TVpCustomDataStore.PlayEventSounds

### AlertDisplayed — property

```
property AlertDisplayed : Boolean
```

Flag that determines whether this event's alert is currently being displayed.

If the value of AlertDisplayed is True, the DataStore will not play the sound or display the alert dialog box.

When manually showing the alert dialog box for an event, set the value of the event's AlertDisplayed property to True, to prevent the DataStore from playing a sound and displaying the dialog box more than once.

### AllDayEvent — property

```
property AllDayEvent : Boolean
```

Flag that denotes this event as an all-day event.

All day events have a StartTime of 12:00AM, and an ending time of 11:59PM on the target day.

AllDayEvent is a persistent field and is stored in the external database.

### Category — property

```
property Category : Integer
```

Provides read/write access to the event's category identifier.

The event's category is an index that indicates the category of the event. The TVpTimeGrid component uses the value of Index to properly color code the event according to the values in the DataStore component's CategoryColorMap.

The value of Category is set in the Event edit dialog box's Category combo box that is populated with category strings. The category strings, are located in the DataStore component's CategoryColorMap.

Category is a persistent field and is stored in the external database.

See Event Edit Dialog on page 291.

See also: TVpCustomDataStore.CategoryColorMap

## Changed — property

```
property Changed : Boolean
```

Flag that determines if this event has been changed.

The value of Changed is checked as data is being saved to the database. If it is True, the event's data is updated in the database. If it is False, this record is skipped.

When one of the event's properties are updated, the changed flag is set accordingly.

## CustInterval — property

```
property CustInterval : Integer
```

CustInterval determines the repeat interval for a repeating event.

CustInterval is ignored if the value of RepeatCode is not rtCustom.

CustInterval is a persistent field and is stored in the external database.

See also: RepeatCode

## Deleted — property

```
property Deleted : Boolean
```

Flag that determines whether the event has been deleted and should be removed from the database.

The value of Deleted is checked as data is being written to the database. If it is True, the event is removed from the database, and released from memory. If it is False, this record is either updated in the database, (depending on the value of Changed) or skipped.

See also: Changed, TVpSchedule.DeleteEvent

## Description — property

```
property Description : string
```

Provides read/write access to the event's description field as a string.

Description is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Description is defined as a 255-character string called "Description."

See Database Design on page 157.

### EndTime — property

```
property EndTime : TDateTime
```

Provides read/write access to the event's end date and time as a TDateTime.

EndTime is a persistent field and is stored in the external database.

### ItemIndex — property

```
property ItemIndex : Integer
```

Provides the index of this particular event.

This value is assigned to the event object when it is created and added to the schedule. It is not a persistent, database field. It will change as the DataStore component's DayBuffer or Date properties change and the events are re-loaded into memory.

### Note — property

```
property Note : string
```

Provides read/write access to the event's Notes field as a string.

Note is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Note is defined as a 1024-character string called "Notes."

See Database Design on page 157.

### Owner — property

```
property Owner : TVpSchedule
```

Provides access to the schedule to which this event belongs.

### RecordID — property

```
property RecordID : Integer
```

Provides read access to the event's RecordID.

The RecordID is used by the external database as an index field. It is a persistent field and is stored in the external database.

## RepeatCode — property

```
property RepeatCode : TVpRepeatType
```

```
TVpRepeatType = (rtNone, rtDaily, rtWeekly, rtMonthlyByDay,
rtMonthlyByDate, rtYearlyByDay, rtYearlyByDate, rtCustom);
```

Determines the repeat pattern for a repeating event.

RepeatCode is a persistent field and is stored in the external database.

In the CustomDBDataStore class, RepeatCode is defined as a Word called "RepeatCode."

See Database Design on page 157.

See also: CustInterval

## SnoozeTime — property

```
property SnoozeTime : TDateTime
```

Specifies how much time to wait, before firing the alert again.

The following code snippet is from the TVpCustomDataStore's dsOnTimer procedure. AlarmTime is determined by subtracting AdvanceTime from StartTime and then adding SnoozeTime.

```
if (Event <> nil) and Event.AlarmSet then begin
  AdvanceTime := GetAlarmAdvanceTime(Event.AlarmAdv,
    Event.AlarmAdvType);
  AlarmTime := Event.StartTime - AdvanceTime
    + Event.SnoozeTime;
```

When the AlertDialog is displayed, it sets a timestamp. If the Snooze button is clicked, the value of SnoozeTime is determined by adding the length of snooze time to the amount of time that the dialog box has been displayed. This insures that the dialog box isn't fired again immediately, if it has been displayed longer than the value of SnoozeTime.

## StartTime — property

```
property StartTime : TDateTime
```

Provides read/write access to the event's start date and time as a TDateTime.

StartTime is a persistent field and is stored in the external database.

# TVpTasks Class

Maintains a list of tasks for one resource.

3

## Hierarchy

TObject (VCL)

TVpTasks (VpData)

## Properties

Owner

## Methods

| | | |
|---|---|---|
| AddTask | CountByDay | GetTask |
| BatchUpdate | DeleteTask | Last |
| ClearTasks | First | LastByDay |
| Count | FirstByDay | Sort |

### AddTask — method

```
function AddTask : TVpTask;
```

Adds a new task to the task list and returns a reference to it.

The following code snippet is from the TVpTaskList's internal tlSpawnTaskEditDialog procedure:

```
if NewTask then begin
  Task := DataStore.Resource.Tasks.AddTask;
  Task.CreatedOn := now;
  Task.DueDate := Now + 7;
end else
  Task := tlTask;
```

### BatchUpdate — method

```
procedure BatchUpdate(Value : Boolean);
```

Sets a flag that prevents flicker while many tasks are updated at once.

When the TVpTaskList component paints itself, it first checks the value of Tasks.BatchUpdate. If it is True, the paint procedure falls through without refreshing the control. Set BatchUpdate to True when you are making lots of changes to the task list at once, such as when you are loading a large amount of task data from a file.

```
procedure LoadLotsOfData;
begin
  Tasks.BatchUpdate(True);
  {Load lots of data into the task list}
  Tasks.BatchUpdate(False);
end;
```

Note: Calls to BatchUpdate are cumulative. Each call to BatchUpdate(True) must have a matching call to BatchUpdate(False) or the TaskList control will not ever refresh itself.

### ClearTasks — method

```
procedure ClearTasks;
```

Removes all tasks from the task list and destroys them.

ClearTasks does not permanently delete the tasks from the database. Rather, it is used in the event list's destructor to de-allocate the memory used by the event objects and clear the list.

See also: DeleteTask

### Count — method

```
function Count : Integer;
```

Returns the number of tasks in the list.

### CountByDay — method

```
function CountByDay(Date : TDateTime) : Integer;
```

Returns the number of tasks with a DueDate that matches the Date portion of the passed in TDateTime.

### DeleteTask — method

```
procedure DeleteTask(Task : TVpTask);
```

Sets the Task's Deleted flag to True.

Calling DeleteTask does not actually remove the task from memory. It simply marks the task as deleted, which will prevent it from being shown in the VpControls. The task will be deleted from the database and removed from memory during the DataStore's next PostTasks operation.

To permanently remove a task from the list, call DeleteTask, followed by a call to the DataStore's PostTasks method.

See also: ClearTask, TVpCustomDataStore.PostTasks

### First — method

```
function First : TVpTask;
```

Returns a reference to the first task in the list.

Useful for iterating the tasklist.

### FirstByDay — method

```
function FirstByDay(Date : TDateTime) : TVpTask;
```

Returns a reference to the first task of the date specified.

Useful for iterating tasks by day.

### GetTask — method

```
function GetTask(Index : Integer) : TVpTask;
```

Returns a reference to the task specified by the value of Index.

### Last — method

```
function Last : TVpTask;
```

Returns a reference to the last task in the list.

### LastByDay — method

```
function LastByDay(Date : TDateTime) : TVpTask;
```

Returns a reference to the last task of the date specified.

Useful for iterating tasks by day.

### Owner — property

```
property Owner : TVpResource
```

Returns a reference to the TVpResource to which this task list belongs.

### Sort — method

```
procedure Sort;
```

Sorts the tasks alphabetically by date.

The TaskList is sorted first by date, and then alphabetically by the tasks description.

# TVpTask Class

The TVpTask class models one of the resource's tasks.

3

## Hierarchy

TObject (VCL)

TVpTask (VpData)

## Properties

| | | |
|---|---|---|
| Changed | Deleted | ItemIndex |
| Complete | Description | Owner |
| CompletedOn | Details | RecordID |
| CreatedOn | DueDate | |

### Changed — property

```
property Changed : Boolean
```

Flag that determines if this task has been changed.

The value of Changed is checked as data is being saved to the database. If it is True, the task's data is updated in the database. If it is False, this record is skipped.

### Complete — property

```
property Complete : Boolean
```

Flag that specifies whether the task has been completed.

If the TaskList's DisplayOptions.ShowCompleted is True, completed tasks are drawn on the TaskList with a line through them, and with a checked check box next to them, for the rest of the day. If it is False, they disappear from the list immediately when they are completed.

Complete is a persistent field and is stored in the external database.

### CompletedOn — read-only property

```
property CompletedOn : TDateTime
```

Read-only property containing the date on which this task was completed.

CompletedOn is automatically set by the TVpTaskList component when the task's Completed check box is checked.

CompletedOn is a persistent field and is stored in the external database.

### CreatedOn — read-only property

```
property CreatedOn : TDateTime
```

Read-only property containing the date on which this task was created.

CreatedOn is a persistent field and is stored in the external database.

### Deleted — property

```
property Deleted : Boolean
```

Flag that determines whether the task has been deleted and should be removed from the database.

The value of Deleted is checked as data is being written to the database. If it is True, the task is removed from the database, and released from memory. If it is False, this task's record is updated in the database, depending on the value of Changed.

See also: Changed, TVpTasks.DeleteTask

### Description — property

```
property Description : string
```

String field that contains the task's description.

The task's description is the field that is displayed in the TVpTaskList component.

Description is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Description is defined as a 255-character string called "Description".

See Database Design on page 157.

### Details — property

```
property Details : string
```

String field that contains the task's details.

Details is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Details is defined as a 1024-character string called "Details".

See Database Design on page 157.

### DueDate — property

```
property DueDate : TDateTime
```

Determines the task's due date.

The tasks DueDate is used to determine whether the task is overdue.

DueDate is a persistent field and is stored in the external database.

## ItemIndex — property

```
property ItemIndex : Integer
```

Provides the index of this particular task.

This value is assigned to the task object when it is created and added to the internal task list. It is not a persistent, database field. It may change as tasks are added and deleted.

## Owner — read-only property

```
property Owner : TVpTasks
```

Read-only property that returns a reference to the TaskList instance to which this particular task belongs.

See also: TVpTasks

## RecordID — property

```
property RecordID : Integer
```

Provides access to the task's ID.

The RecordID is used by the external database as an index field.

RecordID is a persistent field and is stored in the external database.

# TVpContacts Class

Maintains a list of contacts for a resource.

3

## Hierarchy

TObject (VCL)

TVpContacts (VpData)

## Properties

ContactsList

## Methods

| | | |
|---|---|---|
| AddContact | Count | GetContact |
| BatchUpdate | DeleteContact | Last |
| ClearContacts | First | Sort |

## AddContact — method

```
function AddContact : TVpContact;
```

Adds a new contact item to the list and returns a reference to it.

Use AddContact when you want to create a new Contact item and add it to the list. Items in the list are automatically saved to the database.

For example:

```
procedure AddNewContact(Resource : TVpResource);
var
 Contact: TVpContact;
begin
 Contact := Resource.Contacts.AddContact;
 Contact.LastName := 'Blanton';
 Contact.FirstName := 'Phillip';
 Contact.Company := 'TurboPower Software Company';
 Contact.Address := '15 North Nevada Avenue';
 Contact.City := 'Colorado Springs';
 Contact.State := 'Colorado';
 Contact.Zip := '80903';
 Contact.BirthDate :=StrToDateTime('8/8/64');
 Contact.Note := 'Send money for his birthday!';
 {Etc...}
end;
```

## BatchUpdate — method

```
procedure BatchUpdate(Value : Boolean);
```

Sets a flag that prevents flicker as many contacts are updated at once.

When the TVpContactGrid component paints itself, it first checks the value of Contacts.BatchUpdate. If it is True, the paint procedure falls through without refreshing the control. Set BatchUpdate to True when you are making lots of changes to the contacts list at once, such as when you are loading a large amount of contacts data from a file.

```
procedure LoadLotsOfData;
begin
  Contacts.BatchUpdate(True);
  {Load lots of data into the contacts list}
  Contacts.BatchUpdate(False);
end;
```

Note: Calls to BatchUpdate are cumulative. Each call to BatchUpdate(True) must have a matching call to BatchUpdate(False) or the ContactGrid control will not ever refresh itself.

## ClearContacts — method

```
procedure ClearContacts;
```

Removes all contacts from the list and destroys them.

ClearContacts does not permanently delete the contacts from the database. Rather, it is used in the DataStore's destructor to free each contact from memory and clear the contacts list.

To permanently remove contacts, use the DeleteContact method.

See also: DeleteContact

## ContactsList — property

```
property ContactsList : TList
```

Provides direct access to the object's internal list.

The internal list is a TList of the individual contact objects.

### Count — method

```
function Count : Integer;
```

Returns the number of contacts in the list.

### DeleteContact — method

```
procedure DeleteContact(Contact : TVpContact);
```

Flags the contact as deleted.

Does not actually remove the contact from the list. The contact isn't removed from the database and freed from memory until the DataStore's next PostContacts operation.

To permanently remove contacts from the list, call DeleteContact followed by a call to the DataStore's PostContacts method.

See also: TVpContact.Delete, TVpCustomDataStore.PostContacts.

### First — method

```
function First : TVpContact;
```

Provides access to the first item in the list.

See also: Last

### GetContact — method

```
function GetContact(Index : Integer) : TVpContact;
```

Returns a reference to the individual contact specified by the value of Index.

### Last — method

```
function Last : TVpContact;
```

Provides access to the last item in the list.

See also: First

### Sort — method

```
procedure Sort;
```

Sorts the contacts alphabetically.

The ContactsList is sorted first by LastName and then by FirstName.

# TVpContact Class

Models one of the resources contact records. Each instance of TVpContact represents one record in the Contacts database.

3

## Hierarchy

TObject (VCL)

TVpContact (VpData)

## Properties

| | | |
|---|---|---|
| Address | Deleted | PhoneType1 |
| Category | EMail | PhoneType2 |
| Changed | FirstName | PhoneType3 |
| City | LastName | PhoneType4 |
| Company | Note | PhoneType5 |
| Country | Phone1 | Position |
| Custom1 | Phone2 | RecordID |
| Custom2 | Phone3 | State |
| Custom3 | Phone4 | Title |
| Custom4 | Phone5 | Zip |

### Address — property

```
property Address : string
```

Provides read/write access to the contact's Address field.

Address is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Address is defined as a 100-character string called "Address."

See Database Design on page 157.

### Category — property

```
property Category : string
```

Provides read/write access to the contact's Category field.

Category is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Category is defined as an Integer field called "Category."

See Database Design on page 157.

### Changed — property

```
property Changed : Boolean
```

Flag that determines if this resource has been changed.

The value of Changed is checked as data is being saved to the database. If it is True, the resource's data is updated in the database. If it is False, this record is skipped.

### City — property

```
property City : string
```

Provides read/write access to the contact's City field.

City is a persistent field and is stored in the external database.

In the CustomDBDataStore class, City is defined as a 50-character string called "City."

See Database Design on page 157.

## Company — property

```
property Company : string
```

Provides read/write access to the contact's Company field.

Company is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Company is defined as a 50-character string field called "Company."

See Database Design on page 157.

## Country — property

```
property Country : string
```

Provides read/write access to the contact's Country field.

Country is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Country is defined as a 25-character string field called "Country."

See Database Design on page 157.

## Custom1 — property

```
property Custom1 : string
```

Provides read/write access to the contact's Custom1 field.

There are four custom fields that you may use as you wish.

Custom1 is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Custom1 is defined as a 100-character string called "Custom1."

See Database Design on page 157.

## Custom2 property

```
property Custom2 : string
```

Provides read/write access to the contact's Custom2 field.

There are four custom fields that you may use as you wish.

Custom2 is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Custom2 is defined as a 100-character string called "Custom2."

See Database Design on page 157.

## Custom3 property

```
property Custom3 : string
```

Provides read/write access to the contact's Custom3 field.

There are four custom fields that you may use as you wish.

Custom3 is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Custom3 is defined as a 100-character string called "Custom3."

See Database Design on page 157.

## Custom4 property

```
property Custom4 : string
```

Provides read/write access to the contact's Custom4 field.

There are four custom fields that you may use as you wish.

Custom4 is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Custom4 is defined as a 100-character string called "Custom4."

See Database Design on page 157.

### Deleted — property

```
property Deleted : Boolean
```

Flag that determines whether the contact has been deleted and should be removed from the database.

The value of Deleted is checked as data is being written to the database. If it is True, the contact is removed from the database, and released from memory. If it is False, this contact's record is updated in the database, depending on the value of Changed.

See also: Changed, TVpContacts.DeleteContact.

### EMail — property

```
property EMail : string
```

Provides read/write access to the contact's EMail address field.

EMail is a persistent field and is stored in the external database.

In the CustomDBDataStore class, EMail is defined as a 100-character string called "EMail."

See Database Design on page 157.

### FirstName — property

```
property FirstName : string
```

Provides read/write access to the contact's FirstName field.

FirstName is a persistent field and is stored in the external database.

In the CustomDBDataStore class, FirstName is defined as a 50-character string called "FirstName."

See Database Design on page 157.

### LastName — property

```
property LastName : string
```

Provides read/write access to the contact's LastName field.

LastName is a persistent field and is stored in the external database.

In the CustomDBDataStore class, LastName is defined as a 50-character string called "LastName."

See Database Design on page 157.

## Note — property

```
property Note : string
```

Provides read/write access to the contact's Note field.

Note is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Note is defined as a 1024-character string called "Note."

See Database Design on page 157.

## Phone1 — property

```
property Phone1 : string
```

Provides read/write access to the contact's first telephone field.

Phone1 is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Phone1 is defined as a 25-character string called "Phone1."

See Database Design on page 157.

## Phone2 — property

```
property Phone2 : string
```

Provides read/write access to the contact's second telephone field.

Phone2 is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Phone2 is defined as a 25-character string called "Phone2."

See Database Design on page 157.

## Phone3 — property

```
property Phone3 : string
```

Provides read/write access to the contact's third telephone field.

Phone3 is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Phone3 is defined as a 25-character string called "Phone3."

See Database Design on page 157.

## Phone4 property

```
property Phone4 : string
```

Provides read/write access to the contact's fourth telephone field.

Phone4 is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Phone4 is defined as a 25-character string called "Phone4."

See Database Design on page 157.

## Phone5 property

```
property Phone5 : string
```

Provides read/write access to the contact's fifth telephone field.

Phone5 is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Phone5 is defined as a 25-character string called "Phone5."

See Database Design on page 157.

## PhoneType1 property

```
property PhoneType1 : TVpPhoneType
```

```
TVpPhoneType = (
  ptWork, ptHome, ptFax, ptMain, ptPager, ptMobile, ptOther);
```

Provides read/write access to the contact's PhoneType1 field.

The TVpPhoneType is an enumerated type that represents the types associated with each of the telephone fields.

PhoneType1 is a persistent field and is stored in the external database.

In the CustomDBDataStore class, PhoneType1 is defined as an Integer field called "PhoneType1."

See Database Design on page 157.

## PhoneType2 property

```
property PhoneType2 : TVpPhoneType
```

```
TVpPhoneType = (
  ptWork, ptHome, ptFax, ptMain, ptPager, ptMobile, ptOther);
```

Provides read/write access to the contact's PhoneType2 field.

The TVpPhoneType is an enumerated type that represents the types associated with each of the telephone fields.

PhoneType2 is a persistent field and is stored in the external database.

In the CustomDBDataStore class, PhoneType2 is defined as an Integer field called "PhoneType2."

See Database Design on page 157.

## PhoneType3 property

```
property PhoneType3 : TVpPhoneType
```

```
TVpPhoneType = (
  ptWork, ptHome, ptFax, ptMain, ptPager, ptMobile, ptOther);
```

Provides read/write access to the contact's PhoneType3 field.

The TVpPhoneType is an enumerated type that represents the types associated with each of the telephone fields.

PhoneType3 is a persistent field and is stored in the external database.

In the CustomDBDataStore class, PhoneType3 is defined as an Integer field called "PhoneType3."

See Database Design on page 157.

## PhoneType4 — property

```
property PhoneType4 : TVpPhoneType
```

```
TVpPhoneType = (
  ptWork, ptHome, ptFax, ptMain, ptPager, ptMobile, ptOther);
```

Provides read/write access to the contact's PhoneType4 field.

The TVpPhoneType is an enumerated type that represents the types associated with each of the telephone fields.

PhoneType4 is a persistent field and is stored in the external database.

In the CustomDBDataStore class, PhoneType4 is defined as an Integer field called "PhoneType4."

See Database Design on page 157.

## PhoneType5 — property

```
property PhoneType5 : TVpPhoneType
```

```
TVpPhoneType = (
  ptWork, ptHome, ptFax, ptMain, ptPager, ptMobile, ptOther);
```

Provides read/write access to the contact's PhoneType5 field.

The TVpPhoneType is an enumerated type that represents the types associated with each of the telephone fields.

PhoneType5 is a persistent field and is stored in the external database.

In the CustomDBDataStore class, PhoneType5 is defined as an Integer field called "PhoneType5."

See Database Design on page 157.

## Position — property

```
property Position : string
```

Provides read/write access to the contact's Position field.

Position is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Position is defined as a 30-character string field called "Position."

See Database Design on page 157.

## RecordID — property

```
property RecordID : Integer
```

Provides read access to the contact's RecordID.

The RecordID is used by the external database as an index field. It is a persistent field and is stored in the external database.

In the CustomDBDataStore class, RecordID is defined as an Integer field called "RecordID." It is used by the dataset as a primary index.

See Database Design on page 157.

## State — property

```
property State : string
```

Provides read/write access to the contact's State field.

State is a persistent field and is stored in the external database.

In the CustomDBDataStore class, State is defined as a 25-character string field called "State."

See Database Design on page 157.

## Title — property

```
property Title : string
```

Provides read/write access to the contact's Title field.

Title is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Title is defined as a 50-character string field called "Title."

See Database Design on page 157.

## Zip — property

```
property Zip : string
```

Provides read/write access to the contact's zip code field.

Zip is a persistent field and is stored in the external database.

In the CustomDBDataStore class, Zip is defined as a 10-character string field called "Zip."

See Database Design on page 157.

# TVpTimeRange Class

The TVpTimeRange class models a single block of time. It is used by the TVpBaseDataStore component to determine the range of time for which events and tasks will be loaded into memory. It is also used by the events to contain their start and end times.



## Hierarchy

TObject (VCL)

TVpTimeRange (VpData)

## Properties

| | | |
|---|---|---|
| EndTime | Owner | StartTime |

### EndTime — property

```
property EndTime : TDateTime
```

Provides read/write access to the TimeRange's ending time.

The value of StartTime must be less than the value of EndTime. If EndTime is set to a value less than StartTime, the value of StartTime will default to 30 minutes earlier than the new value of EndTime.

### Owner — property

```
property Owner : TObject
```

Provides access to the TimeRange's owner.

Owner is the object to which the TimeRange instance belongs. It is passed to the TimeRange's constructor.

### StartTime — property

```
property StartTime : TDateTime
```

Provides read/write access to the TimeRange's starting time.

The value of StartTime must be less than the value of EndTime. If StartTime is set to a value greater than EndTime, the value of EndTime will default to 30 minutes later than the new value of StartTime.

# Chapter 4: Visual PlanIt Database Design

This chapter provides information on the expected layout of a Visual PlanIt database back end. It is based on the design for the BDEDataStore's database.

Visual PlanIt expects certain fields to be in a certain format, but if you are writing your own DataStore component, you may of course design things as you wish. You just may have to do a lot of converting back and forth while loading data into, or posting data from the Visual PlanIt objects.

The Visual PlanIt BDEDataStore component utilizes the following tables:

- Resources
- Events
- Contacts
- Tasks

They are defined in the following sections.

# Resources Table

The resources table stores the information that makes up each resource (see Table 4.1).

Table 4.1: *Resources table structure*

| FieldName | Type | Size | Key |
|---|---|---|---|
| ResourceID | Integer | | * |
| Description | String | 255 | |
| Notes | String | 255 | |
| ImageIndex | Integer | | |
| Active | Logical | | |

# Events Table

The events table stores information on all of the events in the schedules for each resource (see Table 4.2).

Table 4.2: *Event table structure*

| FieldName | Type | Size | Key |
|---|---|---|---|
| RecordID | Integer | | * |
| StartTime | TimeStamp | | |
| EndTime | TimeStamp | | |
| ResourceID | String | 20 | |
| Description | String | 255 | |
| Notes | String | 255 | |
| Category | Integer | | |
| AllDayEvent | Logical | | |
| DingPath | String | 255 | |
| AlarmSet | Logical | | |
| AlarmAdvance | Integer | | |
| AlarmAdvanceType | Number | | |
| SnoozeTime | TimeStamp | | |
| RepeatCode | Number | | |
| CustomInterval | Integer | | |

# Contacts Table

The contacts table stores the contacts for each resource (see Table 4.3).

Table 4.3: *Contacts table structure*

| FieldName | Type | Size | Key |
|---|---|---|---|
| RecordID | Integer | | * |
| ResourceID | Integer | | |
| FirstName | String | 50 | |
| LastName | String | 50 | |
| Title | String | 50 | |
| Company | String | 50 | |
| Position | String | 30 | |
| Address | String | 100 | |
| City | String | 50 | |
| State | String | 25 | |
| Zip | String | 10 | |
| Country | String | 25 | |
| Note | String | 255 | |
| Phone1 | String | 25 | |
| Phone2 | String | 25 | |
| Phone3 | String | 25 | |
| Phone4 | String | 25 | |
| Phone5 | String | 25 | |
| PhoneType1 | Integer | | |
| PhoneType2 | Integer | | |
| PhoneType3 | Integer | | |
| PhoneType4 | Integer | | |
| PhoneType5 | Integer | | |
| Category | Integer | | |
| EMail | String | 100 | |

Table 4.3: *Contacts table structure (continued)*

| FieldName | Type | Size | Key |
|---|---|---|---|
| Custom1 | String | 100 | |
| Custom2 | String | 100 | |
| Custom3 | String | 100 | |
| Custom4 | String | 100 | |

# Tasks Table

The tasks table stores all of the task list information for each resource (see Table 4.4).

Table 4.4: *Task table structure*

| FieldName | Type | Size | Key |
|---|---|---|---|
| RecordID | Integer | | * |
| ResourceID | Integer | | |
| Complete | Logical | | |
| Description | String | 255 | |
| Details | String | 255 | |
| CreatedOn | TimeStamp | | |
| CompletedOn | TimeStamp | | |
| DueDate | TimeStamp | | |

# Chapter 5: Visual PlanIt Components

This chapter provides information on the components that are installed on the Visual PlanIt palette page. It covers all of the palette page components, except for the DataStore components which are covered in “Chapter 3: Visual PlanIt Data Storage Building Blocks” on page 89.

Important support classes are documented here and other, less important classes, are not. Very simple support classes are only documented in the source.

The following components and their support classes and routines are described in this chapter:

- Control Link Component
- LED Label Component
- Clock Component

  TVpCustomClock

  TVpAnalogOptions

  TVpDigitalOptions
- Calendar Component
- Navigation Bar Component

  TVpCustomNavBar

  TVpNavFolder

  TVpNavBtnItem
- Print Preview Component
- Print Format Combo Box Component
- Date Edit Component

  TVpCustomDateEdit
- Resource Combo Component

- Day View Component

  TVpCHAttributes

  TVpRHAttributes

  TVpTimeSlotColor
- Week View Component
- Month View Component
- Contact Grid Component

  TVpContactHeadAttr
- Task List Component
- Resource Edit Dialog Component
- Event Edit Dialog Component
- Contact Edit Dialog Component
- Task Edit Dialog Component
- Notification Dialog Component
- Print Preview Dialog Component

# TVpControlLink Component

The ControlLink is the component that is in charge of synchronizing the individual components with each other.

When a ControlLink is dropped on the form, it iterates through the form's controls and connects itself to any TVpLinkableControl descendants with an empty ControlLink property.

## Hierarchy

TComponent (VCL)

❶ TVpComponent (VPBase). . . . . 28

TVpControlLink (VPBaseDS)

## Properties

| | | |
|---|---|---|
| Printer | ❶ Version | |

## Methods

| | | |
|---|---|---|
| GetDependentList | TriggerOnGetVariable | TriggerOnPageStart |
| Notify | TriggerOnPageEnd | |

## Events

| | | |
|---|---|---|
| OnGetVariable | OnPageEnd | OnPageStart |

### GetDependentList — method

```
function GetDependentList : TList;
```

Returns a TList object containing an instance of the TVpDependentInfo class for each of the dependent controls.

Iterate the list returned by GetDependentList to access each of the ControlLink's dependent components.



See also: TVpDependentInfo

### Notify — method

```
procedure Notify(Sender : TVpLinkableControl;
 NotificationType : TVpNotificationType; const Value : Variant);
```

```
TVpNotificationType = (
  neDateChange, neResourceChange, neDataStoreChange, neInvalidate);
```

Dependent controls call Notify when they have made changes that need to be communicated to other dependent controls.

### OnGetVariable — event

```
property OnGetVariable : TVpOnGetVariableEvent
```

```
TVpOnGetVariableEvent = procedure(Sender : TObject;
  VarName : string; Found : Boolean; var Value : string;
  var Change : TVpChangeVar) of object;
```

```
TVpChangeVar = (cvRemove, cvIgnore, cvChange);
```

Defines an event handler for when print caption variables are evaluated.

The OnGetVariable fires when anything looking like variable has been found in a print caption. Variable names are defined as '$' followed by alphanumeric characters or a '_'. Any other character terminates the variable name. The Found flag indicates if this variable was found in the variable list.

The current value of the variable is passed in the Value parameter. This can be changed to temporarily override the value of the variable. The actual print caption variable will not be changed as a result of changing value.

The Change parameter determines how the variable will be printed. A value of cvChange will print the value passed in the ValueParameter, cvRemove will not print anything for the variable and cvIgnore will ignore that a variable was found and just print the variable name.

The OnGetVariable event can be used to create variables that otherwise would be hard to define. For example, OnGetVariable could be used to do database lookups to insert data fields into the print job.

This event also allows for variable filtering. Usage of the Change parameter can make variables conditional under certain circumstances.

### OnPageEnd — event

```
property OnPageEnd : TVpOnPageEndEvent
```

```
TVpOnPageEndEvent = procedure(Sender : TObject; PageNum : Integer;
  ADate : TDateTime; LastPage : Boolean) of object;
```

Defines an event handler for when a page is finished printing.

When printing pages using the Print method, this event fires after each page is printed. Note that ADate is the date of the next page to print while PageNum is the page number of the page that just printed. LastPage is True if this is the last page of the print job.

See also: OnPageStart

### OnPageStart — event

```
property OnPageStart : TVpOnPageStartEvent
```

```
TVpOnPageStartEvent = procedure(Sender : TObject;
  PageNum : Integer; ADate : TDateTime) of object;
```

Defines an event handler for just before a page is printed.

When printing pages using the Print method, this event fires just before each page is rendered. The PageNum and ADate parameters indicate the page number and date that is being printed.

See also: OnPageEnd

### Printer — property

```
property Printer : TVpPrinter
```

Defines the print formats.

The Printer property is a TCollection that defines all the print formats and the print elements that make up those formats. In addition this property contains methods for loading and saving the formats, adding print caption variables and data needed for running the print jobs. See the TVpPrinter class on page 75 for more information.



### TriggerOnGetVariable — method

```
procedure TriggerOnGetVariable(
  Sender : TObject; VarName : string; Found : Boolean;
  var Value : string; var Change : TVpChangeVar);
```

Triggers the OnGetVariable event.

This is an internal method that should not be called manually.

### TriggerOnPageEnd — method

```
procedure TriggerOnPageEnd(Sender : TObject;
  PageNum : Integer; ADate : TDateTime; LastPage : Boolean);
```

Triggers the OnPageEnd event.

This is an internal method that should not be called manually.

### TriggerOnPageStart — method

```
procedure TriggerOnPageStart(Sender : TObject;
  PageNum : Integer; ADate : TDateTime);
```

Triggers the OnPageStart event.

This is an internal method that should not be called manually.

# TVpCustomLEDLabel Class

The TVpCustomLEDLabel class is the immediate ancestor of the TVpLEDLabel component. It implements all the methods and properties used by the TVpLEDLabel component and is identical to the TVpLEDLabel component except that no properties are published.

TVpCustomLEDLabel is provided to facilitate creation of descendent label components. For property and method descriptions, see the "TVpLEDLabel Component" on page 170.

## Hierarchy

TGraphicControl (VCL)

    TVpCustomLEDLabel (VPLEDLabel)

# TVpLEDLabel Component

The LEDLabel is the component that is in the TVpClock to create the digital display. It is surfaced here as a bonus for your use in your own applications.

The TVpLEDLabel is only capable of displaying the following characters:

```
Characters: Array[0..72] of Word =
($0000,$3B70,$1320,$0001,$0300,$0002,$0840,$CCCC,$1020,$8784,
{ ' '    *      +    ,      -     .     /      0     1     2  }
 $870C,$4708,$C30C,$C38C,$8408,$C78C,$C70C,$0810,$2040,$C788,
{   3     4     5     6     7     8     9     <     >     A  }
 $952C,$C084,$942C,$C384,$C380,$C18C,$4788,$9024,$048C,$4A90,
{   B     C     D     E     F     G     H     I     J     K  }
 $4084,$6C88,$6498,$C48C,$C780,$C49E,$C790,$C214,$9020,$448C,
{   L     M     N     O     P     Q     R     S     T     U  }
 $48C0,$44D8,$2850,$2820,$8844,$2010,$C788,$952C,$C084,$942C,
{   V     W     X     Y     Z     /     a     b     c     d  }
 $C384,$C380,$C18C,$4788,$9024,$048C,$4A90,$4084,$6C88,$6498,
{   e     f     g     h     i     j     k     l     m     n  }
 $C48C,$C780,$C49E,$C790,$C214,$9020,$448C,$48C0,$44D8,$2850,
{   o     p     q     r     s     t     u     v     w     x  }
 $2820,$8844,$FFFF);
{   y     z     : }
```

The LED Segment map is as follows:

```
        LED Segment Map

   ------------------------
   |          1           |
   ------------------------
   |  | \    |  |    / |  |
   |  |  \   |  |   /  |  |
   |  |   \  |  |  /   |  |
   |2 |\3  \ |4 | /5  /|6 |
   |  | \   \|  |/   / |  |
   |  |  \   |  |   /  |  |
   -----------  -----------
   |   7      \/   8      |
   -----------/\-----------
   |  |  /   |  |   \  |  |
   |  | /   /|  |\   \ |  |
   |9 |/10 / |11| \12 \|13|
   |  |   /  |  |  \   |  |
   |  |  /   |  |   \  |  |
   |  | /    |  |    \ |  |
   ------------------------ |-----|
   |          14          | |  *  |
   ------------------------ |-----|
```

* Period and comma are drawn here. Colon is drawn in the center of segments 4 and 11.

Each segment is made up of 6 points. The segments that don't need 6 points, such as the period and colon dots, return to the coordinates of the initial point for each of the remaining unused points.

## Hierarchy

TGraphicControl (VCL)

TVpCustomLEDLabel (VPLEDLabel). . . . . 169

TVpLEDLabel (VPLEDLabel)

## Properties

| | | |
|---|---|---|
| BgColor | OffColor | Size |
| Caption | OnColor | Version |
| Columns | Rows | |

5

## Reference Section

### BgColor — property

```
property BgColor : TColor
```

Default: clBlack

Specifies the color of the display's background.

### Caption — property

```
property Caption : string
```

Contains the string that will be painted into the display.

Note: Not all characters lend themselves to a segmented LED display. For instance '?', '!', '@', '"', '%' are all examples of characters that won't display in the LED Label.

Following is a list of supported characters:

```
Characters: Array[0..72] of Word =
($0000,$3B70,$1320,$0001,$0300,$0002,$0840,$CCCC,$1020,$8784,
{ ' '    *      +    ,      -     .     /      0     1     2  }
 $870C,$4708,$C30C,$C38C,$8408,$C78C,$C70C,$0810,$2040,$C788,
{   3     4     5     6     7     8     9     <     >     A  }
 $952C,$C084,$942C,$C384,$C380,$C18C,$4788,$9024,$048C,$4A90,
{   B     C     D     E     F     G     H     I     J     K  }
 $4084,$6C88,$6498,$C48C,$C780,$C49E,$C790,$C214,$9020,$448C,
{   L     M     N     O     P     Q     R     S     T     U  }
 $48C0,$44D8,$2850,$2820,$8844,$2010,$C788,$952C,$C084,$942C,
{   V     W     X     Y     Z     /     a     b     c     d  }
 $C384,$C380,$C18C,$4788,$9024,$048C,$4A90,$4084,$6C88,$6498,
{   e     f     g     h     i     j     k     l     m     n  }
 $C48C,$C780,$C49E,$C790,$C214,$9020,$448C,$48C0,$44D8,$2850,
{   o     p     q     r     s     t     u     v     w     x  }
 $2820,$8844,$FFFF);
{   y     z     : }
```

### Columns — property

```
property Columns : Integer
```

Specifies the number of characters in the display.

### OffColor — property

```
property OffColor: TColor
```

Default: $000E3432 (dark green)

Specifies the color of the un-lit (darkened) segments.



### OnColor — property

```
property OnColor : TColor
```

Default: clLime

Specifies the color of the lit segments.

### Rows — property

```
property Rows : Integer
```

Specifies the number of rows in the display.

### Size — property

```
property Size : Integer
```

Determines the size of the display.

Size is a little bit misleading. It is the thickness of the segments, in pixels. The allowable values are between 2 and 10. A value of 10 makes a very large display.

### Version — read-only property

```
property Version : string
```

Displays the library version in the Object Inspector.

The Version property also has a specialized design-time property editor that displays the Visual PlanIt about box when the ellipsis button is clicked.

# TVpCustomClock Class

The TVpCustomClock class is the immediate ancestor of the TVpClock component. It implements all the methods and properties used by the TVpClock component and is identical to the TVpClock component except that no properties are published.

TVpCustomClock is provided to facilitate creation of descendent clock components. For property and method descriptions, see “TVpClock Component” on page 176.

## Hierarchy

TCustomControl (VCL)

- TVpCustomControl (VPBase). . . . . 30
  - TVpCustomClock (VPClock)

# TVpClock Component

The TVpClock component displays an analog clock on a form. The clock can be used as a real-time clock or it can be used as a static time display. When used as a real-time clock, the clock displays the system's current time, updated every second. When used as a static time display, the clock displays a time as determined by the Time property. A static time display can be used to display schedules, meeting times, and so on.

Both the clock face and the clock hands are customizable. The clock face is determined by the ClockFace property. If ClockFace is not assigned, the clock will be drawn with a default clock face. To display a picture on a clock face, assign a bitmap to the ClockFace property. The bitmap supplied will be used as the clock face with the hands drawn on top of the clock face. Visual PlanIt comes with several clock faces that you can use in your applications. These bitmaps can be found in the Examples directory. The clock hands can be customized to modify the hand length, color, and fill style. The second hand can be shown or hidden as desired.

The clock is also capable of displaying time digitally as well. The background color, lit element color and un-lit element colors are all customizable, as well as 24 hour time format.



## Hierarchy

TCustomControl (VCL)

- ❶ TVpCustomControl (VPBase) . . . . . 30
  - TVpCustomClock (VPClock) . . . . . 175
    - TVpClock (VPClock)

## Properties

| | | |
|---|---|---|
| Active | ElapsedDays | HourOffset |
| AnalogOptions | ElapsedHours | MinuteOffset |
| ClockMode | ElapsedMinutes | SecondOffset |
| DigitalOptions | ElapsedSeconds | |
| DisplayMode | ElapsedSecondsTotal | |

## Events

| | |
|---|---|
| OnCountdownDone | OnMinuteChange |
| OnHourChange | OnSecondChange |

## Active — property

```
property Active : Boolean
```

Default: False

Determines whether the clock shows the system time in real time.

The behavior of the Active property depends on the current ClockMode (see the following table).

| Option | Description |
|---|---|
| cmClock | Set Active to True to display the system time in real time. Set Active to False if you want the clock to display a static time. Assign a TDateTime value to the Time property to set the static time. |
| cmTimer | Setting Active to True resets the timer at 12:00 and starts the timer counting up. Setting Active to False stops the timer. Read the ElapsedTime property to determine the elapsed time of the clock in cmTimer mode. |
| cmCountdownTimer | Set the start time with TimeOffset and MinuteOffset. Setting Active to True sets the timer at the time specified by TimeOffset and MinuteOffset, and begins counting down to zero. Setting Active to False before the countdown timer reaches zero will stop the timer. The timer will automatically set Active to False when it reaches zero. |

See also: ClockMode, ElapsedTime, Time

## AnalogOptions — property

```
property AnalogOptions : TVpHandOptions
```

Groups the analog setting properties together in the Object Inspector.

AnalogOptions are the properties that define the rendering of the clock when DisplayMode is set to dmAnalog.

See the “TVpHandOptions Class” on page 183 for more information.

See also: DisplayMode, TVpHandOptions

## ClockMode property

```
property ClockMode : TVpClockMode
```

```
TVpClockMode = (cmClock, cmTimer, cmCountdownTimer);
```

Default: cmClock

The mode in which the clock operates, as shown in the following table:

| Mode | Description |
|---|---|
| cmClock | The clock displays a time of day. |
| cmTimer | The clock shows the current elapsed time of a timer. To start a timer set ClockMode to cmTimer and the Active property to True. To stop the timer, set Active to False. Time is automatically set to 12:00 AM when the Active property is set to True if ClockMode is cmTimer. The ElapsedSeconds property can be used to determine the elapsed time when the clock is used in timer mode. |
| cmCountdownTimer | The clock shows the amount of time left on the timer. Set the amount of time for the countdown timer with TimeOffset and MinuteOffset properties. When the Active property is set to True, the clock's time is set to the defined start time and the clock begins to tick off time backwards toward zero. If the Active property is set to False before the clock reaches zero, the clock stops. When the clock reaches zero, the Active property is set to False automatically. Elapsed time can be read at any time. |

See also: Active, ElapsedSecondsTotal, Time

## DigitalOptions property

```
property DigitalOptions : TVpClockDigitalOptions
```

Determines how the clock appears when it DisplayMode is dmDigital.

DigitalOptions is an instance of the TVpClockDigitalOptions class. Options include the LED segment off color and LED segment on color, display back color, whether to show seconds, and whether the separator flashes with each second.

See the "TVpClockDigitalOptions Class" on page 188 for more information on DigitalOptions.

## DisplayMode — property

```
property DisplayMode : TVpClockDisplayMode
```

```
TVpClockDisplayMode = (dmAnalog, dmDigital);
```

Default: dmDigital

Determines whether the clock will be displayed as a digital or analog clock.

When DisplayMode is dmDigital the clock uses DigitalOptions to determine how it appears. Otherwise, it uses AnalogOptions to determine how the clock appears.

See also: AnalogOptions, DigitalOptions

5

## ElapsedDays — read-only, run-time property

```
property ElapsedDays : Integer
```

Returns the number of elapsed days when ClockMode is cmTimer or cmCountdownTimer.

Read ElapsedDays to determine the number of elapsed days since the timer was started. The elapsed time (in milliseconds) is stored internally in an unsigned long integer variable so the number of elapsed days will reset to 0 after 49.7 days have elapsed.

See also: ElapsedHours, ElapsedMinutes, ElapsedSeconds, ElapsedSecondsTotal

## ElapsedHours — read-only, run-time property

```
property ElapsedHours : Integer
```

Returns the number of elapsed hours when ClockMode is cmTimer or cmCountdownTimer.

ElapsedHours does not return the total elapsed hours, but rather is the hours component of elapsed days, hours, minutes, and seconds. To determine the total elapsed time in hours, read the ElapsedSecondsTotal property and divide by 360.

See also: ElapsedDays, ElapsedMinutes, ElapsedSeconds, ElapsedSecondsTotal

## ElapsedMinutes — read-only, run-time property

```
property ElapsedMinutes : LongInt
```

Returns the number of elapsed minutes when ClockMode is cmTimer or cmCountdownTimer.

ElapsedMinutes does not return the total elapsed minutes, but rather is the minutes component of elapsed days, hours, minutes, and seconds. To determine the total elapsed time in minutes, read the ElapsedSecondsTotal property and divide by 60.

See also: ElapsedDays, ElapsedHours, ElapsedSeconds, ElapsedSecondsTotal



## ElapsedSeconds — read-only, run-time property

```
property ElapsedSeconds : LongInt
```

Returns the number of elapsed seconds when ClockMode is cmTimer or cmCountdownTimer.

ElapsedSeconds does not return the total elapsed seconds, but rather is the seconds component of elapsed days, hours, minutes, and seconds. To determine the total elapsed time in seconds, read the ElapsedSecondsTotal property. The elapsed time (in milliseconds) is stored internally in an unsigned long integer variable so the number of elapsed seconds will reset to 0 after 4,294,967 seconds have elapsed (49.7 days).

See also: ElapsedDays, ElapsedHours, ElapsedMinutes, ElapsedSecondsTotal

## ElapsedSecondsTotal — read-only, run-time property

```
property ElapsedSecondsTotal : LongInt
```

Returns the total number of elapsed seconds when ClockMode is cmTimer or cmCountdownTimer.

The elapsed time (in milliseconds) is stored internally in an unsigned long integer variable so the total number of elapsed seconds will reset to 0 after 4,294,967 seconds have elapsed (49.7 days).

See also: ElapsedDays, ElapsedHours, ElapsedMinutes, ElapsedSeconds

## HourOffset — property

```
property HourOffset : Integer
```

The hour offset from the current time zone.

The clock's time is taken from the system's current time. When HourOffset is 0, the current local time is displayed. To display times in other time zones, set HourOffset to the relative time zone you wish to display. If, for example, you are in the United States Mountain time zone and you want to display a clock showing Pacific time you would set HourOffset to –1. To display Eastern time (two time zones earlier) you would set HourOffset to 2.

HourOffset is also used for the CountDown timer. When the clock's mode is set to cmCountdownTimer, the clock's display is set to 0:00 (12:00 in analog mode). Set the hour and minute offsets, to represent the timer's set time and set Active to True. The clock will display the amount of time specified by HourOffset and MinuteOffset, and run backwards until it reaches 0.00.

See also: Active, ClockMode, MinuteOffset

## MinuteOffset — property

```
property MinuteOffset : Integer
```

The minute offset from the current time zone.

Used in conjunction with the HourOffset. Some time zones are offset from their neighbors by 30 minutes instead of an hour. MinuteOffset allows you to easily set the clock to display those time zones, without having to change the time.

MinuteOffset is also used for the CountDown timer. When the clock's mode is set to cmCountdownTimer, the clock's display is set to 0:00 (12:00 in analog mode). Set the hour and minute offset to represent the timer's set time and set Active to True. The clock will display the amount of time specified by HourOffset and MinuteOffset, and run backwards until it reaches 0.00.

See also: Active, ClockMode, HourOffset

## OnCountdownDone — event

```
property OnCountdownDone : TNotifyEvent
```

Defines an event that is fired when the timer reaches zero in cmCountdownTimer mode.

Provide an event handler for the OnCountdownDone event if you want to be notified each time the countdown timer reaches zero.

See also: ClockMode

### OnHourChange — event

```
property OnHourChange : TNotifyEvent
```

Defines an event that is fired when the hour changes.

Provide an event handler for the OnHourChange event if you want to be notified at the top of each hour.

See also: OnMinuteChange, OnSecondChange

### OnMinuteChange — event

5

```
property OnMinuteChange : TNotifyEvent
```

Defines an event that is generated when the minute changes.

Provide an event handler for the OnMinuteChange event if you want to be notified when the minute changes.

See also: OnHourChange, OnSecondChange

### OnSecondChange — event

```
property OnSecondChange : TNotifyEvent
```

Defines an event that is generated each second.

Provide an event handler for the OnSecondChange event if you want to be notified each time the clock's second hand changes.

See also: OnHourChange, OnMinuteChange

### Time — property

```
property Time : TDateTime
```

The clock's current time.

Read Time to determine the clock's current time. Set Time to a TDateTime value to display a static time on the clock when the Active property is False.

See also: Active

# TVpHandOptions Class

TVpHandOptions determines how a TVpClock's hands are drawn. Clock hand options are surfaced in the TVpClock component's AnalogOptions property on page 177. The options allow you to control the appearance of the second hand, the minute hand, and the hour hand. You can set the hand colors, their length and width, and whether they are filled or outlined. By default the hour and minute hands are black and the second hand is red. The hand length is automatically adjusted when the size of the clock changes.

## Hierarchy

TPersistant (VCL)

TVpHandOptions (VPClock)

## Properties

ClockFace
DrawMarks
HourHandColor
HourHandLength
HourHandWidth
MinuteHandColor
MinuteHandLength
MinuteHandWidth
SecondHandColor
SecondHandLength
SecondHandWidth
ShowSecondHand
SolidHands

### ClockFace — property

```
property ClockFace : TBitmap
```

A bitmap that is displayed for the clock face.

Assign a bitmap to ClockFace to change the appearance of the clock. You can load a bitmap at design time using the ClockFace property editor, or at run time by assigning a TBitmap to ClockFace. The clock hands are drawn on top of the clock face. For aesthetic reasons you may want to set the DrawMarks property to False when supplying a clock face. Several clock faces are provided for you. You will find the clock face bitmaps in the Examples directory.

See also: DrawMarks, TVpClock.AnalogOptions



### DrawMarks — property

```
property DrawMarks : Boolean
```

Default: True

Determines whether minute and hour marks are drawn around the clock face.

When DrawMarks is True, markers are drawn around the clock border. The markers consists of dots at the minute points and blue squares at the hour points. Generally, you will set DrawMarks to False if you are using a bitmap as the clock face.

See also: ClockFace

### HourHandColor — property

```
property HourHandColor : TColor
```

Default: clBlack

Determines the color of the hour hand.

## HourHandLength — property

```
property HourHandLength : TVpPercent
```

```
TVpPercent = 0..100;
```

Default: 60

Determines the length of the hour hand as a percentage of the clock's diameter.

Once the hand length is set, it will retain it's length relative to the clock face if the clock is resized.

## HourHandWidth — property

```
property HourHandWidth : Integer
```

Default: 4

Determines the width of the clock's hour hand.

## MinuteHandColor — property

```
property MinuteHandColor : TColor
```

Default: clBlack

Determines the color of the clock's minute hand.

## MinuteHandLength — property

```
property MinuteHandLength : TVpPercent
```

```
TVpPercent = 0..100;
```

Default: 80

Determines the length of the minute hand as a percentage of the clock's diameter.

Once the hand length is set, it will retain it's length relative to the clock face if the clock is resized.

## MinuteHandWidth — property

```
property MinuteHandWidth : Integer
```

Default: 3

Determines the width of the clock's minute hand.

## SecondHandColor — property

```
property SecondHandColor : TColor
```

Default: clRed

Determines the color of the clock's second hand.

## SecondHandLength — property

```
property SecondHandLength : TVpPercent
TVpPercent = 0..100;
```

Default: 90

Determines the length of the second hand as a percentage of the clock's diameter.

Once the hand length is set, it will retain it's length relative to the clock face if the clock is resized. To eliminate the second hand, set ShowSecondHand to False.

See also: ShowSecondHand

## SecondHandWidth — property

```
property SecondHandWidth : Integer
```

Default: 1

Determines the width of the clock's second hand.

## ShowSecondHand — property

```
property ShowSecondHand : Boolean
```

Default: True

Determines whether the clock's second hand is displayed.

See also: SecondHandLength

## SolidHands property

```
property SolidHands : Boolean
```

Default: True

Determines the drawing style used to paint the clock hands.

If SolidHands is True, hands are filled with the current hand color. If SolidHands is False, clock hands are outlined with the hand color and the interior of the hand is drawn in a three-dimensional style.

# TVpClockDigitalOptions Class

TVpClockDigitalOptions determines how the clock is drawn in dmDigital mode. DigitalOptions are surfaced in the TVpClock component's DigitalOptions property. The options allow you to control the LED segment colors, display back color, whether to show seconds, and whether the separator flashes with each second.

## Hierarchy

TPersistant (VCL)

TVpClockDigitalOptions (VPClock)

## Properties

BackColor
FlashSeparator
LEDOffColor
LEDOnColor
ShowSeconds

## Reference Section

### BackColor — property

```
property BackColor : TColor
```

Default: clBlack

Determines the color of the background area behind the LED segments.

### FlashSeparator — property

```
property FlashSeparator : Boolean
```

Default: True

Determines whether the separator will flash every second.

### LEDOffColor — property

```
property LEDOffColor : TColor
```

Default: $00274026 (dark green)

Determines the color of the un-lit (darkened) LED segments.

### LEDOnColor — property

```
property LEDOnColor : TColor
```

Default: clLime

Determines the color of the lit LED segments.

### ShowSeconds — property

```
property ShowSeconds : Boolean
```

Default: True

Determines whether the seconds are visible on the display.

# TVpCalendar Component

Visual PlanIt provides a flexible monthly calendar (as shown in Figure 5.1) that is specially designed to synchronize with other Visual PlanIt components via Visual PlanIt's ControlLink component. A properly configured TVpCalendar component will reflect date changes from and pass new date selections to all other Visual PlanIt components that share the same ControlLink and DataStore.

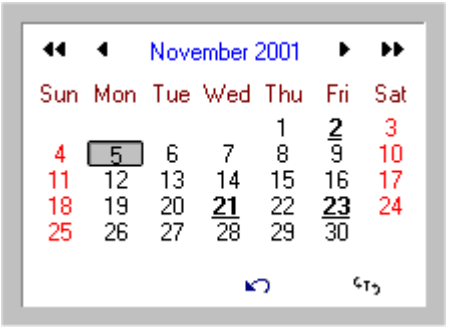



*Figure 5.1: Calendar example.*

The component provides a grid-like display of a month, with each row of the grid representing one week, and each column representing a weekday. You can specify which day is at the left of the calendar grid via the WeekStarts property. You can also specify the colors for the active day, weekend days, days with events, the current month, day names (at the top of the grid), the days of the previous and next months, and the optional header, via the Colors property.

Caution: Days with events show up in the specified color, in bold typeface and underlined. If you have the DataStore's DayBuffer set to a value that is less than 31 days, the calendar may not properly display all the days that actually have events. This is due to the fact that not all of the days covered by the month have been loaded into memory, so the DataStore can't find any events for them. If you are going to use the TVpCalendar component, it is recommended that the DataStore's DayBuffer property is set to at least 31 (the default).

The buttons at the top of the calendar allow navigating to adjacent months (single-arrow) and adjacent years (double-arrows). A right-click over the name of the month displays a pop-up window that allows selection of a month from a list that spans a seven-month period (three months before and three months after the current month).

The lower-left button is used to revert to the last displayed value and the lower-right button is used to jump to the current date (today).

## Hierarchy

TCustomControl (VCL)

❶ TVpCustomControl (VPBase) . . . . . 30

❷ TVpLinkableControl (VPBaseDS) . . . . . 32

TVpCustomCalendar (VPCalendar)

TVpCalendar (VPCalendar)

## Properties

Colors
❷ ControlLink
❷ DataStore
Date
DateFormat
Day
DayNameWidth
DrawHeader
Month
Options
❷ ReadOnly
❶ Version
WeekStarts
Year

## Methods

❷ GetControlType
❷ GetLastPrintLine
IncDay
IncMonth
IncYear
LinkHandler
PaintToCanvas
RenderToCanvas
SetToday

## Events

❶ AfterEnter
❶ AfterExit
OnChange
OnDrawDate
OnDrawItem
OnGetDateEnabled
OnGetHighlight
❶ OnMouseWheel

## Reference Section

### Colors — property

```
property Colors : TVpCalColors
```

Specifies the colors used for displaying different sections of the calendar.

### Date — property

```
property Date : TDateTime
```

Provides read/write access to the calendar's date.

Programmatically set the calendar's date by assigning a new value to this property. Read the calendar's date by reading this value.

### DateFormat — property

```
property DateFormat : TVpDateFormat
TVpDateFormat = (dfShort, dfLong);
```

Determines how the date will be displayed at the top of the calendar.

### Day — read-only, run-time property

```
property Day : Integer
```

Returns the day portion of the calendar's date.

### DayNameWidth — property

```
property DayNameWidth : TVpDayNameWidth
TVpDayNameWidth = 1..3;
```

Determines how many characters will be used to display the day names.

### DrawHeader — property

```
property DrawHeader : Boolean
```

Determines whether the month and day names are drawn at the top of the calendar.

### IncDay — method

```
procedure IncDay(Delta : Integer);
```

Increments the calendar's date by the number of days specified by Delta.

### IncMonth — method

```
procedure IncMonth(Delta : Integer);
```

Increments the calendar's date by the number of months specified by Delta.

### IncYear — method

```
procedure IncYear(Delta : Integer);
```

Increments the calendar's date by the number of years specified by Delta.

### LinkHandler — method

```
procedure LinkHandler(Sender : TVpLinkableControl);
 NotificationType : TVpNotificationType;
 const Value : Variant); override;
```

Called by the ControlLink component in response to a change notification from another component.

Each TVpLinkableControl descendant must define a LinkHandler procedure. It is used solely by the attached ControlLink component. Do not call the LinkHandler procedure programmatically.

### Month — read-only, run-time property

```
property Month : Integer
```

Returns the month portion of the calendar's date.

### OnChange — event

```
property OnChange : TDateChangeEvent

TDateChangeEvent = procedure(
  Sender : TObject; Date : TDateTime) of object;
```

Defines an event handler that is fired when the calendar's date changes.

## OnDrawDate — event

```
property OnDrawDate : TCalendarDateEvent
```

```
TCalendarDateEvent = procedure(Sender : TObject;
  ADate : TDateTime; const Rect : TRect) of object;
```

Defines an event handler that is fired when the calendar's date is drawn.

Create an OnDrawDate event handler to control how the date at the top of the calendar is drawn.

See also: OnDrawItem

5

## OnDrawItem — event

```
property OnDrawItem : TCalendarDateEvent
```

```
TCalendarDateEvent = procedure(Sender : TObject;
  ADate : TDateTime; const Rect : TRect) of object;
```

Defines an event handler that is fired when each day of the calendar is drawn.

Create an OnDrawDate event handler to control how each individual day is drawn on the calendar.

See also: OnDrawDate

## OnGetDateEnabled — event

```
property OnGetDateEnabled : TGetDateEnabledEvent
```

```
TGetDateEnabledEvent = procedure(Sender : TObject;
  ADate : TDateTime; var Enabled : Boolean) of object;
```

Defines an event handler that allows you to determine if ADate can be selected.

If Enabled is set to False, the user will not be allowed to select that particular date.

## OnGetHighlight — event

```
property OnGetHighlight : TGetHighlightEvent
```

Defines an event handler that allows you to determine the color used to draw the text displayed in the cell for the specified date.

## Options — property

```
property Options : TVpCalDisplayOptions
```

```
TVpCalDisplayOption = (cdoShortNames, cdoShowYear, cdoShowInactive,
  cdoShowRevert, cdoShowToday, cdoShowNavBtns, cdoHideActive);
```

```
TVpCalDisplayOptions = set of TVpCalDisplayOption;
```

Default: [cdoShortNames, cdoShowYear, cdoShowInactive, cdoShowRevert, cdoShowToday, cdoShowNavBtns]

Determines several options that determine the look of the displayed calendar.

The following table describes the possible options:

| Option | Description |
|---|---|
| cdoShortNames | Causes dates returned by calling DateString to use short month names. |
| cdoShowYear | Causes the year to be displayed at the top of the calendar. |
| cdoShowInactive | Allows display of days that are not part of the current month. |
| cdoShowRevert | Displays the "Revert" button that allows changing back to the date that was previously displayed. |
| cdoShowToday | Displays the "Today" button that allows selecting today's date directly. |
| cdoShowNavBtns | Displays navigation buttons at the top of the calendar that allows moving between months and years. |

## PaintToCanvas — method

```
procedure PaintToCanvas(ACanvas : TCanvas; ARect : TRect;
  Angle : TVpRotationAngle; ADate : TDateTime);
```

```
TVpRotationAngle = (ra0, ra90, ra180, ra270);
```

Paints the calendar to an arbitrary canvas.

PaintToCanvas will paint the calendar onto ACanvas in the rectangle specified by ARect. The calendar will be rotated inside of ARect by the angle specified in Angle. ACanvas can be either a screen or printer canvas.

```
procedure RenderToCanvas(RenderCanvas : TCanvas;
  RenderIn : TRect; Angle : TVpRotationAngle; Scale : Extended;
  RenderDate : TDateTime; StartLine : Integer; StopLine : Integer;
  UseGran : TVpGranularity; DisplayOnly : Boolean);
  virtual; abstract;
```

```
TVpRotationAngle = (ra0, ra90, ra180, ra270);
```

```
TVpGranularity = (
  gr60Min, gr30Min, gr20Min, gr15Min, gr10Min, gr6Min, gr5Min);
```



Paints the control to an arbitrary canvas.

RenderToCanvas will paint the control onto ACanvas in the rectangle specified by ARect. The control will be rotated inside of ARect by the angle specified in Angle. ACanvas can be either a screen or printer canvas.

Some additional parameters are used to finely specify how the control is painted.

Scale is used to manually increase or decrease the size of the control. Most controls calculate the scale in terms of ARect.

RenderDate is used by TVpCustomCalendar, TVpDayView, TVpWeekView, and TVpMonthView to specify the date to display.

StartLine is used by TVpDayView to specify the first hour to display. If this value is –1, the start of the day (midnight) is used. This value is also used by TVpContactGrid to specify the first contact and TVpTaskList to specify the first task to display. A value of 0 will start with the first contact or task.

StopLine is used by TVpDayView to specify the last hour to display. If this value is –1, the end of the day will be used (11:59pm).

UseGran is used by the TVpDayView to specify the granularity to use.

If DisplayOnly is on, the display will be adjusted slightly for printing.

### SetToday — method

```
procedure SetToday;
```

Sets the calendar's date to Now.

### WeekStarts — property

```
property WeekStarts : TVpDayType
```

```
TVpDayType = (dtSunday, dtMonday, dtTuesday,
  dtWednesday, dtThursday, dtFriday, dtSaturday);
```

Sets the day that will be used by the calendar to represent the first day of the week.

### Year — read-only, run-time property

```
property Year : Integer
```

Returns the year component of the calendar's current date.

# NavBar Component

The TVpNavBar component emulates the control you see on the left side of Microsoft Outlook™. The contents of the component are arranged by folders and items. Each TVpNavBar contains one or more folders. Within each folder are one or more folder items. By default, folders are depicted as buttons on the TVpNavBar. Clicking on a folder's button or tab makes that folder the active folder and displays its contents. Clicking on a folder item generates an event that can be used to perform some action in the program. A typical use for a TVpNavBar is to place the bar in the left side of a form with different item views displayed to the right of the bar.



The NavBar also has the ability for folders to contain other components. Set any folder's FolderType property to ftContainer and it becomes the equivalent of a ScrollBox that accepts other components dropped on it at design time.

Only one folder's contents can be visible at a given time. If an ftDefault type folder contains more items than can be displayed in its client area, spinner buttons will be displayed. If an ftContainer type folder contains more items than can be displayed, it will create scroll bars that can be used to scroll the client area. Clicking the spinner buttons or manipulating the scroll bars allows you scroll the active folder's client area to gain access to the additional items.

In an ftDefault type folder, folder items can be displayed in one of two ways. When large icons are used (images greater than 16 by 16 pixels), the image is centered in the client area of the TVpNavBar with the item's text below the image. When small icons are used, the icon image appears near the left edge of the TVpNavBar with text to the right of the image. Icon is a generic term; the image list can contain either icons or bitmaps. Use a VCL TImageList component to store the item images.

The TVpNavBar supports in-place, run-time editing for folders and ftDefault type folder items. Calling the RenameFolder method invokes in-place editing of folder captions. In-place editing of folder item captions is invoked by calling RenameItem. Users can then directly type a new caption for the folder or item. Pressing Enter on the keyboard accepts the new caption and the updates TVpNavBar. Pressing Esc abandons edits and the previous caption is restored.

Note: RenameItem will not work in the folder's items if the folder's FolderType is ftContainer.

There are four primary folder and item events supplied by the TVpNavBar:

- The OnFolderClick event notifies you when a folder changes.
- The OnItemClick event notifies you when a folder item is clicked. You can use this event to change the view associated with the item clicked.
- The OnFolderChange event notifies you when a folder is about to change.
- The OnFolderChanged event notifies you after a folder has changed.

Folders can be added at design time using the Folder Editor. To invoke the Folder Editor double-click the TVpNavBar on your form. Alternatively, you can double-click the value column next to the FolderCollection property in the Object Inspector. The Folder Editor contains buttons to add, remove, or reorder folders. Click on a folder in the Folder Editor to display that folder's properties in the Object Inspector. Folder items are added at design time via the Folder Item Editor. This property editor is displayed when you double-click next to a folder's ItemCollection property in the Object Inspector. As with the Folder Editor, clicking a folder item in the Folder Items Editor displays that item's properties in the Object Inspector.

# TVpCustomNavBar Class

The TVpCustomNavBar class is the immediate ancestor of the TVpNavBar component. It implements all the methods and properties used by the TVpNavBar component and is identical to the TVpNavBar component except that no properties are published.

TVpCustomNavBar is provided to facilitate creation of descendent components. For property and method descriptions, see the “TVpNavBar Component” on page 201.

## Hierarchy

5

TCustomControl (VCL)

TVpCustomControl (VPBase). . . . . 30

TVpCustomNavBar (VPNavBar)

# TVpNavBar Component

## Hierarchy

TCustomControl (VCL)

❶ TVpCustomControl (VPBase) . . . . . 30

TVpNavBar (VPNavBar)

## Properties

ActiveFolder
ActiveItem
AllowRearrange
BackgroundColor
BackgroundImage
BackgroundMethod
ButtonHeight
DrawingStyle
FolderCollection
FolderCount
Folders
Images
ItemFont
PreviousFolder
PreviousItem
PlaySounds
ScrollDelta
SoundAlias
❶ Version

## Methods

BeginUpdate
EndUpdate
GetFolderAt
GetItemAt
InsertFolder
InsertItem
InvalidateItem
RemoveFolder
RemoveItem
RenameFolder
RenameItem

## Events

❶ AfterEnter
❶ AfterExit
OnDragDrop
OnDragOver
OnFolderChange
OnFolderChanged
OnFolderClick
OnItemClick
OnMouseOverItem
❶ OnMouseWheel

### ActiveFolder — property

```
property ActiveFolder : Integer
```

The index number of the active folder.

Read ActiveFolder to determine the index number of the active folder. Set ActiveFolder to programmatically set the active folder. Folders are 0-based with the first folder at index 0, the second at index 1, etc.

See also: FolderCount, InsertFolder, RenameFolder



### ActiveItem — read-only, run-time property

```
property ActiveItem : Integer
```

The index number of the active folder's active item.

Active Item is 0-based with the first item in each folder at index 0.

Note: Components contained in folders of type ftContainer are not numbered and are not accessible via the ActiveItem property.

See also: ActiveFolder, TVpNavFolder.FolderType, TVpNavFolder.Items

### AllowRearrange — property

```
property AllowRearrange : Boolean
```

Default: True

Determines whether or not the items within a folder can be rearranged by dragging.

When AllowRearrange is True, items can be moved via drag and drop. When AllowRearrange is False, items cannot be rearranged. Folder items can be dragged within a folder or between folders. When dragging between folders, the target folder will be expanded when the mouse cursor is dragged over a folder button.

AllowRearrange only applies to default folders and their items. For instance, you cannot drag a default item to a container folder.

See also: Items, TVpNavFolder.FolderType

## BackgroundColor — property

```
property BackgroundColor : TColor
```

Default: clInactiveCaption

The background color of the Nav Bar.

BackgroundColor is applied to the background when the BackgroundMethod property is set to bmNone.

See also: BackgroundImage, BackgroundMethod

## BackgroundImage — property

```
property BackgroundImage : TBitmap
```

The image that is displayed in the Nav Bar's background.

By default, BackgroundImage is nil and the color of the background area is determined by the BackgroundColor property. Set BackgroundImage to display a bitmap on the background of the component instead. The background fill method (none, normal, or stretch) is determined by the BackgroundMethod property.

See also: BackgroundColor, BackgroundMethod

## BackgroundMethod — property

```
property BackgroundMethod : TVpBackgroundMethod
```

Default: bmNormal

The painting method to paint the background.

The possible values for BackgroundMethod are shown in the following table:

| Value | Behavior |
|---|---|
| bmNone | No background bitmap is applied. The background is filled using the color specified in BackgroundColor. |
| bmNormal | The bitmap specified in BackgroundImage is used to paint the background. The image is not sized to fit the component. |
| bmStretch | The background is filled with the bitmap specified in the BackgroundImage property. The image is stretched to fit the size of the component. |

Setting BackgroundMethod to bmNormal or bmStretch has no effect if BackgroundImage is not assigned

See also: BackgroundColor, BackgroundImage

## BeginUpdate — method

```
procedure BeginUpdate;
```

Temporarily disables repainting of the control.

Call BeginUpdate to disable repainting of the component's items if you are programmatically adding many items to a folder. Call EndUpdate after adding items to force the component to be repainted.

Warning: Calls to BeginUpdate and EndUpdate are cumulative so each call to BeginUpdate must have a corresponding EndUpdate call.



See also: EndUpdate

## ButtonHeight — property

```
property ButtonHeight : Integer
```

Default: 20

The height of the Nav Bar's folder button or tab (depending on the value of DrawingStyle).

See also: DrawingStyle

## DrawingStyle — property

```
property DrawingStyle : TVpFolderDrawingStyle
```

```
TVpFolderDrawingStyle = (
  dsCooltab, dsDefButton, dsEtchedButton, dsStandardTab);
```

The style used to draw the Nav Bar's folder indicators.

The drawing style definitions are shown in the following table:

| DrawingStyle | Result |
|---|---|
| dsDefButton | The border and folder buttons are drawn using the Microsoft Outlook97™ style. |
| dsEtchedButton | The component is drawn using the Outlook98™, etched style. |
| dsCoolTab | The component is drawn using the Microsoft Outlook97™ style except the folder buttons are drawn as rounded, 3D tabs. |
| dsStandardTab | The component is drawn using the Microsoft Outlook97™ style except the folder buttons are drawn as standard, squared off tabs. |



## EndUpdate — method

```
procedure EndUpdate;
```

Repaints the component after adding items with painting disabled.

Call EndUpdate to repaint the component after adding items when repainting has been disabled (by calling BeginUpdate).

Warning: Calls to BeginUpdate and EndUpdate are cumulative so each call to BeginUpdate must have a corresponding EndUpdate call.

See also: BeginUpdate

## FolderCollection — property

```
property FolderCollection : TVpCollection
```

The list of folders for the component.

A TVpNavBar contains one or more folders. FolderCollection contains the list of folders. Each folder's characteristics are set at design time via the FolderCollection property editor or at run time through code. To enumerate a TVpNavBar's folders, use the Folders property. To add folders at run time, call the InsertFolder method. To remove folders at run time, call the RemoveFolder method.

See also: FolderCount, Folders, InsertFolder, RemoveFolder

### FolderCount — read-only, run-time property

```
property FolderCount : Integer
```

The number of folders in the TVpNavBar.

See also: FolderCollection, Folders

### Folders — property

```
property Folders[Index : Integer] : TVpNavFolder
```

An indexed property containing the folders in the TVpNavBar.

Use Folders at run time to access a particular folder by its folder index or to enumerate the folders.

See also: FolderCollection, FolderCount

### GetFolderAt — method

```
function GetFolderAt(X, Y : Integer) : Integer;
```

Returns the folder index of the folder at the given coordinates.

See also: Folders, GetItemAt

### GetItemAt — method

```
function GetItemAt(X, Y : Integer) : Integer;
```

Returns the item index of the default folder item at the given coordinates.

If the active folder is of ftContainer type, GetItemAt returns -1.

See also: GetFolderAt, TVpNavFolder.Items

## Images — property

```
property Images : TImageList
```

The image list that contains the images used to display folder items.

Set Images to the name of the image list that contains the folder item images. Set each folder item's IconIndex property to the index corresponding to the image to display for that item. The folder items' display characteristics are determined by the folder's IconSize property. For large icons 32 by 32 pixels is the recommended size. For small icons, 16 by 16 pixel images are recommended.

Use a VCL TImageList component to store the item images.

See also: TVpNavItem.IconIndex, TVpNavFolder.IconSize



## InsertFolder — method

```
procedure InsertFolder(
  const ACaption : string; AFolderIndex : Integer);
```

InsertFolder adds a folder to the TVpNavBar.

ACaption is the caption for the new Folder. AFolderIndex is the position in the list where the new item will be inserted.

See also: InsertItem, FolderCollection, Folders, RemoveFolder

## InsertItem — method

```
procedure InsertItem(const ACaption : string;
  AFolderIndex, AItemIndex, AIconIndex : Integer);
```

Adds a folder item into a folder of type ftDefault.

Call InsertItem to programmatically add an item to a folder at run time. ACaption is the caption for the new item. AFolderIndex is the index number of the parent folder for the new item. AItemIndex is the position within the folder where the new item will be added. AIconIndex is the index of the image within the image list that will be used to display the new item.

Note: Calls to InsertItem are ignored if AFolderIndex refers to a folder of type ftContainer.

See also: InsertFolder, RemoveItem

### InvalidateItem — method

```
procedure InvalidateItem(FolderIndex, ItemIndex : Integer);
```

Forces a redraw of the specified item.

### ItemFont — property

```
property ItemFont : TFont
```

The font that is used to draw the text associated with a folder item. If ItemFont is unassigned, the TVpNavBar's font will be used.



### OnDragDrop — event

```
property OnDragDrop : TVpLOBDragDropEvent
```

```
TVpLOBDragDropEvent = procedure(Sender, Source : TObject;
  X, Y : Integer; FolderIndex, ItemIndex : Integer) of object;
```

Defines an event handler that is called when a drop occurs from another component.

Source is the component that initiated the drag event. X is the cursor x position and Y is the cursor y position. State is the drag state (see the VCL help for an explanation of TDragState). FolderIndex is the index of the folder on which the drop occurred. ItemIndex is the item index position of the drop within the folder. The value of ItemIndex varies depending on whether you are allowing a drop on folders, on items, or both. If you are allowing drop on folders, ItemIndex is the index where a new item will be inserted (when you call InsertItem, for example). If you are allowing drop on items within a folder, ItemIndex is the index number of the item that was under the cursor when the drop occurred.

See also: OnDragOver

## OnDragOver — event

```
property OnDragOver : TVpLOBDragOverEvent
```

```
TVpLOBDragOverEvent = procedure(
  Sender, Source : TObject; X, Y : Integer; State : TDragState;
  var AcceptFolder, AcceptItem : Boolean) of object;
```

Defines an event handler that is called when an item is being dragged from another component.

Source is the component that initiated the drag event. X is the cursor x position and Y is the cursor y position. State is the drag state (see the VCL help for an explanation of TDragState). AcceptFolder controls whether or not a folder can accept a drop. When AcceptFolder is True a line is drawn on the folder at the cursor position. An item can be dropped on the folder at the indicated position. AcceptItem controls whether or not a folder item can accept a drop. When AcceptItem is True, the folder item under the mouse cursor is highlighted to indicate that a drop on the folder item is valid. If neither AcceptFolder or AcceptItem are True, the no drop cursor is displayed and the OnDragDrop event is not generated. AcceptFolder does not have to be True to accept drops on folder items.

See also: OnDragDrop

## OnFolderChange — event

```
property OnFolderChange : TFolderChangeEvent
```

```
TFolderChangeEvent = procedure(Sender : TObject; Index : Integer;
  var AllowChange : Boolean; Dragging : Boolean) of object;
```

Defines an event handler that is called when the active folder is about to change.

Index is the index number of the folder that is about to become active. AllowChange determines whether or not the folder can change. By default, AllowChange is True. Setting AllowChange to False within your event handler prevents the folder change from taking place. Dragging is a flag that indicates whether a folder item is being dragged. You may want to allow a particular folder to change if the user is attempting to drag an item from one folder to another, but disallow the folder change during normal operation.

See also: OnFolderChanged

## OnFolderChanged — event

```
property OnFolderChanged : TFolderChangedEvent
```

```
TFolderChangedEvent = procedure(
  Sender : TObject; Index : Integer) of object;
```

Defines an event handler that is called when the active folder changes.

Index is the index number of the new active folder. OnFolderChange is a notification event. As such, it is generated after the active folder changes. To detect a folder state change before the folder changes, use the OnFolderChange event.

See also: OnFolderChange

5

## OnFolderClick — event

```
property OnFolderClick : TFolderClickEvent
```

```
TFolderClickEvent = procedure(Sender : TObject;
  Button : TMouseButton; Shift : TShiftState;
  Index : Integer) of object;
```

Defines an event handler that is called when a folder button is clicked.

Button is the mouse button that was use to click the folder. Shift is the keyboard state when the folder was clicked. Index is the folder index of the folder that was clicked. Provide an event handler for OnFolderClick to determine when a folder button is clicked.

See also: OnItemClick

## OnItemClick — event

```
property OnItemClick : TItemClickEvent
```

```
TItemClickEvent = procedure(
  Sender : TObject; Button : TMouseButton;
  Shift : TShiftState; Index : Integer) of object;
```

Defines an event handler that is called when a folder item is clicked.

Button is the mouse button that was use to click the folder. Shift is the keyboard state when the item was clicked. Index is the item index of the item that was clicked. Provide an event handler for OnFolderClick to determine when a folder button is clicked. Read the ActiveFolder property to determine the folder that the item clicked belongs to.

See also: OnFolderClick

### OnMouseOverItem — event

```
property OnMouseOverItem : TVpMouseOverItemEvent
```

```
TVpMouseOverItemEvent = procedure(
  Sender : TObject; Item : TVpNavItem) of object;
```

Defines an event handler that is called when the mouse cursor moves over a TVpNavBar item.

Sender is the object that generated the event. Item is a pointer to the item that the cursor is currently over.

### PlaySounds — property

```
property PlaySounds : Boolean
```

Default: False

Determines whether or not a sound is played when the active folder changes.

Set PlaySounds to True to enable playing of a sound when the active folder changes. The sound played is determined by the SoundAlias property.

See also: SoundAlias

### PreviousFolder — read-only, run-time property

```
property PreviousFolder : Integer
```

The index of the folder that was previously selected.

Read PreviousFolder to determine the index of the folder that was previously selected. If no previous folder was selected, PreviousFolder contains –1.

See also: ActiveFolder, FolderCollection, Folders, OnFolderClick, PreviousItem

### PreviousItem — read-only, run-time property

```
property PreviousItem : Integer
```

The index of the folder item that was previously clicked.

Read PreviousItem to determine the index number of the item that was previously clicked. If no item was previously clicked, PreviousItem contains –1. To determine the folder containing the previously clicked item, read the PreviousFolder or ActiveFolder properties.

See also: OnItemClicked, PreviousFolder, TVpNavFolder.Items

### RemoveFolder — method

```
procedure RemoveFolder(AFolderIndex : Integer);
```

Removes the folder specified by AFolderIndex. All of the folder's contained items or components are destroyed.

### RemoveItem — method

```
procedure RemoveItem(AFolderIndex, AItemIndex : Integer);
```

Removes the specified item from the specified folder.



If AFolderIndex refers to a folder of type ftContainer, all calls to RemoveItem are ignored.

### RenameFolder — method

```
procedure RenameFolder(AFolderIndex : Integer);
```

Enables in-place editing of a folder's caption.

Call RenameFolder to invoke the TVpNavBar's in-place editor. Users can then type a new caption for the folder. Pressing Esc abandons the operation and returns the folder caption to its original state. Pressing Enter changes the folder's caption to the new text.

See also: InsertFolder, RemoveFolder, RenameItem

### RenameItem — method

```
procedure RenameItem(AFolderIndex, AItemIndex : Integer);
```

Enables in-place editing of an item's caption.

Call RenameItem to invoke the TVpNavBar's in-place editor. Users can then type a new name for the item. Pressing Esc abandons the operation and returns the item caption to its original state. Pressing Enter changes the item's caption to the new text.

Note: If AFolderIndex refers to a folder of type ftContainer, all calls to RenameItem are ignored.

See also: InsertItem, RemoveItem, RenameFolder

## ScrollDelta — property

```
property ScrollDelta : Integer
```

Default: 2

Determines the scrolling granularity when a folder changes.

When a folder button is clicked, the new folder scrolls into view. A lower ScrollDelta results in finer (and slower) scrolling. Set ScrollDelta to a higher number to increase the scroll speed.

## SoundAlias — property

```
property SoundAlias : string
```

The sound that will be played when the active folder changes.

SoundAlias can be either a filename or the name of a system sound. System sounds include Minimize, Maximize, MailBeep, and so on. System sound names are listed in the registry. System sound names can be found under the HKEY_CURRENT_USER\AppEvents\Schemes registry key. Using system sounds is advantageous in that it allows you to incorporate the user's current sound scheme. You may, for example, want to associate a folder change with Windows' minimize sound. If the specified sound cannot be found, the default Windows sound is played (usually the ding sound). SoundAlias is ignored if the PlaySounds property is set to False.

See also: PlaySounds

# TVpNavFolder Class

## Hierarchy

TCollectionItem (VCL)

❶ TVpCollectionItem (VPBase) . . . . . 41

TVpNavFolder (VPNavBar)

5

## Properties

| | | | | | |
|---|---|---|---|---|---|
| | Caption | | IconSize | | Items |
| | Container | | ItemCollection | ❶ | Name |
| ❶ | DisplayText | | ItemCount | ❶ | Version |

## Reference Section

### Caption — property

```
property Caption : string
```

The text displayed on a folder's button.

A folder's caption can be changed at run time by assigning a new value to Caption. Additionally, the RenameFolder method of the TVpNavBar component can be called to invoke the TVpNavBar's in-place editor.

See also: TVpNavBar.RenameFolder

### Container — property

```
property Container : TVpFolderContainer
```

Provides access to the folder's component container.

Container is only available if the folder's type is ftContainer. Otherwise, it is unassigned.

### IconSize — property

```
property IconSize : TVpIconSize
```

```
TVpIconSize = (isLarge, isSmall);
```

Default: isLarge

Determines the display characteristics of folder items.

IconSize can be either isLarge or isSmall. When IconSize is set to isLarge, images are displayed centered on the TVpNavBar with the text below the image. When IconSize is set to isSmall the image is drawn on the left side of the TVpNavBar with the text to the right of the image. For large icons an image size of 32 by 32 pixels is recommended. For small icons a 16 by 16 pixel image is recommended. Using images larger than 16 by 16 with the isSmall setting may result in the item's text overlapping the image.

See also: TVpNavBar.Images

## ItemCollection — property

```
property ItemCollection : TVpCollection
```

The list of items in a folder.

A TVpNavBar contains one or more folders with one or more items in each folder. ItemCollection contains the list of items for a folder. Each item's characteristics are set at design time via the ItemCollection property editor or at run time through code. To enumerate a folder's items, use the Items property. To add items at run time, call the InsertItem method of the TVpNavBar component. To remove items at run time, call the RemoveItem method of the TVpNavBar component.



See also: ItemCount, Items, TVpNavBar.InsertItem, TVpNavBar.RemoveItem

## ItemCount — read-only, run-time property

```
property ItemCount : Integer
```

The number of items in the folder.

ItemCount returns 0 if the folder is of type ftContainer.

See also: ItemCollection, Items

## Items — read-only, run-time property

```
property Items[Index : Integer] : TVpNavItem
```

An indexed property that allows access to the items in a folder.

Use Items to enumerate the items in a folder, or to access a particular item by index number. The following example changes the caption of the second item in the first folder of an TVpNavBar:

```
procedure TForm1.Button3Click(Sender : TObject);
begin
  VpNavBar1.Folders[0].Items[1].Caption := 'Outbox';
end;
```

See also: ItemCollection, ItemCount

# TVpNavBtnItem Class

The TVpButtonItem class models the behavior of the default NavBar items. These are the items contained in folders of type ftDefault. These items cannot be used in folders of type ftContainer.

## Hierarchy

TCollectionItem (VCL)

❶ TVpCollectionItem (VPBase) . . . . . 41

TVpNavBtnItem (VPNavBar)

## Properties

| | | | | |
|---|---|---|---|---|
| | Caption | IconIndex | ❶ | Name |
| | Collection | IconRect | ❶ | Version |
| | Description | Index | | |
| ❶ | DisplayText | LabelRect | | |

### Caption — property

```
property Caption : string
```

The display text for a folder item.

An item's caption can be changed at run time by assigning a new value to Caption. Additionally, you can call the RenameItem method of the TVpNavBar component to invoke the TVpNavBar's in-place editor.

See also: TVpNavBar.RenameItem



### Description — property

```
property Description : string
```

Used to specify a text description of a TVpNavBar item.

Description can be used to store any additional text associated with a TVpNavBar item. For example, you can use Description to display hint text for a particular item in your application's status bar. For an example, see the OnMouseOverItem event of the TVpNavBar component on page 211.

See also: TVpNavBar.OnMouseOverItem

### IconIndex — property

```
property IconIndex : Integer
```

The index in the image list used to draw a folder item.

Set each folder item's IconIndex property to the index corresponding to the image to display for that item. Images are contained in the TVpNavBar's Images property.

See also: TVpNavBar.Images

### IconRect — read-only, run-time property

```
property IconRect : TRect
```

Contains the size and location of a folder item's display icon.

See also: LabelRect

### LabelRect — read-only, run-time property

```
property LabelRect : TRect
```

Contains the size and location of a folder's label.

See also: IconRect

# TVpPrintPreview Component

The TVpPrintPreview component provides an easy way to view how the print formats defined in a TVpControlLink component will look when they are printed.

This component can be used in conjunction with the TVpPrintFormatComboBox to select between different print formats.

## Hierarchy

5

TCustomControl (VCL)

❶ TVpCustomControl (VPBase) . . . . . 30

TVpPrintPreview (VPPrtPrv)

## Properties

| | | |
|---|---|---|
| BorderColor | DestPrinter | PageColor |
| BorderStyle | DrawingStyle | StartDate |
| ControlLink | EndDate | ❶ Version |
| CurPage | OffPageColor | ZoomFactor |

## Methods

| | | |
|---|---|---|
| FirstPage | IsLastPage | PrevPage |
| ForceUpdate | LastPage | |
| IsFirstPage | NextPage | |

## Events

| | |
|---|---|
| ❶ AfterEnter | ❶ OnMouseWheel |
| ❶ AfterExit | OnPageChange |

### BorderColor — property

```
property BorderColor : TColor
```

Default: clBlack

Specifies the color of the border drawn around the page.

See also: OffPageColor, PageColor

### BorderStyle — property



```
property BorderStyle : TBorderStyle
```

Default: bsSingle

Determines the style used when drawing the border.

The possible values are bsSingle (a single line border is drawn around the component) or bsNone (no border line).

The TBorderStyle type is defined in the VCL.

See also: DrawingStyle

### ControlLink — property

```
property ControlLink : TVpControlLink
```

Associates a TVpControlLink to this component.

The print formats that the print preview uses are defined in the associated TVpControlLink component.

### CurPage — property

```
property CurPage : Integer
```

Specifies the current page being displayed.

Warning: Several factors go into determining how much data will fit on a page. Some controls, like contact grids use variable amounts of data when rendering. To render a page, all previous pages must also be rendered in order to properly calculate what the page will look like. Setting this value to something large may take some time to calculate. It is safer to use NextPage and IsLastPage to cycle through to higher page numbers.

See also: CurPage, IsFirstPage, IsLastPage, LastPage, NextPage, PrevPage

## DestPrinter — property

```
property DestPrinter : TPrinter
```

Associates the print preview with a printer.

A printer is associated with the print preview component. The paper size from the printer, as well as the resolution of the printer, is used to determine how the page will view when it is printed.

Note: The default printer is automatically associated with the print preview.



## DrawingStyle — property

```
property DrawingStyle : TVpDrawingStyle
```

```
TVpDrawingStyle = (dsFlat, ds3d);
```

Default: ds3d

Determines how the border around the control will be drawn.

If the BorderStyle is bsSingle, the type of border around the control will be determined by this property. Two types of borders can be used, flat (dsFlat) and 3d (ds3d).

See also: BorderStyle

## EndDate — property

```
property EndDate : TDateTime
```

Specifies the last date to be printed.

This is needed to calculate the last page of the print job.

See also: StartDate

## FirstPage — method

```
procedure FirstPage;
```

Goes to the first page of the print job.

See also: CurPage, IsFirstPage, LastPage, NextPage, PrevPage

### ForceUpdate method

```
procedure ForceUpdate;
```

Forces the print preview to match changes in the print format.

When print formats are added and deleted, the print preview will be notified. However, in some cases, the print formats may be changed without the print preview being notified. ForceUpdate will force the print preview to use the change print formats.

### IsFirstPage method

```
function IsFirstPage : Boolean;
```

Determines if the current page is the first page of the print job.

Returns True if the current page is the first page in the print job.

See also: CurPage, FirstPage, IsLastPage, PrevPage

### IsLastPage method

```
function IsLastPage : Boolean;
```

Determines if the current page is the last page of the print job.

Returns True if the current page is the last page of the print job.

See also: CurPage, IsFirstPage, LastPage, NextPage

### LastPage method

```
procedure LastPage;
```

Goes to the last page of the print job.

Warning: Several factors go into determining how much data will fit on a page. Some controls, like contact grids use variable amounts of data when rendering. To render a page, all previous pages must also be rendered in order to properly calculate what the page will look like. For large print jobs, going to the last page may take some time. It is safer to use NextPage and IsLastPage to iterate to the last page.

See also: CurPage, FirstPage, IsLastPage, NextPage, PrevPage

## NextPage — method

```
procedure NextPage;
```

Goes to the next page of the print job.

See also: CurPage, FirstPage, IsLastPage, LastPage, PrevPage

## OffPageColor — property

```
property OffPageColor : TColor
```

Default: clSilver

5

Sets the background color of the control.

If the page displayed is either smaller than the print preview control, or is zoomed out to be smaller than the print preview control, the OffPageColor is used fill in the area that is not part of the page.

See also: BorderColor, PageColor, ZoomFactor

## OnPageChange — event

```
property OnPageChange : TVpPageChange
```

```
TVpPageChange = procedure(
  Sender : TObject; NewPage : Integer) of object;
```

Defines an event handler that is called when the current page changes.

See also: CurPage, FirstPage, LastPage, NextPage, PrevPage

## PageColor — property

```
property PageColor : TColor
```

Default: clWhite

Sets the color of the page.

PageColor sets the background color of the page. Any part of the page that is not overlaid with a print element will be displayed in this color.

See also: BorderColor, OffPageColor

## PrevPage — method

```
procedure PrevPage;
```

Goes to the previous page of the print job.

See also: CurPage, FirstPage, IsFirstPage, LastPage, NextPage

## StartDate — property

```
property StartDate : TDateTime
```

Specifies the starting date of the print job.

See also: EndDate

## ZoomFactor — property

```
property ZoomFactor : TVpPPZoomFactor
```

```
TVpPPZoomFactor = (zfFitToControl, zfActualSize, zf25Percent,
  zf33Percent, zf50Percent, zf67Percent, zf75Percent);
```

Default: zfFitToControl

Specifies the magnification level of the print preview dialog box.

The print preview component supports several levels of magnification of the page in the print preview ranging from 25% to full size.

If the magnification level results in a page that is larger than the print preview control, scroll bars will be added to the control to allow the user to view all parts of the page. If the magnification results in a page smaller than the control, the extra space will be filled with the OffPageColor.

ZfFitToControl will attempt to fit the page to the size of the print preview control as well as possible. The aspect ratio of the page is maintained. This may result in either the bottom or the right side of the control not being completely filled with the page.

See also: OffPageColor, PageColor

# TVpPrintFormatComboBox Component

The TVpPrintFormatComboBox displays all the print formats in the associated TVpControlLink component. Changing the current selection of the TVpPrintFormatComboBox will automatically change the current print format used by the TVpControlLink component and notify other components that the current print format has changed.

When print formats are added or removed, this component is automatically notified of the event and will update the selection of print formats accordingly.



## Hierarchy

TCustomComboBox (VCL)

TVpPrintFormatComboBox (VPPrtFmtCBox)

## Properties

ControlLink

## Methods

UpdateItems

## Reference Section

### ControlLink — property

```
property ControlLink : TVpControlLink
```

Associates a TVpControlLink to this component.

### UpdateItems — method

```
procedure UpdateItems;
```

Forces the component to refresh its list of print formats.

In cases where the print formats in the associated TVpControlLink do not match the print formats displayed in the combo box, this method will force the combo box to match itself to the TVpControlLink component.

See also: ControlLink

# TVpCustomDateEdit Class

The TVpCustomDateEdit class is the immediate ancestor of the TVpDateEdit component. It implements all of the methods and properties of the DateEdit component and is identical to the TVpDateEdit component, except that no properties are published.

The TVpCustomDateEdit class is provided to facilitate your creation of descendant DateEdit components. For property and method descriptions see “TVpDateEdit Component” on page 229.



## Hierarchy

TCustomEdit (VCL)

- TVpEdPopup (VPEdPop)
  - TVpCustomDateEdit (VPDateEdit)

# TVpDateEdit Component

The TVpDateEdit component is a simple, specialized data entry component that allows free-form date entry and provides a pop-up calendar. It was created for use in the standard event edit dialog box, and is provided as a component for you to use in your own applications.

## Hierarchy

TCustomEdit (VCL)

TVpEdPopup (VPEdPop)

TVpCustomDateEdit (VPDateEdit) . . . . . 228

TVpDateEdit (VPDateEdit)

## Properties

| | | |
|---|---|---|
| AllowIncDec | PopupCalColors | ReadOnly |
| Date | PopupCalFont | RequiredFields |
| Epoch | PopupCalHeight | TodayString |
| ForceCentury | PopupCalWidth | WeekStarts |

## Methods

| | |
|---|---|
| FormatDate | PopupOpen |

## Events

| | |
|---|---|
| OnGetDate | OnSetDate |

### AllowIncDec — property

```
property AllowIncDec : Boolean
```

Default: True

Determines if pressing the plus or minus keys will increment and decrement the date.

If AllowIncDec is True, pressing the plus key will increment the date by one day and pressing the minus key will decrement the field by one day. If AllowIncDec is False, the plus and minus keys have no effect.



### Date — run-time property

```
property Date : TDateTime
```

Provides access to the control's value as a TDateTime.

Programmatically change the date of the TVpDateEdit control by setting Date. Retrieve the value of the TVpDateEdit control by reading Date.

### Epoch — property

```
property Epoch : Integer
```

Default: The beginning of the current century.

Specifies the starting year of a 100-year period.

The Epoch defaults to the beginning of the current century. As recently as 1999, this value would have been "1900." However, as a new century has recently begun, the value of Epoch will default to "2000" for the next one hundred years.

### ForceCentury — property

```
property ForceCentury : Boolean
```

Default: False

Determines if the date mask is forced to display four digit years.

If ForceCentury is True, the date mask obtained from Windows is modified to produce a date which includes the display of a four digit year. If False, the date mask obtained from Windows is used without any modification.

### FormatDate — virtual method

```
function FormatDate(Value : TDateTime) : string; virtual;
```

Correctly formats a TDateTime value into a string suitable for display.

FormatDate is used internally by the TVpDateEdit component to properly format the TDateTime value for display in the control.

### OnGetDate — event

```
property OnGetDate : TVpGetDateEvent
```

```
TVpGetDateEvent = procedure(
  Sender : TObject; var Value : string) of object;
```

Defines an event handler that is called when the component's date changes.

Sender is the instance of the component and Value is the text value of the date after is has already been parsed and expanded as necessary. You can change Value to represent any valid date. The format of the date must match the date format used by the control. That is, it must use the Windows DateMask and date separator characters and must honor the setting of the ForceCentury property.

Note: See the source code of the FormatDate method for an example of how to format a date mask in accordance with the ForceCentury property.

See also: ForceCentury

### OnSetDate — property

```
property OnSetDate : TNotifyEvent
```

Defines an event handler that is called when the value of the displayed date changes.

### PopupCalColors — property

```
property PopupCalColors : TVpCalColors
```

Determines the colors used by the pop-up calendar.

See the Colors property of the TVpCalendar Component on page 192 for additional information about the individual color properties that can be set.

See also: TVpCalendar.Colors

## PopupCalFont — property

```
property PopupCalFont : TFont
```

Determines the font used for the pop-up calendar.

See also: TVpCalendar.Font

## PopupCalHeight — property

```
property PopupCalHeight : Integer
```

Determines the height of the pop-up calendar.



## PopupCalWidth — property

```
property PopupWidth : Integer
```

Determines the width of the pop-up calendar.

## ReadOnly — property

```
property ReadOnly : Boolean
```

Determines if the date displayed in the edit field can be changed.

## RequiredFields — property

```
property RequiredFields : TVpRequiredFields
TVpRequiredField = (rfYear, rfMonth, rfDay);
TVpRequiredFields = set of TVpRequiredField;
```

Default: [rfMonth, rfDay]

Determines which of the three possible dates must be entered by the user.

By default, only the month and day fields are required, in which case the year is assumed to be the current year. In general, if a field isn't required, the current date is used to determine the value of the missing portion.

## TodayString — property

```
property TodayString : string
```

Default: The Windows date separator.

Determines the text that is interpreted as meaning, “enter the current date here.”

By default, TodayString is the character used to separate the individual date fields, (day, month and year). Entering the TodayString and exiting the field will cause the current date to be automatically entered. For example, entering ‘/’ and leaving the field will display the current date.

## WeekStarts — property

```
property WeekStarts : TVpDayType
```

```
TVpDayType = (dtSunday, dtMonday, dtTuesday, dtWednesday,
 dtThursday, dtFriday, dtSaturday);
```

Default: dtSunday

Determines which day begins the week.

The value of the WeekStarts property determines which day of the week occupies the left-most column of the calendar.

# TVpResourceCombo Component

Provides a combo box list of available resources. When the DataStore loads, the Combo Boxes items list populates itself with the description of each resource. When the user selects a resource from the list, the ComboBox notifies the ControlLink which forces the other components to refresh themselves.

## Hierarchy

TCustomComboBox (VCL)

TVpResourceCombo (VPBaseDS)

## Properties

ControlLink DataStore Version

## Reference Section

### ControlLink — property

```
property ControlLink : TVpControlLink
```

Synchronizes other TVpLinkableControls with each other.

The ControlLink is responsible for notifying other dependent components when important events have taken place. Other TVpLinkableControls on the form that share the same ControlLink component will synch up with each other.

See also: TVpControlLink

### DataStore — property

```
property DataStore : TVpCustomDataStore
```

Provides access to the Visual PlanIt data.

The DataStore is the component that manages all of the Visual PlanIt data.

See also: TVpCustomDataStore

### Version — read-only property

```
property Version : string
```

Displays the library version in the Object Inspector.

The Version property also has a specialized design-time property editor that displays the Visual PlanIt about box when the ellipsis button is clicked.

# TVpDayView Component

The TVpDayView Component is a visual control that displays one day's worth of events for one resource. It acquires its data from a connected DataStore component and synchronizes with other TVpLinkableControls via a ControlLink component.

## Hierarchy

TCustomControl (VCL)

❶ TVpCustomControl (VPBase) . . . . . 30

❷ TVpLinkableControl (VPBaseDS). . . . . 32

TVpDayView (VPDayView)

## Properties

ActiveRow
Color
❷ ControlLink
❷ DataStore
Date
DateLabelFormat
DefaultTopHour
DrawingStyle
Granularity
GutterWidth
HeadAttributes
LineColor
LineCount
OwnerDrawCells
OwnerDrawColHeader
OwnerDrawRowHeader
❷ ReadOnly
ResourceID
RowHeadAttributes
ShowResourceName
TimeFormat
TimeSlotColors
TopHour
TopLine
❶ Version
VisibleLines

## Methods

AutoScaledPaintToCanvas
DeleteActiveEvent
GetControlType
❷ GetLastPrintLine
HourToLine
LinkHandler
PaintToCanvas
RenderToCanvas

## Events

AfterEdit
❶ AfterEnter
❶ AfterExit
BeforeEdit
IncludeWeekends
NumDays
❶ OnMouseWheel
OnOwnerEditEvent

## Reference Section

### ActiveRow — property

```
property ActiveRow : Integer
```

Specifies which row is currently highlighted.

ActiveRow can only be within the cluster of rows that are currently visible.

### AfterEdit — event

```
property AfterEdit : TVpAfterEditEvent
```

```
TVpAfterEditEvent = procedure(
  Sender : TObject; Event : TVpEvent) of object;
```

Defines an event handler that is called after the event has been edited.

Sender contains a reference to the DayView. Event contains a reference to the event that was just edited.

### AutoScaledPaintToCanvas — method

```
procedure AutoScaledPaintToCanvas(
  PaintCanvas : TCanvas; PaintTo : TRect; Angle : TVpRotationAngle;
  RenderDate : TDateTime; StartLine : Integer; StopLine : Integer;
  UseGran : TVpGranularity);
```

```
TVpRotationAngle = (ra0, ra90, ra180, ra270);
```

```
TVpGranularity = (
  gr60Min, gr30Min, gr20Min, gr15Min, gr10Min, gr6Min, gr5Min);
```

Paints the day view to an arbitrary canvas with correct scaling.

This method will calculate the proper scale needed to paint the day view. In most cases, PaintToCanvas handles this automatically.

StartLine and StopLine specify the start and stop hours of the day view. The values for these can be calculated by calling HourToLine. A value of –1 for StartLine will start at the beginning of the day (midnight) and –1 for StopLine will end at the end of the day (11:59pm).

UseGran specifies the granularity to paint the day view.

See also: HourToLine, PaintToCanvas

## BeforeEdit — event

```
property BeforeEdit : TVpBeforeEditEvent
```

```
TVpBeforeEditEvent = procedure(Sender : TObject; Event : TVpEvent;
  var AllowIt : Boolean) of object;
```

Defines an event handler that is called just before the in-place editor, or the Event Edit Dialog, is displayed.

Sender contains a reference to the DayView, Event contains a reference to the event object that is being edited, and AllowIt determines whether the operation should be allowed.

If you decide that the edit shouldn't be allowed, set AllowIt False and it will be abandoned.



## Color — property

```
property Color : TColor
```

Default: clWindow

Determines the default background color of the client area.

Gridlines that do not fall into one of the TimeSlotColors categories, will be painted this color.

## Date — property

```
property Date : TDateTime
```

Default: Now

Specifies the date for which all events will be displayed.

The DayView is only capable of displaying events for one day. Date specifies the day for which events will be displayed. Set Date to the day for which you wish to display events.

Note: Date can be changed by a TVpCalendar component that is connected to the same ControlLink as the DayView. The Calendar will also reflect date changes made to the DayView either programmatically, or due to the manipulation of the DayView's spinner buttons. See "TVpCalendar Component" on page 190 for more information.

See also: ControlLink

## DateLabelFormat — property

```
property DateLabelFormat : string
```

Default: “dddd, mmmm dd, yyyy”

Determines the formatting string that will be applied to the date string displayed at the top of the control.

## DefaultTopHour — property

```
property DefaultTopHour : TVpHours
```

```
TVpHours = (h12AM, h_1AM, h_2AM, h_3AM, h_4AM, h_5AM, h_6AM, h_7AM,
  h_8AM, h_9AM, h10AM, h11AM, h12PM, h_1PM, h_2PM, h_3PM, h_4PM,
  h_5PM, h_6PM, h_7PM, h_8PM, h_9PM, h10PM, h11PM);
```

Default: h7AM

Defines the hour that is displayed at the top of the grid by default.

See also: TopHour

## DeleteActiveEvent — method

```
procedure DeleteActiveEvent(Verify : Boolean);
```

Deletes the currently selected event.

DeleteActiveEvent permanently removes the event from the schedule by setting the event’s Deleted property to True, and then calling the DataStore’s PostEvents method.

If Verify is True, the user will be asked to confirm the deletion. Otherwise, the deletion will take place silently.

See also: TVpCustomDataStore.PostEvents, TVpEvent.Deleted

### DrawingStyle — property

```
property DrawingStyle : TVpDrawingStyle
```

```
TVpDrawingStyle = (dsFlat, ds3d);
```

Default: ds3d

Determines the style in which the control will be drawn.

### GetControlType — method



```
function GetControlType : TVpItemType; override;
```

```
TVpItemType = (itDayView, itWeekView, itMonthView, itCalendar,
  itShape, itCaption, itTasks, itContacts);
```

Returns the control's ItemType.

GetControlType is used by the printing subsystem for arranging components on the printed canvas.

### Granularity — property

```
property Granularity : TVpGranularity
```

```
TVpGranularity = (
  gr60Min, gr30Min, gr20Min, gr15Min, gr10Min, gr6Min, gr5Min);
```

Default: gr30min

Defines the amount of time represented by one gridline.

See also: LineCount, TopLine

### GutterWidth — property

```
property GutterWidth : Integer
```

Default: 7

Determines the width in pixels of the narrow gutter between the row head and the grid rows.

### HeadAttributes — property

```
property HeadAttributes : TVpCHAttributes
```

Determines the attributes that will be used to paint the column header.

## HourToLine — method

```
function HourToLine(const Value : TVpHours;
  const UseGran : TVpGranularity) : Integer;
```

```
TVpHours = (h12AM, h_1AM, h_2AM, h_3AM, h_4AM, h_5AM, h_6AM, h_7AM,
  h_8AM, h_9AM, h10AM, h11AM, h12PM, h_1PM, h_2PM, h_3PM, h_4PM,
  h_5PM, h_6PM, h_7PM, h_8PM, h_9PM, h10PM, h11PM);
```

Converts a DayView hour into a line number.

HourToLine converts an hour into a line number. This line number can be used in the StartLine and StopLine parameters to AutoScaledPaintToCanvas or RenderToCanvas.

See also: AutoScaledPaintToCanvas

5

## IncludeWeekends — property

```
property IncludeWeekends : Boolean
```

Determines whether to include weekend days in multiple day mode.

IncludeWeekends is ignored unless the value of NumDays is greater than 1.

See also: NumDays

## LineColor — property

```
property LineColor : TColor
```

Default: clGray

Determines the color of the gridlines.

## LineCount — read-only, run-time property

```
property LineCount : Integer
```

Returns the number of time lines currently covered by the day.

The value of LineCount depends on the value of Granularity. For instance, if Granularity is set to gr60Min, the value of LineCount will be 24. If the value of Granularity is gr30Min, the value of LineCount will be 48, etc.

Note: This is not the number of visible lines, rather the total number of lines represented by one day at the current granularity setting. If you need the number of visible lines, see the VisibleLines property.

See also: Granularity, VisibleLines

### LinkHandler — method

```
procedure LinkHandler(Sender : TVpLinkableControl;
  NotificationType : TVpNotificationType;
  const Value : Variant); override;
```

Defines the event that is called by the ControlLink component in response to a change notification.

See also: TVpControlLink

### NumDays — property



```
property NumDays : Integer
```

Determines how many days are displayed in the DayView component.

The DayView component can display multiple, concurrent days worth of events. It begins with the day specified by the value of the Date property, and increments the value of the date accordingly, until each column is filled.

### OnOwnerEditEvent — event

```
property OnOwnerEditEvent : TVpEditEvent
```

```
TVpEditEvent = procedure(Sender : TObject; Event : TVpEvent;
  Resource : TVpResource; var AllowIt : Boolean) of object;
```

Defines an event that is called when the item is being edited.

The Visual PlanIt DayView provides a detailed event editor dialog box that should be satisfactory for most implementations. However, if you wish to supply your own event-editing dialog box, you should set it up and call it from this event handler.

Sender contains a reference to this DayView. Event and Resource contain references to the event being edited, and the resource to which it belongs. If the event is successfully edited, set the value of AllowIt to True. Setting the value of AllowIt to False will cause the application to abandon any changes made to the event.

Caution: Be sure to show your dialog box modally, or execution will fall through causing the application to get out of sync.

Note: You must add TVpData to your form's Uses clause to get the application to find the declarations of TVpEvent and TVpResource.

The following is a suggestion of how to use the OnOwnerEditEvent event:

```
procedure TForm1.TVpDayView1OwnerEditEvent(
  Sender : TObject; Event : TVpEvent; Resource : TVpResource;
  var AllowIt : Boolean);
begin
 Application.CreateForm(
   TMyEventEditForm, MyEventEditForm);
 try
  MyEventEditForm.SaveChanges := False;
  { populate your form's fields with the data }
  { from the Resource and Event objects }
  MyEventEditForm.ShowModal;
  AllowIt := MyEventEditForm.SaveChanges;
  if AllowIt then begin
   { Transfer the data from your edit form to }
   { the Resource and Event objects }
  end;
   finally
    MyEventEditForm.Free;
    MyEventEditForm := nil;
   end;
end;
```

## OwnerDrawCells — property

```
property OwnerDrawCells : TVpOwnerDrawRowEvent
```

Defines an event handler that is fired before the cells are drawn.

Allows the developer to customize the drawing of the cells.

## OwnerDrawColHeader — property

```
property OwnerDrawColHeader : TVpOwnerDrawEvent
```

Defines an event handler that is fired before the column header is drawn.

Allows the developer to customize the drawing of the column header.

## OwnerDrawRowHeader — property

```
property OwnerDrawRowHeader : TVpOwnerDrawRowEvent
```

Defines an event handler that fires before the row header is drawn.

Allows the developer to customize the drawing of the row header.

## PaintToCanvas — method

```
procedure PaintToCanvas(ACanvas : TCanvas; ARect : TRect;
  Angle : TVpRotationAngle; ADate : TDateTime;
  StartHour : TVpHours; EndHour : TVpHours;
  UseGran : TVpGranularity);
```

```
TVpRotationAngle = (ra0, ra90, ra180, ra270);
```

```
TVpHours = (h12AM, h_1AM, h_2AM, h_3AM, h_4AM, h_5AM, h_6AM, h_7AM,
  h_8AM, h_9AM, h10AM, h11AM, h12PM, h_1PM, h_2PM, h_3PM, h_4PM,
  h_5PM, h_6PM, h_7PM, h_8PM, h_9PM, h10PM, h11PM);
```

```
TVpGranularity = (
  gr60Min, gr30Min, gr20Min, gr15Min, gr10Min, gr6Min, gr5Min);
```

Paints the day view to an arbitrary canvas with correct scaling.

PaintToCanvas will paint the day view onto ACanvas in the rectangle specified by ARect. The day view will be rotated inside of ARect by the angle specified in Angle. ACanvas can be either a screen or printer canvas.

StartLine and StopLine specify the start and stop hours of the day view. The values for these can be calculated by calling HourToLine. A value of –1 for StartLine will start at the beginning of the day (midnight) and –1 for StopLine will end at the end of the day (11:59pm).

UseGran specifies the granularity to paint the day view.

## RenderToCanvas — virtual abstract method

```
procedure RenderToCanvas(
  RenderCanvas : TCanvas; RenderIn : TRect;
  Angle : TVpRotationAngle; Scale : Extended;
  RenderDate : TDateTime; StartLine : Integer;
  StopLine : Integer; UseGran : TVpGranularity;
  DisplayOnly : Boolean); virtual; abstract;
```

```
TVpRotationAngle = (ra0, ra90, ra180, ra270);
```

```
TVpGranularity = (
  gr60Min, gr30Min, gr20Min, gr15Min, gr10Min, gr6Min, gr5Min);
```

Paints the control to an arbitrary canvas.

RenderToCanvas will paint the control onto ACanvas in the rectangle specified by ARect. The control will be rotated inside of ARect by the angle specified in Angle. ACanvas can be either a screen or printer canvas.

Some additional parameters are used to finely specify how the control is painted.

Scale is used to manually increase or decrease the size of the control. Most controls calculate the scale in terms of ARect.

RenderDate is used by TVpCustomCalendar, TVpDayView, TVpWeekView and TVpMonthView to specify the date to display.

StartLine is used by TVpDayView to specify the first hour to display. If this value is –1, the start of the day (midnight) is used. This value is also used by TVpContactGrid to specify the first contact and TVpTaskList to specify the first task to display. A value of 0 will start with the first contact or task.

StopLine is used by TVpDayView to specify the last hour to display. If this value is –1, the end of the day will be used (11:59pm).

UseGran is used by the TVpDayView to specify the granularity to use.

If DisplayOnly is on, the display will be adjusted slightly for printing.



### ResourceID — property

```
property ResourceID : Integer
```

Default: -1

The index value of the currently selected resource.

ResourceID can be set either programmatically, or via a TVpResourceCombo component.

See also: TVpResourceCombo

### RowHeadAttributes — property

```
property RowHeadAttributes : TVpRHAttributes
```

Determines the attributes that will be used to paint the row header.

See also: TVpRHAttributes

### ShowResourceName — property

```
property ShowResourceName : Boolean
```

Default: True

Determines whether the resource name will be displayed at the top of the DayView.

### TimeFormat — property

```
property TimeFormat : TVpTimeFormat
```

```
TVpTimeFormat = (tf24Hour, tf12Hour);
```

Default: tf12Hour

Determines whether the grid displays the time in 24-hour format or 12-hour format with am/pm display.

### TimeSlotColors — property

```
property TimeSlotColors : TVpTimeSlotColor
```

Determines the color that will be used to paint the background of different time lines.

### TopHour — property

```
property TopHour : TVpHours
```

```
TVpHours = (h12AM, h_1AM, h_2AM, h_3AM, h_4AM, h_5AM, h_6AM, h_7AM,
  h_8AM, h_9AM, h10AM, h11AM, h12PM, h_1PM, h_2PM, h_3PM, h_4PM,
  h_5PM, h_6PM, h_7PM, h_8PM, h_9PM, h10PM, h11PM);
```

Sets the hour that will be displayed in the top line of the control.

Set TopHour to force the control to begin the display at that time. Read TopHour to determine at which time the display begins.

See also: DefaultTopHour

## TopLine — property

```
property TopLine : Integer
```

Sets the line that will be displayed at the top of the grid.

TopLine differs from TopHour in that each line represents a different amount of time, depending on the value of Granularity. TopHour will always start the display at the beginning of an hour, while TopLine may start the display at any point within an hour.

See also: Granularity, TopHour

## VisibleLines — read-only property

```
property VisibleLines : Integer
```

The number of lines that are currently visible.

If you need the number of lines that are represented by one day at the current granularity settings, see the LineCount property.

# TVpCHAttributes Class

The TVpCHAttributes class groups the properties that control the look of the column header in the Object Inspector.

## Hierarchy

TPersistent (VCL)

TVpCHAttributes (VPDayView)

5

## Properties

Color

Font

Owner

## Reference Section

### Color — property

```
property Color : TColor
```

Provides access to the color that will be used to paint the Column Header's background.

### Font — property

```
property Font : TFont
```

Provides access to the font that will be used in the Column Header.

### Owner — property

```
property Owner : TVpDayView
```

Provides access to the object's owner.

# TVpRHAttributes Class

The TVpRHAttributes class groups the properties that control the look of the row header in the Object Inspector.

## Hierarchy

TPersistent (VCL)

TVpRHAttributes (VPDayView)

5

## Properties

| | |
|---|---|
| Color | MinuteFont |
| HourFont | Owner |

## Reference Section

### Color — property

```
property Color : TColor
```

Provides access to the color that will be used to paint the row header's background.

### HourFont — property

```
property HourFont : TVpFont
```

Provides access to the font that will be used to draw the row header's hour.

### MinuteFont — property

```
property MinuteFont : TVpFont
```

Provides access to the font that will be used to draw the row header's minute.

### Owner — property

```
property Owner : TVpDayView
```

Provides access to the object's owner.

# TVpTimeSlotColor Class

The TVpTimeSlotColor class groups the properties that control the color of the time slots in the Object Inspector.

## Hierarchy

TPersistent (VCL)

TVpTimeSlotColor (VPDayView)

5

## Properties

| | | |
|---|---|---|
| Active | Inactive | Weekend |
| Holiday | Weekday | |

## Reference Section

### Active — property

```
property Active : TColor
```

Defines the color that will be used to paint the background of active time lines.

### Holiday — property

```
property Holiday : TColor
```

Defines the color that will be used to paint the background of holiday time lines.

### Inactive — property

```
property Inactive : TColor
```

Defines the color that will be used to paint the background of Inactive time lines.

### Weekday — property

```
property Weekday : TColor
```

Defines the color that will be used to paint the background of weekday time lines.

### Weekend — property

```
property Weekend : TColor
```

Defines the color that will be used to paint the background of weekend time lines.

# TVpWeekView Component

The WeekView component displays the events of an entire week. It has an in-place editor, as well as a built-in Event Edit dialog box.

## Hierarchy

TCustomControl (VCL)

❶ TVpCustomControl (VPBase) . . . . . 30

❷ TVpLinkableControl (VPBaseDS). . . . . 32

TVpWeekView (VPWeekView)

## Properties

Color
❷ ControlLink
❷ DataStore
Date
DateLabelFormat
DayHeadAttributes
DrawingStyle
EventFont
HeaderFont
LineColor
❷ ReadOnly
ShowEventTime
TimeFormat
❶ Version
WeekStartsOn

## Methods

DeleteActiveEvent
GetControlType
❷ GetLastPrintLine
LinkHandler
PaintToCanvas
RenderToCanvas

## Events

AfterEdit
❶ AfterEnter
❶ AfterExit
BeforeEdit
❶ OnMouseWheel
OnOwnerEditEvent

## Reference Section

### AfterEdit — event

```
property AfterEdit : TVpAfterEditEvent
```

```
TVpAfterEditEvent = procedure(
  Sender : TObject; Event : TVpEvent) of object;
```

Defines an event handler that is called after an edit has taken place.

### BeforeEdit — event

```
property BeforeEdit : TVpBeforeEditEvent
```

```
TVpBeforeEditEvent = procedure(Sender : TObject;
  Event : TVpEvent; var AllowIt : Boolean) of object;
```

Defines an event handler that is called just before an editor is displayed.

If you set AllowIt to False in your event handler, the user will not be allowed to edit the item.

### Color — property

```
property Color : TColor
```

Defines the background color of the client area.

### Date — property

```
property Date : TDateTime
```

Defines the WeekView's display date.

The WeekView displays seven days at a time. When the value of Date changes, the WeekView component will display the week in which that date occurs.

### DateLabelFormat — property

```
property DateLabelFormat : string
```

Determines the format of the dates displayed at the top of each day.

### DayHeadAttributes — property

```
property DayHeadAttributes : TVpDayHeadAttr
```

Clusters the properties that define how the top area of each day will appear.

### DeleteActiveEvent — method

```
procedure DeleteActiveEvent(Verify : Boolean);
```

Deletes the currently selected event.

DeleteEvent permanently removes the event from the schedule by setting the event's Deleted property to True and then calling the DataStore's PostEvents method.

If Verify is True, the user will be asked to confirm the deletion. Otherwise, the deletion will take place silently.

See also: TVpBDEDataStore.PostEvents, TVpEvent.Deleted



### DrawingStyle — property

```
property DrawingStyle : TVpDrawingStyle
```

```
TVpDrawingStyle = (dsFlat, ds3d);
```

Determines if the control is drawn flat or 3d.

### EventFont — property

```
property EventFont : TFont
```

Determines the font that will be used to draw events.

### GetControlType — method

```
function GetControlType : TVpItemType; override;
```

```
TVpItemType = (itDayView, itWeekView, itMonthView,
  itCalendar, itShape, itCaption, itTasks, itContacts);
```

Returns the control's ItemType.

GetControlType is used by the printing subsystem for arranging components on the printed canvas.

### HeaderFont — property

```
property HeaderFont : TFont
```

Determines the font that will be used to draw the heading area in top of the control.

## LineColor — property

```
property LineColor : TColor
```

Default: clGray

Determines the color of the gridlines.

## LinkHandler — method

```
procedure LinkHandler(Sender : TComponent;
  NotificationType : TVpNotificationType;
  const Value : Variant); override;
```

Defines the event that is called by the ControlLink component in response to a change notification.

See “TVpControlLink Component” on page 165 for more information.

## OnOwnerEditEvent

```
property OnOwnerEditEvent : TVpEditEvent
```

```
TVpEditEvent = procedure(Sender : TObject; Event : TVpEvent;
  Resource : TVpResource; var AllowIt : Boolean) of object;
```

Defines an event that is called when the item is being edited.

The Visual PlanIt DayView provides a detailed event editor dialog box that should be satisfactory for most implementations. However, if you wish to supply your own event-editing dialog box, you should set it up and call it from this event handler.

Sender contains a reference to this DayView. Event and Resource contain references to the event being edited, and the resource to which it belongs. If the event is successfully edited, set the value of AllowIt to True. Setting the value of AllowIt to False will cause the application to abandon any changes made to the event.

Caution: Be sure to show your dialog box modally, or execution will fall through causing the application to get out of sync.

Note: You must add TVpData to your form’s Uses clause to get the application to find the declarations of TVpEvent and TVpResource.

The following is a suggestion of how to use the OnOwnerEditEvent event with your own event edit dialog box:

```
procedure TForm1.TVpWeekView1OwnerEditEvent(
  Sender : TObject; Event : TVpEvent; Resource : TVpResource;
  var AllowIt : Boolean);
begin
 Application.CreateForm(
   TMyEventEditForm, MyEventEditForm);
 try
  MyEventEditForm.SaveChanges := False;
  { populate your form's fields with the data }
  { from the Resource and Event objects }
  MyEventEditForm.ShowModal;
  AllowIt := MyEventEditForm.SaveChanges;
  if AllowIt then begin
   { Transfer the data from your edit form to }
   { the Resource and Event objects }
  end;
   finally
    MyEventEditForm.Free;
    MyEventEditForm := nil;
   end;
end;
```

Following is a suggestion of how to do it with a TVpEventEditDialog component:

```
procedure TForm1.TVpWeekView1OwnerEditEvent(
  Sender : TObject; Event : TVpEvent; Resource : TVpResource;
  var AllowIt : Boolean);
begin
  VpEventEditDialog1.Execute(Event, VpDayView1.TimeFormat);
end;
```

Of course, the latter is no different than just allowing the WeekView to handle the dialog box.

## PaintToCanvas method

```
procedure PaintToCanvas(ACanvas : TCanvas; ARect : TRect;
  Angle : TVpRotationAngle; ADate : TDateTime);
```

```
TVpRotationAngle = (ra0, ra90, ra180, ra270);
```

Paints the week view to an arbitrary canvas.

PaintToCanvas will paint the week view onto ACanvas in the rectangle specified by ARect. The week view will be rotated inside of ARect by the angle specified in Angle. ACanvas can be either a screen or printer canvas.

## RenderToCanvas virtual abstract method

```
procedure RenderToCanvas(RenderCanvas : TCanvas;
  RenderIn : TRect; Angle : TVpRotationAngle; Scale : Extended;
  RenderDate : TDateTime; StartLine : Integer; StopLine : Integer;
  UseGran : TVpGranularity; DisplayOnly : Boolean);
  virtual; abstract;
```

```
TVpRotationAngle = (ra0, ra90, ra180, ra270);
```

```
TVpGranularity = (
  gr60Min, gr30Min, gr20Min, gr15Min, gr10Min, gr6Min, gr5Min);
```

Paints the control to an arbitrary canvas.

RenderToCanvas will paint the control onto ACanvas in the rectangle specified by ARect. The control will be rotated inside of ARect by the angle specified in Angle. ACanvas can be either a screen or printer canvas.

Some additional parameters are used to finely specify how the control is painted.

Scale is used to manually increase or decrease the size of the control. Most controls calculate the scale in terms of ARect.

RenderDate is used by TVpCustomCalendar, TVpDayView, TVpWeekView, and TVpMonthView to specify the date to display.

StartLine is used by TVpDayView to specify the first hour to display. If this value is –1, the start of the day (midnight) is used. This value is also used by TVpContactGrid to specify the first contact and TVpTaskList to specify the first task to display. A value of 0 will start with the first contact or task.

StopLine is used by TVpDayView to specify the last hour to display. If this value is –1 the end of the day will be used (11:59pm).

UseGran is used by the TVpDayView to specify the granularity to use.

If DisplayOnly is on, the display will be adjusted slightly for printing.



### ShowEventTime — property

```
property ShowEventTime : Boolean
```

Determines whether the time will be displayed along with the event description.

If the WeekView's space is limited, setting ShowEventTime to False will save space.

### TimeFormat — property

```
property TimeFormat : TVpTimeFormat
```

```
TVpTimeFormat = (tf24Hour, tf12Hour);
```

Default: tf12Hour

Determines whether the grid displays the time in 24-hour format or 12-hour format with am/pm display.

### WeekStartsOn — property

```
property WeekStartsOn : TVpDayType
```

```
TVpDayType = (dtSunday, dtMonday, dtTuesday,
  dtWednesday, dtThursday, dtFriday, dtSaturday);
```

Does not affect the display of the control. Due to Saturday and Sunday being half height, they must be displayed together which limits the weekview's flexibility when it comes to starting day.

WeekStartsOn is used internally to determine when weeks roll over from one to the next.

# TVpMonthView Component

The MonthView component is designed to display an entire month's worth of events at a time, for one resource. It differs from the TVpCalendar in that is has the ability to display events right on the calendar.

If the MonthView's space is limited, setting the ShowEvents property to False will prevent it from attempting to display the events.

## Hierarchy

TCustomControl (VCL)

❶ TVpCustomControl (VPBase) . . . . . 30

❷ TVpLinkableControl (VPBaseDS) . . . . . 32

TVpMonthView (VPMonthView)

## Properties

Color
❷ ControlLink
❷ DataStore
Date
DateLabelFormat
DayHeadAttributes
DayNameStyle
DayNumberFont
DrawingStyle
EventDayStyle
EventFont
KBNavigation
LineColor
OffDayColor
❷ ReadOnly
SelectedDayColor
ShowEvents
ShowEventTime
TimeFormat
❶ Version
WeekStartsOn

## Methods

GetControlType
❷ GetLastPrintLine
LinkHandler
PaintToCanvas
RenderToCanvas

## Events

❶ AfterEnter
❶ AfterExit
❶ OnMouseWheel
OwnerDrawCells

### Color — property

```
property Color : TColor
```

Default: clWindow

Determines the default background color of the client area.

### Date — property



```
property Date : TDateTime
```

Defines the MonthView's display date.

The MonthView displays an entire month at a time. When the value of Date changes, the MonthView component will display the month in which that date occurs.

### DateLabelFormat — property

```
property DateLabelFormat : string
```

Default: "mmmm yyyy"

Determines the formatting string that will be applied to the date string displayed at the top of the control.

### DayHeadAttributes — property

```
property DayHeadAttributes : TVpDayHeadAttr
```

Defines the formatting of the area across the top of the calendar where the day names are listed.

DayHeadAttributes clusters the attributes of the rows of day names together in the object inspector. The properties controlled are the background color and the font.

## DayNameStyle — property

```
property DayNameStyle : TVpMVDayNameStyle
TVpMVDayNameStyle = (dsLong, dsShort, dsLetter);
```

Defines how the day names will be displayed at the top of the calendar.

Provides display options for cases where the MonthView has very limited space, as shown in the following table:

| Option | Description |
|---|---|
| dsLong | Days are spelled completely. |
| dsShort | Days are shortened to their three-letter abbreviations. |
| dsLetter | Days are represented by their first letter only. |

## DayNumberFont — property

```
property DayNumberFont : TFont
```

Defines the font that will be used to draw the MonthView's day numbers.

## DrawingStyle — property

```
property DrawingStyle : TVpDrawingStyle
TVpDrawingStyle = (dsFlat, ds3d);
```

Default: ds3d

Determines the style in which the control will be drawn.

## EventDayStyle — property

```
property EventDayStyle : TFontStyles
```

Defines the style in which days with events are drawn.

TFontStyles is defined in the VCL unit Graphics.pas as follows:

```
TFontStyle = (fsBold, fsItalic, fsUnderline, fsStrikeOut);
TFontStyles = set of TFontStyle;
```

The MonthView component paints each day according to whether there are any events on that day. They are all set to False by default, so days with events are not drawn any different than days without events.

## EventFont — property

```
property EventFont : TFont
```

Specifies the font that will be used to draw events in the MonthView.

If the ShowEvents property is True, the MonthView component will use this font to draw the events. If ShowEvents is False, this property is not used.

See also: ShowEvents

## GetControlType — method



```
function GetControlType : TVpItemType; override;
```

```
TVpItemType = (itDayView, itWeekView, itMonthView,
  itCalendar, itShape, itCaption, itTasks, itContacts);
```

Returns the control's ItemType.

GetControlType is used by the printing subsystem for arranging components on the printed canvas.

## KBNavigation — property

```
property KBNavigation : Boolean
```

Default: True

Determines if the user is allowed to navigate the MonthView by using the keyboard's arrow keys.

## LineColor — property

```
property LineColor : TColor
```

Default: clGray

Determines the color of the gridlines.

### LinkHandler — Method

```
procedure LinkHandler(Sender : TVpLinkableControl;
  NotificationType : TVpNotificationType;
  const Value : Variant); override;
```

Called by the ControlLink component in response to a change notification from another component.

Each TVpLinkableControl descendant must define a LinkHandler procedure. It is used solely by the attached ControlLink component. Do not call the LinkHandler procedure programmatically.



### OffDayColor — property

```
property OffDayColor : TColor
```

Determines the background color of the days that don't belong to the currently active month.

### OwnerDrawCells — event

```
property OwnerDrawCells : TVpOwnerDrawDayEvent
```

```
TVpOwnerDrawDayEvent = procedure(
  Sender : TObject; const Canvas : TCanvas; R : TRect;
  Day : Integer; var Drawn : Boolean) of object;
```

Defines an event handler that is called when each cell is drawn.

Create an OwnerDrawCells event handler if you want to take the drawing of the cells into your own hands.

If you decide to skip a cell, set Drawn to False, and the default drawing mechanism will draw the cell. If you handle the drawing yourself, set the Drawn variable to True to skip default drawing.

### PaintToCanvas — method

```
procedure PaintToCanvas(ACanvas : TCanvas; ARect : TRect;
  Angle : TVpRotationAngle; ADate : TDateTime);
```

```
TVpRotationAngle = (ra0, ra90, ra180, ra270);
```

Paints the month view to an arbitrary canvas.

PaintToCanvas will paint the month view onto ACanvas in the rectangle specified by ARect. The month view will be rotated inside of ARect by the angle specified in Angle. ACanvas can be either a screen or printer canvas.

## RenderToCanvas — virtual abstract method

```
procedure RenderToCanvas(
  RenderCanvas : TCanvas; RenderIn : TRect;
  Angle : TVpRotationAngle; Scale : Extended;
  RenderDate : TDateTime; StartLine : Integer;
  StopLine : Integer; UseGran : TVpGranularity;
  DisplayOnly : Boolean); virtual; abstract;
```

```
TVpRotationAngle = (ra0, ra90, ra180, ra270);
```

```
TVpGranularity = (
  gr60Min, gr30Min, gr20Min, gr15Min, gr10Min, gr6Min, gr5Min);
```

5

Paints the control to an arbitrary canvas.

RenderToCanvas will paint the control onto ACanvas in the rectangle specified by ARect. The control will be rotated inside of ARect by the angle specified in Angle. ACanvas can be either a screen or printer canvas.

Some additional parameters are used to finely specify how the control is painted.

Scale is used to manually increase or decrease the size of the control. Most controls calculate the scale in terms of ARect.

RenderDate is used by TVpCustomCalendar, TVpDayView, TVpWeekView, and TVpMonthView to specify the date to display.

StartLine is used by TVpDayView to specify the first hour to display. If this value is –1, the start of the day (midnight) is used. This value is also used by TVpContactGrid to specify the first contact and TVpTaskList to specify the first task to display. A value of 0 will start with the first contact or task.

StopLine is used by TVpDayView to specify the last hour to display. If this value is –1, the end of the day will be used (11:59pm).

UseGran is used by the TVpDayView to specify the granularity to use.

If DisplayOnly is on, the display will be adjusted slightly for printing.

## SelectedDayColor — property

```
property SelectedDayColor : TColor
```

Specifies the color of the active day.

As the user navigates the MonthView component, the color of the active day changes as a visual indicator. This property determines what color the number of the active day will be drawn in.

## ShowEvents — property

```
property ShowEvents : Boolean
```

Determines if the MonthView component will display events.

Set ShowEvents to False if the MonthView has limited space and is unable to adequately display the events.

This does not affect the EventDayStyle. Days with events will still be painted according to the values of EventDayStyle.

## ShowEventTime — property

```
property ShowEventTime : Boolean
```

Determines whether the time will be displayed along with the event description.

If the MonthView's space is limited, setting ShowEventTime to False will save space by only displaying the event's description.

## TimeFormat — property

```
property TimeFormat : TVpTimeFormat
```

```
TVpTimeFormat = (tf24Hour, tf12Hour);
```

Default: tf12Hour

Determines whether the grid displays the time in 24-hour format or 12-hour format with am/pm display.

## WeekStartsOn — property

```
property WeekStartsOn : TVpDayType
```

Determines which day will be shown as the first day of the week.

# TVpContactGrid Component

The contact grid component displays the contacts list for one resource, in a flexible, multi-column control. It allows for limited in-place editing and includes a pop-up editor for complete contact manipulation.

The supplied pop-up contact editor is the same dialog box that is wrapped up as the TVpContactEditDialog. It may be replaced with a dialog box of your own, by creating an OnOwnerEditContact event handler.

5

## Hierarchy

TCustomControl (VCL)

❶ TVpCustomControl (VPBase) . . . . . 30

❷ TVpLinkableControl (VPBaseDS). . . . . 32

TVpContactGrid (VPContactGrid)

## Properties

| | | | | | |
|---|---|---|---|---|---|
| | BarColor | | ContactHeadAttributes | | ResourceID |
| | BarWidth | | ControlLink | | ShowResourceName |
| | Color | | DataStore | ❶ | Version |
| | ColumnWidth | | DrawingStyle | | |
| | ContactIndex | ❷ | ReadOnly | | |

## Methods

| | | | | | |
|---|---|---|---|---|---|
| | DeleteActiveContact | ❷ | GetLastPrintLine | | PaintToCanvas |
| | GetControlType | | LinkHandler | | RenderToCanvas |

## Events

| | | | | | |
|---|---|---|---|---|---|
| | AfterEdit | ❶ | AfterExit | ❶ | OnMouseWheel |
| ❶ | AfterEnter | | BeforeEdit | | OnOwnerEditContact |

## Reference Section

### AfterEdit — event

```
property AfterEdit : TVpAfterEditContact
```

```
TVpAfterEditContact = procedure(
  Sender : TObject; Contact : TVpContact) of object;
```

Defines an event handler that is called immediately after a contact is edited.

### BarColor — property

```
property BarColor : TColor
```

Default: clSilver

Determines the color of the vertical lines (bars) that separate the columns.

### BarWidth — property

```
property BarWidth : Integer
```

Default: 3

Defines the width of the vertical lines (bars) that separate the columns.

The real width of the bar area includes an invisible area on either side of the visible bar, which makes drag selection easier.

The size of the extra area is defined by a constant in TVpConst.pas called ExtraBarWidth. By default, the value of ExtraBarWidth is 2 pixels, which makes the default width of the bar area 7 pixels (three pixel bar with two extra, invisible pixels on each side).

### BeforeEdit — event

```
property BeforeEdit : TVpBeforeEditContact
```

```
TVpBeforeEditContact = procedure(Sender : TObject;
  Contact : TVpContact; var AllowIt : Boolean) of object;
```

Defines an event handler that is called just before the contact is edited.

Sender contains a reference to the ContactGrid. Contact is a reference to the contact object being edited. AllowIt determines whether the edit will be allowed.

Set AllowIt to False to prevent the edit from taking place.

### Color — property

```
property Color : TColor
```

Default: clWindow

Determines the default background color of the client area.

### ColumnWidth — property

```
property ColumnWidth : Integer
```

Default: 180 pixels

5

Defines the width of the columns in pixels.

ColumnWidth changes as the user drags a column bar horizontally. It may also be set programmatically.

### ContactHeadAttributes — property

```
property ContactHeadAttributes : TVpContactHeadAttr
```

Defines how the control's header will be drawn.

### ContactIndex — property

```
property ContactIndex : Integer
```

Contains the index of the currently selected contact.

### ControlLink — property

```
property ControlLink : TVpControlLink
```

Synchronizes other TVpLinkableControls with each other.

The ControlLink is responsible for notifying other dependent components when important events have taken place. Other TVpLinkableControls on the form that share the same ControlLink component will synch up with each other.

See "TVpControlLink Component" on page 165 for more information.

## DataStore — property

```
property DataStore : TVpCustomDataStore
```

Provides access to the Visual PlanIt data.

The DataStore is the component that manages all of the Visual PlanIt data.

See “TVpCustomDataStore Class” on page 92 for more information.

## DeleteActiveContact — method

```
procedure DeleteActiveContact(Verify : Boolean);
```

Deletes the currently selected contact.

DeleteActiveContact permanently removes the contact from the contact list by setting the contact’s Deleted property to True, and then calling the DataStore’s PostContacts method.

If Verify is True, the user will be asked to confirm the deletion. Otherwise, the deletion will take place silently.

See also: TVpContact.Deleted, TVpCustomDataStore.PostContacts

## DrawingStyle — property

```
property DrawingStyle : TVpDrawingStyle
```

```
TVpDrawingStyle = (dsFlat, ds3d);
```

Default: ds3d

Determines the style in which the control will be drawn.

## GetControlType — method

```
function GetControlType : TVpItemType; override;
```

```
TVpItemType = (itDayView, itWeekView, itMonthView,
  itCalendar, itShape, itCaption, itTasks, itContacts);
```

Returns the control’s ItemType.

GetControlType is used by the printing subsystem for arranging components on the printed canvas.

## LinkHandler — method

```
procedure LinkHandler(Sender : TVpLinkableControl;
  NotificationType : TVpNotificationType;
  const Value : Variant); override;
```

Defines the event that is called by the ControlLink component in response to a change notification.

See “TVpControlLink Component” on page 165 for more information.

## OnOwnerEditContact — event



```
property OnOwnerEditContact : TVpEditContact
```

```
TVpEditContact = procedure(Sender : TObject; Contact : TVpContact;
 Resource : TVpResource; AllowIt : Boolean) of object;
```

Defines an event that is called when the contact item is being edited.

The Visual PlanIt ContactGrid provides a detailed contact editor dialog box that should be satisfactory for most implementations. However, if you wish to supply your own contact-editing dialog box, you should set it up and call it from this event handler.

Sender contains a reference to this ContactGrid. Contact and Resource contain references to the contact being edited, and the resource to which it belongs. If the contact is successfully edited, set the value of AllowIt to True. Setting the value of AllowIt to False will cause the application to abandon any changes made to the contact.

Caution: Be sure to show your dialog box modally, or execution will fall through causing the application to get out of sync.

Note: You must add Visual PlanIt’s Data to your form’s Uses clause to get the application to find the declarations of TVpContact and TVpResource.

Following is a suggestion of how to use the OnOwnerEditContact event:

```
procedure TForm1.TVpContactGrid1OwnerEditContact(
 Sender : TObject; Contact : TVpContact;
 Resource : TVpResource; AllowIt : Boolean);
begin
 Application.CreateForm(
   TMyContactEditForm, MyContactEditForm);
 try
  MyContactEditForm.SaveChanges := False;
  { populate your form's fields with the data }
  { from the Resource and Contact objects }
  MyContactEditForm.ShowModal;
  AllowIt := MyContactEditForm.SaveChanges;
  if AllowIt then begin
   { Transfer the data from your edit form to }
   { the Resource and Contact objects }
  end;
   finally
    MyContactEditForm.Free;
    MyContactEditForm := nil;
   end;
end;
```

## PaintToCanvas — method

```
procedure PaintToCanvas(
  ACanvas : TCanvas; ARect : TRect; Angle : TVpRotationAngle);
```

```
TVpRotationAngle = (ra0, ra90, ra180, ra270);
```

Paints the contact grid to an arbitrary canvas.

PaintToCanvas will paint the contact grid onto ACanvas in the rectangle specified by ARect. The contact grid will be rotated inside of ARect by the angle specified in Angle. ACanvas can be either a screen or printer canvas.

```
procedure RenderToCanvas (RenderCanvas : TCanvas;
  RenderIn : TRect; Angle : TVpRotationAngle; Scale : Extended;
  RenderDate : TDateTime; StartLine : Integer; StopLine : Integer;
  UseGran : TVpGranularity; DisplayOnly : Boolean);
  virtual; abstract;
```

```
TVpRotationAngle = (ra0, ra90, ra180, ra270);
```

```
TVpGranularity = (
  gr60Min, gr30Min, gr20Min, gr15Min, gr10Min, gr6Min, gr5Min);
```



Paints the control to an arbitrary canvas.

RenderToCanvas will paint the control onto ACanvas in the rectangle specified by ARect. The control will be rotated inside of ARect by the angle specified in Angle. ACanvas can be either a screen or printer canvas.

Some additional parameters are used to finely specify how the control is painted.

Scale is used to manually increase or decrease the size of the control. Most controls calculate the scale in terms of ARect.

RenderDate is used by TVpCustomCalendar, TVpDayView, TVpWeekView, and TVpMonthView to specify the date to display.

StartLine is used by TVpDayView to specify the first hour to display. If this value is –1, the start of the day (midnight) is used. This value is also used by TVpContactGrid to specify the first contact and TVpTaskList to specify the first task to display. A value of 0 will start with the first contact or task.

StopLine is used by TVpDayView to specify the last hour to display. If this value is –1, the end of the day will be used (11:59pm).

UseGran is used by the TVpDayView to specify the granularity to use.

If DisplayOnly is on, the display will be adjusted slightly for printing.

## ResourceID — property

```
property ResourceID : Integer
```

Default: -1

The index value of the currently selected resource.

ResourceID can be set either programmatically or via a TVpResourceCombo component.

See also: TVpResourceCombo

## ShowResourceName — property

```
property ShowResourceName : Boolean
```

Default: True

Determines whether the resource name will be displayed at the top of the ContactGrid.

# TVpContactHeadAttr Class

Groups the contact head attributes together in the Object Inspector

## Hierarchy

TPersistent (VCL)

TVpContactHeadAttr (VPContactGrid)

5

## Properties

| | |
|---|---|
| Bordered | Font |
| Color | Grid |

## Reference Section

### Bordered — property

```
property Bordered : Boolean
```

Determines if the contact header cell is drawn with a single-pixel border.

If Bordered is True, the border's color will be cl3dDkShadow.

### Color — property

```
property Color : TColor
```

Defines the color of the contact header cell's background.

### Font — property

```
property Font : TFont
```

Defines the font that will be used in the contact header cell.

### Grid — read-only property

```
property Grid TVpContactGrid
```

Provides access to the ContactHeadAttr's owner.

# TVpTaskList Component

The task list component displays one day's worth of tasks for one resource, in a list-style control. It allows for limited in-place editing and includes a pop-up editor for complete task manipulation.

The supplied pop-up list editor may be replaced with a dialog box of your own if desired.

## Hierarchy

5

TCustomControl (VCL)

❶ TVpCustomControl (VPBase) . . . . . 30

❷ TVpLinkableControl (VPBaseDS). . . . . 32

TVpTaskList (VPTaskList)

## Properties

| | | | | | |
|---|---|---|---|---|---|
| | Color | | DrawingStyle | | ShowResourceName |
| ❷ | ControlLink | | LineColor | | TaskHeadAttributes |
| ❷ | DataStore | | MaxVisibleTasks | | TaskIndex |
| | DisplayOptions | ❷ | ReadOnly | ❶ | Version |

## Methods

| | | | | | |
|---|---|---|---|---|---|
| | DeleteActiveTask | ❷ | GetLastPrintLine | | PaintToCanvas |
| | GetControlType | | LinkHandler | | RenderToCanvas |

## Events

| | | | | | |
|---|---|---|---|---|---|
| | AfterEdit | ❶ | AfterExit | ❶ | OnMouseWheel |
| ❶ | AfterEnter | | BeforeEdit | | OnOwnerEditTask |

## Reference Section

### AfterEdit — event

```
property AfterEdit : TVpAfterEditTask
```

```
TVpAfterEditTask = procedure(
  Sender : TObject; Task : TVpTask) of object;
```

Defines an event handler that is called immediately after a task is edited.

### BeforeEdit — event

```
property BeforeEdit : TVpBeforeEditTask
```

```
TVpBeforeEditTask = procedure(Sender : TObject;
  Task : TVpTask; var AllowIt : Boolean) of object;
```

Defines an event handler that is called just before the task is edited.

Sender contains a reference to the TaskList. Task is a reference to the task object being edited. AllowIt determines whether the edit will be allowed.

Set AllowIt to False to prevent the edit from taking place.

### Color — property

```
property Color : TColor
```

Default: clWindow

Determines the default background color of the grid area.

### DeleteActiveTask — method

```
procedure DeleteActiveTask(Verify : Boolean);
```

Deletes the currently selected Task.

DeleteActiveTask permanently removes the task from the task list by setting the task's Deleted property to True, and then calling the DataStore's PostTasks method.

If Verify is True, the user will be asked to confirm the deletion. Otherwise, the deletion will take place silently.

See also: TVpBDEDataStore.PostTasks, TVpTask.Deleted

## DisplayOptions — property

```
property DisplayOptions : TVpTaskDisplayOptions
```

Controls how tasks are displayed in the TaskList.

The DisplayOptions property controls the following items:

CheckColor, CheckStyle, CompletedColor, DueDateFormat, OverdueColor, NormalColor, ShowAll, ShowCompletedTasks, ShowDueDate.

See "TVpTaskDisplayOptions Class" on page 285 for more information.



## DrawingStyle — property

```
property DrawingStyle : TVpDrawingStyle
```

```
TVpDrawingStyle = (dsFlat, ds3d);
```

Default: ds3d

Determines the style in which the control will be drawn.

## GetControlType — method

```
function GetControlType : TVpItemType; override;
```

```
TVpItemType = (itDayView, itWeekView, itMonthView,
  itCalendar, itShape, itCaption, itTasks, itContacts);
```

Returns the control's ItemType.

GetControlType is used by the printing subsystem for arranging components on the printed canvas.

## LineColor — property

```
property BarColor : TColor
```

Default: clSilver

Determines the color of the gridlines.

## LinkHandler — method

```
procedure LinkHandler(Sender : TVpLinkableControl;
  NotificationType : TVpNotificationType;
  const Value : Variant); override;
```

Defines the event that is called by the ControlLink component in response to a change notification.

See “TVpControlLink Component” on page 165 for more information.

## MaxVisibleTasks — property

```
property MaxVisibleTasks : Word
```

Determines the maximum number of tasks that can be displayed in the TaskList component.

## OnOwnerEditTask — event

```
property OnOwnerEditTask : TVpEditTask
```

```
TVpEditTask = procedure(Sender : TObject; Task : TVpTask;
 Resource : TVpResource; AllowIt : Boolean) of object;
```

Defines an event that is called when the task item is being edited.

The Visual PlanIt TaskList provides a detailed task editor dialog box that should be satisfactory for most implementations. However, if you wish to supply your own task-editing dialog box, you should set it up and call it from this event handler.

Sender contains a reference to this TaskList. Task and Resource contain references to the task being edited, and the resource to which it belongs. If the task is successfully edited, set the value of AllowIt to True. Setting the value of AllowIt to False will cause the application to abandon any changes made to the task.

Caution: Be sure to show your dialog box modally, or execution will fall through causing the application to get out of sync.

Note: You must add TVpData to your form’s Uses clause to get the application to find the declarations of TVpTask and TVpResource.

The following is a suggestion of how to use the OnOwnerEditTask event:

```
procedure TForm1.TVpTaskList1OwnerEditTask(
 Sender : TObject; Task : TVpTask;
 Resource : TVpResource; AllowIt : Boolean);
begin
 Application.CreateForm(
   TMyTaskEditForm, MyTaskEditForm);
 try
  MyTaskEditForm.SaveChanges := False;
  { populate your form's fields with the data }
  { from the Resource and Task objects }
  MyTaskEditForm.ShowModal;
  AllowIt := MyTaskEditForm.SaveChanges;
  if AllowIt then begin
   { Transfer the data from your edit form to }
   { the Resource and Task objects }
  end;
   finally
    MyTaskEditForm.Free;
    MyTaskEditForm := nil;
   end;
end;
```

## PaintToCanvas — method

```
procedure PaintToCanvas(
  ACanvas : TCanvas; ARect : TRect; Angle : TVpRotationAngle);
```

```
TVpRotationAngle = (ra0, ra90, ra180, ra270);
```

Paints the task list to an arbitrary canvas.

PaintToCanvas will paint the task list onto ACanvas in the rectangle specified by ARect. The task list will be rotated inside of ARect by the angle specified in Angle. ACanvas can be either a screen or printer canvas.

```
procedure RenderToCanvas (RenderCanvas : TCanvas;
  RenderIn : TRect; Angle : TVpRotationAngle; Scale : Extended;
  RenderDate : TDateTime; StartLine : Integer; StopLine : Integer;
  UseGran : TVpGranularity; DisplayOnly : Boolean);
  virtual; abstract;
```

```
TVpRotationAngle = (ra0, ra90, ra180, ra270);
```

```
TVpGranularity = (
  gr60Min, gr30Min, gr20Min, gr15Min, gr10Min, gr6Min, gr5Min);
```

Paints the control to an arbitrary canvas.

RenderToCanvas will paint the control onto ACanvas in the rectangle specified by ARect. The control will be rotated inside of ARect by the angle specified in Angle. ACanvas can be either a screen or printer canvas.

Some additional parameters are used to finely specify how the control is painted.

Scale is used to manually increase or decrease the size of the control. Most controls calculate the scale in terms of ARect.

RenderDate is used by TVpCustomCalendar, TVpDayView, TVpWeekView, and TVpMonthView to specify the date to display.

StartLine is used by TVpDayView to specify the first hour to display. If this value is –1, the start of the day (midnight) is used. This value is also used by TVpContactGrid to specify the first contact and TVpTaskList to specify the first task to display. A value of 0 will start with the first contact or task.

StopLine is used by TVpDayView to specify the last hour to display. If this value is –1, the end of the day will be used (11:59pm).

UseGran is used by the TVpDayView to specify the granularity to use.

If DisplayOnly is on, the display will be adjusted slightly for printing.

### ShowResourceName — property

```
property ShowResourceName : Boolean
```

Default: True

Determines whether the resource name will be displayed at the top of the TaskList.

### TaskHeadAttributes

```
property TaskHeadAttributes : TVpTaskHeadAttr
```

Defines the attributes of the Tasklist's header area.

The TaskHeadAttributes contains the font and background color that is used to paint the TaskList's header.

### TaskIndex — property

```
property TaskIndex : Integer
```

Contains the index of the currently selected task.

# TVpTaskDisplayOptions Class

The TVpTaskDisplayOptions class contains properties that determine how the tasks are displayed in the task list. It also clusters these properties together in the object inspector.

## Hierarchy

TPersistent (VCL)

TVpTaskDisplayOptions (VPTaskList)

## Properties

| | | |
|---|---|---|
| CheckColor | DueDateFormat | ShowAll |
| CheckStyle | NormalColor | ShowCompletedTasks |
| CompletedColor | OverdueColor | ShowDueDate |

### CheckColor — property

```
property CheckColor : TColor
```

Determines the color of the event check marks.

### CheckStyle — property

```
property CheckStyle : TVpCheckStyle
```



Determines the style of the event checks.

The options are an 'X' or a check mark.

### CompletedColor — property

```
property CompletedColor : TColor
```

Determines the color of completed tasks.

By default, completed tasks are colored gray and drawn with a line through them. This is where the color can be changed.

### DueDateFormat — property

```
property DueDateFormat : string
```

Determines the format in which the DueDate is formatted.

### NormalColor — property

```
property NormalColor : TColor
```

Determines the color in which normal, incomplete tasks are drawn.

### OverdueColor — property

```
property OverdueColor : TColor
```

Determines the color in which overdue items are drawn.

## ShowAll — property

```
property ShowAll : Boolean
```

Determines whether to show all tasks.

If ShowAll is True, all tasks will be shown regardless of their status.

The task list will only show up to the maximum number of visible tasks as determined by the TaskList's MaxVisibleTasks property.

See also: TVpTaskList.MaxVisibleTasks

## ShowCompletedTasks — property

```
property ShowCompletedTasks : Boolean
```

Determines whether to show completed tasks.

If ShowCompletedTasks is set to True, tasks will continue to be displayed in the list after they are marked completed. They will only be displayed in the TaskList during the day they were completed. They will disappear from the list on the next day.

If ShowCompletedTasks is set to False, the tasks will disappear immediately when they are checked.

## ShowDueDate — property

```
property ShowDueDate : Boolean
```

Determines whether the TaskList displays the task's DueDate along with the task.

If ShowDueDate is False, the task will only display its description. If ShowDueDate is True, the due date will be displayed along with the description.

# TVpBaseDialog Class

The TVpBaseDialog is the ancestor to all of the Visual PlanIt dialog components. It defines the dialog box placement properties, the connection to the DataStore component, and the Version property.

## Hierarchy

TComponent (VCL)

❶ TVpComponent (VPBase) . . . . . 28

TVpBaseDialog (VPDlg)

## Properties

DataStore

Placement

❶ Version

## Methods

Execute

## Reference Section

### DataStore — property

```
property DataStore : TVpCustomDataStore
```

Provides access to the Visual PlanIt data.

The DataStore is the component that manages all of the Visual PlanIt data.

See “TVpCustomDataStore Class” on page 92 for more information.

### Execute — virtual abstract method

```
function Execute : Boolean; virtual; abstract;
```

Executes the dialog box.

This is a virtual, abstract method that isn’t implemented in the base class. It must be implemented by each descendant class.

### Placement — property

```
property Placement : TVpDialogPlacement
```

Determines how the dialog box will be placed on the screen.

See the “TVpDialogPlacement Class” on page 48.

### Version — read-only property

```
property Version : string
```

Displays the library version in the Object Inspector.

The Version property also has a specialized design-time property editor that displays the Visual PlanIt about box when the ellipsis button is clicked.

# TVpResourceEditDialog Component

The Resource Edit Dialog is used to add and modify resources. It descends from the TVpBaseDialog and simply implements the standard dialog box interface.

See the “TVpBaseDialog Class” on page 288 for more information.

## Hierarchy

TComponent (VCL)

5

TVpComponent (VPBase) . . . . . 28

TVpBaseDialog (VPDlg) . . . . . 288

TVpResourceEditDialog (VPResEditDlg)

# TVpEventEditDialog Component

The Visual PlanIt DayView includes the full-featured event-editing dialog box. It is available by default via the DayView, WeekView, and MonthView components. It is also available as a component. It should be sufficient for most users however, if you wish to provide your own dialog box, see the TVpDayView's OnOwnerEditEvent event on page 257 for information on how to substitute the default dialog box with your own.

The TVpEventEditDialog descends from the TVpBaseDialog and simply implements the standard dialog box interface.

See the "TVpBaseDialog Class" on page 288 for more information.



## Hierarchy

TComponent (VCL)

- TVpComponent (VPBase) . . . . . 28
  - TVpBaseDialog (VPDlg) . . . . . 288
    - TVpEventEditDialog (VPEvntEditDlg)

# TVpContactEditDialog Component

The Visual PlanIt ContactGrid includes the full-featured contact-editing dialog box. It is available by default via the ContactGrid component and as a dialog component installed on the palette. It should be sufficient for most users however, if you wish to provide your own dialog box see the ContactGrid's OnOwnerEditContact event on page 272 for information on how to substitute the default dialog box with your own.

The TVpContactEditDialog descends from the TVpBaseDialog and simply implements the standard dialog box interface.



See the "TVpBaseDialog Class" on page 288 for more information.

## Hierarchy

TComponent (VCL)

- TVpComponent (VPBase) . . . . . 28
  - TVpBaseDialog (VPDlg) . . . . . 288
    - TVpContactEditDialog (VPContactEditDlg)

# TVpTaskEditDialog Component

The Visual PlanIt TaskList includes the full-featured task-editing dialog box. It is available by default via the TaskList component and as a dialog component installed on the palette. It should be sufficient for most users however, if you wish to provide your own dialog box see the TaskList's OnOwnerEditTask event on page 281 for information on how to substitute the default dialog box with your own.

The TVpTaskEditDialog descends from the TVpBaseDialog and simply implements the standard dialog box interface.

See the "TVpBaseDialog Class" on page 288 for more information.



## Hierarchy

TComponent (VCL)

- TVpComponent (VPBase) . . . . . 28
  - TVpBaseDialog (VPDlg) . . . . . 288
    - TVpTaskEditDialog (VPTaskEditDlg)

# TVpNotificationDialog Component

The Visual PlanIt Custom DataStore includes the full-featured event alert dialog box. It is available by default via CustomDataStore descendant components and as a dialog component installed on the palette. It should be sufficient for most users however, if you wish to provide your own dialog box see the CustomDataStore's OnAlert event on page 95 for information on how to substitute the default dialog box with your own.

The TVpNotificationDialog descends from the TVpBaseDialog and simply implements the standard dialog box interface.



See the "TVpBaseDialog Class" on page 288 for more information.

## Hierarchy

TComponent (VCL)

TVpComponent (VPBase) . . . . . 28

TVpBaseDialog (VPDlg) . . . . . 288

TVpNotificationDialog (VPAlarmDlg)

# TVpPrintPreviewDialog Component

The PrintPreviewDialog component is based on the PrintPreview and PrintFormatCombo components. It allows the user to select a pre-defined print format, and see how that format will appear on the printed page.

## Hierarchy

TComponent (VCL)

TVpComponent (VPBase) . . . . . 28

❶ TVpBaseDialog (VPDlg) . . . . . 288

TVpPrintPreviewDialog (VPPrtPrvDlg)

## Properties

| | | | | | |
|---|---|---|---|---|---|
| | ControlLink | | EndDate | | StartDate |
| ❶ | DataStore | ❶ | Placement | ❶ | Version |

## Methods

Execute

### ControlLink — property

```
property ControlLink : TVpControlLink
```

Associates the print preview dialog box with a TVpControlLink.

The ControlLink property provides the print preview dialog box with the print formats that it needs.



### EndDate — property

```
property EndDate : TDateTime
```

Specifies the last date to be printed.

This is needed to calculate the last page of the print job.

See also: StartDate

### Execute — method

```
function Execute : Boolean;
```

Displays the print preview dialog box.

Execute will display the print preview dialog box. The user can select different print formats and step through the pages.

Execute will return True when the user selects Ok or Print. It will return False when the user cancels.

### StartDate — property

```
property StartDate : TDateTime
```

Specifies the starting date of the print job.

See also: EndDate

# Chapter 6: Visual PlanIt Example Applications

Visual PlanIt includes a number of example applications that will help you get up to speed with the product. These applications will change and grow over time, and may not be exactly as presented in the manual. Due to this fact, the printed documentation only has a cursory explanation of the example applications. For more detailed information about how to use the Visual PlanIt components, see the tutorial section starting on page 24.

For the latest information on the example applications, please see the Readme.hlp file in the Visual PlanIt installation directory.

The main example application is the Scheduler. The Scheduler is a full-blown personal information manager style application, based completely on Visual PlanIt components. It is located in the Examples directory.

Other examples are small pieces of the Scheduler, broken out into simpler, smaller pieces for the purpose of demonstrating specific tasks.

# Scheduler

The Scheduler application demonstrates how to implement a full-blown personal information manager, using the Visual PlanIt components. It demonstrates how to use the DayView, WeekView, MonthView, TaskList, ContactGrid, ResourceCombo, and Calendar components in conjunction with a BDEDataStore and ControlLink.

The Scheduler application uses the BDE database engine by default, along with a set of Paradox tables located in the Visual PlanIt\Examples\Data directory.

Run Scheduler and select from the list of available resources in the ResourceCombo.

Scroll the display date, by clicking the spinner button at the top of the DayView, or selecting a date in the calendar.



Add and modify the events by either single-clicking, or double-clicking the events in the DayView.

Modify contacts by either clicking or double-clicking in the Contacts list.

Modify the tasks by selecting the task in the task list, or add a new task by double-clicking anywhere in the list.

Notice how all of the components are date-synchronized by selecting a date in the calendar, or by scrolling the date in the DayView? That is the work of the ControlLink component.

# DayView Demo

The DayView application is a very simple application demonstrating only a single DayView component functioning with a BDEDataStore.

## ContactList Demo

The ContactList demo is a simple application showing the use of a single ContactList component, in conjunction with a BDEDataStore.

## TaskList Demo

The TaskList demo is a simple application showing the use of a single TaskList component, in conjunction with a BDEDataStore.

## Printer Demo

The PrinterDemo shows the basics of printing Visual PlanIt components. For more detailed information on using the Visual PlanIt printing subsystem, see the tutorial section on page 24.

Other example applications may be introduced into the product as it evolves. Please see the Readme.hlp file in the Visual PlanIt installation directory for the latest information on the example applications.

# Identifier Index

## A

Active 47, 116, 177, 253
ActiveFolder 202
ActiveItem 202
ActiveRow 237
Add 44
AddContact 141
AddDefaultVariables 76
AddEvent 121
AddResource 113
Address 145
AddTask 133
AddVariable 76
AfterEdit 237, 255, 269, 279
AfterEnter 31
AfterExit 31
AlarmAdv 126
AlarmAdvType 126
AlarmSet 127
AlarmWavPath 127
AlertDisplayed 128
AliasName 104
AllDayEvent 128
AllowIncDec 230
AllowRearrange 202
AnalogOptions 177
AutoConnect 93
AutoCreate 93
AutoCreateAlias 104
AutoScaledPaintToCanvas 237

## B

BackColor 189
BackgroundColor 203
BackgroundImage 203
BackgroundMethod 203
BarColor 269
BarWidth 269
BatchUpdate 121, 133, 142
BeforeEdit 238, 255, 269, 279
BeginUpdate 204
BgColor 173
bmXxx 203
BorderColor 221
Bordered 277
BorderStyle 221
BottomMargin 77
Brush 55
ButtonHeight 204

## C

Caption 60, 62, 173, 215, 218
Category 128, 145
Category (0 - 9) 36
CategoryColorMap 93
CategoryIndex 38
cdoXxx 195
Changed 40, 116, 129, 137, 145
ChangeVariable 77
CheckColor 286
CheckStyle 286
City 145
ClearContacts 142
ClearEvents 122
ClearResources 113
ClearTasks 133
ClearVariables 77
ClockFace 184
ClockMode 178
cmXxx 177, 178
Collection 51, 62, 71
Color 38, 238, 249, 250, 251, 255, 262, 270, 277, 279

Colors 192
Columns 174
ColumnWidth 270
Company 146
Complete 137
CompletedColor 286
CompletedOn 137
Connected 94
ContactHeadAttributes 270
ContactIndex 270
Contacts 116
ContactsDirty 116
ContactsList 142
Container 215
ControlLink 33, 34, 221, 227, 235, 270, 296
Count 44, 114, 134, 143
CountByDay 134
Country 146
CreatedOn 137
CreateTable 104
CurFormat 78
CurPage 221
CustInterval 129
Custom1 146
Custom2 147
Custom3 147
Custom4 147
cvXxx 166

## D

DataStore 33, 235, 271, 289
Date 94, 192, 230, 238, 255, 262
DateFormat 192
DateLabelFormat 239, 255, 262
Day 192
DayBuffer 94, 106
DayEnd 78
DayHeadAttributes 255, 262
DayInc 71
DayIncUnits 71
DayNameStyle 263
DayNameWidth 192
DayNumberFont 263
DayOffset 62
DayOffsetUnits 63
DayStart 78
DefaultEventSound 94
DefaultTopHour 239
DeleteActiveContact 271
DeleteActiveEvent 239, 256
DeleteActiveTask 279
DeleteContact 143
Deleted 117, 129, 138, 148
DeleteEvent 122
DeleteTask 134
DeleteVariable 78
Description 38, 72, 117, 129, 138, 218
DestPrinter 222
Details 138
dfXxx 192
DigitalOptions 178
DisplayMode 179
DisplayOptions 280
DisplayText 42
dmXxx 179
DrawHeader 192
DrawingStyle 204, 222, 240, 256, 263, 271, 280
DrawMarks 184
dsXxx 205, 222, 240, 256, 263, 271, 280
dtXxx 233
DueDate 138
DueDateFormat 286
duXxx 63, 71

## E

ElapsedDays 179
ElapsedHours 179
ElapsedMinutes 180
ElapsedSeconds 180

ElapsedSecondsTotal 180
ElementName 63
Elements 72
EMail 148
EnableEventTimer 95
EndDate 222, 296
EndTime 130, 155
EndUpdate 205
Epoch 230
EventCount 122
EventCountByDay 123
EventDayStyle 263
EventFont 256, 264
EventsByDate 123
EventsDirty 117
Execute 289, 296

## F

Find 79
First 134, 143
FirstByDay 134
FirstName 148
FirstPage 222
FolderCollection 205
FolderCount 206
Folders 206
Font 60, 249, 250, 277
ForceCentury 230
ForceUpdate 223
FormatDate 231
FormatName 72

## G

GetColor 36
GetContact 143
GetControlType 33, 240, 256, 264, 271, 280
GetDependentList 166
GetEditorCaption 44
GetEvent 123
GetFolderAt 206
GetItemAt 206
GetLastPrintLine 33
GetName 36
GetOwner 44
GetResource 114
GetTask 135
Granularity 79, 240
Grid 277
grXxx 79, 196, 237, 244, 259, 266, 274, 283
GutterWidth 240

## H

HaveContactGrid 79
HaveDate 79
HaveTaskList 80
HaveVariable 80
HeadAttributes 240
HeaderFont 256
Height 49, 63
Holiday 47, 253
HourFont 251
HourHandColor 184
HourHandLength 185
HourHandWidth 185
HourOffset 181
HourToLine 241

## I

IconIndex 218
IconRect 219
IconSize 215
Images 207
imXxx 65, 82
Inactive 47, 253
IncDay 193
IncludeWeekends 241
IncMonth 193
IncYear 193

InsertFolder 207
InsertItem 207
InvalidateItem 208
IsFirstPage 223
IsLastPage 223
isXxx 215
itCalendar 64
Item 44
ItemByName 44
ItemCollection 216
ItemCount 216
ItemFont 208
ItemIndex 117, 130, 139
Items 53, 69, 74, 216
ItemType 64
itXxx 256, 280

## K

KBNavigation 264

## L

LabelRect 219
Last 135, 143
LastByDay 135
LastContact 80
LastName 148
LastPage 223
LastTask 80
LEDOnColor 189
Left 49, 65
LeftMargin 81, 224
LineColor 241, 257, 264, 280
LineCount 241
LinkHandler 34, 193, 242, 257, 265, 272, 281
Load 95, 106
LoadFromFile 81
LoginPrompt 107
LookupVariable 82

## M

MarginUnits 82, 224
MaxVisibleTasks 281
Measurement 65
MinuteFont 251
MinuteHandColor 185
MinuteHandLength 185
MinuteHandWidth 185
MinuteOffset 181
Month 193
mpXxx 49

## N

Name 42, 51
NextPage 224
neXxx 166
NormalColor 286
Note 130, 149
Notes 118
Notify 166
NotifyLinked 83
NumDays 242

## O

OffColor 174
OffDayColor 265
OffPageColor 224
OnAlert 95
OnChange 193
OnChanged 44
OnColor 174
OnConnect 97, 107
OnCountdownDone 181
OnDisconnect 97, 107
OnDragDrop 208
OnDragOver 209
OnDrawDate 194
OnDrawItem 194

OnFolderChange 209
OnFolderChanged 210
OnFolderClick 210
OnGetDate 231
OnGetDateEnabled 194
OnGetEditorCaption 45
OnGetHighlight 194
OnGetVariable 166
OnHourChange 182
OnItemClick 210
OnItemSelected 45
OnMinuteChange 182
OnMouseOverItem 211
OnMouseWheel 31
OnOwnerEditContact 272
OnOwnerEditEvent 242, 257
OnOwnerEditTask 281
OnPageChange 224
OnPageEnd 167
OnPageStart 167
OnSecondChange 182
OnSetDate 231
Options 195
OverdueColor 286
Owner 114, 118, 124, 130, 135, 139, 155, 249, 251
OwnerDrawCells 243, 265
OwnerDrawColHeader 243
OwnerDrawRowHeader 243

## P

PageColor 224
PaintToCanvas 55, 60, 195, 244, 259, 265, 273, 282
PaintToCanvasRect 83
Params 107
ParentForm 45
Pen 55
Phone1 149
Phone2 149
Phone3 149
Phone4 150
Phone5 150
PhoneType1 150
PhoneType2 151
PhoneType3 151
PhoneType4 152
PhoneType5 152
Placement 289
PlayEventSounds 97
PlaySounds 211
PopupCalColors 231
PopupCalFont 232
PopupCalHeight 232
PopupCalWidth 232
Position 49, 152
PostContacts 98, 107
PostEvents 98, 107
PostResources 98, 107
PostTasks 99, 108
PreviousFolder 211
PreviousItem 211
PrevPage 225
Print 83
Printer 168
PrintFormats 84

## R

raXxx 55, 66, 237, 244, 259, 265, 273, 274, 282, 283
ReadOnly 34, 45, 108, 232
RecordID 130, 139, 153
RefreshContacts 99
RefreshEvents 99
RefreshResource 99
RefreshTasks 100
RemoveFolder 212
RemoveItem 212
RemoveResource 114
RenameFolder 212

RenameItem 212
RenderPage 84
RenderToCanvas 34, 196, 244, 259, 266, 274, 283
RepeatCode 131
RequiredFields 232
Resource 100
ResourceID 100, 108, 118, 245, 275
Resources 100
rfXxx 232
RightMargin 86, 225
Rotation 66
RowHeadAttributes 245
Rows 174

## S

SaveToFile 86
Schedule 118
ScrollDelta 213
SecondHandColor 186
SecondHandLength 186
SecondHandWidth 186
SelectedDayColor 266
SetResourceByName 101
SetToday 197
Shape 56, 66
ShowAll 287
ShowCompletedTasks 287
ShowDueDate 287
ShowEvents 267
ShowEventTime 260, 267
ShowResourceName 246, 275, 284
ShowSecondHand 186
ShowSeconds 189
Size 174
SnoozeTime 131
SolidHands 187
Sort 124, 135, 143
SoundAlias 213
StartDate 225, 296
StartTime 131, 155
State 153

## T

TaskHeadAttributes 284
TaskIndex 284
Tasks 119
TasksDirty 119
Time 182
TimeFormat 246, 260, 267
TimeRange 101
TimeSlotColors 246
Title 153
TodayString 233
Top 49, 66
TopHour 246
TopLine 247
TopMargin 87, 225
TriggerOnGetVariable 168
TriggerOnPageEnd 168
TriggerOnPageStart 168
TVpAttributeItem 50
TVpAttributes 52
TVpBaseDialog 288
TVpBDEDataStore 103
TVpCalendar 190
TVpCategoryColorMap 35
TVpCategoryInfo 37
TVpCHAttributes 248
TVpClock 176
TVpClockDigitalOptions 188
TVpCollection 43
TVpCollectionItem 41
TVpComponent 28
TVpContact 144
TVpContactEditDialog 292
TVpContactGrid 268
TVpContactHeadAttr 276
TVpContacts 140
TVpContainerList 46

TVpControlLink 165
TVpCustomClock 175
TVpCustomControl 30
TVpCustomDataStore 92
TVpCustomDateEdit 228
TVpCustomDBDataStore 102
TVpCustomLEDLabel 169
TVpCustomNavBar 200
TVpDateEdit 229
TVpDayView 236
TVpDialogPlacement 48
TVpEvent 125
TVpEventEditDialog 291
TVpFont 39
TVpHandOptions 183
TVpLEDLabel 170
TVpLinkableControl 32
TVpMonthView 261
TVpNavBar 201
TVpNavBtnItem 217
TVpNavFolder 214
TVpNotificationDialog 294
TVpPrintCaption 57
TVpPrinter 75
TVpPrintFormat 73
TVpPrintFormatComboBox 226
TVpPrintFormatElement 68
TVpPrintFormatElementItem 61
TVpPrintFormatItem 70
TVpPrintPreview 220
TVpPrintPreviewDialog 295
TVpPrintShape 54
TVpResource 115
TVpResourceCombo 234
TVpResourceEditDialog 290
TVpResources 112
TVpRHAttributes 250
TVpSchedule 120
TVpTask 136
TVpTaskDisplayOptions 285
TVpTaskEditDialog 293
TVpTaskList 278
TVpTasks 132
TVpTimeRange 154
TVpTimeSlotColor 46, 252
TVpWeekView 254

## U

UpdateDateVariables 87
UpdateItems 227
ustXxx 56

## V

ValidFormat 87
Value 51
Version 29, 31, 42, 174, 235, 289
VisibleLines 247

## W

Weekday 47, 253
Weekend 47, 253
WeekStarts 197, 233
WeekStartsOn 260, 267
Width 49, 67

## Y

Year 197

## Z

zfXxx 225
Zip 153
ZoomFactor 225

# Subject Index

## A


accessing
- base dialog data 289
- category identifier 128
- column header background color 249
- column header font 249
- column header owner 249
- contact grid data 271
- contact header owner 277
- contact's Address field 145
- contact's Category field 145
- contact's City field 145
- contact's Company field 146
- contact's Country field 146
- contact's Custom1 field 146
- contact's Custom2 field 147
- contact's Custom3 field 147
- contact's Custom4 field 147
- contact's EMail address field 148
- contact's fifth telephone field 150
- contact's first telephone field 149
- contact's FirstName field 148
- contact's fourth telephone field 150
- contact's LastName field 148
- contact's Note field 149
- contact's PhoneType1 field 150
- contact's PhoneType2 field 151
- contact's PhoneType3 field 151
- contact's PhoneType4 field 152
- contact's PhoneType5 field 152
- contact's Position field 152
- contact's RecordID field 153
- contact's second telephone field 149
- contact's State field 153
- contact's third telephone field 149
- contact's Title field 153
- contact's zip code field 153

accessing (continued)
- contacts list 116
- control's value 230
- data 33, 235
- description field 117
- end date and time 130
- end time 155
- event owner 130
- event schedule 118
- event's description field 129
- event's Notes field 130
- first contact in list 143
- folder's component container 215
- items by index 216
- last item in contacts list 143
- list of resources 100
- Notes field 118
- object's internal list 142
- RecordID 130
- resource ID 100, 118
- row header background color 251
- row header hour font 251
- row header minute font 251
- row header owner 251
- selected resource 100
- start date and time 131
- task's ID 139
- tasks list 119
- TimeRange's owner 155
- TimeRange's start time 155

active time block, defining color 47

adding
- contacts 141
- date variables 87
- default variables 76
- event to schedule 121
- events 117
- folder to navigation bar 207

adding (continued)
item into folder 207
new item to collection 44
print caption variable 76
resource contacts 116
resources 113
tasks 119, 133
Address field, accessing 145
adjusting, date of print element 62, 63
alarm
sounding 126, 127
alert, displaying 128
alias name, setting 104
alias, creating 104
all-day event, specifying 128
allowing changes to collection 45
Analog clock 176
attribute item
collection 51
name 51
value 51
attribute, items 53

## B

background color
column header 249
contact grid 270
defining active 253
defining for contact header 277
defining for display 173
defining for task list 279
defining holiday 253
defining in week view 255
defining inactive 253
defining weekday 253
defining weekend 253
determining for month view 262
row header 251
setting 224
background color (continued)
specifying color of navigation bar 203
specifying for day view 238
background image, displaying 203
background painting method 203
bar
defining width 269
determining color 269
base dialog
accessing data 289
determining placement 289
executing 289
version 289
batch
updating 121, 133
updating contacts 142
BDEDataStore
connecting 107
creating alias 104
creating tables 104
disconnecting 107
limiting to read-only 108
loading data 106
posting contact changes 107
posting event changes 107
posting resource changes 107
posting task changes 108
setting alias name 104
setting resource's ID number 108
specifying connection parameters 107
specifying login prompt 107
specifying number of days in buffer 106
border
defining 277
determining style 221
specifying color 221
specifying style 222
brush, defining 55
buffer
determining number of days in 94
specifying number of days in 106

# C


calendar
- changing date 193
- determining date format 192
- determining day name width 192
- determining header 192
- determining highlight color 194
- determining if ADate can be selected 194
- determining options 195
- drawing date 194
- drawing day 194
- increment by months 193
- increment by year 193
- increment date by day 193
- link handler 193
- painting to canvas 195
- providing read/write access to date 192
- rendering to canvas 196
- retrieving day portion 192
- retrieving month 193
- retrieving year 197
- setting date to Now 197
- setting first day of week 197
- specifying color 192

caption
- defining 62
- LED label 173
- navigation button item 218
- navigation folder 215
- printing to canvas 60

caption variable, evaluating 166

category color map
- grouping name and color 36
- retrieving color value 36
- retrieving name 36

Category field, accessing 145

category identifer, accessing 128

category information
- color-code 38
- index 38
- name 38

cell, drawing 243, 265

changes, allowing 45

changing
- active folder 210
- calendar's date 193
- contact 145
- current page 224
- date 232
- event 129
- folder 209
- font 40
- item in collection 44
- print caption variable 77
- print formats 83
- resource 116
- task 137
- value of displayed date 231

check mark, determining color 286

City field, accessing 145

clicking
- folder button 210
- item 210

clock
- analog 176
- analog display 179
- current time 182
- determining digital apperance 178
- determining display mode 179
- determining if shown in real time 177
- digital display 176, 179
- grouping analog setting properties 177
- hour notification 182
- hour offset 181
- minute changes 182
- minute offset 181
- mode 178
- real-time 176
- retrieving elapsed days 179
- retrieving elapsed hours 179
- retrieving elapsed minutes 180
- retrieving elapsed seconds 180
- retrieving total elapsed seconds 180

clock (continued)
second change 182
static time 176
timer reaches zero 181
clock digital options
determining background color 189
determining flash separator 189
determining lit color 189
determining un-lit color 189
displaying seconds 189
clock face bitmap 184
collection
adding new item 44
allowing changes 45
calling GetEditorCaption 45
changing item 44
retrieving count 44
retrieving editor caption 44
retrieving item 44
retrieving item by name 44
retrieving owner 44
retrieving parent form reference 45
selecting item 45
specifying 71
collection item
displaying text 42
name 42
version 42
color
determining 194
retrieving value 36
specifying 231, 257
specifying sections of calendar 192
color-code of category 38
column header attributes
accessing background color 249
accessing font 249
accessing owner 249
column header, drawing 243
column, defining width 270
Company field, accessing 146
completing task 137
component version 29
connecting
database 107
to database 93, 97
contact
accessing Address field 145
accessing Category field 145
accessing City field 145
accessing Company field 146
accessing Country field 146
accessing Custom1 field 146
accessing Custom2 field 147
accessing Custom3 field 147
accessing Custom4 field 147
accessing EMail address field 148
accessing fifth telephone field 150
accessing first telephone field 149
accessing FirstName field 148
accessing fourth telephone field 150
accessing LastName field 148
accessing list 116
accessing Note field 149
accessing PhoneType1 field 150
accessing PhoneType2 field 151
accessing PhoneType3 field 151
accessing PhoneType4 field 152
accessing PhoneType5 field 152
accessing Position field 152
accessing RecordID field 153
accessing second telephone field 149
accessing State field 153
accessing third telephone field 149
accessing Title field 153
accessing zip code field 153
adding 116
changing 145
deleting 148
posting changes to 107
contact grid
accessing data 271
control link 270
defining bar width 269

contact grid (continued)
defining column width 270
defining header attributes 270
deleting active contact 271
determining background color 270
determining bar color 269
determining drawing style 271
displaying name at top 275
editing 269
editing item 272
index 270
link handler 272
painting to canvas 273
rendering to canvas 274
resource ID 275
retrieving control type 271
specifying component 79
contact header attributes
accessing owner 277
defining background color 277
defining border 277
defining font 277
ContactList, example application 299
contacts
accessing first item in list 143
accessing last item in list 143
accessing object's internal list 142
adding 141
deleting 143
refreshing 99
removing from list 142
retrieving 143
retrieving number in list 143
sorting 143
table structure 160
updating batch 142
container, accessing 215
control link
communicating changes 166
defining print formats 168
evaluating caption variables 166
finishing printing 167
control link (continued)
printing 167
triggering OnGetVariable event 168
triggering OnPageEnd event 168
triggering OnPageStart event 168
control type
retrieving for contact grid 271
retrieving for day view 240
retrieving for task list 280
controlling
database connection state 94
controlling display options 280
converting hour to line number 241
Country field, accessing 146
creating
BDE alias 104
database tables 93
tables 104
custom control
getting focus 31
losing focus 31
scrolling mouse wheel 31
version 31
Custom1 field, accessing 146
Custom2 field, accessing 147
Custom3 field, accessing 147
Custom4 field, accessing 147

## D

data
accessing 33, 235, 271, 289
loading 106
loading into memory 95
data edit
decrementing date 230
data storage overview 90
data store
.wav file path name 94
accessing list of resources 100
accessing resource ID 100

data store (continued)
accessing selected resource 100
connecting 93
connecting to database 97
controlling connection state 94
creating tables 93
determining number of days in buffer 94
disconnecting from database 97
enabling event timer 95
iterating through contacts 98
iterating through events 98
iterating through resources 98
iterating through tasks 99
loading data 95
mapping color to category names 93
playing event sounds 97
posting changes to database 98, 99
refreshing contacts 99
refreshing events 99
refreshing resources 99
refreshing tasks 100
setting resource by name 101
specifying date 94
specifying time range 101
start time 95
database
connecting 107
design 157
disconnecting 107
date
adjusting 62, 63
changing 193, 232
changing value 231
defining in week view 255
determining format 192
displaying 262
drawing 194
formatting 231
modifying 231
providing read/write access to 192
setting to Now 197
specifying 94, 238
date (continued)
specifying components 79
specifying last to be printed 222, 296
specifying required 232
specifying start of print job 225, 296
task completed 137
task created 137
date edit
accessing control's value 230
changing date 232
changing value of displayed date 231
determining today string 233
determining which day begins week 233
displaying in four digit years 230
formatting date 231
incrementing date 230
modifying date 231
specifying pop-up calendar colors 231
specifying pop-up calendar font 232
specifying pop-up calendar height 232
specifying pop-up calendar width 232
specifying required fields 232
specifying starting year 230
date label format, determining 239, 255, 262
day
defining header attributes 262
determining name width 192
determining number displayed 242
determining which begins week 233
drawing 194
incrementing 193
retrieving 192
setting first of week 197
specifying color 266
specifying end 78
specifying name style 263
specifying number displayed 71
specifying number font 263
specifying start 78
day view
converting hour to line number 241
defining granularity 240

day view (continued)
defining top hour 239
deleting currently selected event 239
determining date label format 239
determining gutter width 240
determining header attributes 240
determining number of days displayed 242
determining time format 246
displaying name at top 246
drawing cells 243
drawing column header 243
drawing row header 243
editing 237, 238
editing item 242
including weekends 241
link handler 242
number of currently visible lines 247
painting to canvas 237, 244
rendering to canvas 244
resource ID 245
retrieving control type 240
retrieving line count 241
setting hour 246
setting top line 247
specifying background color 238
specifying date 238
specifying drawing style 240
specifying highlighted row 237
specifying line color 241
specifying row header attributes 245
specifying time line background color 246
DayView, example application 298
DCU files 7
decrementing date 230
defining
active background color 253
active time block color 47
bar width 269
border 277
brush used in drawing 55
caption 62
contact grid column width 270
defining (continued)
contact grid header attributes 270
contact header background color 277
contact header font 277
day header attributes 262
font used to render caption 60
granularity 240
height of dialog form 49
holiday background color 253
holiday color 47
inactive background color 253
inactive time block color 47
left side placement of dialog form 49
month view's display date 262
pen used in drawing 55
placement of dialog form 49
print formats 84, 168
print shape 56
printer variable 80
shape 66
task list attributes 284
task list background color 279
top hour 239
top placement of dialog form 49
week view background color 255
week view display date 255
weekday background color 253
weekday color 47
weekend background color 253
weekend color 47
width placement of dialog form 49
deleting
active contact 271
active event 256
contact 148
contacts 143
currently selected event 239
currently selected task 279
event 129
print caption variable 78
record 117
resource 114

deleting (continued)
schedule events 122
task from database 138
tasks 134
demonstration application
ContactList 299
DayView 298
Printer 299
TaskList 299
demonstration programs 9
description
accessing field 117, 129
print format 72
task 138
design conventions 12
details, task 138
determining
background color 189
bar color 269
border style 221
calendar header 192
calendar options 195
check mark color 286
clock digital apperance 178
clock display mode 179
clock hand drawing style 187
completed tasks color 286
contact grid background color 270
date format 192
date label format 239, 255, 262
day name width 192
day week starts 267
dialog box placement on screen 289
drawing style 271
DueDate format 286
flash separator 189
gutter width 240
header attributes 240
highlight color 194
hour hand color 184, 189
hour hand width 185
icon size 215
determining (continued)
if clock is shown in real time 177
if items can be rearranged 202
if page is first in print job 223
if page is last in print job 223
if resource available to user 116
length of hour hand 185
lit color 189
maximum number of visible tasks 281
minute hand color 185
minute hand length 185
minute hand width 185
month view background color 262
normal task color 286
number of days displayed 242
number of days in buffer 94
overdue task color 286
repeat interval 129
repeat pattern 131
scroll speed 213
second hand color 186
second hand length 186
second hand width 186
time format 246, 260, 267
today string 233
units of time 126
un-lit color 189
whether to include weekends 241
whether to show all tasks 287
which day begins week 233
dialog form
defining height 49
defining left side placement 49
defining placement 49
defining top placement 49
defining width placement 49
dialog placement
defining height of form 49
defining left side placement 49
defining placement of form 49
defining top placement 49
defining width placement 49

disabling repainting of control 204
disconnecting
  database 107
  from database 97
display
  controlling options 280
  determining mode 179
  specifying size 174
displaying
  alert 128
  background image of navigation bar 203
  clock face 184
  contact name 275
  event time 267
  month view events 267
  name at top of day view 246
  second hand 186
  seconds 189
  task list resource name 284
  text 42
  time 260
  version 174
  year in four digit years 230
dragging item 209
drawing
  calendar date 194
  calendar day 194
  cells 243, 265
  column header 243
  minute and hour marks on clock face 184
  row header 243
drawing style
  navigation bar 204
  specifying 240, 271
  specifying in month view 263
  specifying used in task list 280
due date of task 138

# E

editing
  contact 269
  contact item 272
  day view 237, 238
  item 242
  linkable control 34
  task 281
  task list 279
  week view 255
  week view item 257
editor caption, retrieving 44
element
  specifying in print format 72
  specifying width 67
EMail address field, accessing 148
enabling
  event timer 95
  repainting of control 205
end date, accessing 130
end time, accessing 130, 155
evaluating caption variables 166
event
  .wav file path name 127
  accessing category identifier 128
  accessing description field 129
  accessing end date and time 130
  accessing event's Notes field 130
  accessing owner 130
  accessing RecordID 130
  accessing schedule 118
  accessing start date and time 131
  adding to schedule 121
  changing 129
  deleting 122, 129
  deleting active 256

event (continued)
deleting currently selected 239
determining repeat interval 129
determining repeat pattern 131
determining units of time 126
displaying alert 128
displaying for month view 267
displaying time 267
enabling timer 95
index 130
playing sounds 97
posting changes to 107
refreshing 99
retrieving 123
retrieving count by day 123
retrieving number 122
sorting 124
sounding alarm 126, 127
specifying all-day 128
specifying day style 263
specifying font 264
specifying wait time 131
table structure 159
example application
ContactList 299
DayView 298
overview 297
Printer 299
TaskList 299
Example Application, Scheduler 298
example programs 9
executing
dialog box 289
print preview dialog box 296

## F

fifth telephone field, accessing 150
finding
display icon 219
folder's label 219
print caption variable 79
first telephone field, accessing 149
FirstName field, accessing 148
FlashSeparator 189
folder
adding to navigation bar 207
changing 209
changing active 210
clicking button 210
collection 205
count 206
finding label 219
index number 202
index of previously selected 211
removing 212
renaming 212
retrieving index 206
specifying tab height 204
Folder Editor 199
font
accessing row header 251
changing 40
column header 249
defining 60
defining for contact header 277
item 208
specifying 232
specifying in week view 256

forcing
  item update 227
  print preview update 223
forcing redraw of item 208
format
  changing 83
  defining 84
  loading from file 81
  painting to canvas 83
  saving to file 86
  validating 87
formatting date 231
fourth telephone field, accessing 150

## G

going to first page of print job 222
going to last page of print job 223
going to next page of print job 224
going to previous page of print job 225
granularity, specifying 79
gridline color
  specifying for month view 264
  specifying for task list 280
grouping
  analog setting properties 177
  category name and color 36
gutter width, determining 240

## H

hand options
  clock face 184
  determining drawing style 187
  determining hour hand color 184
  determining hour hand width 185
  determining minute hand color 185
  determining minute hand length 185
  determining minute hand width 185
hand options (continued)
  determining second hand color 186
  determining second hand length 186
  determining second hand width 186
  displaying second hand 186
  drawing minute and hour marks 184
hardware requirements 6
header
  defining attributes 270
  determining 192
  determining attributes 240
  drawing 243
  specifying attributes 245
  specifying day attributes 255
help, installing 9
holiday, defining color 47
hour
  accessing font 251
  converting to line number 241
  defining top 239
  setting top 246
hour hand
  determining color 184, 189
  determining length 185
  determining width 185
hour notification 182

## I

icon
  determining size 215
  finding 219
image list
  index 218
  navigation bar 207
inactive time block, defining color 47
incrementing
  day 193
  month 193
  year 193

incrementing date 230
index
  category 38
  contact grid 270
  event 130
  resource 117
  task 139
  task list 284
index number
  active folder 202
  active item 202
in-place editing, NavBar 198
Installation
  C++Builder 8
installing
  Delphi 7
  for multiple compilers 7
  integrated help 9
  packages 8
item
  accessing by index 216
  adding into folder 207
  adding to collection 44
  attributes 53
  clicking 210
  count 216
  dragging 209
  editing 242, 257, 272
  font 208
  forcing redraw 208
  forcing update 227
  index number 202
  index of previously selected 211
  list 216
  moving mouse cursor over 211
  removing 212
  renaming 212
  retrieving by name 44
  retrieving from collection 44
  retrieving index 206
  selecting in collection 45
iterating
  through database contacts 98
  through database events 98
  through database resources 98
  through database tasks 99

## L

last contact, specifying 80
LastName field, accessing 148
LED label
  caption 173
  displaying version 174
  specifying background color 173
  specifying color of lit segments 174
  specifying color of un-lit segments 174
  specifying display size 174
  specifying number of characters in display 174
  specifying number of rows 174
LEDLabel
  background color 173
  LED segment map 171
  Non-displaying characters 173
LEDOffColor 189
line
  number of currently visible 247
line color
  specifying 241
line count, retrieving 241
link handler, day view 242
linkable control
  accessing data 33
  editing 34
  printing subsystem 33
  rendering control 34
  stopping control 33
  synchronizing 33, 34
linkable controls, base class 32

list
accessing 142
accessing last item in 143
retrieving number of contacts in 143
loading
data 106
data into memory 95
print formats from file 81
login, specifying prompt 107

## M

mapping color to category names 93
minute font, accessing 251
minute hand
determining color 185
determining length 185
determining width 185
mode, clock 178
modifying
date 231
events 117
resource contacts 116
tasks 119
month
incrementing 193
retrieving 193
month view
defining day header attributes 262
defining display date 262
determining background color 262
determining background color of off days 265
determining date label format 262
determining day week starts 267
determining time format 267
displaying event time 267
displaying events 267
drawing cells 265
link handler 265
painting to canvas 265
month view (continued)
rendering to canvas 266
retrieving control type 264
specifying active day color 266
specifying day name style 263
specifying day number font 263
specifying drawing style 263
specifying event day style 263
specifying event font 264
specifying gridline color 264
using keyboard to navigate 264
moving mouse cursor over item 211

## N

name
attribute item 51
category 38
collection item 42
displaying at top of day view 246
displaying at top of grid 275
print format 72
print format element 63
retrieving 36
naming conventions 11
NavBar
large icons 198
NavBar component overview 198
navigation bar
adding folder 207
adding item into folder 207
background painting method 203
changing active folder 210
changing folder 209
clicking folder button 210
clicking item 210
determining if items can be rearranged 202
determining scroll speed 213
disabling repainting of control 204
displaying background image 203
drag and drop 208

navigation bar (continued)
dragging item 209
drawing style 204
enabling repainting of control 205
folder count 206
forcing redraw of item 208
image list 207
index number of active folder 202
index number of active item 202
item font 208
list of folders 205
moving mouse cursor over item 211
playing sounds 211
playing system sounds 213
previously selected folder index 211
previously selected item index 211
property containing folders 206
removing folder 212
removing item 212
renaming folder 212
renaming item 212
retrieving folder index 206
retrieving item index 206
sound scheme 213
specifying background color 203
specifying folder tab height 204
navigation button item
caption 218
finding display icon 219
finding folder's label 219
image list index 218
navigation folder
accessing container 215
accessing items by index 216
caption 215
determining icon size 215
item count 216
item list 216
specifying text description 218
newsgroups, technical support 13
Note field, accessing 149
Notes field, accessing 118, 130

## O

on-line help 11
option, determining 195
overview
Schedule 120
Tasks 132
overview, resources 112
owner
accessing 130, 155, 251, 277
column header 249
retrieving 44, 114, 118, 124, 135

## P

packages, installing 8
page
changing current 224
determining if first in print job 223
determining if last in print job 223
going to first 222
going to last 223
going to next 224
going to previous 225
setting color 224
specifying current displayed 221
painting
calendar to canvas 195
contact grid to canvas 273
day view to canvas 237, 244
month view to canvas 265
print caption to canvas 60
print format to canvas 83
shape to canvas 55
task list to canvas 282
to canvas 84
week view to canvas 259
parameter, specifying 107
parent form, retrieving 45
pen, defining 55
PhoneType1 field, accessing 150

PhoneType2 field, accessing 151
PhoneType3 field, accessing 151
PhoneType4 field, accessing 152
PhoneType5 field, accessing 152
playing event sounds 97
playing sounds 211
playing system sounds 213
populating TList 123
Position field, accessing 152
positioning
  left side of element 65
  top of element 66
posting
  changes to database 98, 99
  contact changes 107
  event changes 107
  resource changes 107
  task changes 108
print caption
  defining font 60
  painting to canvas 60
  printing caption 60
  variables 57
print format
  items 74
  specifying 78
print format combo box
  control link 227
  forcing item update 227
print format element item
  adjusting date 62, 63
  defining caption 62
  defining shape 66
  name 63
  positioning left side 65
  positioning top 66
  rotating 66
  specifying height 63
  specifying measurement units 65
  specifying print format 62
  specifying type 64
  specifying width 67
print format element, items 69
print format item
  description 72
  name 72
  specifying collection 71
  specifying DayInc units 71
  specifying elements 72
  specifying number of days displayed 71
print format, defining 168
print preview
  associating with a printer 222
  changing current page 224
  control link 221
  determining border style 221
  determining if first page 223
  determining if last page 223
  forcing update 223
  going to first page 222
  going to last page 223
  going to next page 224
  going to previous page 225
  setting background color 224
  setting page color 224
  specifying border color 221
  specifying border style 222
  specifying current page 221
  specifying end date 222
  specifying starting date 225
print preview dialog
  control link 296
  executing 296
  specifying last date to be printed 296
  specifying start date 296
print shape
  defining 56
  defining brush used 55
  defining pen used 55
  painting to canvas 55
printer
  adding date variables 87
  adding default variables 76
  adding variable 76

associating with print preview 222
changing caption variable 77
changing formats 83
defining formats 84
defining variable 80
deleting variable 78
finding variable 79
loading formats from file 81
painting format to canvas 83
painting to canvas 84
removing variables 77
retrieving variable value 82
saving formats to file 86
specifying bottom margin 77
specifying contact grid components 79
specifying date components 79
specifying end of day 78
specifying granularity 79
specifying last contact 80
specifying last task 80
specifying left margin 81
specifying margin units 82
specifying print format 78
specifying right margin 86
specifying start of day 78
specifying task list components 80
specifying top margin 87
validating format 87
Printer, example application 299
printing
caption 60
finishing 167
starting 167
printing subsystem 33
providing read/write access to date 192

## R

raXxx 195, 196
read-only, limiting data store to 108
record, deleting 117
RecordID field, accessing 153
RecordID, accessing 130
refreshing
contacts 99
events 99
resources 99
tasks 100
removing
contacts from list 142
folder 212
item 212
resources 113
tasks from list 133
variables 77
renaming
folder 212
item 212
rendering
contact grid to canvas 274
control 34
day view to canvas 244
month view to canvas 266
task list to canvas 283
week view to canvas 259
repeat interval, determining 129
repeat pattern, determining 131
requirements, hardware and software 6
resource
accessing 100
accessing contacts list 116
accessing description field 117
accessing event schedule 118
accessing ID 118
accessing Notes field 118
accessing tasks list 119
adding contacts 116
adding events 117
adding tasks 119
changing 116
deleting record 117
determining if available to user 116
index 117

resource (continued)
modifying contacts 116
modifying events 117
modifying tasks 119
posting changes to 107
retrieving owner 118
setting by name 101
resource combo
accessing data 235
control link 235
displaying version 235
resource ID
accessing 100, 118
contact grid 275
day view 245
setting number 108
resource name, displaying at top of task list 284
resources
accessing list of 100
adding 113
deleting 114
overview 112
refreshing 99
removing 113
retrieving number 114
retrieving owner 114
retrieving reference 114
table structure 158
retrieving
category name 36
collection count 44
collection owner 44
color value 36
contact grid control type 271
contacts 143
control type 240, 256
day portion of calendar's date 192
editor caption 44
retrieving (continued)
elapsed days 179
elapsed hours 179
elapsed minutes 180
elapsed seconds 180
event 123
event count by day 123
folder index 206
item by name 44
item from collection 44
item index 206
line count 241
month 193
month view control type 264
number of contacts in list 143
number of events 122
number of resources in list 114
number of tasks in list 134
parent form reference 45
reference to first task in list 134
resource owner 114, 118
resource reference 114
schedule owner 124
task list control type 280
task owner 135, 139
total elapsed seconds 180
variable value 82
year 197
Returns 143
rotating element 66
row
specifying active 237
specifying number in display 174
row header attributes
accessing background color 251
accessing hour font 251
accessing minute font 251
accessing owner 251
specifying 245
row header, drawing 243

## S


saving print formats to file 86
Schedule
  Batch update 121
schedule
  adding event 121
  deleting events 122
  marking deleted flag True 122
  populating TList 123
  retrieving event 123
  retrieving event count by day 123
  retrieving number of events 122
  retrieving owner 124
  sorting events 124
  updating batch 121
Schedule overview 120
scroll speed, determining 213
second hand
  determining color 186
  determining length 186
  determining width 186
  displaying 186
second telephone field, accessing 149
selecting item in collection 45
setting
  BDE database alias name 104
  calendar's date to Now 197
  control background color 224
  first day of week 197
  hour displayed in top line 246
  page color 224
  resource by name 101
  resource's ID number 108
  top line 247
setup 7
shape, defining 66
shape, painting to canvas 55
showing
  task DueDate 287
software requirements 6
sorting
  contacts 143
  schedule events 124
  tasks 135
sound, playing 213
sounding
  alarm 126
  event alarm 127
specifying
  active day color 266
  all-day event 128
  background color 173
  border color 221
  border style 222
  bottom margin 77
  collection 71
  color of lit segments 174
  color of sections of calendar 192
  color of un-lit segments 174
  contact grid components 79
  current page displayed 221
  database connection parameters 107
  date 94, 238
  date components 79
  day header attributes 255
  day name style 263
  day number font 263
  day view background color 238
  day view line color 241
  DayInc units 71
  display size 174
  drawing style 240
  elements in print format 72
  end date to be printed 222
  end of day 78
  event day style 263
  event font 264
  folder tab height 204
  gridline color 257
  height of element 63
  highlighted row 237

specifying (continued)
last contact 80
last date to be printed 296
last printer task 80
left margin 81
login prompt 107
margin units 82
measurement units 65
month view drawing style 263
month view gridline color 264
navigation bar background color 203
number of characters in LED display 174
number of days displayed 71
number of days in buffer 106
number of rows in display 174
pop-up calendar colors 231
pop-up calendar font 232
pop-up calendar height 232
pop-up calendar width 232
print format 62, 78
print format granularity 79
required date fields 232
right margin 86
row header attributes 245
start date of print job 296
start of day 78
starting date of print job 225
starting year of 100-year period 230
task list components 80
task list drawing style 280
task list gridline color 280
text description 218
time format 260
time line background color 246
time range 101
top margin 87
type of print element 64
wait time 131
week view drawing style 256
week view font 256
week view header font 256
width of element 67
start date, accessing 131
start time
accessing 131, 155
alerting 95
State field, accessing 153
synchronizing controls 33, 34

## T

table, creating 104
task
accessing ID 139
adding 119
changing 137
completing 137
date completed 137
date created 137
deleting currently selected 279
deleting from database 138
description 138
details 138
determining completed color 286
due date 138
editing 281
index 139
modifying 119
posting changes to 108
retrieving owner 139
specifying last 80
task display options
determining check mark color 286
determining completed tasks color 286
determining DueDate format 286
determining normal color 286
determining overdue color 286
determining whether to show all 287
showing completed 287
showing DueDate 287
task list
controlling display options 280
defining attributes 284

task list (continued)
defining background color 279
deleting currently selected task 279
displaying resource name 284
editing 279
editing task 281
index 284
link handler 281
maximum number of visible tasks 281
painting to canvas 282
rendering to canvas 283
retrieving control type 280
specifying component 80
specifying drawing style 280
specifying gridline color 280
TaskList
example application 299
tasks
adding 133
counting by day 134
deleting 134
refreshing 100
removing from list 133
retrieving number in list 134
retrieving owner 135
retrieving reference specified by Index 135
retrieving reference to first 134
retrieving reference to first task of specified date 134
retrieving reference to last task in list 135
retrieving reference to last task of specified date 135
sorting 135
table structure 162
updating batch 133
tasks list, accessing 119
Tasks, overview 132
technical support 13
third telephone field, accessing 149
time format, determining 246, 260, 267
time line, specifying background color 246
time range
accessing end time 155
accessing owner 155
accessing start time 155
specifying 101
time slot color
defining active background color 253
defining active time block color 47
defining holiday background color 253
defining holiday color 47
defining inactive background color 253
defining inactive time block color 47
defining weekday background color 253
defining weekday color 47
defining weekend background color 253
defining weekend color 47
time, displaying 260
TimeGrid
Event Edit Dialog 293
Title field, accessing 153
TList, populating 123
top line, setting 247
triggering
OnGetVariable event 168
OnPageEnd event 168
OnPageStart event 168

## U

updating
batch 121, 133
contacts 142
using keyboard to navigate month view 264

## V

validating print format 87
value
attribute item 51
retrieving 82

variable
adding 76, 87
adding default 76
changing 77
defining 80
deleting 78
finding 79
removing 77
retrieving value 82
version 31
base dialog 289
collection item 42
component 29
LED label 174
resource combo 235

## W

wait time, specifying 131
wav file path name 94, 127
week view
defining background color 255
defining display date 255
deleting active event 256
determining date label format 255
determining day week starts on 260
determining time format 260
displaying time 260
week view (continued)
editing 255
editing item 257
link handler 257
painting to canvas 259
rendering to canvas 259
retrieving control type 256
specifying day header attributes 255
specifying drawing style 256
specifying font 256
specifying gridline color 257
specifying header font 256
specifying time format 260
weekday, defining color 47
weekend
defining color 47
determining whether to include 241

## Y

year
incrementing 193
retrieving 197
specifying starting 230

## Z

zip code field, accessing 153

The TurboPower family of tools—
Winners of 6 *Delphi Informant* Readers' Choice Awards for 2001! Company of the Year in 2000 and 2001.

*For over fifteen years you've depended on TurboPower to provide the best tools and libraries for your development. Now try FlashFiler and Orpheus—two of TurboPower's best selling products risk free. Both are compatible with Borland & Microsoft development environments, and both are backed by expert support and our 60-day money back guarantee.*

DEVELOP, DEBUG, OPTIMIZE
FROM START TO FINISH, TURBOPOWER
HELPS YOU BUILD YOUR BEST

## FLASHFILER 2™

*FlashFiler 2 is the high performance database engine that's ideal for storing Visual PlanIt data. It's amazingly easy to use, easy to license, and easy to deploy no matter how big your development project is. And if you're looking for a database with advanced features that will grow with you, there's no better choice than FlashFiler 2.*

## ORPHEUS 4™

*Make the first impression count with new Orpheus 4, the award-winning user interface construction kit from TurboPower Software Company. Your customers will feel comfortable the minute your application starts, resulting in them spending less time learning how to use your program and more time getting to know what makes it special.*

Try the full range of TurboPower products. Download free Trial-Run Editions from our Web site.

**www.turbopower.com**

Visual PlanIt requires Microsoft Windows (XP, 2000, Me, NT, or 9X), and Borland Delphi 4 and above, or C++Builder 4 and above.

TURBOPOWER®
Software Company

©2002, TurboPower Software Co.