Gravitybox Schedule.NET

Capabilities and Technologies Overview

Gravitybox Software

Gravitybox Software
PO Box 2462
Alpharetta, GA 30023 USA
Voice mail & Fax : (+1) 866.GRAVBOX
Email : feedback@gravitybox.com
Website : http://www.gravitybox.com

Contact Information
For inquiries about redistribution of this document or the GbSchedule.NET component please contact Gravitybox Software.

Gravitybox Software
PO Box 2462
Alpharetta, GA 30023

Voice-Mail and FAX (Toll-Free in the US):
866.GRAVBOX (866.472.8269)

Website:
http://www.gravitybox.com

Email:
Please send questions, comments, or bugs to feedback@gravitybox.com

Downloads
The Gravitybox Schedule ActiveX component and other Gravitybox products can be downloaded from the Gravitybox website at http://www.gravitybox.com.

The direct download URL for GbSchedule.NET is http://www.gravitybox.com/download/Schedule.NET.msi.

Source Code Note
You may notice that the examples of this book are not complete. The relevant code has been listed for discussion; however it is not necessary to list thousands of lines of code that you are never going to read. Only the relevant code snippets have been included. I hope that other writers in the future take this same methodology.

Table Of Contents

Part I
Getting Started

The most likely way for the world to be destroyed, most experts agree, is by accident. That's where we come in; we're computer professionals. We cause accidents.

-Unknown

Enthusiasm is contagious -- and so is the lack of it.

-Unknown

Introduction

Why Gravitybox Schedule.NET?

Gravitybox Schedule is a third-party tool written in response the to the vacuum of third-party scheduling components. Few companies have any type of scheduling software available. The selection that is available is incomplete at best and non-functional at worst. Many of the existing applications deal mainly with scheduling in a very narrow context, such as employee scheduling or task scheduling. Many of these applications display the schedules as Gantt graphs. Most people who have created schedules by hand are familiar with the appointment book model or grid display. This appears to be an obvious way to display scheduled information; however the lack of software that displays information in these formats is noticeably missing.

In fill the void, Gravitybox offers a general-purpose software component that may be used to display scheduled information of almost any type, in many, different formats. The component will display information in the common grid format with time and dates on opposing axes. It will also allow you to transpose the axes, as well as specify the increments on each one. It can display an arbitrarily large schedule, allowing for years at a time to be view and scrolled. There are conflict resolution routines that can determine if a change will cause a conflict with other appointments. A warning may also be given the user in this situation in his native language, if need be.

GbSchedule.NET will not only display the information in a grid with multiple configurations but it may also be used to display scheduled information in a month view format as well, popularized by MS-Outlook. This view allows the user to see a month at a glance. For those who are accustomed to viewing information in an appointment book format, they will not be disappointed because the GbSchedule.NET offers this view as well.

Most of the functionality available may be used with little or no coding. There are many properties that may be set to configure the behavior of the schedule. These properties control almost all of the functionality and behavior of the software. Customizations may be added in code. Opportunities are given to override the default behavior of the schedule by using the provided events. There are “Before” and “After” procedures for most events. For example there is a BeforeMove and AfterMove event. The first may be used to cancel an appointment move. The latter maybe used for some sort of confirmation code that an object has been moved.

In all, the component is truly general purpose. It allows the developer full control over the display and behavior of every aspect of its existence. The control was designed to allow maximum flexibility over almost any type of scheduling scenario.

The need for GbSchedule.NET

In today's world, there is a need for a general-purpose scheduling utility. More applications are requiring at least some scheduling as part of their functionality. Many developers dreading this task save it for last only to realize that it is a much larger task than at first they thought. A high-quality schedule inside of your application is an application in and of itself. You could spend your entire allotted time developing just this part of your application.

This is where the Gravitybox Schedule component comes in. All of the complicated scheduling routines have been incorporated into it, including conflict resolution and warning. Also added is the ScheduleProperties control that allows you to create in minutes a customized property sheet for appointments. Almost all default behaviors may be overridden with customizations. And most importantly, very little code is needed to perform even complex tasks. The schedule may be dropped on a form, be fully drag-drop enabled, moving appointments across windows or even different programs with file loading and saving functionality in 30 lines of code or less!

All of this creates a component that has a small learning curve. Most of the properties are self-explanatory. An intermediate developer can read though the properties and look at examples and be an expert in no time. This allows you to add first-rate scheduling to your application with ease.

Goals of GbSchedule.NET
The goals of GbSchedule.NET are several. These are the guidelines on which the software was developed.

To create a high-quality component
To have a small learning curve
To have the maximum functionality with a minimal of code
To add user requests in a timely fashion

Real-World Uses

When evaluating a software product, its usefulness is its most important attribute. No matter the time and care expended to create a software program, if it does not solve any real-world problem, it is useless. GbSchedule.NET has been designed to resolve many scheduling situations. The original goal of the component was to produce a module that could be dropped into an application to create a scheduling application for a doctor's office. This did prove an ambitious goal in and of itself. The necessary functionality has been added to accomplish this task.

As stated, this component may be used to create applications that have significant, differing scheduling purposes. The first use is an office application. Some of the program features follow. It should graphically display clients that are scheduled to come into the office on any given day. It should be able to list services provided and cost. It should be able to assign an appointment to a category and to a provider. The developer should define the categories and providers collections. It should have print functionality so that the user can crate a hard copy of his schedule. There are other more advanced features of coarse but these criteria can be used to create a basic office application. This is probably the most used scenario for application development using the GbSchedule.NET product.

There are as many uses as developers can conceive. I know of one developer using it to schedule airplanes at an airport. I think this was a very small airport, as I do hope that the FAA has its own proprietary program written for this express purpose. One TV studio has created a schedule of TV programs. It lists all shows and allows engineers to see their air schedule at a glance. And as if to test its flexibility, some have used the component to keep their schedule for a year at time. After all it can be scrolled through any length of time.

Chapter 1

Collections, Lists and Objects

This heart of the schedule is its collections. They hold all of the information necessary to display a schedule. All of the rooms, categories, providers, icons, and appointments are held in collections. Collections are nothing more than a group of similar objects. All collections in the GbSchedule.NET software are derived from the class “System.Collections.CollectionBase”. They provide a common interface with certain properties and methods. For example all collections have methods: Add, Contains, IndexOf, Remove and RemoveAt.

Lists differ from collections in that lists holds references to previously created objects and collection actually create objects. Most objects present in the GbSchedule.NET object modal are not directly creatable. The collections serve as object factories as well as repositories. You may add an existing object to a list but may not create a new object with the list. For example, to create a Category you would use the CategoryCollection object located on the schedule. Each Appointment object has a CategoryList. This list is not used to create new Category objects but is used to store references to categories that exist in the schedule’s CategoryCollection. In this way you may associate existing categories with an appointment. If a category is removed from the schedule’s CategoryCollection, the Category object is also removed from all lists in which it resides since the object does not exist after its removal. The same rules hold for all lists. Once an object is removed from its parent collection, it is also removed from all lists that contain it.

Collections

Overview

Collections in the schedule component are the repositories of data. These form the backbone of the schedule since their only purpose is to create and hold data. The schedule is not bound to any external data source and only displays the data in its collections.

AppointmentCollection

All appointments for a schedule are held in the AppointmentCollection. The collection itself is used to create appointments as appointments may not be created directly with the “new” keyword. You must use the “Add” method of the collection to specify a number of parameters and it will create, add, and return the new object.

RoomCollection

The RoomCollection defines the rooms that are available on the schedule. Rooms may be used in the display a schedule’s axis. These objects are used when the user needs to view room specific information on one of the axes. Room objects are not directly creatable and as such must be created using the “Add” method of a schedule’s RoomCollection object. Room objects may be associated with appointments. In this way an Appointment object knows to render itself in a specific room if the schedule is being displayed with rooms on one of the axes. Rooms are optional and are not required for a schedule to display correctly as long as the Viewmode does not include rooms.

CategoryCollection
The CategoryCollection defines the categories that are available to group appointments. These objects are for grouping as there is no way currently to display categories on either axis. You may associate any number of categories to an appointment using an appointment’s CategoryList object. Any number of Category objects may be added to this list.

ProviderCollection
The ProviderCollection defines the providers that are available on the schedule. Providers may be used in the display a schedule’s axis. These objects are used when the user needs to view provider specific information on one of the axes. Provider objects are not directly creatable and as such must be created using the “Add” method of a schedule’s ProviderCollection object. Provider objects may be associated with appointments using an appointment’s ProviderList object. Any number of Provider objects may be added to this list. In this way an Appointment object knows to render itself for a specific Provider if the schedule is displaying itself with providers on one of the axes. Providers are optional and are not required for a schedule to display correctly as long as the Viewmode does not include providers.

ScheduleAreaCollection
This collection is used to defined no-drop areas and other colored areas. The schedule has an object named NoDropAreaCollection that defines all areas of a schedule where appointments are not allowed. The user cannot schedule appointments in these areas. Another predefined collection is the ColoredAreaCollection, which is used to color areas of a schedule but where appointments are allowed.

A very useful method of this object is “IsAreaOverlap” method, which allows you to query whether a defined area overlaps any area in the collection. For example, you might define a no-drop area from 11am through 1pm on January 1. You can then use one of the overloaded versions of this method to determine if a specified date and time overlaps the no-drop zone.

```
Schedule1.NoDropAreaCollection.Add(Color.Red, _
     #1/1/2004#, #11:00:00 AM#, 120)
Dim b As Boolean = Schedule1.NoDropAreaCollection.ToList.IsAreaOverlap( _
     #1/1/2004#, #10:00:00 AM#, 70)
```

In the code above the variable “b“ would be set to true because the area 10am – 11:10am does overlap 11am-1pm.

Lists

Overview

The lists are complementary to the collections in that they hold the same type of objects but allow much more functionality on its contents. There are three operation defined for each list object: Intersect, Union, and Subtract. These actions may be used in any combination to construct arbitrarily complex queries. Given two or more lists you may union their contents to get a list of all objects in both lists. Moreover, you may subtract one list from the other list to get all objects in first that are not in the second. Finally you can find only common objects to both lists using the Intersect method. Each collection that has a matching list allows you to create a list of all objects by calling its ToList method. However the real potential comes from the ability to work with subsets of collections.

AppointmentList

This is the most important list in the namespace. It allows you to query a group of appointments. The AppointmentCollection allows you to form some simple, pre-made queries using the GetBlocked and GetUnBlocked methods. Each returns a list of appointments. However the advanced functionality of the schedule is rooted in the AppointmentCollection’s Find method. It allows to you ask many different questions of the collection and get an AppointmentList in return. You may ask as many questions as you wish each time getting a list of appointments for each. You may then create a master lists by using one or more of the defined list actions: Union, Subtract, or Intersect.

Using the AppointmentCollection “Find” method, the following code snippet creates three AppointmentList objects. The first retrieves a list of all appointments on January 1, 2004. The second get a list of all appointments with the subject “MySubject”. The third is the union of both lists. We could have just as easily subtracted one list from the other.

```
Dim list1 As AppointmentList
Dim list2 As AppointmentList
Dim list3 As AppointmentList
list1 = Schedule1.AppointmentCollection.Find(#1/1/2004#, #1/1/2004#)
list2 = Schedule1.AppointmentCollection.Find("MySubject")
list3 = list1.Union(list2)
```

Using this simple example you can see the power of lists. It basically allows you to apply set arithmetic to sets of schedule objects.

The AppointmentList has a three other methods of special interest: IsAreaAvailable, IsConflict, and NextAreaAvailable. The IsAreaAvailable method determines whether an appointment is scheduled in a specific schedule area. An area is defined by the set of parameters passed into this method. It is overloaded multiple times to allow almost any region to be queried. If any part of one or more appointments overlaps any portion of the defined area, this method returns false. The IsConflict method allows you to determine if a specified appointment conflicts with any other appointments in the list.

There is an overloaded version of the method that takes two appointment parameters and compares them to each other and no other appointments. The NextAreaAvailable method starts at a specified spot on a schedule and returns the next available schedule area where the specified appointment may be placed outside of any no-drop areas and which does not conflict with any other appointment in the list.

CategoryList

The CategoryList is very similar to the AppointmentList except that it contains Category objects. It allows the same set arithmetic as above. However there is little use in this because there will most likely be few categories in an application and in general categories are just labels that you will apply to appointments.

ProviderList

The ProviderList has the same functionality as the list described above.

The Visibility Object

Overview

When viewing a schedule, you will need to know what is visible. If a particular date, time, room, etc is not visible you will want to scroll it into view programmatically. The Visibility object allows you to query and set display areas of a schedule.

In depth

The Visibility object provides numerous methods to determine if certain aspects of a schedule are in the viewable area. If you need to ensure that a particular date, time, room, provider, or appointment is visible then this object is quite useful. Depending on the viewmode of the schedule you can query the first or total number of columns, rows, dates, etc.

Visibility Properties

FirstVisibleColumn	Determines the first visible column number. The valid values are [0..N-1], where N is the total number of columns. The total number of columns depends on many factors. If columns are dates, then the value is the number of days between the MinDate and MaxDate. If the columns are Rooms then the value is the number of objects in the RoomCollection object.
FirstVisibleDate	Determines the first date that is visible on the schedule. This is the date that is displayed in the top row of left column, depending on the viewmode. If the viewmode does not support dates, the method does not return a valid value.
FirstVisibleProvider	Determines the first provider that is visible on the schedule. This is the provider that is displayed in the top row of left column, depending on the

	viewmode. If the viewmode does not support providers, the method returns -1.
FirstVisibleRoom	Determines the first room that is visible on the schedule. This is the room that is displayed in the top row of left column, depending on the viewmode. If the viewmode does not support rooms, the method returns -1.
FirstVisibleRow	Determines the first visible row number. The valid values are [0..N-1], where N is the total number of rows. The total number of rows depends on many factors. If rows are dates, then the value is the number of days between the MinDate and MaxDate. If the rows are Rooms then the value is the number of objects in the RoomCollection object.
FirstVisibleTime	Determines the first time value that is visible on the schedule. This is the time that is displayed in the top row of left column, depending on the viewmode. If the viewmode does not support times, the method does not return a valid value.
IsDateVisible	Determines if the specified date is visible in the viewing area.
IsProviderVisible	Determines if the specified provider is visible in the viewing area.
IsRoomVisible	Determines if the specified room is visible in the viewing area.
IsTimeRangeVisible	Determines if the specified time range is visible in the viewing area.
IsTimeVisible	Determines if the specified time is visible in the viewing area.
TotalColumnCount	Determines the total number of columns that are currently displayed on the schedule.
TotalHeight	Determines the total height in pixels, including headers, of the schedule. This value does not describe the visible area, but the entire schedule as if it were all displayed at once.
TotalRowCount	Determines the total number of rows that are currently displayed on the schedule.
TotalWidth	Determines the total width in pixels, including headers, of the schedule. This value does not describe the visible area, but the entire schedule as if it were all displayed at once.
VisibleColumns	This method returns the number of complete and partial columns that are visible in the viewing area.

VisibleRows	Overloaded. This method returns the number of complete and partial rows that are visible in the viewing area.

Visibility Methods

GetAppointmentFromCor	Given a set of coordinates this method will return the Appointment object under that spot. If there is no Appointment object at the specified location, null is returned.
GetColumnFromCor	Given a set of coordinates this method will return the column index under that spot. If no column exists at the specified location, the returned value is -1.
GetDateFromCor	Given a set of coordinates this method will return the date under that spot. If no date exists at the specified location, the returned value is not valid.
GetDateFromRowCol	Given a row or column value, this method returns the date at that location. This only applies to viewmodes that show dates. Also the value, row or column, depends on the viewmode as well. If the dates are displayed on top, then the specified value should be the column.
GetProviderFromCor	Given a set of coordinates this method will return the Provider object under that spot. If there is no Provider object at the specified location, null is returned.
GetProviderFromRowCol	Given a row or column value, this method returns the provider at that location. This only applies to viewmodes that show providers. Also the value, row or column, depends on the viewmode as well. If the providers are displayed on top, then the specified value should be the column.
GetRoomFromCor	Given a set of coordinates this method will return the Room object under that spot. If there is no Room object at the specified location, null is returned.
GetRoomFromRowCol	Given a row or column value, this method returns the room at that location. This only applies to viewmodes that show rooms. Also the value, row or column, depends on the viewmode as well. If the rooms are displayed on top, then the specified value should be the column.
GetRowColFromDate	Given a date, this method will return the row or column that displays the specified date. If the dates

	are displayed on top, then the returned value will be the column, otherwise the row.
GetRowColFromTime	Given a time, this method will return the row or column that displays the specified time. If the times are displayed on top, then the returned value will be the column, otherwise the row.
GetRowFromCor	Given a set of coordinates this method will return the row index under that spot. If no row exists at the specified location, the returned value is –1.
GetTimeFromCor	Given a set of coordinates this method will return the time value under that spot. If no time exists at the specified location, the returned value is not valid.
GetTimeFromRowCol	Given a row or column value, this method returns the time value at that location. This only applies to viewmodes that show times. Also the value, row or column, depends on the viewmode as well. If the times are displayed on top, then the specified value should be the column.
ShowAppointment	Given an appointment, this method will scroll the schedule's viewing area to ensure that the appointment is visible. If the appointment is not displayed in the current viewmode, then no scrolling occurs. An example of this would be as follows. The viewmode is showing rooms and times and the specified appointment has no associated Room object. It is not displayed on the schedule at all and as such cannot cause a scroll.
ShowDate	This method will scroll the viewing area to include the specified date if possible. If the specified date is not a valid value, no scrolling occurs. An example of an invalid date would be as follows. The schedule's MinDate and MaxDate are set to Jan 1, 2004 and Jan 10, 2004 respectively. If you tried to display Jan 20, 2004, the schedule cannot possibly show this date, since it is out of range.
ShowProvider	This method will scroll the viewing area to include the specified provider. If the current viewmode does not support providers, this method will not cause any scrolling to occur.
ShowRoom	This method will scroll the viewing area to include the specified room. If the current viewmode does not support rooms, this method will not cause any scrolling to occur.
ShowTime	This method will scroll the viewing area to include

	the specified time value if possible. If the specified time value iss not a valid value, no scrolling occurs. An example of an invalid time would be as follows. The schedule's StartTime is set to 8:00AM and the DayLength is set to 10 hour. This means that the schedule only displays to 6:00PM. If you tried to display 10:00PM, the schedule cannot possibly show this time, since it is out of range.

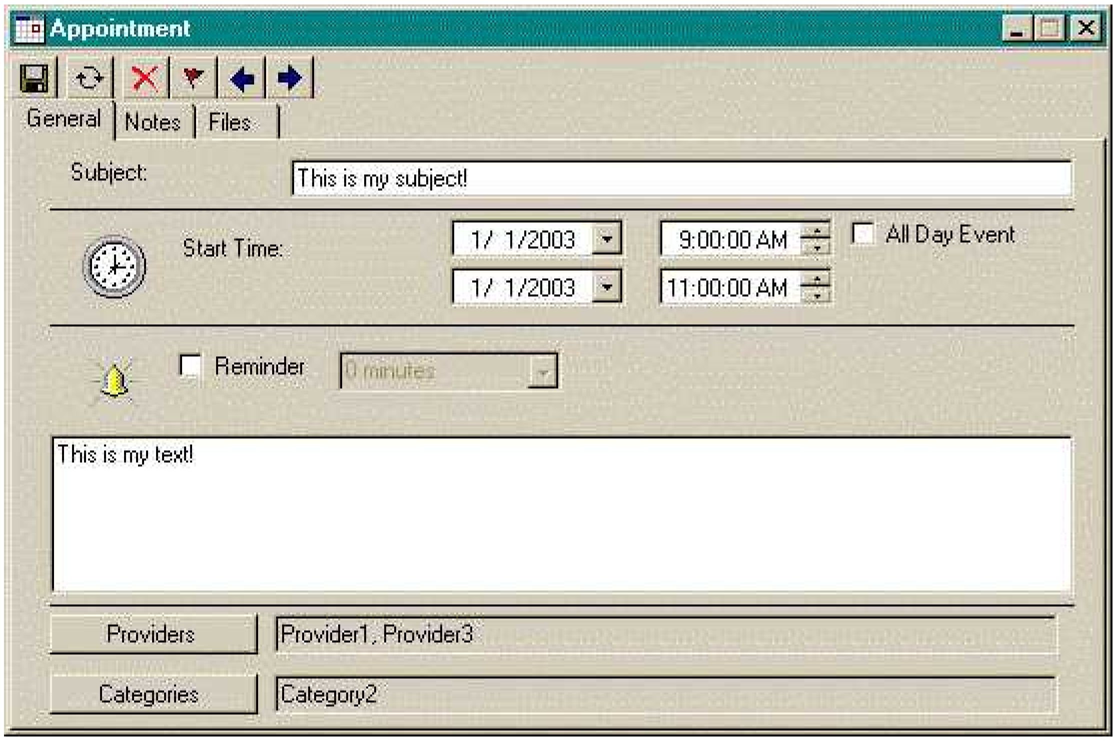
The Dialogs Object

Overview

A Dialogs object provides an interface for displaying dialog screens. GbSchedule.Net has many built-in dialog screens that may be used to display information to the user or prompt the user for information. None of the dialogs must be shown. They are all optional as you may build you own dialog screens to suit your custom application needs.

Appointment Dialog

The appointment dialog is used to display the properties of an appointment by default. The user utilizes this dialog to view and set properties of an appointment. It reconfigures itself based on the schedule's viewmode. For example, if the viewmode does not display room information on either axis, the room associated with the appointment would not be displayed, as it is irrelevant. Also the screen reconfigures itself based on missing information. For example, if the schedule's CategoryCollection is empty, there would be no need in displaying the category list at the bottom of the screen. In this case, the list would not appear.



Subject: This is my subject!

Start Time: 1/ 1/2003 9:00:00 AM

1/ 1/2003 11:00:00 AM

All Day Event

Reminder 0 minutes

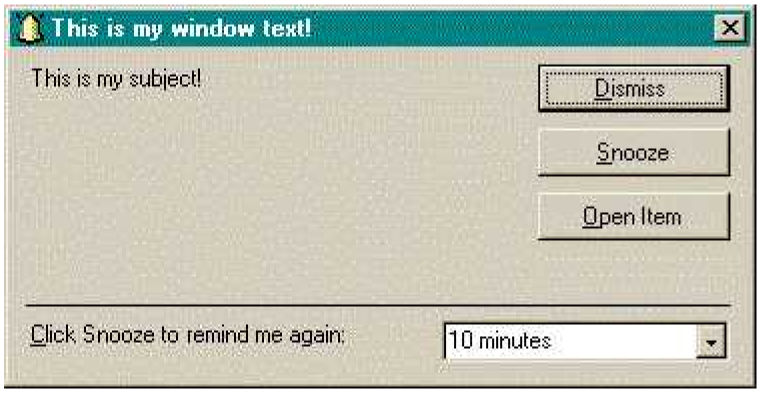
This is my text!

Providers: Provider1, Provider3

Categories: Category2

Alarm Dialog

The alarm dialog is displayed when an appointment comes due. Each appointment has an Alarm object. If this appointment's Alarm object's property IsArmed is set to true, then the alarm is set. When the appointment's date and time match the date and time of the system clock, this dialog will be displayed. The alarm object also has a reminder property. If this property is greater than zero then it will generate a reminder window that many minutes before the appointment comes due. If the reminder value is less than zero then the reminder window will be displayed after the appointment comes due.



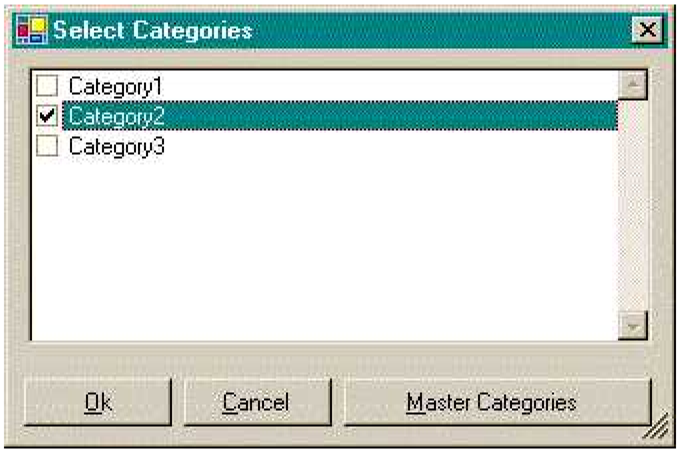
This is my subject!

Click Snooze to remind me again: 10 minutes

Category Dialog

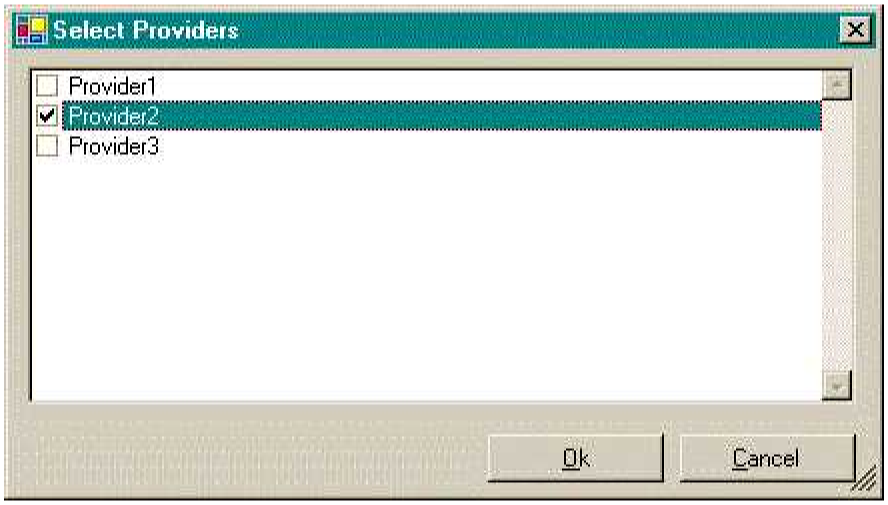
The category dialog is displayed from an appointment's property dialog screen. If the appointment's category list is displayed in the dialog there will be a button labeled

"Categories" that when pressed will launch this window. It allows the user to associated any number of categories with the appointment.



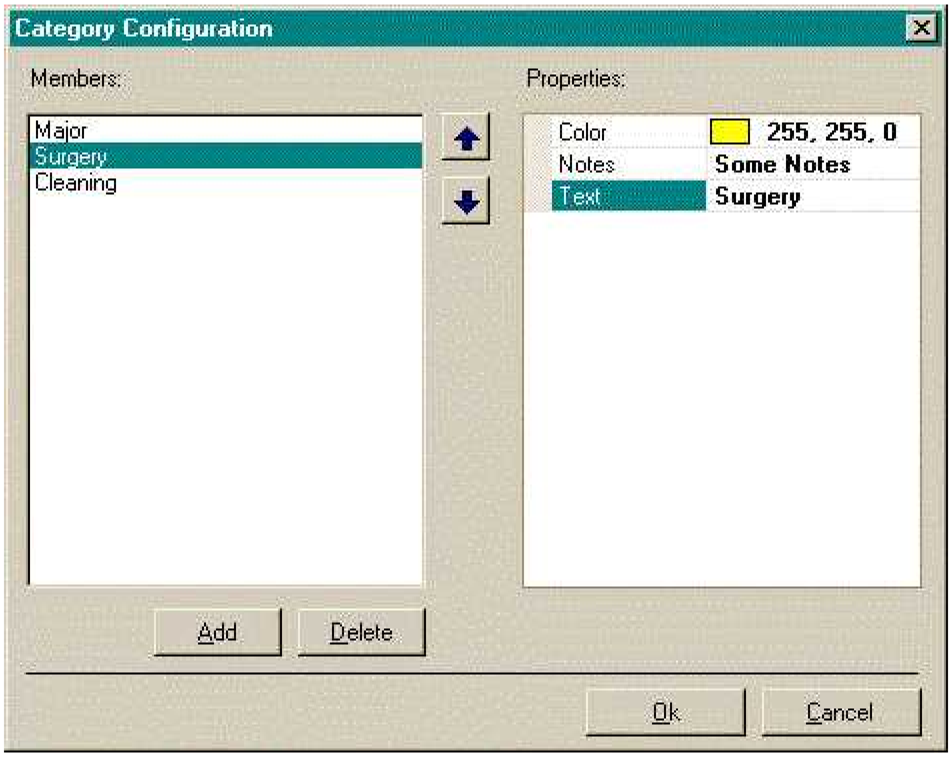
Provider Dialog

The provider dialog is displayed from an appointment's property dialog screen. If the appointment's provider list is displayed in the dialog there will be a button labeled "Providers" that when pressed will launch this window. It allows the user to associate any number of providers with the appointment.



Category Configuration

The category configuration dialog cannot be displayed by the user but can only be called in code. It allows the user to add, edit, remove, and reorder categories from the schedule's CategoryCollection. This allows the user direct access to this collection.

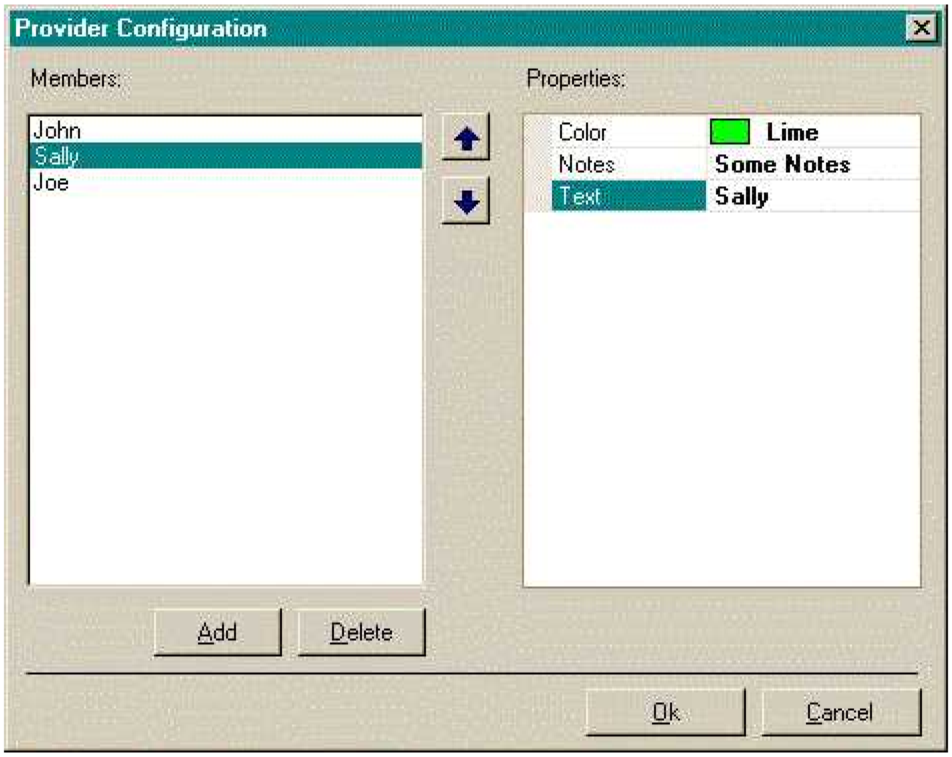


Major
Surgery
Cleaning

Color	🟨 255, 255, 0
Notes	Some Notes
Text	Surgery

Provider Configuration

The provider configuration dialog is just like the category configuration dialog but it edits the ProviderCollection of a schedule.

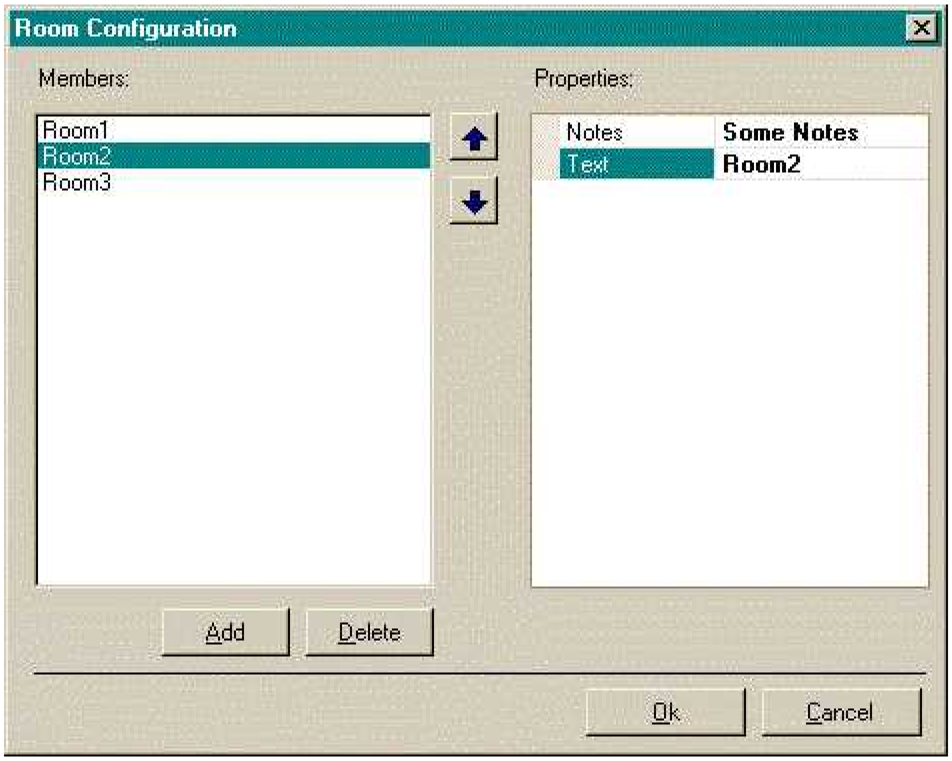


John
Sally
Joe

Color	🟩 Lime
Notes	Some Notes
Text	Sally

Room Configuration

The room configuration dialog is just like the category configuration dialog but it edits the RoomCollection of a schedule.



About Dialog

The about dialog is a screen that displays information about the schedule itself. This will most likely never be used in your commercial application.

Displaying Dialogs from code

Only some of the dialogs may be displayed by users actions; however all of them may be shown programmatically. The schedule has a “Dialogs” object that may be used to do this. There are many “Show…” methods to display dialogs. Below is VB.NET code to display the appointment properties dialog. The code creates an appointment and then displays its property dialog.

```
Dim appointment As Appointment
appointment = Schedule1.AppointmentCollection.Add("", #1/1/2004#, _
                                                  #8:00:00 AM#, 60)
Call Schedule1.Dialogs.ShowPropertyDialog(appointment)
```

All of the dialog methods are overloaded to allow you to specify a configuration object. This object permits you to set many attributes about the appearance and behavior of various dialogs. The following code performs the same action as above but create a configuration object as well. In the configuration the appointment categories are turned off, the dialog is displayed center-screen, and no navigation is allowed within the dialog.

```
Dim appointment As Appointment
Dim dialogSettings As New AppointmentDialogSettings
dialogSettings.StartPosition  =  FormStartPosition.CenterScreen
dialogSettings.AllowCategory = False
dialogSettings.AllowNavigate = False

appointment  =  Schedule1.AppointmentCollection.Add("",  #1/1/2004#, _
                                                   #8:00:00 AM#, 60)
Call   Schedule1.Dialogs.ShowPropertyDialog(appointment)
```

Overriding Dialogs

As stated above some of the dialogs may be displayed through user interaction. Many times the default behavior is fine; however there are times when you want to override the default functionality and override certain aspects of the dialogs or cancel the dialog all together.

There are three dialogs that may be displayed through user interaction: appointment properties, category list, and provider list dialogs. The latter two are called from the appointment property dialog. When the user double clicks on an appointments if starts an edit of the appointment. An edit occurs by displaying an appointment property dialog. Before the dialog is shown the event “BeforePropertyDialog” is raised. Its mouse event parameter has a cancel property that when set to true will cancel the edit. The appointment property dialog is not shown. If you wish to display the dialog but change the default UI then the “DialogSettings” mouse event property is the actual settings object used to configure the dialog. All you need do is change any relevant properties on this object to apply them to the dialog before it is shown.

If either the category or provider button on the appointment property screen is pressed the appropriate dialog is shown. The event “BeforeCategoryListDialog” is called before the category dialog is shown and the “BeforeProviderListDialog” event is called before the provider dialog is shown. Both of these events allow you to cancel and configure the respective dialog screens.

Part II Using GbSchedule.NET

My cup runneth over.

-Psalm xxiii. 5

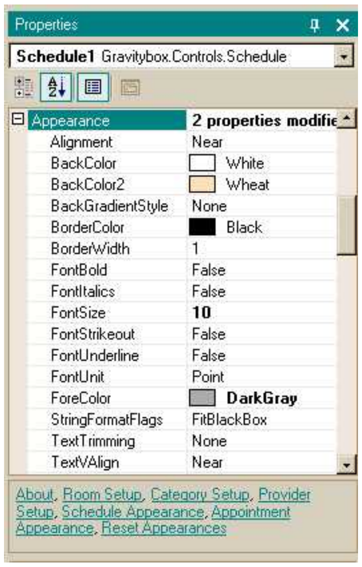
Chapter 2

Schedule Display

Overview

The schedule has many different ways in which it can be displayed. All visible elements have a presentation definition object named Appearance. This allows you to set a common set of properties to change the settings of any object in the package. You may control the look-and-feel of headers, bars, and appointments.

The property grid displayed shows the Appearance object of the schedule. It contains the settings for the back color and gradients. If this object were attached to a schedule area, the font properties would control the text drawn. This object gives you full control over the alignment of text its color and the background on which it is displayed.



Schedule1 Gravitybox.Controls.Schedule

⊟ Appearance	**2 properties modifie**
Alignment	Near
BackColor	⬜ White
BackColor2	🟧 Wheat
BackGradientStyle	None
BorderColor	⬛ Black
BorderWidth	1
FontBold	False
FontItalics	False
FontSize	**10**
FontStrikeout	False
FontUnderline	False
FontUnit	Point
ForeColor	▨ **DarkGray**
StringFormatFlags	FitBlackBox
TextTrimming	None
TextVAlign	Near

About, Room Setup, Category Setup, Provider Setup, Schedule Appearance, Appointment Appearance, Reset Appearances

Display Objects

Overview

There are two headers displayed at all times on a schedule, the column header and row header. These display different types of information based on the Viewmode property, such as date, time, room, or provider. Defined by the header is the actual grid. It houses all appointments inside of a scrollable region. Various colored bars and user-drawn components may be added to enhance the presentation of the schedule.

ColumnHeader and RowHeader

These two objects are of the class type Gravitybox.Objects.Header. They define how the column and row header are displayed. The viewmode determines information displayed in the top and left margins. However these two header objects determine the look of the headers. Headers can hold information about the date, time, room, or provider. This information determines the actual text of each header cell.

AllowResize	Determines if the user may use the mouse to resize the header.
Appearance	Determines if the look-and-feel of the header.
AutoFit	Determines if the header is resized automatically to fit the schedule size. If this is true the AllowResize property is

	ignored.
FrameIncrement	Determines the number of columns or rows that are scrolled at one time. The default is one but this allows you to make the user scroll only N rows or columns at a time.
Size	Determines the size, in pixels, of the header. For column header this value is the width and for row headers it is the height.

NoDropAreaCollection and ColoredAreaCollection
Schedule areas can be defined to present some type of color-coded information to the user. For example, you may wish to define 12pm through 1pm as lunchtime and June 3 as surgery day. You would use the NoDropAreaCollection for the first example and the ColoredAreaCollection for the second. During lunchtime you do not want any appointments to be scheduled. You define this time range and it is colored. If a user tries to move an appointment over this position, it turns to a no-drop icon. In the second example, surgery, you wish to define a day to schedule surgeries. Here you want to color-code the schedule but you do want appointments to be scheduled there.

The following code demonstrates how to add to two examples above.

```
'Define lunchtime for every day
Schedule1.NoDropAreaCollection.Add(Color.Red, #12:00:00 PM#, 60)

'Define surgeries for 6/3/2004
Schedule1.ColoredAreaCollection.Add(Color.LightBlue,  #6/3/2004#)
```

The entire functionality of these two collections lies in their Add methods. Once an object is created it cannot be modified only removed. The Add method is overloaded to allow many different types of areas to be defined. For example, you may wish to define a date, date/time, date/room, room/time, etc. Use the Add method that matches the information need to define your area.

SelectedItems
The SelectedItems list allows multiple appointments to be selected at one time. If the MultiSelect property is set to true, then multiple appointments can be selected at the same time. The user can add appointments to this collection by holding the <CTRL> key and clicking on any number of appointments. You can programmatically select an appointment by adding it to this collection.

ScheduleIcons
This is a fixed list of predefined icons that may be used on the schedule. You may use icons in this list to as appointment icons or place in an appointment header. You do not have to use these icons. They are provided for convenience only. The code below creates an appointment and adds an icon that will be displayed in its left margin.

```
Dim appointment As Appointment
```

```
appointment = Schedule1.AppointmentCollection.Add("", #6/3/2004#, _
                                                     #8:00:00 AM#, 60)
appointment.IconCollection.Add("",   Schedule1.ScheduleIcons.IconInfo)
```

DefaultIcons

The DefaultIcons is also a predefined collection of icons however it differs in that these icons are actually used in the schedule. There are places on the schedule where icons are displayed to provide information. This collection holds all of the icons used by the schedule component. For example, on the toolbar of the appointment properties screen, the icons displayed are stored in the DefautlIcons collection. Also when an appointment is in a recurrence pattern, the recurrence icon used comes form this collection as well.

Display Properties

Overview

There are many properties involved in the display of the schedule. Listed below are some other more important ones. They control how bars and colors are displayed as well as local display settings for data such as time.

AppointmentBar

This object defines the bar that may be displayed in the left margin of an appointment. The bar may describe either the provider or category of an appointment. Since appointments may actually have more than one category or provider assigned to it, only the first one is used to draw the bar. The setting of the bar is by default none. No bar is displayed in the appointment margin at all. There is also a fourth setting: user-drawn. This allows you to draw your own bar on the appointment. The event UserDrawnAppointmentBar is raised with a rectangle of the drawing area. The schedule will not draw anything over this area, as it is your responsibility to draw there.

BarType	Determines the type of bar to draw: None, Category, Provider, or UserDrawn.
Size	This property defines the width, in pixels, of the bar

This property is provided so that you might convey more information to the user if desired by using a color-code bar. There is also a matching bar in the schedule's left margin defined by the property TimeBar. It is just like this property object except that its settings are used to draw the bars in the schedule's left margin.

AppointmentSpace

This property defines the number of pixels between and appointment's right edge and the column divider. This value must be greater than or equal to zero. If the value is zero then there is no space between the two and appointments are positioned flush with the column divider. This property is useful to allow the user to create appointments that conflict with each other. For example, if an appointment is schedule from 8a – 10a and fills its column, the user has no way to click on the background to create an appointment at 9a. With a space on the right side of the appointment the user can click on the background at the 9a row to create an appointment.

BlockoutColor
Each appointment has a Blockout property. When this property is set to true the appointment is displayed as a solid color with no text, header, or icons. You would use this property when you wish for the user to know that an appointment exists at a position but do not want him to interact or see any information about the appointment. The schedule property BlockoutColor is used to draw the solid color for any appointments that have been blocked-out.

ClockSetting
Because different countries have different time displays, the ClockSetting property allows you to define this display. The settings allow for time to be displayed in 12-hour or 24-hour format. This affects the schedules, its property screens, etc.

ConflictDisplay
Invariably when building schedules, conflicting appointments occur. The schedule allows for conflicts to be displayed in 3 different ways: overlap, side-by-side, and stagger. The overlap setting is the least useful of all. It allows conflicting appointments to completely obscure each other. No attempt is made to show all appointments if there are conflicts. The side-by-side setting guarantees that no part of any appointment is obscured by another. If a conflict occurs between two appointments, each becomes half as wide and they are displayed in columns within columns. The stagger setting is not as useful but can be used to add graphic splendor to your schedule. Some parts of conflicts appointments are obscured but some part of them is guaranteed to be visible. In this way it is not possible to totally lose sight of an appointment even if most of it is covered. Used with an appointment's transparency property, this settings can make a schedule quite unique.

DayLength
This DayLength property defines the length of the day displayed by a schedule. This is measured in hours. A schedule can only be of integral hour length starting and ending on an hour boundary. For example, if the StartTime is 8a and then DayLength is 10 hours, the last visible time will be 6p.

Appearance
This property object defines the look of the main schedule. It contains settings for the backcolor as well as the gradient style. This Appearance object is used to draw all areas that are not margins, headers, or appointments. In other words it defines the schedule grid itself.

EventAppearance
A schedule may have any number of appointments marked as events, which are displayed in the top margin in certain viewmodes. When the event header is visible these appointments are displayed in the header, which is between the column header

and the main schedule grid. The EventAppearance object controls the display of this area. It defines the background color as well gradient type if desired.

HeaderDateFormat
Every application has a different requirement on how to display dates. The HeaderDateFormat property allows you to define how dates appear in the date headers. This property is used to format the dates in the row or column header. It is only used if the viewmode is set to a value that includes dates, of course.

MaxDate
This property defines the last date to which the user can scroll. A schedule must have some date range and it is defined by the MinDate and MaxDate.

MinDate
This property defines the first date displayed on a schedule.

MinutesShown
The time margin may be drawn one of two different ways: All or FirstOnly. When this property value is set to All, all minute increments are displayed. For example if the ScheduleIncrement is set to 20, there are three increments per hour. The time margin would show the hour number and then “00”, “20”, and “40” for each hour. If the property value is set to FirstOnly, then “00” is the only valued displayed for each hour. The other two increments are blank. This is the style of Microsoft Outlook.

ScrollBars
If a schedule’s range is greater than that which can be displayed, scroll bars are used. The ScrollBars property defines which bars are visible if need be: none, horizontal, vertical, or both. By default the settings is “both”. You may selectively turn off the scrollbars if you wish; however there is no need to do this as they are disabled if not needed.

SelectedItem
This object is a reference to the appointment currently selected. If the MultiSelect property is set, this property is the last appointment that was (and still is) selected. This object will necessarily be in the SelectedItems collection. If MutiSelect property is set to false, this appointment will be the only one in the SelectedItems collection.

StartTime
This property defines the first time value displayed on the schedule. It must be an even hour. Used in conjunction with the DayLength property, it defines the time margin of a schedule.

TimeIncrement
This property defines into how many increments an hour is divided. For example if its settings is 30 minutes, an hour is split into two increments. This setting can be any factor of 60 from 1 to 60.

ViewMode
This is a most useful property as it defines the look of the schedule. It determines the type of information displayed on it. There are numerous modes available that display different configurations of dates, times, room, and providers. Depending on how you wish to display information to your user, you can use this property to define the view of the information.

Selector Object

Overview
The Selector is an object used to define a user-highlighted area on the schedule. The user may click the mouse or press the arrow keys to navigate and highlight areas of the schedule. The selector defines the size of a new appointment, if the <ENTER> key is pressed. In this way the user may define an area and create an appointment in it.

AllowSelector
This schedule property defines if the Selector object is used. The selector allows you to select an area of the schedule with the keyboard. When this property is set to true the keyboard can be used to highlight areas. For example, you may want to select multiple rows (or columns) on the schedule. You can hold down the <SHIFT> button and use the navigation keys to highlight the area of the grid over the desired area. If you wish to disallow keyboard navigation then set this property to false.

Selector.Column
This property gets or sets the starting column in which the selector resides.

Selector.Length
This property gets or sets the length in cell units of the selector. If the time margin is on the left and the Length setting is 2, the selector covers 2 rows.

Selector.Row
This property gets or sets the starting row in which the selector resides.

Appearances

All objects that are displayed on the schedule have an associated Appearance object. It defines the look-and-feel of the object being drawn. Not all properties of the Appearance object are used at all times. For example, the EventAppearance object defines the drawing attributes of the event header at the top of the screen. There is no text drawn in this region and as such the font properties of the object are not used.

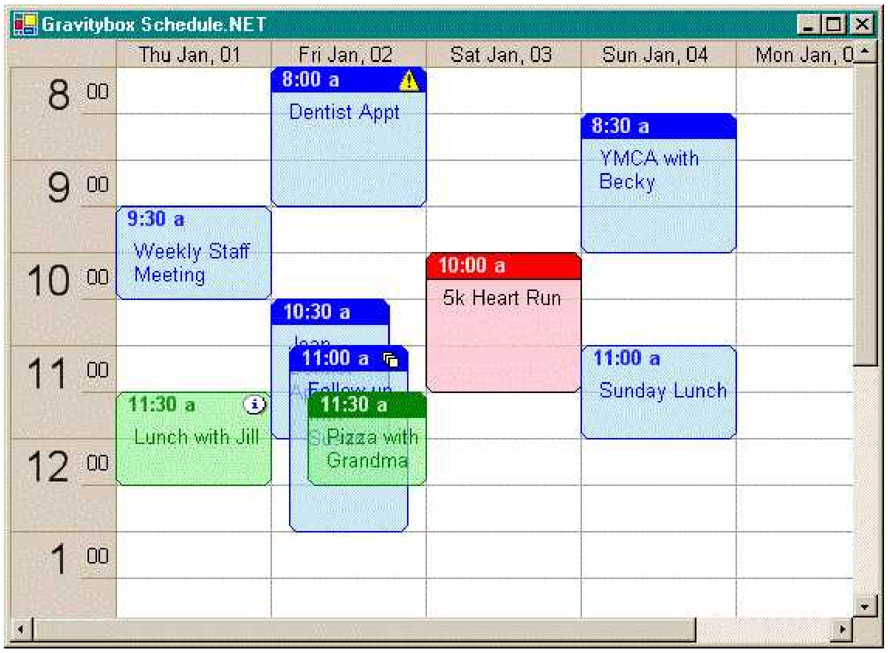
Appearance objects come in different varieties. The standard Appearance object is used to draw the headers and areas. The AppointmentAppearance object defines a few more properties and is used to draw appointments. The

AppointmentHeaderAppearance object is used to draw appointment headers. The last two are derived from the base object and define a few more properties for drawing.

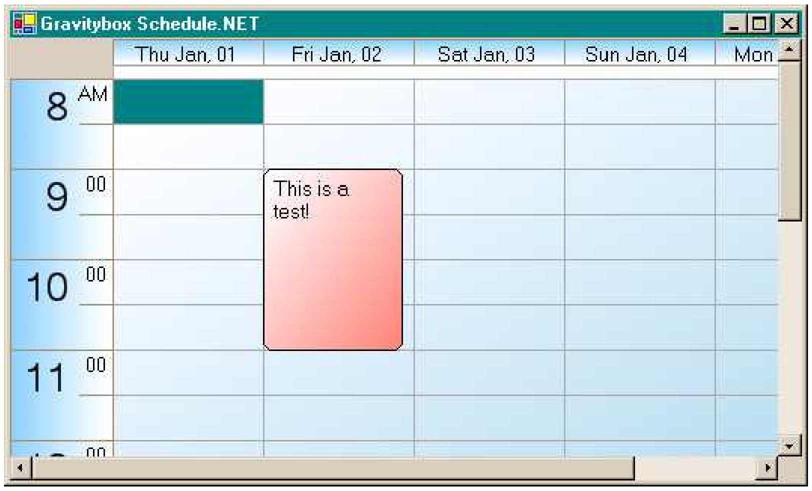
Alignment	Determines the alignment of the text drawn.
BackColor	Determines if the background color of the object.
BackColor2	Determines the secondary color to which the background fades when the BackGradientStyle is not set to None.
BackGradientStyle	Determines if the background fades from one color to another.
BorderColor	Determines the border color of the object.
BorderWidth	Determines the size of the border in pixels.
FontBold	Determines if the font used is bold.
FontItalics	Determines if the font used is italic.
FontSize	Determines the size of the font used in units defined by FontUnit.
FontStrikeout	Determines if the font used is strikeout.
FontUnderline	Determines if the font used is underlined.
FontUnit	Determines the units to use when using the FontSize.
ForeColor	Determines the text color.
StringFormatFlags	Determines the formatting options of text.
TextTrimming	Determines the way in which text is trimmed if there is not enough space to display it.
TextVAlign	Determines the vertical alignment of text.

The Appearance object does create quite a robust interface for specifying drawing attributes. The two derived versions are only slightly different. Objects that contain an Appearance property include Schedule Control, Header, ScheduleArea, Appointment, AppointmentHeader, and ScheduleSelector.

An example of the types of schedule displays possible using this object follows. This screen shot was created employing numerous effects. Headers were added to some appointments. All of the appointments had their backcolor and transparency customized.



Another interesting way to display appointments is using gradients. The BackColor2 property of the AppointmentApperance class allows you to define the fade color used to fade from the primary backcolor. When used with the BackGradientStyle property, the background of an appointment can fade from one color to another. This is a very nice visual effect to utilize if your application is going for the XP look-and-feel.



Notice in the screen shot above that the column header, row header, schedule grid, and appointment background are all gradients. Each was created with just three lines of code. Virtually any appearance you wish to create is possible with Appearance objects. The following code snippet was used to create the screen shot above.

```
Schedule1.StartTime = #8:00:00 AM#
Schedule1.DayLength = 10
Schedule1.SetMinMaxDate(#1/1/2004#, #1/5/2004#)
```

```
Schedule1.ColumnHeader.Appearance.BackColor = Color.LightSkyBlue
Schedule1.ColumnHeader.Appearance.BackColor2  =  Color.White
Schedule1.ColumnHeader.Appearance.BackGradientStyle   =
Gravitybox.Objects.GradientStyleConstants.Vertical

Schedule1.RowHeader.Appearance.BackColor = Color.LightSkyBlue
Schedule1.RowHeader.Appearance.BackColor2  =  Color.White
Schedule1.RowHeader.Appearance.BackGradientStyle =
Gravitybox.Objects.GradientStyleConstants.Horizontal

Schedule1.Appearance.BackColor = Color.White
Schedule1.Appearance.BackColor2 = Color.LightBlue
Schedule1.Appearance.BackGradientStyle = _
      Gravitybox.Objects.GradientStyleConstants.ForwardDiagonal

Dim appointment As Gravitybox.Objects.Appointment
appointment = Schedule1.AppointmentCollection.Add("", #1/2/2004#, _
     #9:00:00 AM#, 120)
appointment.Appearance.BackColor = Color.White
appointment.Appearance.BackColor2 = Color.Salmon
appointment.Appearance.BackGradientStyle =
Gravitybox.Objects.GradientStyleConstants.ForwardDiagonal
appointment.Appearance.IsRound = True
appointment.Subject = "This is a test!"
```

Chapter 3

Behavior

Overview

Many of the schedule properties deal specifically with behavior. Described below are numerous properties that define how the schedule will behave when responding to user actions. These properties are quite useful when you wish to toggle off some of the robust built-in functionality of the schedule.

AllowAdd

This property determines if appointments can be added to the schedule when the user double-clicks or pressed the <ENTER> key. Appointments can always be added in code of course but you may not wish the user to add appointments to a schedule. When this property is true, the user can highlight a schedule area and press the <ENTER> key to create an appointment that is added to the AppointmentCollection.

AllowCopy

This property determines if the user can copy (clone) an appointment. When true, the user can hold the <CTRL> key and drag an appointment and drop it to create a copy of the source appointment at the new position. All other properties of the appointment are copied except for the information relevant to the drag: date, time, room, and/or provider.

AllowDragTip

This property determines if a tooltip is displayed while an appointment is being dragged. When this property is set to true, a tooltip is displayed in the top, left corner with the date, time, room, and/or provider of the position under the mouse while it is being dragged.

AllowGrid

This property allows you to toggle the schedule grid on or off. The grid is displayed so that the user might easily determine where appointments reside in relation to the margins. If you do not want to display a grid, set this property false and only the schedule Appearance object’s backcolor is used to draw the grid area.

AllowInPlaceEdit

This property determines if the user can click in an appointment to start an edit of its Subject property. When this occurs, the user may change the Subject property by typing inside of the appointment and pressing clicking off of it to save the new text. The user can cancel the edit by pressing the <ESC> key.

AllowMove

The AllowMove property determines if the user may click the mouse on an appointment and drag it to a new location. If this property is set to false, the user cannot move any appointment on the schedule. You may want to use this property to make a schedule read-only. However keep in mind that this property only controls moves. The user may still perform other appointment operations like resize.

AllowRemove
The AllowRemove property is used to control user defined appointment deletes. If you wish to disallow the user to remove an appointment then set this property to false.

AppointmentResizing
This property allows you to specify the type of resizing that is allowed: None, StartTime, Length, or All. The value StartTime allows appointments to be resized such that only their starting time is changed. The value Length allows appointments to be resized such that only their length is changed. The All value allows both end of the appointment to be resized.

AutoRedraw
This property allows you to turn off screen painting while performing some lengthy schedule operations. For example, when an appointment is added to the AppointmentCollection, the schedule is repainted to reflect the change. If you are loading many appointments this can slow down your load procedure considerably. To combat this issue you can set the AutoRedraw property to false, load all of your appointments, and then reset it to true. This will temporarily turn off screen painting while you load. Be sure to reset this property to true of the screen will not repaint its image.

MultiSelect
The schedule also supports multi-selection. By setting the MultiSelect property to true, the user can hold the <CTRL> key and use the mouse to select multiple appointments. This is useful if you wish to allow the user to remove multiple appointments at one time. Moreover, you may have defined a custom operation that you wish to run on many appointments at one time.

Chapter 4

Event Notifications

Overview

In response to the numerous actions that can occur on a schedule, there are events that are raised to inform the container of these actions. When an appointment is added, removed, etc, an event is raised in which you can add custom code to handle the situation. You may wish to commit data to a database or display information to the user. In any case, these events provide a means to add custom functionality.

AfterAppointmentAdd

The event is raised after an appointment has been added to the AppointmentCollection. This can happen in code by calling the collection’s Add method. It is also raised when the user adds an appointment by pressing the <ENTER> key or double clicking on the schedule background.

AfterAppointmentCopy

This event is raised after the user has made a copy of an appointment by holding the <CTRL> key and dragging an appointment. This action causes a clone of the appointment to be made and added to the AppointmentCollection after which this event is raised.

AfterAppointmentMove

This event is raised after the user has moved an appointment by dragging it. It is not raised when the start time, length, or any other property of an appointment is changed in code. It is only raised by user interaction.

AfterAppointmentRemove

This event is raised after an appointment has been removed from the AppointmentCollection. The user can perform this by selecting an appointment and then pressing the <DELETE> key. It is also raised when the “Remove” or “RemoveAt” method of the AppointmentCollection is called to remove an appointment.

AfterAppointmentResize

This event is raised after the user has successfully resized an appointment. Changing any of an appointment’s properties in code does not raise this event. Dragging either edge of an appointment to change the start time or length triggers this event.

AfterInPlaceEdit

This event is raised after the user has edited the subject of an appointment in-place. An in-place edit occurs when the user selects an appointment and then clicks again on the appointment to display an edit textbox.

AfterPropertyDialog
When a user double clicks on an appointment, the default appointment, property, dialog is displayed. If the user presses the save button on this dialog, the AfterPropertyDialog event is fired. This event is notification that the property dialog was displayed and saved.

AppointmentClick
This event is raised when the mouse is clicked on an appointment.

AppointmentDoubleClick
This event is raised when the mouse is double clicked on an appointment.

AppointmentHeaderInfoClick
This event is raised when the mouse is clicked on an appointment header.

BeforeApplicationExecute
This event is raised before an external application is started by an appointment. Each appointment's Alarm property object has two properties ApplicationName and ApplicationArgs that allow you to specify an application, with its arguments, to be executed when an appointment comes due.

BeforeAppointmentAdd
This event is raised before an appointment is added to the AppointmentCollection. An event parameter allows you to cancel the operation before the appointment is added. If the operation is successful the AfterAppointmentAdd event is raised afterwards.

BeforeAppointmentCopy
This event is raised after an appointment has been copied. The user can perform this action by holding the <CTRL> key and dragging an appointment to a new location. When dropped the appointment is not moved but copied. An event parameter allows you to cancel the operation before the appointment is copied.

BeforeAppointmentDue
This event is raised before an appointment comes due. More specifically this event occurs before the BeforeApplicationExecute event and before the alarm dialog appears. You can cancel these other actions by setting the cancel parameter to this event to true. This will effectively cancel the alarm dialog that would have been raised if the alarm were armed.

BeforeAppointmentMove
This event is raised before an appointment is moved. An event parameter allows you to cancel the operation before the appointment is removed. If the operation is successful the AfterAppointmentMove event is raised afterwards.

BeforeAppointmentReminder

This event is raised before the appointment reminder screen is displayed. A reminder may occur before an appointment comes due. It is basically a screen that is displayed to remind the user some minutes before an appointment is due. You may cancel the reminder screen by setting the cancel parameter of this event to true.

BeforeAppointmentRemove
This event is raised before an appointment is removed from the AppointmentCollection. An event parameter allows you to cancel the operation before the appointment is removed. If the operation is successful the ValidateAppointmentRemove and AfterAppointmentRemove events are raised afterwards in that order.

BeforeAppointmentResize
This event is raised before an appointment is resized. An event parameter allows you to cancel the operation before the appointment is removed. If the operation is successful the AfterAppointmentResize event is raised afterwards.

BeforeAppointmentTextDraw
This event is raised before the text of an appointment is drawn. You may override the default text with your own text if desired. This is raised for every appointment.

BeforeInPlaceEdit
The event is raised before the textbox is shown for the in-place edit. When a selected appointment is clicked again it is toggled into edit mode. The user may type directly into the appointment space to edit the Subject field of the appointment. You may cancel the edit by setting the cancel parameter of this event to true.

BeforePropertyDialog
This event is raised before an appointment property dialog is displayed. An event parameter allows you to cancel the operation before the dialog is shown. If the operation is successful the AfterPropertyDialog event is raised afterwards.

ValidateAppointmentRemove
This event is raised after the BeforeAppointmentRemove event, which allows you to cancel, and before the AfterAppointmentRemove event. The event is raised to inform you that the appointment is guaranteed to be removed and there is no way to stop it. This is a good place to add database code that needs to remove an appointment. The appointment is still in the AppointmentCollection when this event is raised so you can reference the appointment.

Appointment Dialog Notifications

Overview
The built-in, appointment dialog provides a robust interface to edit an appointment. It allows the user to change many attributes of an appointment. This may pose a problem as the user may perform actions that you wish to disallow. These events are defined to let you handle these situations.

PropertyDialogRemoveFiles

The dialog has the ability to display a list of files associated with an appointment. When and if the user removes one of these files, the PropetyDialogRemoveFiles event is raised. This gives notification of this action as well as a chance to stop it.

PropertyDialogSaveInvalidArea

If the user changes an appointment property such as the date, time, room, or provider in such a way to overlap a NoDropArea, this event is raised to notify you of this condition. No-drop area objects are defined in the schedule NoDropAreaCollection object. Each is an area of the schedule where appointments are not allowed to exist.

PropertyDialogSaveValueOutOfRange

If the property dialog is used to change the date or time of an appointment to a value outside of the MinDate through MaxDate range or StartTime through StartTime + DayLength range, this event is raised. This is not necessary an error condition but you may wish to be notified of it for some reason. You can cancel the save from this event if desired.

Category Dialog Notifications

Overview

The category dialog can be displayed from code or by the user on the default appointment dialog. It consists of a list of all category objects that are present in the schedule CategoryCollection. The user may assign any number of categories to an appointment from the dialog.

AfterCategoryListDialog

This event is raised after the category dialog has been displayed. It is merely for notification that this action has occurred successfully.

BeforeCategoryListDialog

This event is raised before the category dialog is displayed. You may cancel the action from this event.

Provider Dialog Notifications

Overview

Much like the category dialog, the provider dialog can be displayed from code or by the user on the default appointment dialog. It consists of a list of all provider objects that are present in the schedule ProviderCollection. The user may assign any number of providers to an appointment from the dialog.

AfterProviderListDialog

This event is raised after the provider dialog has been displayed. It is merely for notification that this action has occurred successfully.

BeforeProviderListDialog
This event is raised before the provider dialog is displayed. You may cancel the action from this event.

Configuration Dialog Notifications

Overview
The configuration dialogs are used in design-time to allow you to setup and persist objects to the schedule's collections. They may also be called at run-time to allow the user to setup the collections. Of course in many instances you will wish to provide a custom configuration screen for your applications; however in some cases the default dialogs may be sufficient and if so, they are available for use at run-time. There is a configuration dialog to setup Categories, Providers, and Rooms.

AfterCategoryConfigurationDialog
This event is raised after the category configuration screen has been saved. If the screen was canceled this event is not fired.

AfterProviderConfigurationDialog
This event is raised after the provider configuration screen has been saved. If the screen was canceled this event is not fired.

AfterRoomConfigurationDialog
This event is raised after the room configuration screen has been saved. If the screen was canceled this event is not fired.

BeforeCategoryConfigurationDialog
This event is raised before the category configuration screen has been displayed. The event allows you to cancel the action before the screen is shown.

BeforeProviderConfigurationDialog
This event is raised before the provider configuration screen has been displayed. The event allows you to cancel the action before the screen is shown.

BeforeRoomConfigurationDialog
This event is raised before the room configuration screen has been displayed. The event allows you to cancel the action before the screen is shown.

Row and Column Header Notifications

Overview
The column and row headers define information about how a schedule is displayed. Each displays information like date, time, room, or provider. The schedule generates the strings that are displayed in the header margins but the container is given a chance to override these strings in the "Before…HeaderDraw" events.

Each header has properties to define behavior and appearance: AllowResize, AutoFit, Size, Appearance, and FrameIncrement. The AllowResize property determines if the user can move the mouse over a header break, grab the edge, and resize it. When a user performs this action, the Size property changes. When the AutoFit property is true, all columns or rows are stretched to fit the grid area. Keep in mind that the Size property becomes read-only when AutoFit is true as the size of the columns or rows is calculated by the schedule. Setting the Size property while in this state throws an exception. The Appearance object defines the colors, borders, etc of the column or row area being drawn. The FrameIncrement property determines the number of columns or rows that are scrolled at one time. The default is one but this allows you to make the user scroll only N rows or columns at a time.

AfterColumnResize
This event is raised after a column header is resized. The schedule has already been redrawn and the action cannot be canceled at this point.

AfterRowResize
This event is raised after a row header is resized. The schedule has already been redrawn and the action cannot be canceled at this point.

BeforeColumnResize
This event is raised before a column header is resized. You may choose to cancel this action and no redraw will be performed. If you do not wish to allow column resizing, set the AllowResize of the ColumnHeader object to false.

BeforeDateHeaderDraw
This event gives you a chance to override the string to be displayed in each of the date header cells.

BeforeProviderHeaderDraw
This event gives you a chance to override the string to be displayed in each of the provider header cells.

BeforeRoomHeaderDraw
This event gives you a chance to override the string to be displayed in each of the room header cells.

BeforeRowResize
This event is raised before a row header is resized. You may choose to cancel this action and no redraw will be performed. If you do not wish to allow row resizing, set the AllowResize of the RowHeader object to false.

Schedule Notifications

Overview

There are numerous miscellaneous events that are raised in response to some action. These range from clicking on a schedule area to changing the appointment selection. These events are quite useful to create custom code that responds to various user actions.

AfterSelectedItemChange
This event is raised when the SelectedItems collection changes. If the schedule is single-select then the collection can have exactly zero or one item in it. Each time the user clicks on an appointment that is not highlighted the currently selected appointment is deselected and the clicked appointment is added to the SelectItems collection. If the schedule property MultiSelect is set to true, the user can hold the <CTRL> key and click multiple appointments to select more than one appointment. All highlighted appointments are in this collection and may you may add other programmatically. Any addition or subtraction cases this event to be raised.

BackgroundClick
This event is raised if the user clicks anywhere inside the grid area not on a header or appointment. This is anywhere on the background.

BeforeSelectedItemChange
This event is raised just before another appointment is added to the SelectedItems collection. You may cancel the action thus canceling the redrawing of the schedule. Immediately after the collection has been modified the AfterSelectedItemChange event is raised.

BeforeToolTip
This event occurs just before a tooltip is displayed. The text is passed as a parameter to the event may be changed before the tip appears. If the text is set to empty string then the tip does not appear, effectively canceling the tooltip display.

ColoredAreaClick
This event occurs when the mouse is clicked inside of an area defined in the ColoredAreaCollection. This is a predefined collection of type ScheduleAreaCollection that exists on a schedule.

DateRangeChange
This event is raised when either the MinDate or MaxDate of a schedule changes. This informs the container that a new viewable range is now visible. Calling the SetMinMaxDate method also causes this event to occur.

NoDropAreaClick
This event occurs when the mouse is clicked inside of an area defined in the NoDropAreaCollection. This is a predefined collection of type ScheduleAreaCollection that exists on a schedule.

ViewModeChange

This event is raised to notify the container that the ViewMode property has changed. The property controls the display of information.

Part III Advanced

Fast, fat computers breed slow, lazy programmers.

-Unknown

Chapter 5

Putting it all together

Overview
Now that you understand where the schedule information resides, you can create some data. In this section code snippets will demonstrate how to add data at run-time and display some needed, built-in dialogs.

Adding Appointments

All appointments that are displayed on the schedule are held in the schedule's AppointmentCollection. It has an Add and Remove method so there is no confusion on how to use the collection.

You may clear the AppointmentsCollection at any time by calling its Clear method.

```
Schedule1.AppointmentCollection.Clear()
```

Also adding appointments to the collection is quite easy. The collection is not only an object that holds a set of appointments but it is also the factory that creates them. Appointment objects may not be created freely outside of the collection. The AppointmentCollection creates and destroys all appointments. There is no such thing as a non-parented appointment. The following code snippet adds a 1-hour appointment on January 1, 2004 at 8am.

```
Schedule1.AppointmentCollection.Add("", #1/1/2004#, #8:00:00 AM#, 60)
```

The Add method creates the appointment and adds it to the collection. It returns a reference to the newly created object so you may set any of its many properties. Only the necessary properties are set through the constructor like date, time, and length. Most of an appointment's properties are optional and as such as not set during its creation. However after creating it you may wish to set its text, color, etc. The first parameter to the Add method is the appointment key. Notice that it is an empty string. This tells the collection to create a random key and assign to the appointment. If you are loading appointments from a database you will most likely set the key so that you may map the appointment back to its corresponding record in the database. In the code snippet below an appointment is added and then its Subject property is set.

```
Dim appointment As Gravitybox.Objects.Appointment
appointment = Schedule1.AppointmentCollection.Add("", _
        #1/1/2004#, #8:00:00 AM#, 60)
appointment.Subject = "John Doe"
```

Deleting Appointments

Appointments are just as easy to remove. You may remove by key, index, or object. The RemoveAt method takes an index [0..N-1] of the object to remove in the collection. If the index does not exist, an out of range exception is thrown.

```
Schedule1.AppointmentCollection.RemoveAt(5)
```

The Remove method is overloaded to take a key or an appointment object. The code snippet below removes an appointment object and an appointment with the key "abc123".

```
Schedule1.AppointmentCollection.Remove(appointment)
Schedule1.AppointmentCollection.Remove("abc123")
```

Displaying Dialogs

Now that you can create appointments, you will need to edit them with dialogs. The schedule comes with a built-in appointment dialog much like MS-Outlook. It provides an easy to use and stylish editor for an appointment. In the following code snippet an appointment is edited by passing its reference to the ShowPropertyDialog method of the Dialogs object. The Dialogs object contains methods for displaying all built-in dialogs.

```
Schedule1.Dialogs.ShowPropertyDialog(appointment)
```

If Categories and Providers are present in their respective collections then the default appointment dialog will allow the user to associate these objects types to an appointment. However you may display these two dialogs separately without using the appointment dialog. The following code snippet displays the Categories and Providers for an existing appointment.

```
Schedule1.Dialogs.ShowCategoryDialog(appointment)
Schedule1.Dialogs.ShowProviderDialog(appointment)
```

Canceling / Custom dialogs

Some applications are highly customized. They cannot use the defaults provided by the schedule. If you wish to write your own dialog screen for editing appointments and other objects of the schedule, you will need to cancel the defaults so that the user never sees them. Before the appointment dialog is actually shown, the BeforePropertyDialog event is raised. In this event you may cancel the edit and display your own custom form. In the code snippet below, the BeforePropertyDialog event is canceled and a custom form named CustomForm is displayed.

```
Private Sub Schedule1_BeforePropertyDialog(ByVal sender As Object, _
      ByVal e As Gravitybox.Objects.EventArgs.BeforePropertyDialogEventArgs) _
     Handles Schedule1.BeforePropertyDialog
  e.Cancel = True
```

```
   Dim F As New CustomForm(e.Appointment)
   F.ShowDialog()
End Sub
```

Chapter 6

Advanced Functionality

Overview

The schedule in addition to holding and displaying data also has advanced querying abilities. The storage of information is not very useful if there is not a way to easily sort and query the information. The collections and lists expose methods to extract specific information from what may be a vast set of data. You can produce subsets of information and perform set arithmetic operations to derive even more specific sets of information.

Searching

One of the most useful things about the defined lists is their ability to return objects from their complementary collections based on querying information. The AppointmentCollection has an overloaded Find method that returns a subset of appointments based on some search criteria. The CategoryCollection and ProviderCollection classes have only basic functionality as they can only search by text. Both of these object types are really nothing more than a piece of text and a color, so there is really nothing much on which to search. The real search functionality is built into the AppointmentCollection. Each appointment has many properties and as such you will want to find objects by different criteria.

Let us look at a real-world example. In a doctor’s office you may wish to find all appointments in a date range. The following code snippet will returns a list of appointments between January 1, 2004 and January 5, 2004, inclusive.

```
Dim list1 As Gravitybox.Objects.AppointmentList
list1 = Schedule1.AppointmentCollection.Find(#1/1/2004#, #1/5/2004#)
```

This is useful if you wish to provide functionality to a user to query appointments by date. You could retrieve a date range from the user in a custom dialog and then, after querying the AppointmentCollection, display the results on screen in a listview control perhaps. This ensures that you do not have to go re-query your database for information that you have already have.

This functionality is useful in and of itself; however in the real world you will most likely need to construct much more complicated queries. Following the example above, image that you wish to find all appointment in that date range for “John Doe”. Let’s assume that a patient’s name is displayed in appointment’s Subject property. Now following the example above we search by Subject. This new list consists of all the appointments for John Doe.

```
Dim list2 As Gravitybox.Objects.AppointmentList
List2 = Schedule1.AppointmentCollection.Find(“John Doe”)
```

Now comes the interesting part. These are the three operations defined on lists: Union, Intersection, and Subtraction. You can use these list operations to define arbitrarily complex queries.

Following the two examples above, we can now build a list that contains all items in list1 and also all items in list2. This is a set operation known as Intersection. Essentially finding all appointments for John Doe between the dates of January 1, 2004 and January 5, 2004.

```
Dim list3 As Gravitybox.Objects.AppointmentList
list3 = list1.Intersect(list2)
```

The new “list3” variable will hold the intersection of the two lists, meaning appointments that are in both lists, with no duplicates. If you wanted to find all appointments for that date range (for any patient) or any appointments for “John Doe”, you would use the “Union” method. This would basically concatenate both lists into one. If you wanted all appointments in the date range except for “John Doe” appointments then you would use the “Subtract” method to remove any appointments in list2 from list1.

Keep in mind when building lists that the Find method will never return duplicate objects in the same list. Although you can programmatically add the same object to a list multiple times, the schedule will not do this. For example, if the “Union” method is called to concatenate two lists, the result will not have duplicates. So two joined lists may have less elements than their combined source lists.

Conflicts

Conflict resolution is the most interesting and aggravating part of scheduling. Many applications require that conflicts do not exist. In other situations it is necessary and expected. The schedule does not disallow conflicts but does provide you with functionality to expose them and correct for them if necessary. By default, the schedule displays conflicting appointments side-by-side so that the user can see all appointments. The appointments are shown partial width to allow other appointments in the same slot to be seen.

The AppointmentList class exposes methods that facilitate conflict searching. The method “IsConflict” allows you to specify an appointment and determine if it conflicts with any appointments in the list. In the following code snippet, an appointment is specified to check if it conflicts with any appointments in a defined list.

```
Dim b As Boolean
b = list1.IsConflict(appointment)
```

If you wish to disallow conflicts then you will wish to not create them programmatically. Before creating an appointment with code, you can check to determine if the proposed appointment would cause a conflict. If is does you can prompt the user or cancel the operation, etc. The AppointmentList method “IsAreaAvailable” allows you to check for this condition. This overloaded method takes a variety of parameters based on the viewmode setting. If the viewmode is date and time then you will need to query based on dates and times to get a meaningful result from the method. Given a set of parameters the method determines if there is any appointment overlapping and if so returns false. If the proposed appointment area does not overlap any other appointment in the list the method return true, indicating the proposed area is available for scheduling.

The following code snippet checks to determine if any appointments overlap the schedule area January 2, 2004 from 10am through 11am.

```
Dim b As Boolean
b = list1.IsAreaAvailable(#1/2/2004#, #10:00:00 AM#, 60)
```

Checking for availability is important but in an office environment many times you wish to know when is the next available time slot for an appointment. Instead of building complicated algorithms to find schedule availability, you can use the built-in functionality already provided. The method “NextAreaAvailable” exists as an enhancement to the “IsAreaAvailable” method. The latter just checks a specific schedule slot, while the former starts a specific schedule slot and returns the next available slot after it.

For example if you were on the phone with a client and he said “The earliest I can come in is next Tuesday.” You would then call this function with next Tuesday date and the proposed appointment length and it would return the next schedule slot that would fit the proposed appointment. The method uses the schedule StartTime and DayLength properties to narrow the calculations. If the schedule only displays 8am through 6pm, the “NextAreaAvailable” method would never returns a slot outside of this range. It also uses the MinDate and MaxDate schedule properties as well as boundaries.

In the following code snippet, the search will begin on January 6, 2004 at 8am. The schedule will continue to search for an appointment slot to fit a 1-hour appointment. If it finds a free schedule area it will be returned in an appointment object. This appointment does not on the schedule it only is a holder for the information. If you wish to add the returned value as an appointment to the AppointmentCollection then simply call the AppointmetnCollection’s Add method with the returned object.

```
Dim proposed As Gravitybox.Objects.Appointment
proposed = list1.NextAreaAvailable(#1/6/2004#, #8:00:00 AM#, 60)
If Not (proposed Is Nothing) Then
  Debug.WriteLine(proposed.StartDate)
  Debug.WriteLine(proposed.StartTime)
  Schedule1.AppointmentCollection.Add(proposed)
End If
```

Schedule Areas

The NoDropCollection and ColoredAreaCollection are created from the ScheduleAreaCollection class. These two class instances define the no-drop areas and other colored areas of a schedule. There is also a matching ScheduleAreaList class.

The ColoredAreaCollection object allows you to define colored areas of a schedule. There is no behavior associated with this collection. The areas are just colored to convey information to the user while still allowing all appointment actions like add, edit, and delete.

The NoDropAreaCollection object holds schedule areas that are defined to disallow appointment drops. This is useful to define holidays, lunch hours, etc where appointments should not exist. However you may still programmatically add appointments to these areas. So as not to confuse your users you may wish to check to determine if a proposed appointment is in a no-drop area. You cannot query the collection directly but use its list. In the following code snippet, the NoDropAreaCollection is converted to ScheduleAreaList and then queried to determine if a new proposed appointment will be inside of it.

```
Dim list As Gravitybox.Objects.ScheduleAreaList
list = Schedule1.NoDropAreaCollection.ToList
Dim b As Boolean = list.IsOverlap(#1/2/2004#, #8:00:00 AM#, 60)
```

The code above queries the ScheduleAreaList to determine if a 1-hour hour slot on January 2, 2004 at 8am overlaps any area in the list. If any appointment overlaps any portion of the defined area the method returns true, otherwise false.

Recurrences

Another advanced concept is that of recurrences. A recurrence is set of appointments that recur at some specified interval for some specific period. An example of this would be an appointment that occurs every Monday at 8:00AM until June 27, 2005.

The default property dialog has a recurrence sub-dialog that can be displayed from the property screen. You can also access recurrences from code. Keep in mind that in the background the schedule actually creates zero or more appointments for each recurrence pattern. Appointments are said to be in a recurrence pattern if they have the same RecurrenceId. When an appointment is created it is given a unique RecurrenceId. So in essence, every appointment is part of a recurrence pattern even if is in a group of one. You can set the RecurrenceId property to any string you wish. This makes it easy to put two appointments in a recurrence pattern, just set their RecurrenceId property to the same value. When you store appointment information to your database you will want to create a field for this value or the next time you load your appointments they will have unique RecurrenceId values and you will have no recurring appointments.

There are three types of recurrences that can be created: daily, weekly, and monthly. For example, you may wish to create an appointment for every Monday at 8:00AM until June 27, 2005 or every Monday at 8:00AM for three weeks. Notice that I added some way to end the recurrence pattern. The first end on a date and the second ends after some number of times. You cannot add a recurrence that goes on forever. This is a limitation currently. Since a recurrence pattern is made of atomic appointments, a nonending recurrence pattern would create an infinite number of appointments and this is not going to fit in any database.

In code you can create a recurrence pattern, and in the process create more appointments in the AppointmentCollection by manipulating a Recurrence object. This object can be created and then used in conjunction with an existing appointment to create a group of appointments. For instance, to follow the example above we will create an appointment and then repeat it until June 27, 2005.

```
    Dim appointment As Appointment
     appointment = Schedule1.AppointmentCollection.Add("", #6/6/2005#, _
                 #8:00:00 AM#, 60)
    appointment.Subject = "John Doe"

    'Create and setup the recurrence
    Dim recurrence As New Recurrence

    'End the recurrence by date and on this date
    recurrence.StartDate = appointment.StartDate
    recurrence.EndDate = #6/27/2005#
     recurrence.EndType = RecurrenceEndConstants.recEndByDate
    recurrence.RecurrenceInterval = RecurrenceIntervalConstants.ricDaily

    'Setup the RecurrenceDay object
    'Every 7 days for all days including weekends
    recurrence.RecurrenceDay.DayInterval = 7
    recurrence.RecurrenceDay.RecurrenceMode = _
             RecurrenceDayConstants.DayInterval

    'Add the recurrence
     Schedule1.AppointmentCollection.AddRecurrence(appointment, recurrence)
```

The above code snippet creates an appointment on June 6, 2005. Then a recurrence object is created for a daily recurrence every 7 days to stop on June 27, 2005. This will add four appointments to the AppointmentCollection. If the DayInterval had been set to 3 then the appointments would have been created on Monday, Wednesday, Friday, Sunday, Tuesday, etc. until June 27, 2005. You can even skip weekends if you wish by setting the RecurrenceMode property to DayWeekdays. This will ensure that no appointments are scheduled on Saturday or Sunday.

The same effect as above can be created with a weekly recurrence of coarse, since the above pattern created an appointment once every seven days, which is a week. The code snippet below does the same thing but with a weekly recurrence.

```
    Dim appointment As Appointment
     appointment = Schedule1.AppointmentCollection.Add("", #6/6/2005#, _
                 #8:00:00 AM#, 60)
    appointment.Subject = "John Doe"

    'Create and setup the recurrence
    Dim recurrence As New Recurrence
    'End the recurrence by date and on this date
    recurrence.StartDate = appointment.StartDate
    recurrence.EndDate = #6/27/2005#
     recurrence.EndType = RecurrenceEndConstants.recEndByDate
     recurrence.RecurrenceInterval = RecurrenceIntervalConstants.ricWeekly

    'Setup the RecurrenceDay object
    'Every 7 days for all days including weekends
    recurrence.RecurrenceWeek.WeekInterval = 1
    recurrence.RecurrenceWeek.UseMon = True

    'Add the recurrence
     Schedule1.AppointmentCollection.AddRecurrence(appointment, recurrence)
```

The UseMon property is set because by default all of the “Use” properties (UseSun, UseMon, etc) are false. I knew this appointment was on Mondays so I set it to true to allow appointments to be scheduled on Mondays. These properties are useful when you want to schedule weekly appointments by do not want them to be scheduled on specific days.

The most complicated mode is Month. This has different and somewhat confusing properties that have values that are not intuitive. You can setup a recurrence by interval or ordinal. By Interval, specifies the day of the month on which to recur, like the third (3) day of the month. By Ordinal, specifies a named day like the second Tuesday of each month. In the following example the recurrence pattern is monthly, by interval. The appointment will be repeated on the third of the month, every two months. Only three appointments will be created by the recurrence pattern as set by the EndIterations property.

```
    Dim appointment As Appointment
     appointment = Schedule1.AppointmentCollection.Add("", _
        #6/6/2005#, #8:00:00 AM#, 60)
    appointment.Subject = "John Doe"

    'Create and setup the recurrence
    Dim recurrence As New Recurrence

     'End the recurrence by interval after 3 appointments added
    'and on this date
    recurrence.StartDate = appointment.StartDate
    recurrence.EndType = RecurrenceEndConstants.recEndByInterval
    recurrence.EndIterations = 3
     recurrence.RecurrenceInterval = RecurrenceIntervalConstants.ricMonthly

    'Setup the RecurrenceDay object
    'Add an appointment the 3 day of every 2 month
    recurrence.RecurrenceMonth.MonthInterval = 2
    recurrence.RecurrenceMonth.RecurrenceMode = _
        RecurrenceMonthConstants.MonthInterval
    recurrence.RecurrenceMonth.DayInterval = 3

    'Add the recurrence
     Schedule1.AppointmentCollection.AddRecurrence(appointment, recurrence)
```

You may also choose to recur based on an ordinal position in a month like the second Tuesday of each fourth month. The following code snippet demonstrates this configuration.

```
    Dim appointment As Appointment
    appointment = Schedule1.AppointmentCollection.Add("", _
        #6/6/2005#, #8:00:00 AM#, 60)
    appointment.Subject = "John Doe"

    'Create and setup the recurrence
    Dim recurrence As New Recurrence

    'End the recurrence by interval after 3 appointments added
    'and on this date
    recurrence.StartDate = appointment.StartDate
    recurrence.EndType = RecurrenceEndConstants.recEndByInterval
    recurrence.EndIterations = 3
    recurrence.RecurrenceInterval = RecurrenceIntervalConstants.ricMonthly

    'Setup the RecurrenceDay object
    'Add an appointment the second(2) Tuesday every 4 months
    recurrence.RecurrenceMonth.MonthInterval = 4
    recurrence.RecurrenceMonth.RecurrenceMode = _
        RecurrenceMonthConstants.MonthOrdinal
    recurrence.RecurrenceMonth.DayOrdinal = _
        RecurrenceMonthOrdinalConstants.Second
    recurrence.RecurrenceMonth.DayPosition = _
        RecurrenceMonthOrdinalDayConstants.Tuesday

    'Add the recurrence
    Schedule1.AppointmentCollection.AddRecurrence(appointment, recurrence)
```

As you can see, the recurrence objects are a bit complicated but they give you great flexibility. You can create virtually any recurrence scenario that you can conceive. The most important fact to remember is that each appointment is a separate, distinct appointment in and of itself and they all share the same RecurrenceId.

Chapter 7

Printing

Overview

After creating a schedule there are many times when you will wish to print a hard copy. The GbSchedule.NET component has this functionality built-in. It will print the schedule as you se it on the screen across multiple pages if necessary. There is even a print preview to view the look of the print before it occurs.

Printing is very easy. There are only two methods: GoPrint and GoPreview. Each can take a settings object to define the printable area. If either is called with no parameter then the entire schedule is printed or previewed as defined by the properties MinDate, MaxDate, StartTime, and DayLength. Also depending on the viewmode Providers or Rooms might be shown as well.

To print an area of the schedule just setup the settings object and send it to the GoPrint method as demonstrated in the following code snippet.

```
Dim settings As New PrintDialogSettings(#1/1/2004#, #8:00:00 AM#, _
        #1/5/2004#, #6:00:00 PM#)
Schedule1.GoPrint(settings)
```

You may define any area of the schedule as long as it is valid. This means that if the MinDate is June 1, 2004, you cannot print from May 20, 2004 since it is not generated by the schedule. The same is true for times as well. If the schedule displays 8am through 6pm then you cannot print a schedule area from 3am in the morning.

There are numerous events that are raised during the course of a print. The order of events is as follows: PrintSetup, BeforePrintHeader, BeforePrintFooter, BeforePrintPageNumber, and PrintProgress. If the print is canceled then the PrintCanceled is raised as well in no particular order.

PrintSetup

This event is raised before any printing starts and allows you to define a custom header and footer height for all pages. It you do not wish to have a header or footer then you can set the heights of these two sections to zero with the event parameter.

BeforePrintHeader

This event is raised before the header is drawn. You may cancel the default header by setting the handled to true and drawing your own graphic in the defined rectangle. This is called one time for each page.

BeforePrintFooter

This event is raised before the footer is drawn. You may cancel the default footer by setting the handled to true and drawing your own graphic in the defined rectangle. This is called one time for each page.

BeforePrintPageNumber
This event is raised before the header is drawn. You may cancel the default header by setting the handled to true and drawing your own graphic in the defined rectangle. This is called one time for each page.

PrintProgress
This event is raised to inform the container of the progress of the print. The event has a parameter with the total number of pages to print and the current page being printed. This event is raised for each page printed. You may take any action desired, produce your own custom progress bar, etc. When the current page matches the total pages then the printing is complete.

PrintCanceled
When the schedule is being printed a progress dialog is displayed. If the user presses the cancel button, the print is canceled and the PrintCanceled event is raised. This informs the container that the print failed and was not completed. You may take appropriate action in your application if desired.

Chapter 8

Data Binding

Overview

Data binding is the process of binding a schedule directly to a dataset. VS.NET defines an object named Dataset. This object can effectively hold multiple database tables. This discussion assumes that you have used datasets before.

Since every application has its own structure and data, you may think that it would be impossible to bind a schedule to an unknown database. The schedule is quite flexible in that it can bind to a dataset with minimal data. In addition a dataset can be expanded to encompass many optional fields. There are also some other constraints on the dataset structure to make it compatible with the schedule. However none of these are overbearing and you can create a compatible dataset from your database quite easily.

A dataset can be defined visually using the Visual Studio environment. In this fashion you can create a strongly typed dataset with appropriate columns and constraints. You can view the dataset on screen and modify it. A strongly typed dataset is derived from the "System.Data.Dataset" object. Alternatively you can create a dataset in code. This second method is described in the example below.

When creating the dataset there are a few constraints on its structure. First, there must be a table defined in the dataset named "Appointment". This table must contain the columns Appointment_Guid, StartDate, StartTime, and Length with the data types string, date, date, and integer respectively. All of these columns must be marked to disallow DbNull and the Appointment_Guid column must be marked as unique. This is the minimal set of requirements. If any of these requirements are not met, the schedule will throw an error when the dataset is bound.

The following code snippet defines a minimal dataset in code.

```
'Create a dataset with an "Appointment" table
Dim ds As New System.Data.DataSet
Dim dt As System.Data.DataTable = ds.Tables.Add("appointment")

'Create the necessary columns
dt.Columns.Add("appointment_guid", GetType(String))
dt.Columns.Add("start_date", GetType(Date))
dt.Columns.Add("start_time", GetType(Date))
dt.Columns.Add("length", GetType(Integer))

'Mark to disallow DBNull
dt.Columns("appointment_guid ").AllowDBNull = False
dt.Columns("start_date").AllowDBNull = False
dt.Columns("start_time").AllowDBNull = False
dt.Columns("length").AllowDBNull = False
```

```
'Mark the 'appointment_guid' column as a unique key
dt.Columns("appointment_guid").Unique = True
```

This empty dataset can be bound to the schedule as follows, where the name of the schedule is Schedule1.

```
'Bind the dataset to the schedule
Schedule1.DataSource = ds
Schedule1.Bind()
```

When the schedule is bound, all appointments on the schedule are cleared, so be sure there are no appointments that you need. After the AppointmentCollection is cleared the newly bound dataset is used to populate the schedule. In the example above, the dataset is empty so no new appointments are created. The dataset always reflects the state of the schedule. This means that as the user operates on an appointment by add remove, move, etc, the dataset is updated. The opposite is also true. If the dataset is modified outside of the schedule while it is bound, the changes will be reflected on the schedule as well. This means that you can commit the dataset back to your database at any time and the changes will be current.

This also creates an interesting corollary. If a dataset is bound to two or more schedules simultaneously, all schedules will be synchronized at all times. This occurs because if one schedule is changed it updates the bound dataset. However since the dataset is bound to another schedule, the second schedule is notified that the dataset was updated and the second schedule is updated as a result. This functionality allows you to synchronize multiple schedules simply by binding each to the same dataset.

Appointment Data Table

Field	Data Type	Required / Non-Null
Appointment_Guid	String	Yes (pk)
Start_Date	DateTime	Yes
Length	Integer	Yes
Blockout	Boolean	No
IsEvent	Boolean	No
IsFlagged	Boolean	No
IsReadOnly	Boolean	No
MaxLength	Integer	No
MinLength	Integer	No
Notes	String	No
Priority	Integer	No
Recurrence_Guid	String	No
Room_Guid	String	No
Subject	String	No
Text	String	No
ToolTipText	String	No
Visible	Boolean	No
Alarm_Window_Text	String	No

Alarm_Is_Armed	Boolean	No
Alarm_Reminder	Integer	No
Alarm_Allow_Open	Boolean	No
Alarm_Allow_Snooze	Boolean	No
Alarm_Was_Dismissed	Boolean	No

Although appointments are the most important table, you may also bind to other dataset tables as well. The Room table must have a room_guid and name, both of type string, set to not accept null values. It may also have an optional notes field of type string. The Appointment table has a direct reference to the Room table. It has a Room_guid field that may be null, signifying no room is associated with the appointment or it may have a room_guid mapping to the Room table. This is a one-to-many relationship. An appointment is associated with exactly zero or one room.

Room Data Table

Field	Data Type	Required / Non-Null
Room_Guid	String	Yes (pk)
Name	String	Yes
Notes	String	No

The Provider table has the same format and rules of the Room table with the additional, non-nullable, required integer field of Color. Both the Provider and Category tables differ in their association to the Appointment table as compared to the Room table. While an appointment can have at most one room, it can have any number of providers or categories associated with it. This is a many-to-many relationship. Because of this more complicated relationship, there is needed an intermediary table necessary. For the Provider table the intermediary is the Appointment_Provider table. This table holds nothing more a reference to the appointment and provider tables. These two fields make up a composite, non-unique key. The Category and Appointment_Category tables are much the same.

Provider Data Table

Field	Data Type	Required / Non-Null
Provider_Guid	String	Yes (pk)
Name	String	Yes
Color	Integer	Yes
Notes	String	No

Appointment_Provider Table

Field	Data Type	Required / Non-Null
Appointment_Guid	String	Yes (pk)
Provider_Guid	String	Yes (pk)

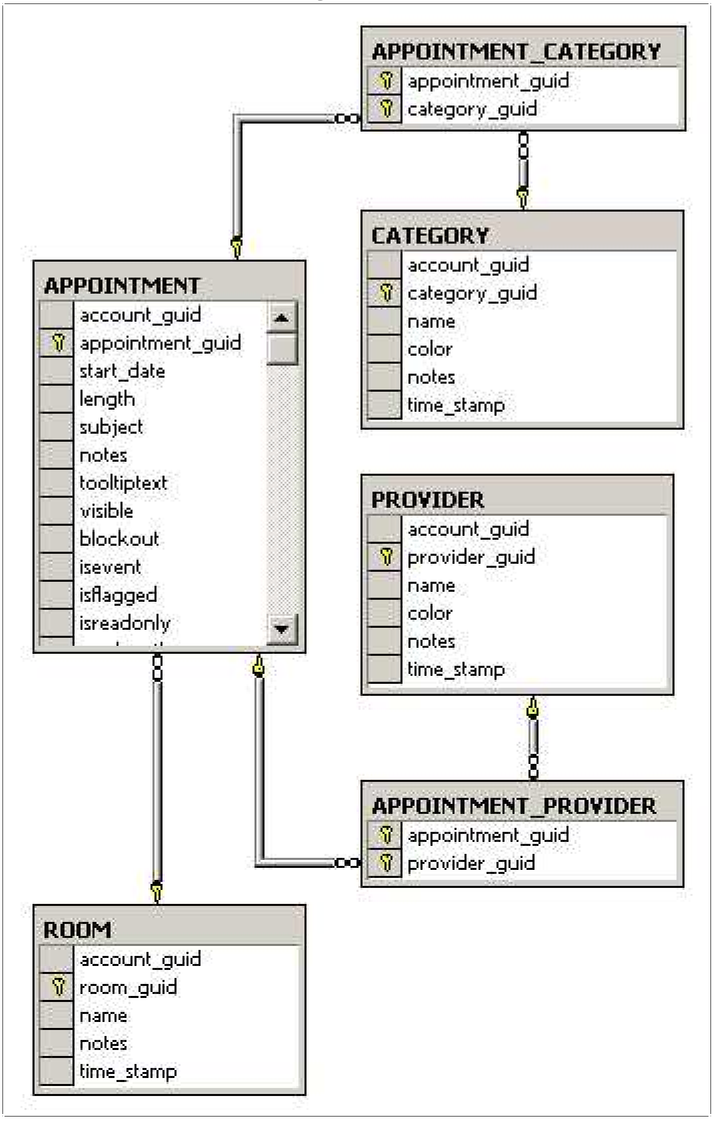
Category Data Table

Field	Data Type	Required / Non-Null
Category_Guid	String	Yes (pk)
Name	String	Yes
Color	Integer	Yes
Notes	String	No

Appointment_Category Table

Field	Data Type	Required / Non-Null
Appointment_Guid	String	Yes (pk)
Category_Guid	String	Yes (pk)

Figure 8.1



Non-Schedule Data

In the course of building an application that supports scheduling, you will undoubtedly want to store data that the schedule does not support. For example, you may want to associate extra information with each appointment like person_id, visit_notes, last_update, etc. The schedule does not support these properties. There is a remedy for this situation. The PropertyItemCollection is a collection of objects of type PropertyItem. These objects are nothing more than a name and a value. You can think of this collection as a set of name/value pairs. You may associate any data you wish to an appointment using this collection.

The collection has the methods Add and Remove. You can add or remove any value you wish and not receive any error. You do not need to data bind to use this collection to store extra information with an appointment; however it becomes even more useful when used with data binding. The database Appointment table can contain any fields you wish. The defined ones map to specific appointment properties link start_date, length, etc. However if you add a non-standard field like "person_id" to the Appointment table, the schedule component does not recognize it, so it is placed in the PropertyItemCollection. In other words all non-standard fields in the Appointment table are created as a name/value pairs and placed in the PropertyItemCollection of the associated appointment. If you set the value of one of these objects, it will be persisted back to the database with no developer interaction! You do not need to write any code to get database persistence functionality of custom database fields. This is built-in.

The following code assumes that you have a field in your Appointment table named "person_id". It is pulled off of the appointment and put in a string variable.

```
    'Get the first appointment in the collection
    Dim appointment As Appointment
     appointment = Schedule1.AppointmentCollection(0)

    'Get the property value and store it for later
    Dim personId As String = ""
     personId = appointment.PropertyItemCollection("person_id").Setting()
```

This functionality does not only apply to appointments. Any extra fields added to the Provider, Category, or Room table triggers the same functionality. The extra fields are added to the PropertyItemCollection of the respective provider, category, or room objects.

Non-Standard Datasets

When developing an application that needs to interface with an existing database, you will most likely want to use your existing database format. As described above the schedule expects a very specific dataset format. Unless your database model has the Appointment, Room etc tables defined with the appropriate field names, the schedule will not bind to it. For many developers this is unacceptable. Fortunately, the schedule has a built-in way to address this problem. The DataBindings object lets you define the datatable for each bindable collection and each field of datatable.

In the example below I assume that you have a table name “Engagement” instead of the standard “Appointment” table. I will also map the field names in the “Engagement” table to the appropriate fields that the schedule expects.

“Appointment” Table Field	**“Engagement” Table Field**
Appointment_Guid	Key
Start_Date	Starting
Length	Duration
Subject	Text

The code below assumes that you have loaded a dataset named “MyDataset” with the “Engagement” table from your database.

```
Dim MyDataset As DataSet

'Populate the Dataset
'...

'Manually bind the datatable to the AppointmentCollection
schedule1.DataBindings.AppointmentBinding.DataSource  =  ds.Tables("Engagement")

'Setup the AppointmentBinding object
schedule1.DataBindings.AppointmentBinding.DataFieldBindingCollection.Add("appointment_guid
"Key")
schedule1.DataBindings.AppointmentBinding.DataFieldBindingCollection.Add("start_date",
"Starting")
schedule1.DataBindings.AppointmentBinding.DataFieldBindingCollection.Add("length",
"Duration")
schedule1.DataBindings.AppointmentBinding.DataFieldBindingCollection.Add("subject",   "Text"

Schedule1.DataSource = MyDataset
Schedule1.Bind()
```

There is a binding object for each table to which data can be bound. You may bind up to eight tables: Appointment, Category, Provider, Room, Resource, Appointment_Category, Appointment_Provider, and Appointment_Resource. When a DataTable does not match the standard name, you must set the table binding as shown above with the “AppointmentBinding” object. There is one for each of the eight tables. You may map as many or as few fields as you wish so long as you map the required ones. If a field in your DataTable happens to have the standard name, there is no need to remap it in the DataBindingCollection object as it will be automatically recognized.

There are a few rules to remember when working with mappings. First, you cannot create a mapping and add it to more than one collection. You can create a binding and add it to a DataBindingCollection object or you can specify the binding settings from the Add method. Also once a DataBinding object has been added to a collection it cannot be changed. Lastly, only valid fields can be added to the DataBindingCollection object based on the table to which it maps. For example, you cannot map the room property “PhoneNumber” to anything since the room has no such property.

Controllers

If you are data binding to a local database, you can bind simply by selecting the appointment table, with the proper fields. This is quick and will work, but you will most likely want more robust functionality. The schedule can natively process a dataset with the tables: Appointment, Provider, Room, Category, Resource, Appointment_Provider, Appointment_Category, and Appointment_Resource. If you use the database scripts provided with the install, you can create all of these tables with the proper format. They are related as described in Figure 8.1 above.

ScheduleController

Gravitybox provides a companion component named ScheduleController that is included in with the install and allows you to connect to a database with no code. The controller, given a connection string, connects to the specified database and queries for providers, rooms, categories, and appointments. You need only one line of code to load and one line of code to save. The code below demonstrates.

```
'Get all appointments
ScheduleDomainController1.GetScheduleDataSet("")

'Get appointments from a date range
Schedule1.DataSource = ScheduleDomainController1.GetScheduleDataSet( _
                       "", #1/1/2005#, #1/31/2005#)
Schedule1.Bind()

'Get a specific appointment by IDSchedule1.DataSource =
Schedule1.DataSource = ScheduleDomainController1.GetScheduleDataSet( _
                       "", "163c61b3-504e-4e45-b1af-ebfe205fa141")
Schedule1.Bind()
```

There are currently three different ways to query the database based on different criteria. They are shown above. You only need to call one of them to load the schedule. There is however only one way to update the database as follows.

```
ScheduleDomainController1.UpdateData("",  Schedule1.DataSource)
```

When the dataset is updated, you need to immediately re-query the database to avoid getting an error.

You will also notice that the first parameter is set to empty string. In the database schema defined there is an “account_guid” that parses all objects into account buckets. This allows you to build applications that load only data specific to one account. This is quite useful if you are building an application that you wish to use to host different, distinct users. For example, you may want to build an application for dentists and also you want to host the dentist’s data. If you sell your application to multiple dentists then you need some way to separate each dentist data. None of the doctor practices need (nor do you want them) to know about each other. You could host all of the offices’ data in the same database and no office would be aware of any other office, since each piece of data is marked with a specific account. In the above example, I assume that you have only one account and it is empty string.

The ScheduleController makes all of this seamless. You need only setup your database properly. This means making sure that at least the minimal amount of fields are in table and the proper stored procedures are present. This is all provided as SQL scripts in the installation application.

WebController

The ScheduleController is fine if you have direct access to the database server. However, in today's world of distributed applications and global networks, often it is desirable to build robust client applications that integrate with remote databases. You can write your own remoting objects to create proxies that your application accesses across the Internet, or you can use the Gravitybox WebController. This companion component works directly with the Gravitybox Enterprise Service. The use of this software is beyond the scope of this document. It is designed to build enterprise level applications. If you are interested in building distributed scheduling applications please view our Enterprise document.

Part IV How do I…

Applying computer technology is simply finding the right wrench to pound in the correct screw.

-Unknown

The computer only crashes when printing a document you haven't saved.

-Unknown

The only thing in life achieved without effort is failure.

-Unknown

Chapter 9 How do I…

Chapter 9

How do I…

How do I add an appointment (room, category, etc.) in code?

Adding an appointment or any other object in code is easy. All you need to do is call the "Add" method on the collection to which you wish to add an object. Code to add an appointment to the Appointments collection is below. The added appointment will be scheduled for Jan1, 2004 at 10:00AM for a 1-hour duration.

```
Schedule1.AppointmentCollection.Add("", #1/1/2004#, #10:00:00 AM#, 60)
```

To add a category, provider, or room is may be performed with one line of code as well as follows.

```
Call Schedule1.CategoryCollection.Add("", "MyCategory")
Call Schedule1.ProviderCollection.Add("", "MyProvider")
Call Schedule1.RoomCollection.Add("", "MyRoom")
```

Each Add method's first parameter is a key (string). If this parameter is not specified (empty string) then one is assigned to the object when it is created.

How do I associate a room with an appointment?

Each appointment may be assigned to exactly 0 or 1 room. If no room is associated then the appointment wil not be displayed when the schedule is in a viewmode that shows room information. However if a room is associated with an appointment, it will be shown in its proper room in these viewmodes. The following example adds 1 room and then adds 1 appointment. It sets the appointment's room property to the newly created room object.

```
Dim room As Room = Schedule1.RoomCollection.Add("", "MyRoom")
Dim appointment As Appointment = Schedule1.AppointmentCollection.Add("",
                                 #1/1/2004#, _
                                 #10:00:00 AM#, 60)
appointment.Room = room
```

How do I add categories and providers to an appointment?

Unlike rooms where only 1 object may be associated with an appointment, any number of categories or providers may be associated with an appointment. The following example adds 10 categories and provider to the schedule's CategoryCollection and ProviderCollection. These collections must be populated because appointments cannot create new category or provider objects. Each appointment can only point at existing objects. The example adds 3 categories to the appointment’s CategoryList object and 3 providers to the appointment’s ProviderList object.

```
Dim ii As Integer
For ii = 1 To 10
  Call Schedule1.CategoryCollection.Add("", "Category" & ii.ToString())
  Call Schedule1.ProviderCollection.Add("", "Provider" & ii.ToString())
Next

Dim appointment As Appointment = Schedule1.AppointmentCollection.Add("",
#1/1/2004#, #10:00:00 AM#, 60)
Call appointment.CategoryList.Add(Schedule1.CategoryCollection(2))
Call appointment.CategoryList.Add(Schedule1.CategoryCollection(5))
Call appointment.CategoryList.Add(Schedule1.CategoryCollection(7))
Call appointment.ProviderList.Add(Schedule1.ProviderCollection(1))
Call appointment.ProviderList.Add(Schedule1.ProviderCollection(6))
Call appointment.ProviderList.Add(Schedule1.ProviderCollection(9))
```

How do I import/export an appointment to/from a VCAL file?
The VCAL file format is an international standard that is used to exchange calendar information. The GbSchedule.NET component supports importing and exporting this file format. The code snippet below exports the entire AppointmentCollection to this file format. The method “ToVCAL” is overloaded and may save any number of appointments to this file format. When importing a file, all appointments in the file are added to the AppointmentCollection object.

```
Call Schedule1.AppointmentCollection.ToVCAL("c:\test.vcal")
Call Schedule1.AppointmentCollection.FromVCAL("c:\test.vcal")
```

How do I get an appointment from coordinates?
The method “GetAppointmentFromCor” on the ScheduleVisibilty object will return an appointment under a pair of coordinates, if one exists. The code snippet below demonstrates this.

```
Dim appointment As Appointment =
Schedule1.Visibility.GetAppointmentFromCor(150, 200)
```

How do I save/load an appointment to/from an XML stream?
One or more appointments may be saved to an XML stream. This is done using the “ToXML” function of an appointment object.

```
Dim s As String = Schedule1.AppointmentCollection.ToXML()
```

Also the AppointmentCollection object allows you save the entire collection is an XML file using the “SaveXML” method.

```
Call Schedule1.AppointmentCollection.SaveXML("c:\test.xml")
```

A single appointment may be saved to an XML stream using its “ToXML” method.

```
Dim s As String = appointment.ToXML()
```

How do I get the index of an object in its parent collection?
There are times when you may need to know an object’s index in its parent collection. This information may be obtained from the parent collection itself. This paradigm holds true for all object collections. A return value of (-1) indicates that the object does not exist in the collection.

```
Dim index As Integer = Schedule1.AppointmentCollection.IndexOf(appointment)
```

How do I query the appointment collection?
The AppointmentCollection has a method “Find” that may be used to return an AppointmentList. The code snippet below demonstrates two ways to call the method. The first returns all appointments for Jan 1, 2004. The second example shows how to find all appointments with a subject set to the value “MySubject”.

```
Dim al As AppointmentList
al = Schedule1.AppointmentCollection.Find(#1/1/2004#, #1/1/2004#)
al = Schedule1.AppointmentCollection.Find("MySubject")
```

How do I combine multiple lists to get a single list?
The AppointmentCollection’s method “Find” can be used to return list based on many different criteria. Quite often you may wish to combine lists in different ways. For example you may wish to find all appointments between Jan 1, 2004 and Jan 10, 2004. In addition you may wish to narrow it further by appointments in some room. You could also go further in depth based on provider or category. This can be done easily by combining AppointmentList objects. The code snippet below finds all objects in the date range Jan 1, 2004 through Jan 10, 2004. It also finds appointments in a particular room.

```
Dim room As Room
room = Schedule1.RoomCollection("Room1")

Dim list1 As AppointmentList
Dim list2 As AppointmentList
list1 = Schedule1.AppointmentCollection.Find(#1/1/2004#, #1/10/2004#)
list2 = Schedule1.AppointmentCollection.Find(room)

Dim listUnion As AppointmentList
Dim listIntersection As AppointmentList
Dim listDifference As AppointmentList

'All appointment in either list
listUnion = list1.Union(list2)
'Only object that are in both lists
```

```
listIntersection = list1.Intersect(list2)
'All object in list 1 minus all object in list2
listDifference = list1.Subtract(list2)
```

How do I cancel the default property dialog?

Before the default dialog is displayed the event "BeforePropertyDialog" is raised. It has a Boolean cancel parameter that may set to true to cancel the dialog.

How do I cancel other dialog screens?

Each dialog is displayed only after some "Before…" event is raised. You may cancel any dialog from its associated "Before…" event. The relavant events are as follows: BeforePropertyDialog, BeforeCategoryListDialog, BeforeProviderListDialog, BeforeProviderConfigurationDialog, BeforeCategoryConfigurationDialog, BeforeRoomConfigurationDialog.

How do I display the default dialogs?

You may display any of the built-in dialogs in code using the schedule's "Dialogs" object. There are eight built-in dialogs and they may be displayed with the following methods: ShowAboutDialog, ShowAlarmDialog, ShowCategoryConfiguration, ShowCategoryDialog, ShowPropertyDialog, ShowProviderConfiguration, ShowProviderDialog, ShowRoomConfiguration.

How do I display the configuration screens?

All of the configuration screens may be displayed from code. In fact they may only be displayed from code since there is no way for the user to show them. The following code snippet displays the category, provider, and room configuration screen respectively.

```
Call Schedule1.Dialogs.ShowCategoryConfiguration()
Call Schedule1.Dialogs.ShowProviderConfiguration()
Call Schedule1.Dialogs.ShowRoomConfiguration()
```

How do I ensure a day (time, room, etc.) is visible in the viewport?

The schedule has a visibility object that can bring any portion of the schedule into the viewport. The "Show…" method take a parameter that identifies an area to display. If possible the area is brought into the viewing area. The code snippet below ensures that a date and a time are visible.

```
Call Schedule1.Visibility.ShowDate(#1/1/2004#)
Call Schedule1.Visibility.ShowTime(#11:00:00 AM#)
```

How do I get the first visible date (time, room, etc) in the viewport?

The schedule's visibility object also can be used to determine what is the first date, time, etc that is visible in the top, left corner of the schedule. The code snippet below retrieves the first visible date and time for the schedule.

```
Dim theDate As Date = Schedule1.Visibility.FirstVisibleDate()
Dim theTime As Date = Schedule1.Visibility.FirstVisibleTime()
```

How do I check for space availability?

The AppointmentList method IsAreaAvailable allows you to check if any appointment overlaps an area. If no appointment in the specified list overlaps the defined space then the method return true. If one or more appointments cover any part of the defined area then false is returned. The code below creates a list from the entire AppointmentCollection effectively verify if a schedule area is available for the whole schedule.

```
If  Schedule1.AppointmentCollection.ToList.IsAreaAvailable(#1/1/2004#,  _
      #10:00:00 AM#, 60) Then
  'Do Something...
End If
```

How do I determine if a time slot is enabled?

Before you add an appointment in code, you may wish to determine if the time slot is inside of a NoDropArea. The schedule's NoDropAreaCollection object may be used to determine if a slot if enabled or not. The code snippet converts the NoDropAreaCollection to a ScheduleAreaList and then checks it to determine if Jan 1, 2004, 10:00AM for 1 hour is enabled or not.

```
If   Schedule1.NoDropAreaCollection.ToList.IsOverlap(#1/1/2004#, _
         #10:00:00 AM#, 60) Then
  'Do Something...
End If
```

How do I get a list of conflicts for an appointment?

The schedule's AppointmentCollection can be converted to an AppointmentList and searched for a list of conflicts. Given a specific appointment, the list's GetConflicts method returns another AppointmentList of appointments that conflict with it. If there are no conflicts, the returned list has no objects in it.

```
Dim al As AppointmentList
al  =  Schedule1.AppointmentCollection.ToList.GetConflicts(appointment)
```

How do I determine if two appointments conflict?

The AppointmentCollection object can be used to determine if a specific appointment conflicts with any other appointment in the collection. The code snippet below demonstrates this.

```
Dim appointment As Appointment
'Add some appointments...
```

```
If  Schedule1.AppointmentCollection.IsConflict(appointment)  Then
  'Do Something...
End If
```

If you wish to only check conflicts between two appointments and not the whole collection, you may use the IsConflict method of an appointment object.

```
Dim appointment1 As Appointment
Dim appointment2 As Appointment
'Add some appointments...
If appointment1.IsConflict(appointment2) Then
  'Do Something...
End If
```

How do I attach additional, custom information to an object (appointment, category, etc.)?
Most objects in the GbSchedule.NET framework have a child object: PropertyCollection. This is a collection of name/value pairs. You may store any number of values with a key, name, and value. These are all string values. You may reference the collection by key, name, or index [0..N-1]. This collection is very useful if you have custom information that you need to associate with an object.

```
Dim appointment As Appointment
'Add some appointments...
appointment.PropertyCollection.Add("", "MyProperty", "MyValue")
```

How do I ensure that an appointment's length will be in a specified range?
When the user is resizing an appointment, he can set its length to any value. If you need an appointment appointment's length to be in a specified range this can be a problem. Each appointment has a MinLength and MaxLength property that is used to enforce its length. When either value is (-1) that value, either min or max, is not enforced. The code snippet below ensures that the appointment's length will always be between 1 and 2 hours.

```
Dim appointment As Appointment
‘Add some an appointment…
appointment.MinLength = 60
appointment.MaxLength = 120
```

How do I ensure that a specified area never receives appointments?
The NoDropAreaCollection allows you to add areas of the schedule where appointments are invalid. These defined areas will be a specified color and appointment moves or copies are not allowed. The code snippet below adds Jan 1, 2004 to the NoDropAreaCollection. You may also define combinations of dates, times, room, providers, etc.

```
Call  Schedule1.NoDropAreas.Add(Color.Red,  #1/1/2004#)
```

How do I set the alarm of an appointment in code?

An alarm may be displayed when an appointment comes due. The alarm may be set from the default property screen or in code. In code it can be set with the code snippet below.

```
appointment.Alarm.IsArmed = True
```

How do I customize an appointment, i.e. make it round, change the color, Transparency, etc?

An appointment has many properties to customize it appearance. The code snippet below demonstrates some of them.

```
'Make the appointment half see-through
appointment.Appearance.Transparency = 50
'Make the backcolor light blue
appointment.Appearance.BackColor = Color.LightBlue
'Make the text color black
appointment.Appearance.ForeColor = Color.Black
'Make the border this a system color
appointment.Appearance.BorderColor  =  SystemColors.ControlDark
'Make the font bold
appointment.Appearance.FontBold = True
'Draw the appointment as round not rectangle
appointment.Appearance.IsRound = True
```

Each appointment also has a header object. This object may be used to define a header for the appointment and set it properties.

```
'Draw a line under the header
appointment.Header.Appearance.AllowBreak = False
'Make the backcolor red
appointment.Header.Appearance.BackColor = Color.Red
'Draw the appointment's date in the header
appointment.Header.HeaderType =
AppointmentHeader.HeaderTypeConstants.DateHeader
'Set the header icon to the default "Info" icon
appointment.Header.Icon = Schedule1.DefaultIcons.IconInfo
'Make the appointment half see-through
appointment.Header.Appearance.Transparency = 50
```

How do I determine how many days an appointment spans?

Appointments may overlap any number of day boundaries. To determine the number of breaks use the “DaySpan” appointment property. This property is not the number of full days that the appointment covers. It is the number 12:00AM boundaries that the appointment covers. For example, an appointment from 1:00AM on Jan 1 to 11:00PM Jan 2 covers 46 hours but it only crosses 1 day boundary.

```
Dim days As Integer = appointment.DaySpan
```

How do I display a colored bar next to an appointment?

There are four ways to configure an appointment's left side bar: none, category, provider, or user-drawn. The first, none, displays no bar at all. The second, category, displays a colored bar the same color as the first category object in the appointment's CategoryList object. If no categories are present in this list, no bar is drawn. The third, provider, displays a colored bar the same color as the first provider object in the appointment's ProviderList object. If no providers are present in this list, no bar is drawn. The last, user-drawn, is quite interesting. It allows you receive notification through the "UserDrawnBar" event when an appointment bar needs to be drawn and allows you to draw your own custom bar.

```
Schedule1.AppointmentBar = _
      Gravitybox.Controls.Schedule.AppointmentBarConstants.None
Schedule1.AppointmentBar = _
     Gravitybox.Controls.Schedule.AppointmentBarConstants.Category
Schedule1.AppointmentBar = _
      Gravitybox.Controls.Schedule.AppointmentBarConstants.Provider
Schedule1.AppointmentBar = _
      Gravitybox.Controls.Schedule.AppointmentBarConstants.UserDrawn
```

How do I draw a custom bar next to an appointment?

When the AppointmentBar property is set the UserDrawn, the "UserDrawnBar" event is raised when the bar needs to be drawn. You, the developer, are responsible for drawing the bar at the proper location. The event exposes the graphics object used to draw the bar. It also exposes the valid rectangle in which to draw. The rectangle defines the appointment bar rectangle. Do not draw outside the rectangle as your changes may or may not be overdrawn.

How do I speed-up appointment loading?

If many appointments are loaded at one time you may experience a slowness or drag. This is because the screen may be repainting many times upon loading. To ensure the fastest loading use the "AutoRedraw" property. Toggle it to false to turn off screen painting and then load the schedule. Be sure to toggle it back to true or the screen will never repaint.

```
Schedule1.AutoRedraw = False
'Add Appointments...
Schedule1.AutoRedraw = True
```

How do I mark an appointment as read-only and have is displayed with no details visible?

If you wish to show an appointment space as taken, but do not want the user to see any details of the appointment, you should use an appointment's BlockOut property. It ensures that the appointment drawn as a solid colored block defined by the schedule's BlockOutColor. No text or icons are displayed on the appointment. The user cannot move, copy, or otherwise interact with the appointment. The appointment simply becomes a placeholder to inform the user that the space is taken but he has no business know any other information about the appointment.

```
appointment.Blockout = True
```

How do I override the displayed header (column or row) text?

There are three events that are displayed before header are drawn: BeforeRoomHeaderDraw, BeforeProviderHeaderDraw, and BeforeDateHeaderDraw. These events allow you to change the text drawn in the header if desired. The change will not change the text permanently. It will only be changed on the screen and not in the header object.

How do I get notification when an appointment's alarm comes due?

The event "BeforeAppointmentDue" is raised just before the default alarm dialog is displayed. You may cancel the dialog from this event. In addition the "BeforeAppointmentReminder" event is raised some number of minutes before (or after) the "BeforeAppointmentDue" event determined by the appointment's "Reminder" property. The Reminder property is the number of minutes before an appointment to display the reminder screen. If this value is 0 then no reminder is displayed. If this value is negative then the reminder occurs after the appointment comes due.

How do I override the displayed appointment text?

The event "BeforeAppointmentTextDraw" is raised just before each appointment's subject text is drawn. You may change the displayed text from this event is necessary. Overriding the text will not change the appointment object Subject property. It will only change the text drawn on the screen. The change has no persistence.

How do I get notification when an appointment is added, copied, moved, or deleted?

There are numerous events that provide notification upon an action occurring. Many times there are complementary events that begin with "Before" and "After". These events work in pairs to notify the container before an action occurs and after that action has occurred. In most cases the action may be canceled in the "Before…" event. Below are a few event actions and there associated events.

Action: Appointment Add
BeforeAppointmentAdd
AfterAppointmentAdd

Action: Appointment Copy
BeforeAppointmentCopy
AfterAppointmentCopy

Action: Appointment Move
BeforeAppointmentMove
AfterAppointmentMove

Action: Appointment Remove
BeforeAppointmentRemove
AfterAppointmentRemove

How do I get notification when the column or row headers are resized?
When a column or row header is resized a pair of events are raised to notify the container. Listed below is the event sequence for each.

Action: Column Resize
BeforeColumnResize
AfterColumnResize

Action: Row Resize
BeforeRowResize
AfterRowResize

How do I get notification when a non-appointment is dropped on the schedule?
When a non-appointment object is dropped on a schedule the event pair "BeforeForeignAdd" and "AfterForeignAdd" is raised. When the container receives these events it is a certainty that some other object has been dropped on the schedule. If this is an illegal operation then cancel the drop in the BeforeForeignAdd event.

How do I get notification of the user edits an appointment's text in grid?
If the property "AllowInPlaceEdit" is set to true, the user may click a selected appointment to start an in-grid edit. When this text is saved the event "AfterInPlaceEdit" is raised. To cancel an in-place edit before it starts use the "BeforeInPlaceEdit" event.

How do I get notification of the user clicking an appointment's header icon?
The event "AppointmentHeaderInfoClick" is raised when the user clicks in the header icon of an appointment. For this event to happen an appointment must have a header and it must have a header icon. The following code snippet adds an appointment and creates a header and icon for it. You may wish to use a header icon to inform the user of some information. If you wish to perform any action on its click use this event.

```
Dim appointment As Appointment
appointment.Header.HeaderType = _
     AppointmentHeader.HeaderTypeConstants.DateHeader
appointment.Header.Icon  =  Schedule1.DefaultIcons.IconInfo
```

How do I toggle the schedule time format between the 12 and 24-hour clock?

Different countries have different clock formats. The schedule will display time in both the 12-hour or 24-hour clock formats. Use the “ClockSetting” property to change the time format.

```
Schedule1.ClockSetting = _
     Gravitybox.Controls.Schedule.ClockSettingConstants.Clock12
```

How do I resize column and row header in code?

The user may resize the column and row header in code by using the two objects: ColumnHeader and RowHeader. Each has a size property that is measured in pixels. Set the size of each using this property.

```
Schedule1.RowHeader.Size = 30
Schedule1.ColumnHeader.Size = 80
```

How do I make the columns (or rows) resize automatically?

If you wish for either the rows or columns to resize automatically use the AutoFit property of the ColumnHeader and RowHeader object. Keep in mind that columns and rows have a minimum size and the AutoFit will not always fit them perfectly. If you have 200 columns on your schedule, scroll bars will still be necessary and the columns will be the minimum allowable size. However for smaller sets of data this property will ensure that the number of columns or rows fits the total schedule area on screen.

```
Schedule1.RowHeader.AutoFit = True
Schedule1.ColumnHeader.AutoFit = True
```

How do I determine if the mouse is over an appointment?

If you need to determine if the mouse is over an appointment, use the HitTest method. This is quite useful in the MouseMove event. You may wish to display some information when the user moves over an appointment. The following code assumes that it is in the Schedule’s MouseMove event. It checks the “e” parameter to determine if an appointment is at the coordinates and if so displays a message.

```
Dim appointment As Gravitybox.Objects.Appointment
appointment  =  Schedule1.AppointmentCollection.HitTest(e.X,  e.Y)
If Not (appointment Is Nothing) Then
  Debug.WriteLine(“Over an appointment!”)
End If
```

How do I customize conflict displays?

In many cases, conflicting appointments are a part of life. GbSchedule.Net allows you to display conflicts in three different ways. (1) They can be overlapped. Essentially there is no processing since an appointment merely overlaps others that conflict. (2) They can be displayed side-by-side. This ensures that all appointments are seen. Each appointment is drawn with a smaller width to allow other conflicting appointments to be seen. (3) An advanced display is staggering. Appointments are slightly overlapped but their widths as slightly smaller as well. This looks much like the iCal application from Apple.

```
Schedule1.ConflictDisplay = _
     Gravitybox.Controls.Schedule.ConflictDisplayConstants.Overlap
Schedule1.ConflictDisplay = _
       Gravitybox.Controls.Schedule.ConflictDisplayConstants.SideBySide
Schedule1.ConflictDisplay = _
       Gravitybox.Controls.Schedule.ConflictDisplayConstants.Stagger
```

How do I print or preview a schedule?

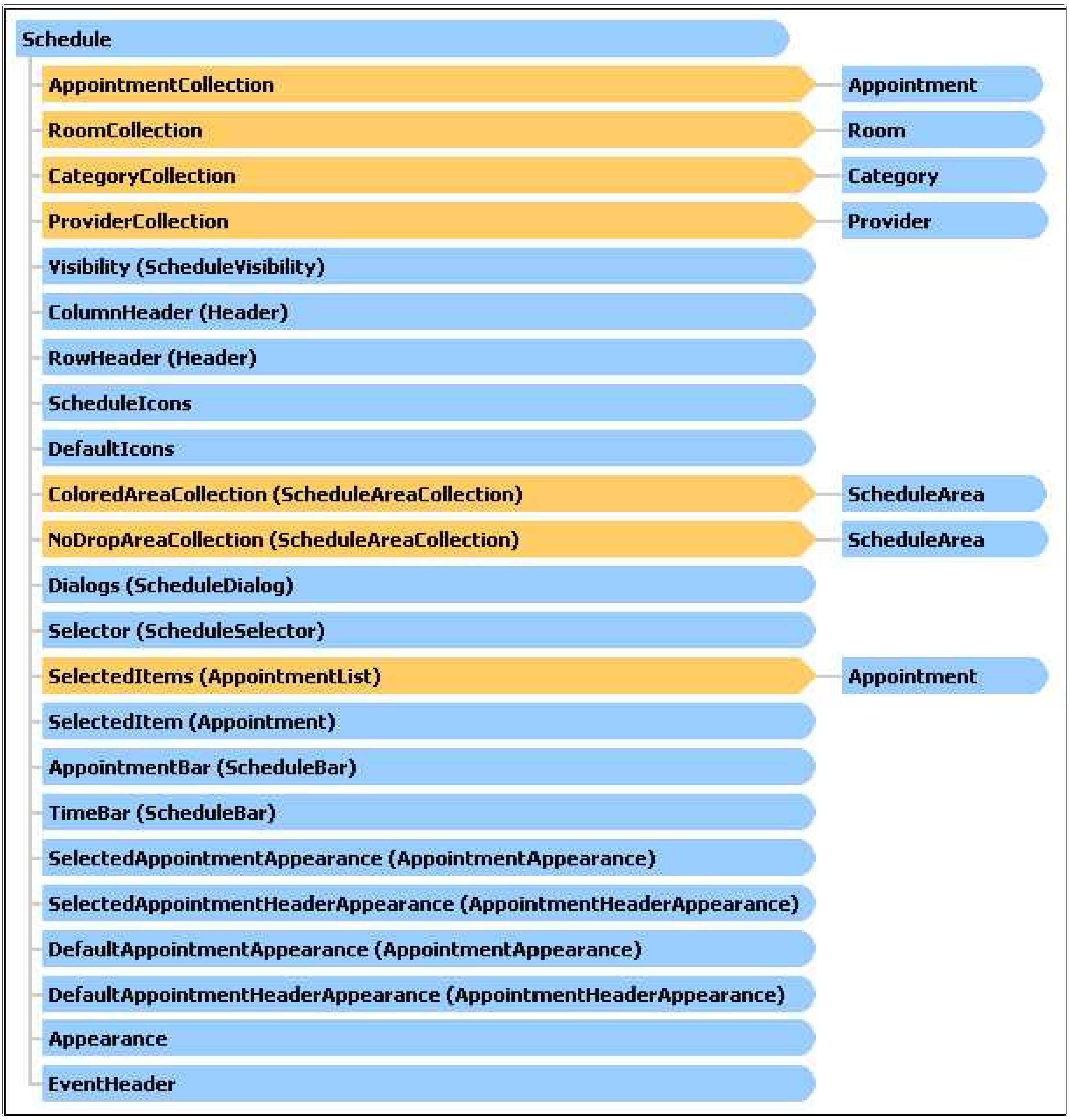
A schedule may be printed as easily as calling the GoPrint method. However this method is overloaded and may take a “PrintDialogSettings” object to customize the area of print. Previewing the schedule works the same way. It is overloaded in the exact same way as the GoPrint method.

```
Schedule1.GoPrint()
Schedule1.GoPreview()
```

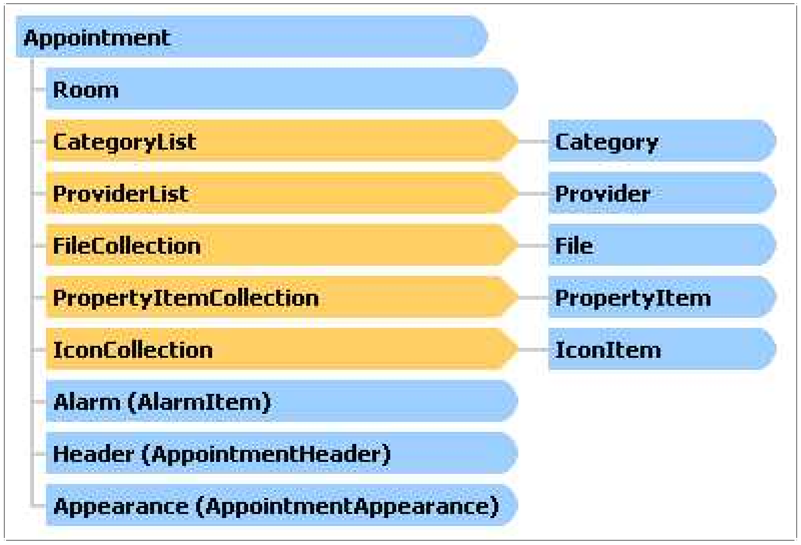
Part V
Appendix

Appendix

Schedule Control


Schedule
AppointmentCollection
Appointment
RoomCollection
Room
CategoryCollection
Category
ProviderCollection
Provider
Visibility (ScheduleVisibility)
ColumnHeader (Header)
RowHeader (Header)
ScheduleIcons
DefaultIcons
ColoredAreaCollection (ScheduleAreaCollection)
ScheduleArea
NoDropAreaCollection (ScheduleAreaCollection)
ScheduleArea
Dialogs (ScheduleDialog)
Selector (ScheduleSelector)
SelectedItems (AppointmentList)
Appointment
SelectedItem (Appointment)
AppointmentBar (ScheduleBar)
TimeBar (ScheduleBar)
SelectedAppointmentAppearance (AppointmentAppearance)
SelectedAppointmentHeaderAppearance (AppointmentHeaderAppearance)
DefaultAppointmentAppearance (AppointmentAppearance)
DefaultAppointmentHeaderAppearance (AppointmentHeaderAppearance)
Appearance
EventHeader

Appointment Object


Appointment
Room
CategoryList
Category
ProviderList
Provider
FileCollection
File
PropertyItemCollection
PropertyItem
IconCollection
IconItem
Alarm (AlarmItem)
Header (AppointmentHeader)
Appearance (AppointmentAppearance)