Developing with Gravitybox Schedule Gravitybox Software

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

Gravitybox Software
550 Arncliffe Court
Suite 100
Alpharetta, GA 30005

Voice-Mail and FAX (Toll-Free in the US):
877.GRAVCOM (877.472.8266)
(+1) 215.243.7686 (outside US)

Fax (UK):
0870.138.9550 (inside UK)
(+44) 870.138.9550 (outside UK)

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 is
http://www.gravitybox.com/download/gbschedule.exe.

Source Code Note
You may notice that the examples of this book are not complete. The relevant code has been listed for discussion; however I do not find it necessary to list thousands of lines of code that you are never going to read. All of the code and examples are available online, so there is no reason to print it. I hope that other writers in the future take this same methodology.

Table Of Contents

PART III Display Modes

PART IV Advanced Functionality

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 GbSchedule?

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 the 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 will not only display the information in a grid with multiple configurations but it may also be used to display scheduled information in MonthView or ListView formats as well. The component may show information in the MonthView format 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 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 event 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

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 almost be an expert in less than an hour. This allows you to add first-rate scheduling to your application with ease.

Goals of GbSchedule
The goals of GbSchedule 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

The software has been translated into many languages. The actual language text is small when compared to a commercial application. The only places in the component that are language specific are prompts. There are default dialogs, which may be overridden by the developer if need be.

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 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 and there is even a sample application provided for this purpose.

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 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 lets them 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

The GbSchedule Object Model

The Gravitybox Schedule has an extensive object model. It may seem complex at first but is actually very intuitive. There are seven main collections. Each holds some part of the information to be displayed on the screen.

ScheduleItems

Quick Tip
Appointments are stored in the ScheduleItems collection.

The main collection and the one that you cannot live without is the ScheduleItems collection. This is the group of appointments for a schedule. In its simplest form each object in the collection needs a starting time, date, and length. These three basic properties are used to display an appointment at its proper position. Many other properties are also available to define customized attributes of each object, but these three are the necessary ones for an object to exist. If a schedule is defined as the display of dates on one axis verses time on the other, then each one of these ScheduleItem objects has a particular position on which to be placed.

Table 1.1
ScheduleItem Object Definition

Alarm	This property determines if the appointment should raise an event at its StartTime. If this property is set, there is an icon displayed in the upper, left corner of the appointment and the ScheduleItemStart event is raised when the appointment's starting time comes due.
AlarmReminder	This property is used to fire the event ScheduleItemReminder event some time before an appointment comes due. This is a positive number of minutes. If the Alarm property is true, the AlarmReminder number minutes is subtracted from the appointment's start time. If the calculated time is the current time, this event is raised. This property basically allows you to receive an event some time prior to an appointment starting. This is useful for builder reminders that an appointment is due in X minutes.
BackColor	This property allows for the specification of a custom backcolor for the appointment. This may be useful, if you want a particular appointment to stand out. When an appointment is created, the backcolor is defaulted to white.
BlackOut	This property allows an appointment to be into a placeholder. The BlackOutColor will determine the appointment's color. No text will be displayed on the appointment and the user may not move, copy, or perform any operation on it.

Category	This property is a pointer into a collection of user-defined categories in the Categories collection. The appointment is said to be in the specified category.
ClusterId	This is a string value that allows you group appointments together without a recurrence pattern. If multiple appointments share the same GroupId property, they are displayed as recurring. However there are times when you may wish to group appointments programmatically without displaying the relationship on screen. You may use the “GetClustered” method to retrieve a list of all appointments with a unique “ClusterId” property value.
DisplayText	This is text that may be used to identify an appointment. Either the DisplayText or Subject may be used as the text displayed on the schedule canvas.
ExtraProperties	This is a collection of name/values pairs. You may add any number of elements to this collection and the values may be referenced by name or index. This allows you to store any amount of information with each appointment.
FontBold	Controls the appointment font’s bold property
FontItalics	Controls the appointment font’s italics property
FontStrikethru	Controls the appointment font’s strikethrough property
FontUnderline	Controls the appointment font’s underline property
ForeColor	This property allows for the specification of a custom color to be specified for the text of an appointment.
GroupId	This is a string value that uniquely identifies a group of appointments. Many appointments may have the same GroupId. This property is generated when a recurring appointment pattern is defined. The ScheduleItems "AddRecurrence" method may create many appointments with the same GroupId, to specify that the appointments are part of a group. This value may never be an empty string. You may use the “GetGrouped” method to retrieve all appointments with the same GroupId property value.
Id	This is a 32-bit integer that may be used to store extra information about an appointment. This is for developer convenience only. It has no effect on scheduling whatsoever.
IsActivity	This read-only property determines if the appointment overlaps a day boundary. If an appointment is scheduled past 12:00 AM on any day it becomes an Activity. Activities are displayed in the event header at the top of the screen, when time is displayed on the left side of the screen.
IsDirty	This property is a Boolean value that is set to True any time one of the object’s properties is changed. It is designed to be false when the object is loaded. It may be checked at any time as a way of identifying if the object has changed.

IsEvent	This property determines if the appointment is an event. Events do not have a StartTime or Length. They exist for a day only and do not have a definite start or end. An example of an event is a birthday. When true, the StartTime and Length will not have any genuine value. These appointments will be displayed only if the AllowEventHeader property is true. This property maps to the “All day event” text in the default property window.
IsFlagged	This Boolean value determines if a small flag is displayed next to the appointment. This icon may be used to indicate some condition to the user.
ItemData	This is a 32-bit integer that may be used to store extra information about an appointment. This is for developer convenience only. It has no effect on scheduling whatsoever.
Length	This property defines the length of the appointment. It is measured in minutes and must be greater than zero.
MaxLength	This property will ensure that a user may never resize an appointment to more than a certain size. The value represents the maximum number of minutes allowed for an appointment. The default value is –1. This default value places no restriction on the appointment. It effectively has no maximum length.
MinLength	This property is a supplement to the MaxLength property. It will ensure that a user may never resize an appointment less than a certain size. The default value is –1. This value places no restriction on the appointment. It effectively has no minimum length.
Name	This is an optional parameter used when adding an appointment to the ScheduleItems collection. If specified it must be unique or an error occurs. After an appointment is added, this property is read-only. You may use this identifier to access an element in the ScheduleItems collection.
Notes	This is an extra string value that may be used to save some additional information associated with this appointment object. It is not used by the schedule for any purpose.
Priority	This is a 32-bit integer that may be used to store an appointment’s priority. This is for developer convenience only. It has no effect on scheduling whatsoever.
Provider	This property maps to the Providers collection, a custom-defined collection of people that participate with the current schedule. This property may be set to a member of the Providers collection or left blank, if desired.
ReadOnly	This property determines if the appointment is read-only. If this is true then the user may not move, copy, or modify this appointment

Room	When the schedule is setup to display Rooms, only appointments with a valid Room property will be displayed. When the schedule is not displaying Rooms, this property is not used.
StartDate	This property is the date on which the appointment will be displayed. The valid date range is the MinDate property to the MaxDate property.
StartTime	This property defines the time an appointment is to start. Used in conjunction with the StartDate property, it determines the position of the appointment on the schedule. If the appointment is an event, this property has no genuine value.
Subject	This is text that may be used to identify an appointment. Either the DisplayText or Subject may be used as the text displayed on the schedule.
Tag	This is an extra string value that may be used to save some additional information associated with this appointment object. It is not used by the schedule for any purpose.
ToolTipText	This text is used to display a tooltip over an appointment when the mouse is at rest over it. This is the default value for the appointment's tooltip. If this property is not specified, it's DisplayText or Subject property is used, depending on the schedule's ToolTipDisplay property.
UniqueKey	This read-only property is a unique 32-bit integer that identifies this appointment from all the rest in the ScheduleItems collection. When an appointment is added, a unique key is generated and assigned to this property. It may not be changed and will be different on every load. However, the most important thing is that it never changes as long as the schedule is loaded. Even if dragged to another window, the appointment's UniqueKey will not change. It is provided to give the developer a way to uniquely identify an appointment in a schedule session.

A change in many of these properties will result in a display change of the appointment or the schedule. Some other properties of an appointment, though still important, are not display related. A change in non-display properties will not result in a repaint of the appointment or screen. They represent associated data that may be saved with an appointment. An example of this is the Id or ItemData property.

Some of the more interesting properties include Alarm, BlackOut, MinLength, and MaxLength. The Alarm property allows for the creation of applications that raise an event, at the start of an appointment. The ScheduleItemStart event is raised to inform of this happening. The index of the object in the ScheduleItems collection is returned, as a parameter to this event. The following code display a message box at each appointment's starting time.

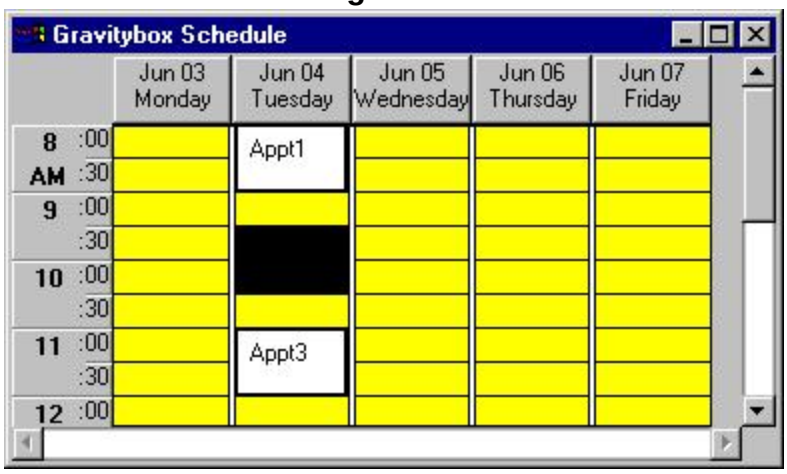
```
Private Sub Schedule1_ScheduleItemStart(ByVal Index As Long)

Dim sText As String

  sText = "The following appointment has started!" & vbCrLf
  sText = sText & Schedule1.ScheduleItems(Index).Subject
  Call MsgBox(sText, vbInformation)

End Sub
```

You may also receive notification some time before an appointment occurs. Each appointment has an “AlarmReminder” property. This is a non-negative number that represents the number of minutes ahead of an appointment starting time to notify the container. If this property value is non-zero, the ScheduleItemReminder event is raised this many minutes before the actual appointment start. This is most useful when building appointment reminders that prompts the user about an impending appointment.

Also notice that each appointment object has its own FontBold, FontItalics, FontStrikethru, and FontUnderline property. This allows you to control the look of each individual appointment. Use this in conjunction with the backcolor and forecolor properties to bring attention to any appointment. You may also use this to define appointment groups. Perhaps you have need to display multiple sets of appointments together on the same schedule but want to the user to be able to easily distinguish between them. You could give each group a unique set of colors and fonts.

The Blackout property permits appointments to be displayed that are in no way editable by the user. Also, the user may not view the contents of a blacked-out appointment either. This property is useful when an area needs to be shown as filled but the user has no access to the appointment. An example of this is a doctor office that has many doctors, each with his own secretary. In a networked system, one secretary can see that a particular room is filled at a particular time, but she has no access to view or modify the other doctor’s appointment. When the BlackOut flag of an appointment is set, it will be a solid color, determined by the BlackOutColor property of the schedule. Its text will not be shown. It is effectively blacked out.

Figure 1.1



In Figure 1.1, there are three, one-hour appointments. The first and third are visible and editable. The second has its BlackOut property set to true. The text of the appointment is set to “Appt2”, but this cannot be seen. The user cannot modify anything about the second appointment, nor can he even see the appointment’s text.

The ReadOnly property also performs this behavior, though in a different fashion. The ReadOnly property makes the appointment non-editable, but still displays the appointment’s information. This property is useful to display read-only versions of a schedule, while still allowing on one person to lock the schedule.

The MinLength and MaxLength properties complement each other. The default value for each of these is –1. This indicates that there are no restrictions imposed on the appointment’s length. In some applications, it may be necessary to limit the appointment’s length; minimum, maximum, or both. These two properties can be set independently and they influence each other in only one respect. The MaxLength must be greater than or equal to the MinLength (if MinLength is not -1) and the MinLength must be less than or equal to the MaxLength (if MaxLength is not -1). If either property is –1, it the value of the other property can be any positive integer.

An interesting new feature is the ability to display appointments that begin on one day and extend over a day boundary into another day. This is currently only available when the time is on left and day only is on top. The appointment is displayed in the event header at the top of the screen. To be displayed the “AllowEventHeader” property must be true. Also the schedule property “AllowActivities” must be true as well.

Rooms

In many circumstances, a date and time is enough to define a schedule for an application’s needs. However there are situations that require more than just a date and time. For example, in a doctor’s office, there may be two patients scheduled for 10:00 AM on October 23. If a schedule is used that defines only with dates and times, there

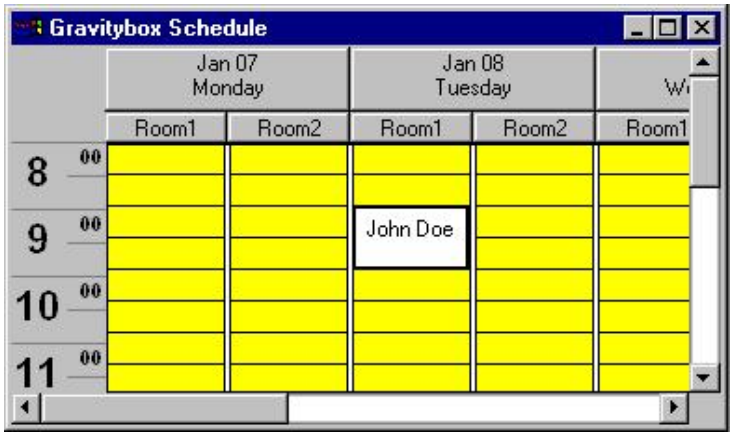
will be two conflicts at this particular time. Conflicts are not necessarily an error. There are times when a conflict is valid and even wanted. Nonetheless, there is a better way to display this information, for this office example. There exist two rooms in this office; this is the reason why two appointments are present for the same time. This means that in reality there is no conflict.

The Rooms collection defines a set of rooms that may be associated with an appointment. When the Viewmode is set to a property that supports dates and rooms, each day will have as a subcategory, the defined Rooms. If there are two rooms defined, the schedule will have the two rooms displayed, for each day on the schedule. Now when the two patients are displayed, there is no conflict, since each appointment's associated room makes it unique on the schedule.

Table 1.2
Room Object Definition

Id	This is a 32-bit integer that may be used to store extra data.
IsDirty	This property determines if an object has been modified. This property may be set to false upon load and checked before a conditional save. Changing the Id property or renaming the Room object through the Rooms collection will cause this property to be true.
Index	This read-only property is the numeric index of this object in its parent collection. The valid values are 1 to N, where N is the number of objects in the Rooms collection.
Name	This is the name, as well as the caption, of the Room object. This property is read-only after the addition; however it can be set indirectly by using the Rename method of the Rooms collection.

Figure 1.2



Each appointment has a Room property that may be set to an existing Room object's name or index. The property does not have to be mapped to anything; however if it does not map to anything and rooms are being displayed on the schedule, this appointment will not be shown. This property will never return an error for an incorrect

value. If the value does not map to any valid value in the Rooms collection, nothing will happen. The only error that may happen is one of logic. If you set this property to “Room1” and the name of the room you wish to map is “Room 1”, the appointment will not be displayed in the proper room. This may cause puzzling results, if you make this error, but no run-time error will be raised.

A room may be added to the Rooms collection with the Add method in the following fashion.

```
Call Schedule1.Rooms.Add(“Room1”)
```

The new room will be named “Room1”. This object may be referenced by name or index [1..N]. The following code fragments are equivalent.

```
Debug.Print Schedule1.Rooms(“Room1”).Name
Debug.Print Schedule1.Rooms(1).Name
```

A Room object’s name may not be numeric. This is because there would be no way to determine if the parameter being used is a name or an index. Since you may not add a numeric name the code below is invalid but I use it as an illustration. If an object’s name were numeric, the following code would produce an error, since there is only one object in the collection and its name is “100”. When referenced by the name “100”, the collection will try to return index 100, which does not exist, and produce an error.

```
Call Schedule1.Rooms.Add(“100”)
Debug.Print Schedule1.Rooms(“100”).Name ‘ERROR
```

This logic also applies to all object collections in this product. The name of no object may be numeric.

Rooms may not be renamed directly. By this I mean that the following code will not execute. It will not rename the first Room object in the collection to “NewName”.

```
Schedule1.Rooms(1).Name = “NewName”
```

This code will produce an error, because the Name property is read-only. If the name of existing Room needs to be changed, the collection’s Rename method must be used. If, as in the previous example, the first Room object is to be renamed “NewName”, the following code would work.

```
Call Schedule1.Rooms.Rename(1, “NewName”)
```

The Rooms collection actually has many useful methods. These may be used to manipulate Room objects within the collection. There are also search functions that make finding a particular object much easier. Additional language support for the displayed schedule may be provided as well, with the collection’s DisplayName

property. If an application's language is not English or different text needs to be displayed other than the default, the DisplayName property may be set to the desired text. This text will be used as the header of Room columns, on the schedule.

Table 1.3
Rooms Collection Definition

Add	This method takes parameters to create a new Room object and adds it to the Rooms collection. A reference it the created object is returned.
Clear	This method will clear the collection of all Room objects.
Count	This method will return the number of elements in the collection.
DisplayName	This is the text that will appear as the room header on the schedule. The default text is "Rooms", but may be set to any text preferred.
Exists	Given a Room object's name or index, this method will return true if a Room object exists, at the specified position in the collection.
Index	This method is very similar to the Exists method only it returns a number. Given a Room object's name or index, this method returns its index. Though valid syntax, it is not very useful to give the index in order to retrieve it. This function was designed to take the Name property as a parameter.
IsDirty	This property will determine if any modifications have occurred, since the last time it was set to false. If any property of any Room object is modified this property is automatically set to true.
Item	Given a Room object's name or index, this method will return a reference to the object. If the given parameter does not exist in the collection, an error is raised. To avoid an error, use the Exists method in conjunction with this method.
NewIndex	This read-only property is the index of the newest object added to the collection with the Add method. After the Remove method is called this property is reset to zero.
Remove	Given a valid index or name, this method will remove the specified room object.
Rename	This method will rename a Room object. It takes a parameter of an existing object's index or name and the new name to give the object. If the new name already belongs to another Room object, an error is raised.

NOTE: An appointment's Room property is not aware of changes in the Rooms collection. If an appointment has a Room property of "2", referring to the second room in the Rooms collection and this element is removed, the appointment will still point to the

old (now non-existent) Room object. If there is not a second Room and the schedule is displaying rooms, the appointment will not be displayed.

Categories

The schedule object also has a Categories collection. This collection may be filled with custom categories that define a business. This functionality is very useful when categorizing appointments. If the appointments can be grouped into categories, this will give the users a much better view of the schedule.

To go back to the doctor's office demo, say there are three categories, "Surgery", "Major", and "Misc". These categories are added to this collection and an associated color with each.

```
Call Categories.Add("Surgery", vbRed)
Call Categories.Add("Major", vbYellow)
Call Categories.Add("Misc", vbBlue)
```

Much like the Rooms collection, a Category object may be referenced by its name or index. Each appointment has a Category property that maps to this collection. An appointment need not have an associated category, but this functionality is provided for convenience.

This category information is used to display the appointment with a slightly different appearance. If its Category property of an appointment maps to a valid object of the Categories collection, the associated Category object's color is displayed on the left side of the appointment. In addition to displaying a category, colored bar on the appointment, there will be a matching bar in the left margin of the schedule. This allows the user to see, at a glance, the categories of appointments. In the left margin of the schedule, there is a column for each defined Category object. When an appointment is mapped to a Category object, its corresponding color bar, in the left margin, will be filled for the length of the appointment. This may be toggled on/off by setting the CategoryBar property of the schedule true or false. If CategoryBar is false, the Categories collection is not displayed in the left margin of the schedule. The associated mapped colors may still be displayed on the appointment's margin by setting the CategoryBar property to true. There exists a tab set in the top, left corner of the schedule. The top tab must be "checked" for the Categories collection to be displayed. This effectively puts the schedule into category mode. This tab is a toggle for the display of Categories or Providers (as described below).

A Category object's name may not be numeric. This is because there would be no way to determine if the parameter being used is a name or an index. Since you may not add a numeric name the code below is invalid but I use it as an illustration. If an object's name were numeric, the following code would produce an error, since there is only one object in the collection and its name is "100". When referenced by the name "100", the collection will try to return index 100, which does not exist, and produce an error.

```
Call Schedule1.Categories.Add(“100”)
Debug.Print Schedule1.Categories(“100”).Name ‘ERROR
```

This logic also applies to all object collections in this product. The name of no object may be numeric.

NOTE: An appointment’s Category property is not aware of changes in the Categories collection. If an appointment has a Category property of “2”, referring to the second category in the Categories collection and this element is removed, the appointment will still point to the old (now non-existent) Category object.

Figure 1.3

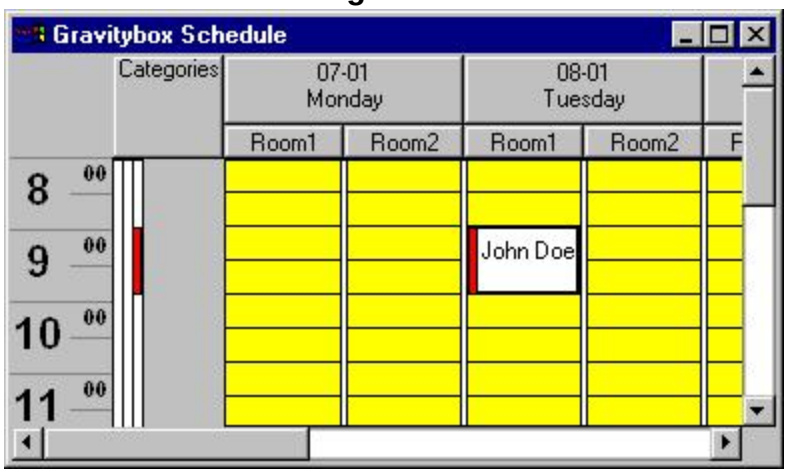


Table 1.4
Category Object Definition

Color	This property stores the color associated with this Category object. The category bar for this Category object will be displayed in this color.
Id	This is a 32-bit integer that may be used to store extra data.
IsDirty	This property will determine if any modifications have occurred, since the last time it was set to false. If any property of any Category object is modified this property is set to true.
Name	This is the name as well as the caption of the Category object. This property is read-only; however it may be changed with the Categories collection, using its Rename method.

The Categories collection has the same methods as the Rooms collection. They may be used to manipulate Category objects within the collection. There are also search functions that make finding a particular object much easier. Additional language support may be provided for the same reasons and by the same methods as explained for the Rooms collection.

Table 1.5
Categories Collection Definition

Add	This method takes parameters to create a new Category object and adds it to the Categories collection. A reference to the newly created object is returned.
Clear	This method will remove all of the Category objects from the collection.
Count	This method will return the number of elements in the collection.
Exists	Given a Category object's name or index, this method will return true if a Category object exists at the specified position.
Index	This method is very similar to the Exists method only it returns a number. Given a Category object's name or index, this method returns its index. Though valid syntax, it is not very useful to give the index in order to retrieve it. This function was designed to take the Name property as a parameter.
IsDirty	This property will determine if any modifications have occurred, since the last time it was set to false. If any property of any Category object is modified, this property is set to true.
Item	Given a Category object's name or index, this method will return a reference to the object. If the given parameter does not exist in the collection, an error is raised. To avoid an error, use the Exists method in conjunction with this method.
Name	This is the text that will appear as the category bar header on the schedule. The default text is "Categories", but may be changed to any desired text.
NewIndex	This read-only property is the index of the newest object added to the collection with the Add method. After the Remove method is called, this property is reset to zero.
Remove	Given a valid Category object's index or name, this method will remove the specified object.
Rename	This method will rename a Category object. It takes a parameter of an existing object's index or name and the new name to give the object. If the new name already belongs to another Category object, an error is raised.

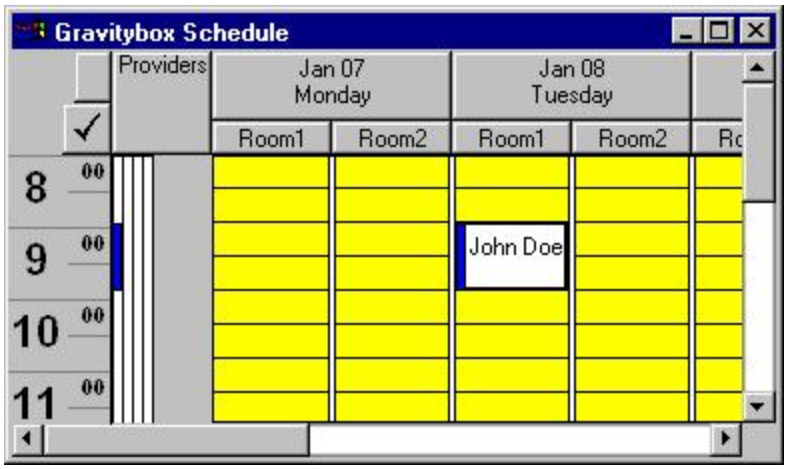
Providers

Another important collection of the GbSchedule is the Providers collection. This collection represents a group of people who provide services for appointments. This is very useful when an appointment needs to be assigned to a person. This also aids in resource management. It allows for a provider to view a schedule and know the times of his responsibilities. Since there are color-coded bars in the left margin, the provider may view a schedule and know by his colored bar when he must work and when he is free.

In the doctor's example, there are four providers, "John", "Sue", "Fred", and "Jane". These people work in the office and each has a set of appointments. First, these providers need to be added to the Providers collection.

```
Call Schedule1.Providers.Add("John", vbBlue)
Call Schedule1.Providers.Add("Sue", vbYellow)
Call Schedule1.Providers.Add("Fred", vbRed)
Call Schedule1.Providers.Add("Jane", vbGreen)
```

Figure 1.4



Now that the providers are part of the schedule, appointments may be assigned to each. This may be accomplished by setting the Provider property of an appointment to the desired Provider object's name or index. This will associated the specified Provider and appointment. The user may view this association on the schedule by selecting the bottom tab on the tab strip in the top, left corner of the schedule, if a tab strip exists. This will toggle the schedule into provider mode. Also the schedule property ShowProviderScheduledTime must be true to for the Providers collection's information to be displayed in the left margin of the schedule.

Table 1.6
Provider Object Definition

Color	This property stores the color associated with this Provider object. The provider bar for this Provider object will be displayed as this color.
Id	This is a 32-bit integer that may be used to store extra data.
Name	This is the name, as well as the caption, of the Provider object. This property is read-only; however renaming the object, through the Providers collections using its Rename method will change it.

The Providers collection exposes much of the same interface as the Rooms and Categories collections. These methods and properties may be used to manipulate

objects within the collection. Included are search functions for locating particular objects. Language support may be offered here as well.

Table 1.7
Providers Collection Definition

Add	This method takes parameters to create a new Provider object and adds it to the Providers collection. A reference to the newly created object is returned.
CaptionAvailable	This is the text that will appear above the Provider AvailableTimes bars, in the left margin of the schedule. This collection holds the valid times for which an appointment may be scheduled, for a Provider. The default text is “Available”, but it may be changed to anything desired.
CaptionScheduled	This is the text that will appear above the times for which a Provider is scheduled. Each appointment associated with a Provider will cause the provider to be scheduled for the duration of the appointment. The default text is “Providers”, but it may be changed to anything desired.
Clear	This method will remove all Provider objects.
Count	This method will return the number of elements in the collection.
Exists	Given a Provider object’s name or index, this method will return true if a Provider object exists at the specified position.
Index	This method is very similar to the Exists method only it returns a number. Given a Provider object’s name or index, this method returns its index. Though valid syntax, it is not very useful to give the index in order to retrieve it. This function was designed to take the Name property as a parameter.
IsDirty	This property will determine if any modifications have occurred, since the last time it was set to false. If any property of any Provider object is modified this property is set to true.
Item	Given a Provider object’s name or index, this method will return a reference to the object. If the given parameter does not exist in the collection, an error is raised. To avoid an error, use the Exists method in conjunction with this method.
Name	This is the text that will appear as the header for the provider columns on the schedule. The default text is “Providers”, but may be changed to any desired text.
NewIndex	This read-only property is the index of the newest object added to the collection with the Add method. After the Remove method is called, this property is reset to zero.
Remove	Given a valid Provider object’s index or name, this method will remove the specified object.
Rename	This method will rename a Provider object. It takes a

	parameter of an existing object's index or name and the new name to give the object. If the new name already belongs to another Provider object, an error is raised.

Quick Tip
You may store any number of name/value pairs for each appointment its "ExtraProperties" collection. Use it to store any amount of information.

An important feature not to be overlooked is the ExtraProperties collection. You may store any amount of extra information using an appointment object's "ExtraProperties" collection. This is a collection of name / value pairs that be used to store anything. You may address the collection by name or index to retrieve its associated value setting. You may not need this functionality but if you have a large amount of information to store with a task this collection is quite useful.

NoDropAreas

Quick Tip
Mark areas on a schedule as unavailable for appointment by using the "NoDropAreas" collection. Appointments cannot be added, moved, or copied to these areas.

There are circumstances that require that certain area of a schedule be blocked from creating and receiving appointments. The NoDropAreas collection provides this functionality. This is an added convenience for developers so that they might not be required to write these complicated routines themselves. The functionality essentially defines reserved areas of a schedule.

If the user attempts to drag (move or copy) an appointment to a NoDrop zone, the mouse will turn to a no-drop pointer to indicate that the appointment may not dropped. When an appointment's edge touches a NoDrop zone, the appointment will no longer move, as this would move it into the reserved zone. If the user continues dragging until the appointment has passed the zone, it will jump to the other side of the NoDrop zone. There is no way to move an appointment or portion thereof into one of these zones. There must be a valid portion of the schedule grid available for the entire length of the appointment.

NOTE: Appointments may be scheduled in these zones by using the Add method of the ScheduleItems collection, however the user may not drop appointments in these areas. There is a way to determine with code, if an appointment slot is free. You may use the GetNextFreeSlot method. This will be covered later in chapter 8.

There may be days that need to be invalid, for example weekends. A developer might constructed a loop and walk through all the days on the schedule, check to see if the date is a weekend day, and add it to this collection. The following code blocks the first weekend in June 2002.

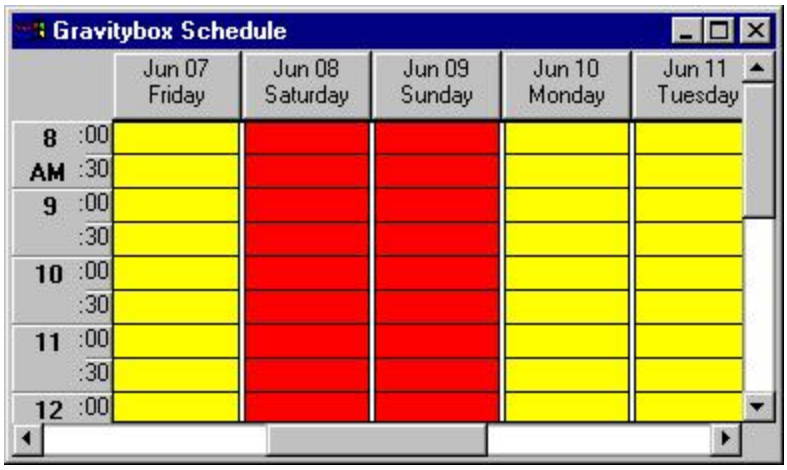
```
Call NoDropAreas.Add(#6/8/2002#)
Call NoDropAreas.Add(#6/9/2002#)
```

Code to block out all weekend days may be constructed with a simple loop. The following code loops through all the dates on a schedule and defines the weekend days as NoDrop zones.

```
Dim i As Long
Dim lLoopCount As Long
Dim dtCurDate As Date

  'Loop through how many days?
  lLoopCount = DateDiff("d", Schedule1.MinDate,
Schedule1.MaxDate)

  'Perform loop
  For i = 0 To lLoopCount - 1

    'Get the current date
    dtCurDate = DateAdd("d", i, Schedule1.MinDate)

    'If the date is Saturday (Weekday function = 7)
    'or Sunday (Weekday function = 1)
    'Then add it to our no drop collection
    If (Weekday(dtCurDate) = 1) Or _
       (Weekday(dtCurDate) = 7) Then
      Call Schedule1.NoDropAreas.Add(dtCurDate)
    End If

  Next I
```

When the schedule is displayed and these dates come into the viewable window, they will be a special color defined by the color of the added NoDrop object. This will allow the user to easily see that this is not a valid section of the schedule.

Figure 1.5

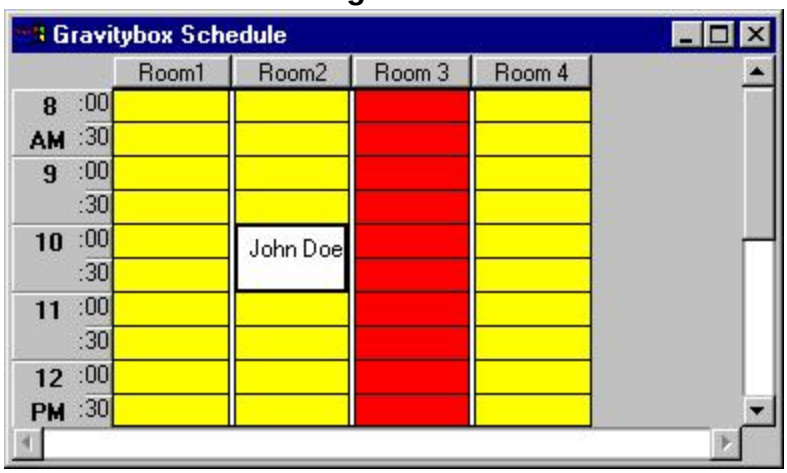


The collection can be used to define a NoDrop zone for a room also. If five rooms are defined and Room3 must be guaranteed to never accept any appointments, the room may be added to this collection and the user will no be allowed to add, copy or move any appointments to the specified room. If the user attempts to drag an appointment to this schedule section, the mouse will turn to a no-drop pointer, to indicate that this action is not valid.

It may be very useful to block rooms. Perhaps you have an office that is being remodeled and you do not wish to schedule any appointments in a particular room. Adding the specified room as the second parameter to the collection does this.

```
Call NoDropAreas.Add(, 3)
```

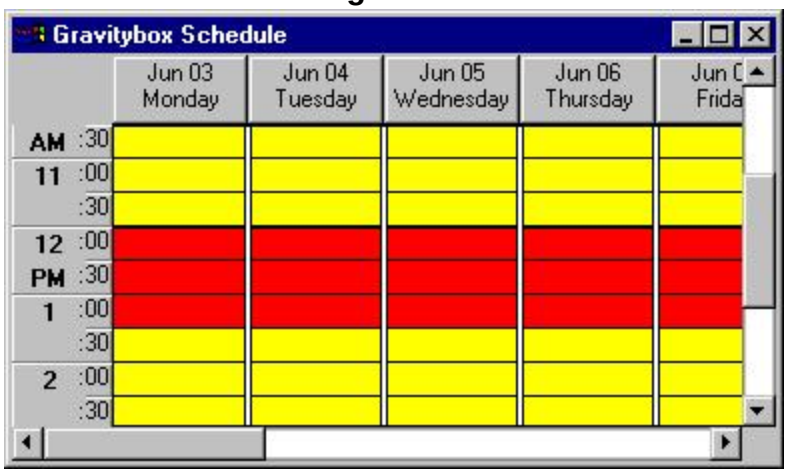
Figure 1.6



To block a time interval two parameters must be specified, StartTime and Length. The zone will begin at the specified time and last for the duration of the Length property, in minutes. The usefulness of this may be seen in the doctor's demo by defining 12PM-1PM as the "lunch hour". No appointments should be scheduled for this time slot, as the office is closed. The following code sets 12:00PM until 1:30PM as a NoDrop zone.

```
Call NoDropAreas.Add(, , #12:00:00 PM#, 90)
```

Figure 1.7



This collection is most useful; however it has much more power. You have seen how to block entire columns and rows, but in the real world many times we need to block very specific sections of our schedule such as Thursdays the 9 from 10AM – 11AM. This may greatly increases the functionality of the collection and the schedule component.

Using multiple parameters of this collection may block any section of a schedule. If you wish to block August 9, 2002 from 8AM through 12PM then you need to add a NoDropArea for this section of the schedule. Notice that second parameter was omitted. There is no room component to this schedule block so we leave it blank. Keep in mind that this block will only show up if both the date and time is displayed by the schedule's viewmode. If the ViewMode is displaying dates versus rooms only, with no time component, this NoDropArea will not be displayed even if the proper date is displayed. If a NoDrop zone is defined with a date, room, or time element then these same components must be in the schedule's viewmode. A good example of this is a NoDrop zone for a specified date, room and time. It would only be shown is the viewmode was set to vmcNormalDayRoomTopTimeLeft or vmcNormalDayRoomLeftTimeTop. These are the only two viewmodes that display all of the NoDrop zones parts.

```
Call NoDropAreas.Add(#8/9/2002#, , #8:00:00 AM#, 240)
```

This collection may be of no use to some developers, but it will be extremely valuable to others. Since GbSchedule does not allow for the selective removal of dates and times from its display, this is the next best thing. The dates and times will be displayed, but only as placeholders.

Part II
Using GbSchedule

Fast, fat computers breed slow, lazy programmers.

-Unknown

Chapter 2

Creating your first Scheduling Application

So now that you are familiar with the object model we can begin to construct our first scheduling application. It is easy to create a program that actually has schedule functionality with no code! If there is no underlying code, you will not be able to load or save schedules. However I state this merely to display the ease of use. We begin by opening Visual Basic and creating and new project. Next we need to add the software component to the project. On the Project menu choose “Components…”. This will display the add component screen. Use this to add the “Gravitybox Schedule” component.

Figure 2.1

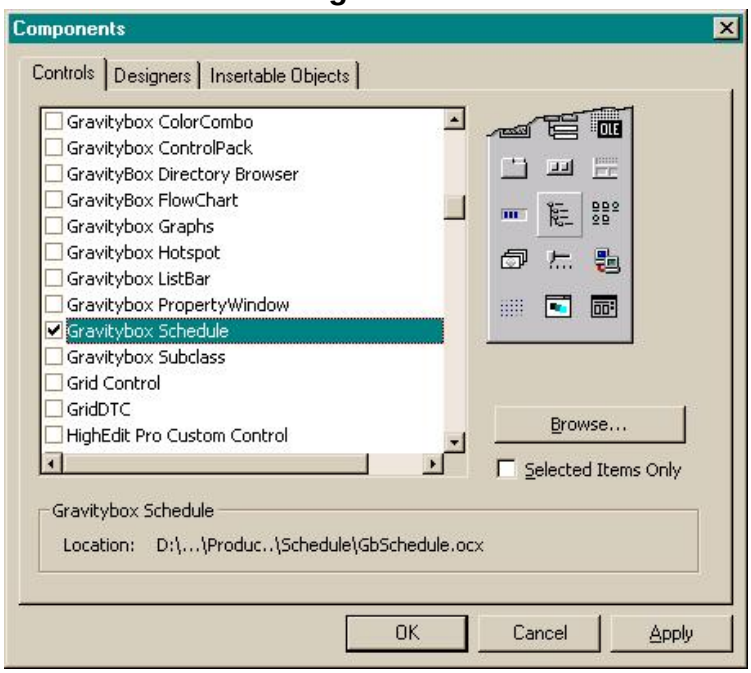


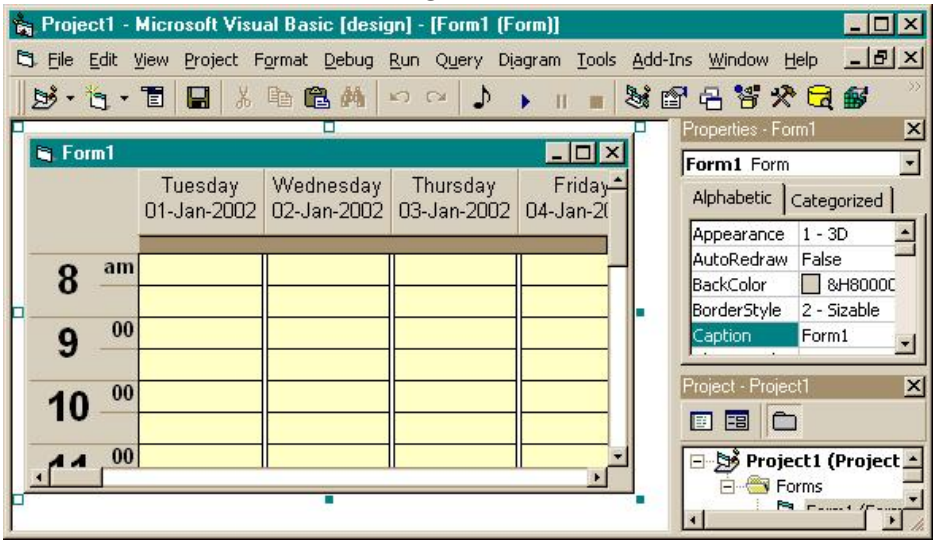
Figure 2.1 shows the Component screen with the schedule added. Press the “OK” button and we are ready to start scheduling. When the schedule is added to a form it is displayed with times on the left opposing days and rooms on the top. This is a very common configuration of a schedule. At design-time there are two rooms for each day to display the day-room configuration functionality. As described in Chapter 1 there many different modes in which the schedule may be displayed. If your application does not require rooms (or objects), this feature may be removed by changing the schedule’s Viewmode property to one of the “day only” settings. This will display only dates and no rooms. Just to keep things simple in our first program, we are going to turn off the Room interface.

Creating a GbSchedule form

To create a display that has a more common interface we need to set the following properties.

StartTime = #8:00 AM#
DayLength = 10
MinDate = #1/1/2002#
MaxDate = #1/31/2002#

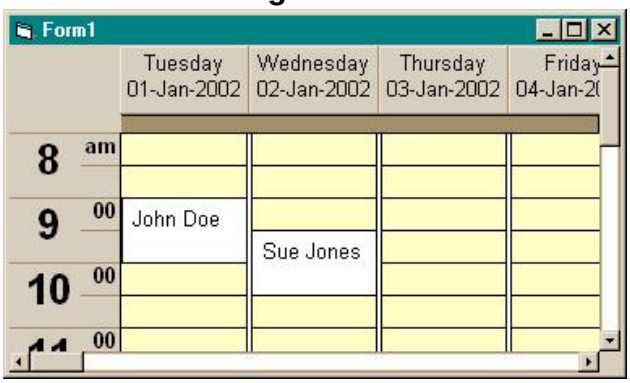
Figure 2.2



This will create a schedule that starts at 8 in the morning and displays times until 6 at night. The start date is the first of January and it lasts for 31 days, the end of the month. This will generate a schedule that starts at a normal time and lasts for the business day. It will also display the entire month so that we can make appointments for the month in its entirety.

Now that we have set the properties to some meaningful values, lets run the program. In run-mode the user may click the mouse and create an appointment. To create an appointment of a desired length, you must press the left mouse button and drag the mouse the length of the desired appointment. When the mouse button is released the appointment is created. The schedule's standard properties box appears when adding and editing by default. This may be turned off. This will be addressed later.

Figure 2.3



In Figure 2.3, I have created an appointment from 9:00 AM on January 1 for one hour. I have created another appointment at 9:30 AM on January 2 for an hour as well. The displayed text may be edited by clicking the appointment. It will become editable after a small delay. You may type directly into the appointment when it is editable. It you wish to edit more properties of an appointment, you may double-click on it and the default properties window will be displayed. Again, this is configurable and may be toggled off if desired.

Moving Appointments

Now that we have two appointments on the schedule we may need to move them. The whole purpose behind an electronic schedule is that maintenance should be easy. On paper you would need to erase the old appointment, then pencil it back in at a new date and time. The process is far easier now. Grab the appointment by clicking it with the left mouse button. While the button is pressed drag the appointment to another position. As it is being dragged, an outline will be drawn to show you exactly where the appointment will be dropped. It you move the mouse close to any margin, the schedule will scroll in that direction. This ensures that if you want to move an appointment to a position that is not currently on the screen, you can still get there by scrolling. When the appointment is dropped, you will be prompted to confirm the move. This functionality may also be overridden if desired.

Now that you know how to move an appointment, it is a good time to describe another very similar action: copy. Moving is a common occurrence for any schedule; however there are times you may want to copy an appointment as well. If you hold the <CTRL> key while moving the mouse, the icon will have a small plus sign next to it. This indicates that you are coping not moving an appointment. When the appointment is dropped, the original appointment does not change. There will be one more appointment on the schedule than before the drag. This feature might be used to add a follow-up for the same person, with all of the same information. For example if John Doe came in for an exam this week and he needed to come back for another exam next week, you could copy his appointment to next week. Now all of the information is the same and you do not need to retype any of it.

Moving Appointments between Windows

The previous move and copy example displayed the actions being performed on a single screen. It is possible to drag appointments across windows or even applications. As in the example you may drag an appointment to another form that has a GbSchedule component on it. Since this is another instance of GbSchedule, it does not matter if it is the same screen, a child window, or a different application. The drag will work no matter what! This feature is nice when building a MDI application. You can construct an application that allows the user to open different days or rooms or some combination of both in different windows. If the user opens two, child forms say one for January 12 and the other for January 13; he will have two instances of GbSchedule. He may then drag one of the existing appointments to the other window. The original appointment in the source windows will disappear and the appointment will reappear in the destination window. There is a little caveat though. For an instance of the schedule to receive drops from outside of itself, the AllowOtherDrops must be toggled to true. This property ensures that the schedule is allowed to receive outside drops. If this property is false, no appointments may be dropped from other schedules. The beauty of this cross-window dragging is that the source and destination schedules need not be the same format. The source may display time at the top and the destination may display time on the left. One may display Rooms while the other may not. They may be displayed in any combination.

Chapter 3

Adding Code

So far we have created and application with a schedule in it. We have learned to create an application with multiple schedules in it and move appointment between those schedules. However we have not as of yet added any code. Now creating these schedules are nice, but without code, we can really build nothing but trivial applications. Chapter 1 showed how to set some other the properties but now we need to learn how to add code and use some of the events.

Reference Creation

We will begin by looking at the ScheduleItems collection, since this collection is the most frequently used. The collection has an Add method that allows for the addition of a new element to the collection. Many properties may be set as parameters in the method call. There are some exceptions and this is the reason for creating an object reference to each. In collections, all of the properties of an object may not be set from its parent Add method. To set all properties would be cumbersome for objects with many properties and would break compatibility, if more properties were added in future versions.

There is no parameter in the Add method to set the Id property of a ScheduleItem. An object reference must be made to set the property contents of the newly added object. The following code will add an appointment, to the ScheduleItems collection, with the name of “Appt1”. It will be scheduled on June 3, 2002 in Room 1 at 8:00 AM. It will display the text “John Doe's Appointment” on the screen.

```
Call Schedule1.ScheduleItems.Add("Appt1", #6/3/2002#, 1, _
#8:00:00 AM#, 60, "John Doe's Appointment")
```

As can be seen, there is no parameter to set the Id property, in the Add method. The developer must reference the last added element and set it. This may be done with the following code. The Count method returns the total number of item in its collection. Since the object was the last one added, it will be the last one in the collection. The Id property may then be referenced and its value set.

```
Schedule1.ScheduleItems(Schedule1.ScheduleItems.Count).Id =
newId
```

This syntax is rather cumbersome and could get very messy, if several properties needed to be set. Also, it issues at least 4 object lookups to find the element. When setting many properties, this syntax would create slow code very quickly, not to mention

code that is difficult to read. An alternative is to get a set an object variable to the newly added object and set its properties. The following code declares an object variable to point to the newly added ScheduleItem.

```
Dim oAppt As CScheduleEl

  Set oAppt = Schedule1.ScheduleItems.Add("Appt1", _
              #6/3/2002#, 1, _
         #8:00:00 AM#, 60, "John Doe's Appointment")
  oAppt.Id = newId
  oAppt.ReadOnly = True
  Set oAppt = Nothing
```

Clearly, the code is much easier to read. This code will also, in theory, run faster. I say “in theory”, because with this amount of code, no difference will be noticed in speed. The object variable was not declared with the “New” keyword because a new object variable of this type may not be created, nor does one ever need to do so. The Add method of the ScheduleItems collection will create the object, add it to the collection, and returns a reference. This reference may be used to manipulate the new object’s properties.

Collection Looping

The same philosophy may be applied to the other collections as well. A Room object may be added to the Rooms collection in must the same way. A reference to it will be returned that may be used to manipulate its properties. A Room object has an Id property that is not defined in the Add method of the Rooms collection. The only way to set it is after it has been added.

```
Dim oRoom As CRoomEl

  Set oRoom = Schedule1.Rooms.Add("Green Room")
  oRoom.Id = 57
  Set oRoom = Nothing
```

This Rooms collection will create a Room object and return a reference to the object. The object reference is stored it in the “oRoom” object variable. Its “Id” property may then be set.

This syntax may be used for all the collections of GbSchedule. An object variable for each collection element type can be created and used to jockey the property values. Alternately, object variables may be used in loops. This is by far the easiest way to set property values of objects.

Many times an object’s properties will need to be set using loops. There are three ways to loop through a collection. The first has been discussed, referencing an object with an index. An integer, looping variable is declared and a loop is constructed to walk from the

first to the last element, each time referencing the Item method of the collection by index.

```
Dim i As Long

  For i = 1 To Schedule1.ScheduleItems.Count
    Schedule1.ScheduleItems(i).Id = i
  Next I
```

The second way to loop through a collection is by creating an object variable, of an appropriate type, and using it, to set the property values. The object variable is set to a member of the collection, used, and released. This is much easier to read, but still rather awkward.

```
Dim oAppt As CScheduleEl
Dim i As Long

  For i = 1 To Schedule1.ScheduleItems.Count
    Set oAppt = Schedule1.ScheduleItems(i)
    oAppt.MinLength = 60
    oAppt.MaxLength = 120
    Set oAppt = Nothing
  Next I
```

The third way is the easiest to read. It lets the collection do the loop. There is no loop counter, as in the other two examples. An object variable is declared and the collection loops through all of its own items. The following syntax allows the collection to do the looping work and takes some of the coding away from the developer.

```
Dim oAppt As CScheduleEl

  For Each oAppt In Schedule1.ScheduleItems
    oAppt.MinLength = 60
    oAppt.MaxLength = 120
  Next
```

In this code, is can be seen that there is no explicit call the to the Item method of the collection. The call is implicit. The following two code fragments are equivalent.

```
Debug.Print ScheduleItems(1).Name
Debug.Print ScheduleItems.Item(1).Name
```

The Item method is the default method of all collections in GbSchedule. There is no need to specify a call to this method explicitly. There is also a speed issue to consider when looping. The third looping example is the fastest way to create a loop. An object

reference is executed much faster than looping with an object's index as in the second example. This difference becomes more drastic as the number of items in the loop increases.

Adding

Quick Tip
When adding many appointments, toggle the "AutoRedraw" property to false. This will turn off screen paints for each object add. When you have completed the loading, toggle this property to true, as the screen will not update when this property is false. This action will speed your loads dramatically.

A good place to start adding code is to manually add an appointment. This may be accomplished in code, with the ScheduleItems collection's Add method. A user can add appointments, if the AllowAdd property of the schedule has been set to true. If false, user adds are not permitted. The user may click the mouse's left mouse button on the starting time that he wishes the appointment to begin. He must then drag the mouse the length of the appointment. The times of the schedule are displayed in the margin of the schedule. The times are clearly marked for the user to know exactly the position of the mouse. When the mouse button is released, an appointment is created. This assumes that the user did make a non-zero length appointment. If the length of his drag was zero minutes, nothing happens.

If the property AllowAddDialog is set to true, the default property box will appear. This allows the user to modify key properties of the appointment. This dialog is not configurable by the developer. If special functionality needs to be added to the property box, a custom screen needs to be constructed and the default property box canceled. Alternatively, you may use the ScheduleProperties control. This control is discussed later and may be used to construct a customized property screen. After the user completes the editing of properties in the default dialog, he will have two options. First he may press the "Ok" button, at which point the modified appointment information is saved. If the user presses the "Cancel" button on the property box, no save will be performed.

If a property box should not be displayed at all, the AllowAddDialog should be set to false. When the user creates an appointment, it will appear on the schedule with default appointment values. There will be no text displayed inside of it. The user may modify the appointment values later by double-clicking the appointment. The event sequence will follow the order: BeforeAdd, AfterAdd.

The BeforeAdd event is raised before the appointment is actually added to the ScheduleItems collection. It will return as parameters the proposed appointment's date, room, start time, and length. In addition, there is a Boolean Cancel parameter that will allow for the cancellation of the Add operation, if so desired. You may add code to this event to perform whatever error checking you wish. If the AllowAddDialog property is true, this event may also be used to cancel the dialog by setting the "Cancel" parameter to false and displaying your own custom dialog. The event may also be used to error check the parameters and Cancel the addition if necessary. This may be required if the user tried to create an appointment outside of some bounds that you have set. Perhaps you wish to cancel all appointments made on weekends.

The AfterAdd event is raised after an item has been added to the ScheduleItems collection. The event will have the new appointment's index as its parameter. You may display your custom dialog box by using this event if it is more appropriate. Make sure that the AllowAddDialog property to false. This will ensure that the default dialog is not displayed at all. The AfterAdd event has the index of the newly added ScheduleItem. So you may create a custom property box and use it to modify the properties of the object specified by the index parameter. The BeforeEdit and AfterEdit events will be raised as well, if the default property box is used. The reason the edit events are called is that after the Add, the operation is a normal edit. So the default behavior for adding an appointment is BeforeAdd, AfterAdd, BeforeEdit, and AfterEdit.

Moving

A move is defined as changing the StartTime, Date, or Room property of a ScheduleItem, by dragging it, with the mouse. After an appointment is created, the user may move the appointment to any other part of the schedule (except NoDrop zones), with the mouse. This assumes the schedule property AllowMove is set to true, since no user-initiated moves may occur if this property is false. Also the appointment's ReadOnly property may not be true, since moving it would change its position. This section does not refer to the action taken when using the property box or setting an appointment's properties in code. These actions are considered edits. When using a property box, the events BeforeEdit and AfterEdit are raised. The following refers to the event sequence when dragging an appointment, with the mouse.

During a move, there are a number of events that come into play. The first event that is raised is the BeforeDrag event. This occurs to inform you that the user is dragging an appointment. A Cancel parameter is provided so that the operation may be canceled if necessary. There is also a drop operation parameter that will inform you as to whether the operation is a copy or a move. The DragOverScheduleItem event is raised every time the mouse moves while dragging an appointment. The DragDropScheduleItem event is finally raised to inform that something was dropped on the schedule (perhaps not even an appointment).

Quick Tip
To turn off the move prompt, set the "DoPrompt" parameter in the BeforeMove event to False.

The next event in sequence is the BeforeMove event. This event is raised so that any code that needs to be executed, before a move, is performed. An interesting parameter in the BeforeMove event is DoPrompt. This is a Boolean value that determines whether the user will be prompted for a move confirmation. The default value is true, but this property may be set to false and the appointment will be moved, with no questions asked. Also the schedule property ConflictWarn allows for the determination of whether the user is to be prompted about conflicts. When moving an appointment, there is the possibility that a conflict may occur. If a conflict does occur because of this move and ConflictWarn is true, the user is prompted by a confirmation dialog to ensure that a conflict is indeed wanted at this position. If ConflictWarn is false then the appointment will be moved and a conflict will occur, with no warning to the user.

After the user has been prompted for the move (or perhaps not prompted) and the appointment's properties have been modified, the AfterMove event is raised. Any cleanup, saving, etc that needs to be done after an appointment move may be performed here. This event has a single parameter, which is a reference to the newly moved appointment object.

Coping

Quick Tip
To copy an appointment, press the <CTRL> button and move an appointment. When dropping the appointment, a new one will be created and the original one will not be removed, thus a copy.

A copy is special case of move. A copy will add another ScheduleItem object to the ScheduleItems collection and set its StartTime, Date, and Room properties to the new position. The original appointment will not be changed. The Copy action is accomplished by holding the <CTRL> key while dragging an appointment. If the user presses the <CTRL> key while dragging, he will notice that the mouse pointer has changed from a normal arrow, to an arrow with a plus sign. This indicates that there will be an extra appointment added to the schedule, by performing this action.

Much like moving an appointment, coping raises its own sequence of events. The first event that is raised is the BeforeDrag event, just like the move operation. As with the previous, there is a Cancel parameter and a drop operation parameter. The DragOverScheduleItem event is raised every time the mouse moves while dragging an appointment. The DragDropScheduleItem event is then raised to inform that something was dropped on the schedule.

The next event in sequence is the BeforeCopy event. This event is raised so that any code that needs to be executed, before a copy is performed, is done so. It also has a DoPrompt parameter, which is a Boolean value that determines whether the user will be prompted, for a copy confirmation. Also remember the schedule property ConflictWarn allows for the determination of whether the user is prompted about conflicts.

After the user has been prompted for the copy and the appointment's properties have been modified, the AfterCopy event is raised. Any cleanup, saving, etc that needs to be executed, after an appointment copy, is performed here. This event has a single parameter, which is a reference to the newly copied appointment object.

Editing

The user may edit an appointment, in a variety of ways. The most common way is to double-click the appointment, for the display of its property box. This assumes that the AllowEdit property is set to true. The default property box will display the StartTime, Length, Room, Date, Category, and DisplayText for the selected appointment. There is also a Notes tab that allows the user to associate some additional text to an appointment.

If you wish to build your own property screen, you will probably need to cancel the default property screen. The following code illustrates the actions that should be performed if a custom property box is to be used. In the BeforeEdit event, the Cancel parameter must be set to true. This will cancel the default edit. A custom property box may then be shown from this event. The following code displays an example of how to display a custom property box. The form “frmProperty” is a customized screen that has been built to display an appointment’s properties as deemed fit.

```
Private Sub Schedule1_AfterEdit(ByVal Index As Long)
  Cancel = True
  Load frmProperty
  frmProperty.ScheduleIndex = Index
  frmProperty.Show vbModal
End Sub
```

It is also possible to modify the ScheduleItem’s DisplayText property without using a property box. If the schedule’s AllowInPlaceEdit property is true, the user may make the display portion of an appointment editable by single clicking the mouse on an appointment. This will start an edit sequence whereby the user may edit the text displayed on the appointment. The BeforeEditText event will be raised before the text area becomes editable. This event has a Boolean Cancel parameter and the ScheduleItem’s index. A successful edit is achieved when the user clicks off of the appointment or it loses focus. If the <ESCAPE> key is pressed, the edit is canceled and the property value is not modified. Upon a successful edit, the AfterEditText event is raised to confirm that the edit was successful.

Deleting

Quick Tip
To remove an appointment, move the mouse over the appointment and pres the <DELETE> key.

The user may remove an appointment from a schedule (and from the ScheduleItems collection) by moving the mouse over the appointment and pressing the <DELETE> key. This assumes the schedule property AllowDelete is true. The user will be prompted to remove the appointment. The default remove text may be overridden. The verbiage or even the language may be changed, if need be. There is a prompt text parameter sent in to the BeforeDelete event. If the default text is to be changed, set this string to the desired text here.

```
Private Sub Schedule1_BeforeDelete(ByVal Index As Long, _
sPrompt As String, Cancel As Boolean)

  sPrompt = "Are you sure that you wish to remove “ & _
“the selected appointment?"

End Sub
```

Before deleting an appointment the BeforeDelete event is raised and will allow for the cancellation of the operation. Upon completion of the delete operation, the AfterDelete

event is raised to inform that the delete operation was completed successfully. The event will have the appointment's old index in the ScheduleItems collections as a parameter. Do not try to reference this appointment at this point. It has been removed. If this index exists, it is the appointment that was next in sequence in the ScheduleItems collection.

Resizing

Quick Tip
To resize an appointment, drag one of its resizable edges to the newly desired position.

You may set the size of an appointment in code but this is rather prohibitive in the real world. When a schedule is displayed, the user may be allowed to resize an appointment. He may do this by moving the mouse over the borders of the appointment then clicking and dragging the desired length. If resizing is allowed, the mouse pointer will turn to an arrow, either North-South or East-West, depending on which is appropriate. The pointer will change when the mouse is over the border of the appointment, not in its interior. Keep in mind that the schedule's AllowResize property must be true for any resizing to be done at all.

There AllowItemResizing property selectively allows resizing of all non-read-only appointments. The possible values of the property are as follows: ircNone, ircBottomRight, ircTopLeft, or ircAll. The value will determine how much functionality is allowed. If this property is set to ircNone, no user resizing will be allowed. If it is set to ircBottomRight, the appointment's length may be modified. Depending on the Viewmode property, dragging the right or bottom appointment edge may modify the length. If the value is ircTopLeft, only StartTime is editable. Again, depending on the Viewmode, the StartTime may be modified, by dragging the top or left appointment edge. Lastly, by using the ircAll value, the StartTime and Length of the appointment may be modified. Essentially this says that both edges of the appointment may be resized. This property affects all appointments.

```
‘Length modification only
Schedule1.AllowItemResizing = ircBottomRight
```

Quick Tip
Bound an appointment's size by using its MinLength and MaxLength properties.

There is additional functionality for an appointment that may be used. Every appointment has two properties that control resizing: MinLength and MaxLength. When the user creates an appointment these values are set to their default value of (–1). This indicates that the appointment has no restrictions on its size. You have the option of setting these values to restrict the user from breaking any business rules that have been defined in your application. If the MinLength is non-zero, the user may not resize the appointment smaller than this value. Conversely, if the MaxLength is non-zero, the user may not resize the appointment larger than this value.

```
'No smaller than 30 minutes
Schedule1.ScheduleItems(3).MinLength = 30
'No larger than 2 hours
```

```
Schedule1.ScheduleItems(3).MaxLength = 120
```

This code will set the third element in the ScheduleItems collection, so that it may not be smaller than 30 minutes and no larger than 2 hours.

Quick Tip
Cancel a user resize in the "BeforeItemResize" event.

There are two of events raised as a consequence of resizing an appointment. The first is the BeforeItemResize event. This event has as its parameters the appointment's index in the ScheduleItems collection and a Boolean Cancel parameter. Once an appointment is resized the AfterItemResize event is raised with the appointment's index as a parameter.

When the user is dragging an edge of an appointment, the WhileItemResize event will be raised repeatedly. As the user drags an appointment edge, it may be resized many times before he releases the mouse button. Each resize triggers this event.

AllowOtherDrops

A special Add may be performed if required. A drag of some external object to a schedule may be desired. This could be a file, picture, etc. Although these items are not appointments, they might have some significance in your application. In this case, the schedule property AllowOtherDrops will allow dropping of non-GbSchedule items on the schedule. This will be performed as follows. The DragDropScheduleItem event will be raised when the drop has taken place. The BeforeAdd and AfterAdd events will then be raised, providing an opportunity to do any verification and also to cancel the Add if necessary. Afterwards the BeforeEdit and AfterEdit events will be raised, in response to the editing with the default property box.

This action actually takes a bit of work. The source control must be setup as OLEDrag enabled. This may be accomplished by using three events of the source control. For example, assume there is a form that has a Listbox and a Schedule on it and the user should be able to drag a list item from the Listbox and drop it on the Schedule. This may be accomplished by using the source's MouseDown event to initiate a drag as follows.

```
Private Sub List1_MouseDown(Button As Integer, Shift As
Integer, X As Single, Y As Single)

  Call List1.OLEDrag

End Sub
```

This action will start the drag; however nothing will happen until there is code in other events as well. The schedule expects a particular type of drag, so this needs to be defined in the source's OLEStartDrag event.

```
Private Sub List1_OLEStartDrag(Data As DataObject,
AllowedEffects As Long)

  Call Data.SetData(, Schedule1.OLEDragFormat)
  AllowedEffects = vbDropEffectCopy

End Sub
```

The OLEStartDrag event is called by the Schedule to test whether this source object is dragging a valid type. The format may be set to the Schedule's OLEDragFormat property.

```
Private Sub List1_OLESetData(Data As DataObject, DataFormat As
Integer)

  'This will create a 90 minute appointment
  Call Data.SetData(Schedule1.CreateByteArray(90),
Schedule1.OLEDragFormat)

End Sub
```

The source control's OLESetData event is called by the Schedule when dragging or a drop has occurred. In this event, the drag format is set to the Schedule's OLEDragFormat property. Also the Schedule expects an appointment length for this format. The data must be a byte array. So that every developer will not need to create his own byte array creation routine, one is implemented by the Schedule itself. The CreateByteArray method takes a string value and returns a byte array with its ASCII codes. If this is sent into the Data object's SetData method, the schedule will draw an appointment of this length on itself. If the data entered is invalid or not a number, the ScheduleIncrement property is used to determine the length of the appointment.

This is not the only way to use the AllowOtherDrops property. When true, any drop will be valid. The user could drop files from Windows Explorer if he chose to do so. The only difference is that all of this code would not have been needed and the appointment would be set to the ScheduleIncrement property. Using the defined technique, allows the developer to have control over the size of the appointment to be created.

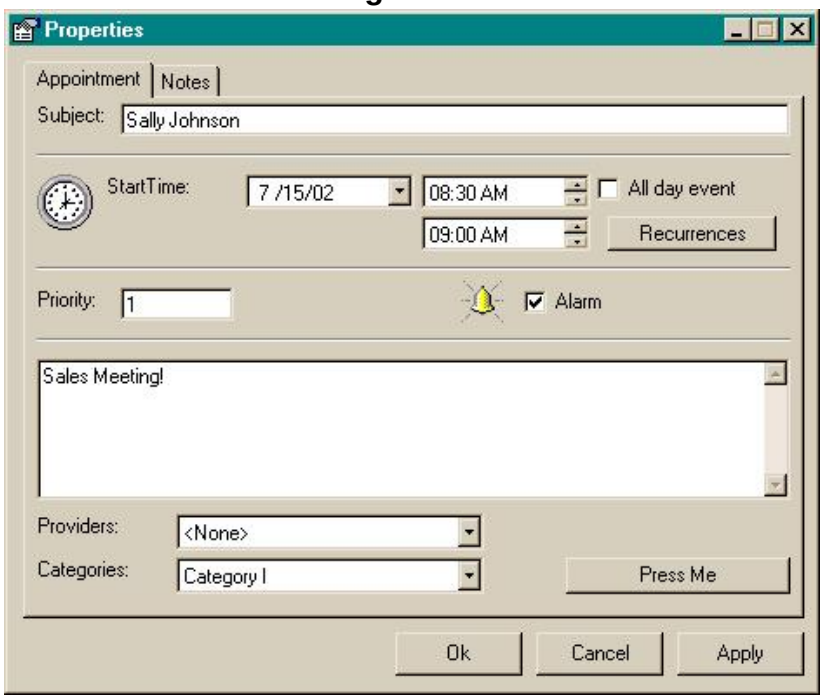
Default Property Window

Quick Tip
You may use the default property window instead of building your own custom edit screen.

You may choose to use the default property window included in the control. If the other properties like AllowAdd and AllowEdit permit, the default property window will be displayed when the user adds or double-clicks and appointment. You may cancel this functionality in the BeforeAdd or BeforeEdit events by setting the "Cancel" parameter to false. The AllowAddDialog property also plays a part. Assuming that the property settings of the control permit, the user will see the default property window. The window

may be shown modal or non-modal depending on the property “DefaultDialogModal”. In Figure 3.1, the window is shown non-modal. Notice that there is an “Apply” button. This will allow the user to save changes to the main schedule without closing the property window. Also notice that there is a “Minimize” button for the window. This allows the user to minimize the property window. When the default property window is shown in a non-modal fashion, multiple windows may be open at the same time. No more than one per appointment may be open, but there may be several windows open each for a different appointment.

Figure 3.1



If the default property window is set to open in a modal fashion then only one per schedule may be open and the user may not minimize the window at any time. The screen is displayed and requires user interaction before it may be closed. Also the “Apply” button does not appear on a modal window since the user must either choose to save or cancel, both of which close the property window.

Custom Icons

Quick Tip
Add custom icons to any appointment by using its “Icons” collection and an ImageList control.

In many instances you may wish to add custom icons to your appointments. These icons may signify special events or information relevant to your application. GbSchedule provides a way to do this simply. The Microsoft common controls are required for this action. Put an ImageList on a form and load it with all of the icons that you want to be displayed on any of the appointments. Make sure that the ImageList is set to display 16x16 icons and set its “UseMaskColor” to true.

Each ScheduleItem has an “Icons” collection. This is merely a collection of names. If you wish to use a particular icon from the ImageList, simply add its key to the ScheduleItem’s Icons collection. When a schedule is repainted it will check to determine if the associated icon key exists in the ImageList. If it does it is painted on the appointment. Icons are displayed in the order [1..N] that they are present in the appointment’s Icons collection. Invalid icons keys are skipped and no error occurs. Keep in mind that the standard icons are dislayed first and the custom icons are displayed thereafter. A standard icon is one that is predefined by the schedule; these include the Alarm property icon, partially displayed icon, and Recurrence icon.

Chapter 4

File Maintenance

> **Quick Tip**
> If you need to save a schedule but do not wish to distribute DAO or ADO database layers, you may use the "ImportXML" and "ExportXML" methods to save/load information.

The schedule has the capability to save its data to a file and retrieve it later. Although most applications will probably use a database to save and load information, there may be times when a database is too much overhead. The added file functionality will make it very easy to build "lightweight" applications. An application may be constructed to allow a user to open and save files, with the common dialog control or you may wish to handle all of the specifics in the background.

ImportXML

The ImportXML method will load a saved schedule. It takes a CXMLParameters object as a parameter. This object will allow the properties that control the load, to be set. This method will load the ScheduleItems collection previously committed to file. It will also selectively load other collections: Categories, Providers, Rooms, and NoDropAreas. These collections may be toggled to load using the following properties of the parameter object.

Table 4.1
ImportXML / ExportXML Parameters

EmployGMT	When True, this property will save the appointment's StartTime property in Greenwich Mean Time (GMT). This may be used when trading schedules across time zones. When saved in this format, the schedule file may be loaded on another machine and the schedule will reflect the current time zone.
FileName	This is the filename of the saved schedule.
Overwrite	This flag determines if the file is overwritten if it exists. (ExportXML only)
UseCategories	This Boolean flag determines whether the Categories collection is loaded and saved.
UseRooms	This Boolean flag determines whether the Rooms collection is loaded or saved.
UseProviders	This Boolean flag determines whether the Providers collection is loaded and saved.
UseNoDropAreas	This Boolean flag determines whether the NoDropAreas collection is loaded and saved.
UseAllCollections	This method takes a Boolean parameter and sets all the "Use…" properties to the specified value.

The following code will load all the data a file contains.

```
Public Sub Save()

Dim oXMLParameters As CXMLParameters

  Call oXMLParameters.UseAllCollections(True)
  oXMLParameters.FileName = "c:\schedule.gcf"
  Call Schedule1.ImportXML(oXMLParameters)
  Set oXMLParameters = Nothing

End Sub
```

The UseAllCollections method was invoked to set all the load flags for all the collections to true. This will load the data for each of those collections, if data exists in the file.

ExportXML

A schedule may be saved using the ExportXML method. It also takes a CXMLParameters object as a parameter, though more of its properties are used. The object may be setup just as if calling the ImportXML routine, with the exception that the Overwrite Boolean flag may be set to true, to overwrite the file, if it exists.

```
Dim oXMLParameters As CXMLParameters

  Call oXMLParameters.UseAllCollections(True)
  oXMLParameters.FileName = "c:\schedule.gcf"
  oXMLParameters.Overwrite = True
  Call Schedule1.ExportXML(oXMLParameters)
  Set oXMLParameters = Nothing
```

Do remember when loading a file, no information on a schedule is removed. If a schedule has one appointment on it and a saved file is loaded with two appointments in it, there will be total of three appointments on the schedule, after the load. If there is previous information in a schedule collection, it will probably need to be cleared before a file is loaded. Clearing first will ensure that after a load, the only items in a schedule collection are the items loaded from file. This action is not necessary however. If two or more schedules are being merged, multiple loads without first clearing is desirable.

Which collections to save is a concern as well. It may be desired to save only the ScheduleItems collection. Perhaps there is no need for information in the other collections. There may be a standard format an application has, with common Providers, Categories, etc. In this case, it is redundant to save the information that is already known. The other collections may be toggled off. The ScheduleItems collection

can never be toggled off. It is always saved and loaded when these methods are used. The two code fragments that follow are equivalent. The first sets all the ”Use…” properties individually and the second sets them all at once.

```
’Set Seperately
oXMLParameters.UseCategories = True
oXMLParameters.UseRooms = True
oXMLParameters.UseProviders = True
oXMLParameters.UseNoDropAreas = True


’Set all at once
Call oXMLParameters.UseAllCollections(True)
```

The second code fragment is just easier to read and more compact. If some collections are to be saved and not others, the ”Use…” properties must to be set separately.

The file format is a markup language. The file uses tags like HTML and XML. Each element in a collection is saved, wrapped in its property name. This makes the file easier to read than binary file. It may also be edited, if need be. Since the file is text, other programs may be created to read, modify, or save schedules. An auxiliary application may be created that generates a GbSchedule file that is read into the GbSchedule ActiveX control. An example of the file tags is displayed here. The example is a file fragment of that describes one appointment.

```
<Appointment>
  <Alarm>False</Alarm>
  <BackColor>16777215</BackColor>
  <BlackOut>False</BlackOut>
  <Category>Category1</Category>
  <DisplayText>qwe</DisplayText>
  <ForeColor>0</ForeColor>
  <Id>0</Id>
  <ItemData>0</ItemData>
  <StartDate>1/2/02</StartDate>
  <Length>90</Length>
  <MaxLength>-1</MaxLength>
  <MinLength>-1</MinLength>
  <Priority>0</Priority>
  <ReadOnly>False</ReadOnly>
  <Room>Room1</Room>
  <StartTime>10:00:00 AM</StartTime>
  <ScheduleCategories>
  </ScheduleCategories>
</Appointment>
```

Quick Tip
If you are distributing schedules to people in other time zones, you may use Greenwich Mean Time (GMT). This will ensure that time differences are not lost.

Also the StartDate and StartTime properties may be influenced by the EmployGMT property. If EmployGMT is True, the export routine will add (or subtract) the necessary number of hours to make the StartDate and StartTime correspond to GMT. When a schedule is loaded with this property also set to true, the calculation will be done in reverse, based on the time zone of the computer performing the load. For example, if an appointment at 2:00 PM is saved on a computer in the Eastern Standard Time (EST), which is GMT –5 hours, the saved file will have 7:00 PM for its StartTime property. This is because at 2:00 PM EST it is 7:00 GMT. Now this file is sent to California, which is Pacific Standard Time (PST) or GMT –8 hours. When loaded the 7:00 PM file setting will be converted to 11:00 AM. So the west coast user will see that this appointment starts at 11:00 AM local time, which is correct. This is the same as 2:00 PM EST for the east coast user. If the EmployGMT property is False (default value), the time loaded and saved will be exactly as it appears on the schedule.

Notice that each element is described by its tags. The schedule uses these tags to save and load information into the correct elements. The language is a simple markup language. It conforms for the most part to the XML standard. It differs in that you cannot add a data definition template header. However since the software is creating the files and not the user, this should not be a problem. The markup language is all that is needed for the uses of this scheduling software. To create an application that can read or save a ”gcf” (Gravitybox Calendar File) file, its file format must be used. The code below defines all the collections saved in the file. If this code were saved to a file, it would load successfully, however there would be no data loaded.

```
<?xml version='1.0'?>
<?Properties ScheduleIncrement='30'?>

<!-- This is a GbSchedule saved schedule. -->

<GbSchedule version='6.2.342'>
  <ScheduleRooms/>
  <ScheduleCategories/>
  <ScheduleProviders/>
  <ScheduleNoDropAreas/>
  <ScheduleItems/>
</GbSchedule>
```

The order of loading is determined by how the file is organized. If the Categories collection is first in the file and Providers second, this is the order of loading. Unless manually modified, it will be loaded as it was saved. The procedure for saving is as follows: Rooms, Categories, Providers, NoDropAreas, and ScheduleItems.

Rooms

If there are objects in the collections, they will be saved inside their respective object tags. The Rooms collection with one Room object named ”Room1” would be saved as follows.

```
<ScheduleRooms>
  <Room>
    <Name>Room1</Name>
    <Id>0</Id>
  </Room>
</ScheduleRooms>
```

If more rooms were present, there would be more ”<Room></Room>” tags pairs defined.

Categories

Categories are defined in much the same way. A Category object has more properties than a Room object. This is solved by adding more tags.

```
<Category>
  <Name>Category1</Name>
  <Id>0</Id>
  <Color>16711680</Color>
</Category>
```

As was seen earlier, with the ScheduleItems object, a Category object defines each of its properties with a tag and wraps the property’s data in the tag. This example shows that a Category object has three properties: Name, Id, and Color.

Providers

The Providers collection defines the people that work with a schedule. Each has multiple properties that need saving. The properties that are present on the Provider object are the same tags that are defined in the file. This makes it obvious that there is a one-to-one map of object properties to file tags.

```
<Provider>
  <Name>John</Name>
  <Id>0</Id>
  <Color>16711680</Color>
  <ScheduleTimes/>
</Provider>
```

NoDropAreas Collection

Finally, there is the NoDropAreas collection. This collection saves the data that define NoDrop zones.

```
<NoDropAreas>
  <NoDropArea>
    <SelDate>6/12/02</SelDate>
    <Room>2</Room>
    <StartTime>12:00:00 PM</StartTime>
    <Length>60</Length>
  </NoDropArea>
</NoDropAreas>
```

Note: The file format saves some characters in hexadecimal format. The letters, numbers and selected punctuation are displayed in the file as plain ASCII. All other characters are saved in the format ”&00”, where the ”00” is its hexadecimal ASCII code.

This ability to create XML files open doors for other functionality. You may wish to construct an application that opens a port to listen for other scheduling applications that send information to you. You may construct a set of applications that reside at different locations and use the Internet to coordinate schedules back and forth. Since you can use the ImportXML method to merge other appointments from a file with the current configuration of the schedule, this type of application could conceivability be built. You may even build a server component that coordinates all client programs. The server may receive XML schedules and send them to their respective destinations. The possibilities are endless.

Chapter 5

Database Access

In Chapter 4, we learned how to save and load information to files. This may work on a limited scale. In this scenario, all schedules are saved in their own separate files. This makes searching through multiple schedules extremely difficult. For a serious application, you need the power of a database. A database can be your repository of appointments and other information that may be loaded, searched, and saved at will. You may also only load the information you need. It is possible to accumulate years of appointment data. You defiantly would not want to load every appointment for 10 years just to see today's appointments. This is the muscle of a database used in conjunction with the schedule. This chapter basically creates an example application and perhaps should be in the examples section; however it pertains more to the database access category than it does to the example category. This is actually another example application and its code may be downloaded from the Gravitybox website.

Creating the Table Structure

The first thing to be done is to create the table structure. We need to decide on a structure that will save all of our information in an efficient format. Let us assume that we are building an application to schedule patients. We will need to keep up with the patients as well as their appointments.

We will define a simple scheduling database. We will build an application that displays the rooms of an office, so the schedule will be in room-only mode. We will display each day in a separate window and make all schedules MDI child windows. This will organize the display somewhat so that the user may view one day in a window. We will assume that we need to add the functionality to setup categories and rooms as well.

We will need three tables: Category, Room, and Schedule. These will save all of the information that is needed to construct a fully functional database scheduling application. The first two tables are configuration tables and the last table stores the appointments. For a bare-bones application we would really only need the Schedule table, but we are going to allow the user to configure categories and rooms as well.

We start with the "Room" table. It will store each room that is to be displayed. The table structure follows.

```
RoomId  AutoNumber
Name  String
SortOrder  Long Integer
```

The “RoomId” field will create a unique key by which each room may be identified. The “Name” field is the text that is displayed for a room on the schedule. Finally the “SortOrder” field allows the user to control the order in which the rooms are displayed. After adding a room the user may set the order in which each appears The table structure, though simple, will serve our needed purpose of storing room configurations.

The other configuration table is the “Category” table. Each appointment may have an associated category. This will appear as a colored bar in its left margin. This aids the user in summarizing information about an appointment without having to look at a detail screen. A category is nothing more than a name and a color. The table definition may be defined as follows.

```
CategoryId  AutoNumber
Name  String
Color  Long Integer
SortOrder  Long Integer
```

The table is similar to the “Room” table with the addition of the “Color” field. This field is used to store the numeric value of a color. This is the color of the category bars drawn on the schedule to visually represents the category. This other fields serve the same purposes as they do in the “Room” table, to provider a unique key, display name, and order.

Finally the main table is named “Schedule”. This table will store all of the necessary information for an appointment. The following fields define an appointment: Date, StartTime, Length, Description, Room, and Category.

```
ScheduleId  AutoNumber
StartDate  Date
StartTime  Time
Length  Long Integer
Description  String
RoomId  Long Integer
CategoryId  Long Integer
```

The “ScheduleId” field will create a unique key for this appointment. The “RoomId” property is a pointer to the “Room” table. This will be a reference to some record in that table. The “CategoryId” field does much the same. It references some record in the “Category” table. The other properties are self-explanatory.

Loading Appointments

Now that the table structure is defined, we will build the code that accesses those tables to load the schedule. After the user chooses a date to load, it is handed to a routine that will open the database and retrieve the appropriate records. On the MDI child form the “LoadSchedule” performs this action.

```
Private Sub LoadSchedule()

'This procedure Loads a Schedule from the database
'and populates this form. This method assumes a
'database with a table named “tbl_Schedule” and
'also a schedule control named “Schedule1”

Dim Db As ADODB.Connection
Dim rs As ADODB.Recordset

  'Load this day's information from the Database
  Set Db = New ADODB.Connection
  Db.ConnectionString = GetConnectString & _
         AppPath & "schedule.mdb"
  Call Db.Open
  Set rs = New ADODB.Recordset
  Set rs = Db.Execute("select * from tbl_Schedule where
StartDate = #1/1/2002#")

  'Loop through and display all the appointments
  While Not rs.EOF
    Call Me.Schedule1.ScheduleItems.Add("", _
         #1/1/2002#, rs!RoomName, _
         rs!StartTime, rs!Length, _
         rs!Description, rs!CategoryName & "")
    Call rs.MoveNext
  Wend

End Sub
```

This code will load the all of the appointments that have been scheduled for this particular date.

Before the appointments are loaded, there is code to load the rooms and categories of the schedule. This is necessary for the schedule is display correctly, of course. The rooms are loaded from the database where they have been previously setup from the room configuration screen. That screen allows the user to specify a number of rooms and a name for each.

```
Private Sub LoadRooms()

'This procedure Loads a Schedule from the database

Dim Db As ADODB.Connection
Dim rs As ADODB.Recordset
Dim objRoom As CRoomEl
```

```
  'Load this day's information from the Database
  Set Db = New ADODB.Connection
  Db.ConnectionString = GetConnectString & AppPath _
         & "schedule.mdb"
  Call Db.Open
  Set rs = New ADODB.Recordset
  Set rs = Db.Execute("select * from Room order by SortOrder")

  'Loop through and display all the appointments
  Call Me.Schedule1.Rooms.Clear
  While Not rs.EOF
    Set objRoom = Me.Schedule1.Rooms.Add(rs!Name)
    objRoom.Id = rs!RoomId
    Call rs.MoveNext
  Wend

  Set Db = Nothing
  Set rs = Nothing

End Sub
```

In this example, we are going to give the user the capability of assigning a category to each appointment. This will give a graphical use to the schedule. The viewer can see at a glance to which category an appointment belongs by looking at the colored bar in its left margin. Category setup is very similar to room setup with the addition of a color parameter. The following code will load the defined categories from the database into the schedule's Categories collection.

```
Private Sub LoadCategories()

'This procedure Loads a Schedule from the database

Dim Db As ADODB.Connection
Dim rs As ADODB.Recordset
Dim objCategory As CCategoryEl

  'Load this day's information from the Database
  Set Db = New ADODB.Connection
  Db.ConnectionString = GetConnectString & _
          AppPath & "schedule.mdb"
  Db.Open
  Set rs = New ADODB.Recordset
  Set rs = Db.Execute("select * from Category order by
SortOrder")

  'Loop through and display all the appointments
```

```
  Call Me.Schedule1.Categories.Clear
  While Not rs.EOF
    Set objCategory = Me.Schedule1.Categories.Add(rs!Name,
rs!Color)
    objCategory.Id = rs!CategoryId
    Call rs.MoveNext
  Wend

  Set Db = Nothing
  Set rs = Nothing

End Sub
```

Saving Appointments

Now that we have addressed loading, we need to think about how to get all of that information into the database in the first place. The saving routines should not be very difficult to understand after viewing the loading routines. The save method will remove the appointments for the specified date and save the ones on the schedule as that date's new appointments.

This is not a perfect scheme. Some will disagree with removing appointments that have not changed only to read them. This is a valid point but the code was written this way for a few reasons. First this is an example of how to save to a database, not a tutorial on algorithm perfection. This said, I agree that examples should teach the "correct" way to build code and this is a perfectly valid way to build it. There is a more important reason that we are removing all appointments first; appointment moves. Posit, in an MDI environment, the user may open multiple dates. This database example application opens one window for each day. If the user opens two (or more) days and moves an appointment from one day to the other, there is a problem at save time. If you choose to merely loop through the appointments and save those appointments that have their "IsDirty" property set to true, you will add an extra appointment each time you move across windows. This is because the source schedule will lose an appointment and since no appointments on the schedule have changed, you save nothing. The destination schedule has one new appointment, so you save it. Now think about it, the source date still has the moved appointment in the database. Now you save the same appointment with the destination date. You have one more appointment than the number with which you started. You may use the appointment's UniqueKey property to overcome this limitation, but this is beyond the current scope.

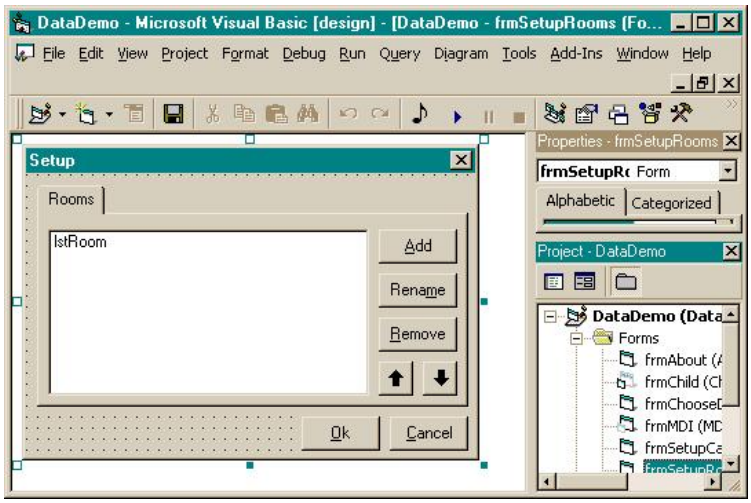
```
Public Sub SaveSchedule()

'This procedure saves a schedule to the database

Dim Db As ADODB.Connection
Dim sSql As String
Dim lCategoryId As Long
```

```
Dim oAppt As CScheduleEl

  Set Db = New ADODB.Connection
  Db.ConnectionString = GetConnectString & AppPath & _
                        "schedule.mdb"
  Call Db.Open

  sSql = "delete from tbl_Schedule where StartDate “ & _
                       “ = #1/1/2002#"
  Call Db.Execute(sSql)
  For Each oAppt In Schedule1.ScheduleItems

    If Schedule1.Categories.Exists(oAppt.Category) Then
      lCategoryId = Schedule1.Categories(oAppt.Category).Id
    Else
      lCategoryId = 0
    End If

    sSql = "insert into tbl_Schedule (StartDate, " & _
           "StartTime , Length, Description, RoomId, " & _
           "CategoryId) values (" & _
           "#1/1/2002#," & _
           "#" & oAppt.StartTime & "#," & _
           oAppt.Length & "," & _
           "'" & oAppt.DisplayText & "'," & _
           Schedule1.Rooms(oAppt.Room).Id & "," & _
           lCategoryId & ")"
    Call Db.Execute(sSql)
  Next

 Changed = False

End Sub
```

This example illustrates the reason we remove all appointments and resave them. There are other schemes of course, some of which may be quite complicated. You could, for example, store the appointment’s database key in the ScheduleItem’s Id property. Then on save the source date’s schedule, you could update the appointment by adding this “Id” to a SQL where clause, to uniquely identify the appointment. However you must also loop and create a delete query to remove all appointments that are not now in the set of appointments for a particular date. Does that sound complicated? It is a bit and depending on the database and number of appointments could be quite time consuming as well. I have chosen the quick and dirty approach. You may feel free to construct arbitrarily, complex algorithms that accomplish the same thing as I have done here with minimal code.

I think that this demonstrates quite well how to build a simple database program with the schedule. It does however only show the loading and saving of appointments. Perhaps you wish to configure rooms, categories, providers, etc. We have seen how to load room and categories, but how do we configure and add them? We will start by creating a simple Room configuration screen.

Figure 5.1



We start by creating the screen. I have chosen to use standard controls like list boxes and buttons because this is the lowest common denominator for learning and also ensures that everyone can create the screen, without any other third-party components. The setup screen will have a listbox and five configuration buttons, in addition to the Ok and Cancel buttons. The configuration buttons will consist of an add, rename, and remove button, as well as the up and down buttons. The first three do exactly what they are named. The up and down buttons allow the user to specify the order in which the rooms are displayed on the schedule, by reordering previously added rooms.

We will store the database information in a temporary collection while the configuration screen is visible. This is done so that we may make changes and they will not be saved until we wish for them to be saved. The program contains the “CItemCol” and “CItemEl” class definitions. The first one will be used to create a collection of objects that will store the necessary information for each room.

```
Option Explicit

Dim Rooms As New CItemCol
Dim Changed As Boolean
Dim arrDeleted() As Long
```

Also notice in the form’s declaration section there is a “Changed” variable. This stores the dirty state of the screen. It is used to determine if the screen needs to be saved

upon exit. The deleted array will store the Id of any room that has been removed while using the screen.

```
Public Sub LoadForm()

Dim NewEl As clsItemEl
Dim Db As ADODB.Connection
Dim rs As ADODB.Recordset

  Set Db = New ADODB.Connection
  Db.ConnectionString = GetConnectString & AppPath _
             & "schedule.mdb"
  Db.Open
  Set rs = New ADODB.Recordset
  Set rs = Db.Execute("select * from [Room] order by
sortorder")

  Call Rooms.Clear
  While Not rs.EOF
    Set NewEl = Rooms.Add(rs!Name)
    NewEl.Id = rs!RoomId
    rs.MoveNext
  Wend

  'Deallocate objects
  Call rs.Close
  Call Db.Close
  Set rs = Nothing
  Set Db = Nothing

  Call Redraw

End Sub
```

After the Rooms collection is loaded, we need populate the listbox with the room names. This is performed in the “Redraw” method. It will clear the listbox and load it with the contents of the Rooms collection.

```
Private Sub Redraw()

'Reloads the combo with the Rooms

Dim i As Integer

  Call lstRoom.Clear
  For i = 1 To Rooms.Count
    Call lstRoom.AddItem(Rooms(i).Name)
```

```
  Next i

End Sub
```

The “SaveForm” method will save the configuration the user has specified. It loops through the Rooms collection, to determine if the room has a database entry. If it does its “Id” property is non-zero. On load from the database, each room has its unique non-zero number stored in its “Id” property. If the room was added in this session of the configuration, it has not been saved to the database yet. If there is no database record then one must be created. It there is a record then the record must be updated. After the save have been performed, we loop through the deleted array and remove all of these room entries from the database.

```
Private Sub SaveForm()

Dim i As Integer
Dim NewEl As CItemEl
Dim Db As ADODB.Connection
Dim sSql As String
Dim oRoom As CRoomEl

  Set Db = New ADODB.Connection
  Db.ConnectionString = GetConnectString & _
            AppPath & "schedule.mdb"
  Call Db.Open

  For Each oRoom In Rooms
    If oRoom.Id = 0 Then
      sSql = "insert into [Room] (Name, SortOrder) " & _
             "values ('" & DoubleChar(oRoom.Name, "'") & _
             "', " & i & ")"
    Else
      sSql = "update [Room] set Name ='" & _
             DoubleChar(oRoom.Name, "'") & _
             "', SortOrder=" & i & _
             " where RoomId = " & oRoom.Id
    End If

    Call Db.Execute(sSql)

  Next i

  'Delete all the one we removed in this session
  For i = 0 To UBound(arrDeleted) - 1
    sSql = "delete from [Room] " & _
           "where RoomId = " & arrDeleted(i)
    Call Db.Execute(sSql)
```

```
  Next i

  'Deallocate objects
  Call Db.Close
  Set Db = Nothing

End Sub
```

The Categories screen is setup in a very similar way. This only difference being that it has the extra property of “Color” defined. The loading and saving of the categories is almost identical. Though not a part of this database example, a Providers collection could also be defined similarly. Each ScheduleItem on the schedule has a “Provider” property that may be mapped to an element in the schedule’s “Providers” collection. This is used to define a set of people associated with a schedule. They too may have a related color, that displays in the appointment’s left margin, if need be.

This concludes the database example. There are many ways to implement this functionality. The proposed methods are only guidelines. There are schemes that differ in complexity but accomplish the same task: database saves and loads. This was simple scheme and may not scale well. However for most applications there may be no advantage to spending countless weeks optimizing convoluted algorithms. Then again may be there will be for your situation.

Part III
Display Modes

My cup runneth over.

-Psalm xxiii. 5

Chapter 6

What is a DisplayMode?

There is more than one way to display a schedule. GbSchedule makes multiple display modes available. A display mode is nothing more than a set of properties that define a “look-and-feel” of a particular schedule layout. Many users require specific ways to display data tailored to their exact needs. To keep the control as general-purpose as possible, very few properties may be used to control the display in a remarkable variety of ways. The property that most notably controls the display behavior of the schedule is the Viewmode.

ViewMode

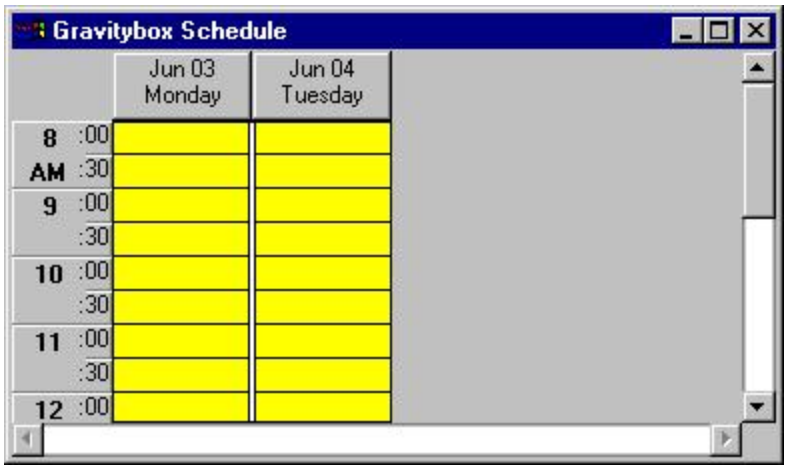
The Viewmode property determines the actual layout of the schedule. A wide variety of displays may be had. You may control the display of the days, rooms, and times with this property. The visibility of each may be toggled on and off. Also the position of each may be changed. For example the time may be displayed in the top margin and the dates in the left margin.

Viewmode Property Settings

vmcNormalDayTopTimeLeft – The days are displayed on the top and the time increments are displayed in the left margin, There are no rooms displayed.
vmcNormalRoomTopTimeLeft - The rooms are displayed on the top and the time increments are displayed in the left margin, There are no days displayed.
vmcNormalDayRoomTopTimeLeft – The days and rooms are displayed on top and the time increments are displayed in the left margin.
vmcNormalDayLeftTimeTop - The days are displayed in the left margin and the time increments are displayed in the top margin, There are no rooms displayed.
vmcNormalRoomLeftTimeTop – The rooms are displayed in the left margin and the time increments are displayed in the top margin, There are no days displayed.
vmcNormalDayRoomLeftTimeTop - The days and rooms are displayed in the left margin and the time increments are displayed in the top margin.
vmcNormalDayLeftRoomTopNoTime – The days are displayed in the left margin and the rooms are displayed in the top margin. There are no times displayed.
vmcNormalDayTopRoomLeftNoTime - The rooms are displayed in the left margin and the days are displayed in the top margin. There are no times displayed.
vmcMonth – The schedule is displayed one month at a time. This is a calendar with appointments listed in the day cells.
vmcList – The schedule is displayed one day at a time. Each day has times listed vertically and the appointments placed at their respective times.

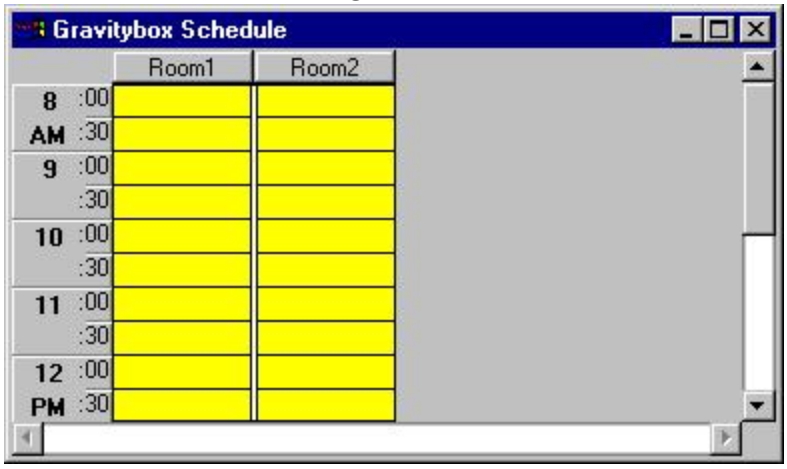
The many view modes allow the schedule to be displayed in almost any fashion that you may need. The display the days on the top axis and the times, you set the Viewmode property to the “vmcNormalDayLeftTimeTop” setting. This creates a schedule similar to the schedule in Figure 6.1.

Figure 6.1



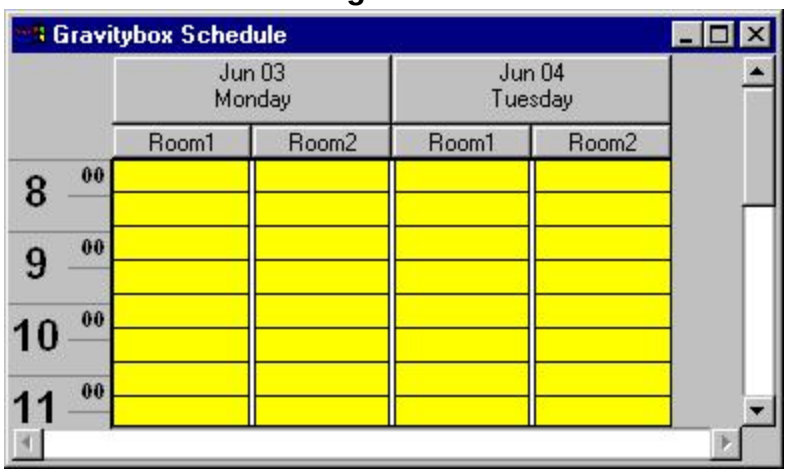
If Viewmode is set the “vmcNormalRoomTopTimeLeft” setting, the control displays only room information. There will be no dates displayed on either margin. Rooms as defined by the Rooms collection and will appear opposed by times in the opposite margin.

Figure 6.2



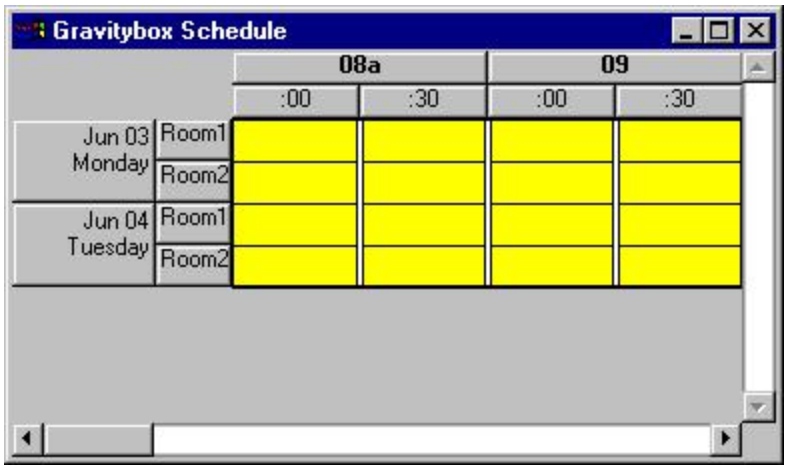
Another useful way to display the schedule is to combine days with rooms. This will allow you to define a number of days that are associated with each day. For each day, every Room in the Rooms collection will be displayed. This means that Room1 for Day1 may be scheduled separately from Room1 for Day2.

Figure 6.3



Each of the above property settings also has a reverse setting in their positioning. Instead of displaying the time in the left margin, you may choose to display it in the top margin. There is no right or wrong way. The correctness of the positioning depends on the needs of your application. In contrast to the previous figure, the times of Figure 6.4 are displayed in the top margin.

Figure 6.4



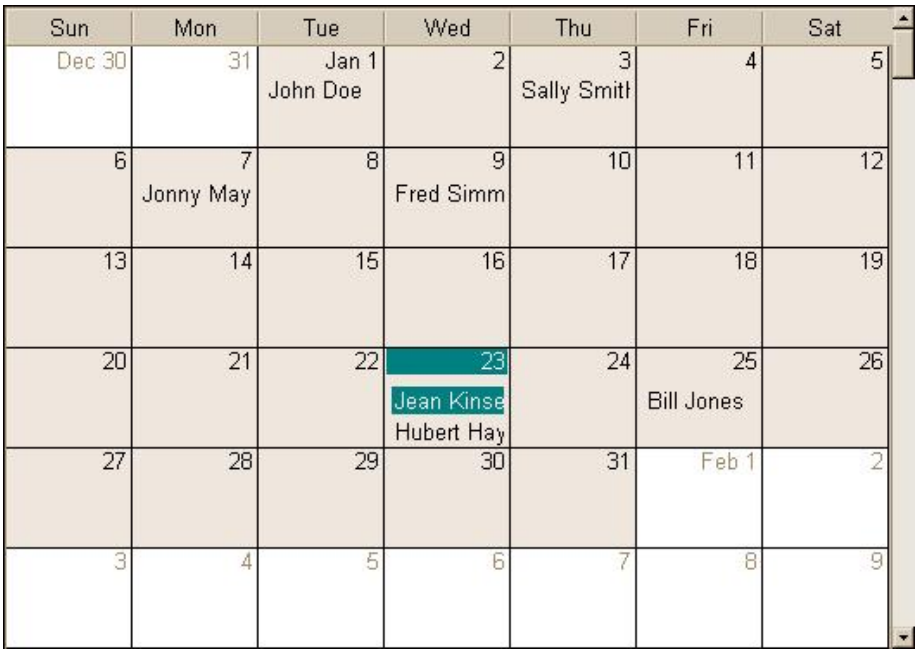
The schedule may also be displayed without any times at all. The rooms and days may be plotted against each other, with one in the left margin and the other in the top margin bringing the total to eight “normal” modes.

Quick Tip
Provider and Category information is not displayed in every ViewMode.

The Provider and Category collections will be displayed only when the times are visible and they are displayed in the left margin. When the time is displayed on the top of the schedule, only basic schedule information is displayed. This is a limitation of the schedule of this time. The colored bars that describe providers and categories are only visible if the time is displayed on the left as well.

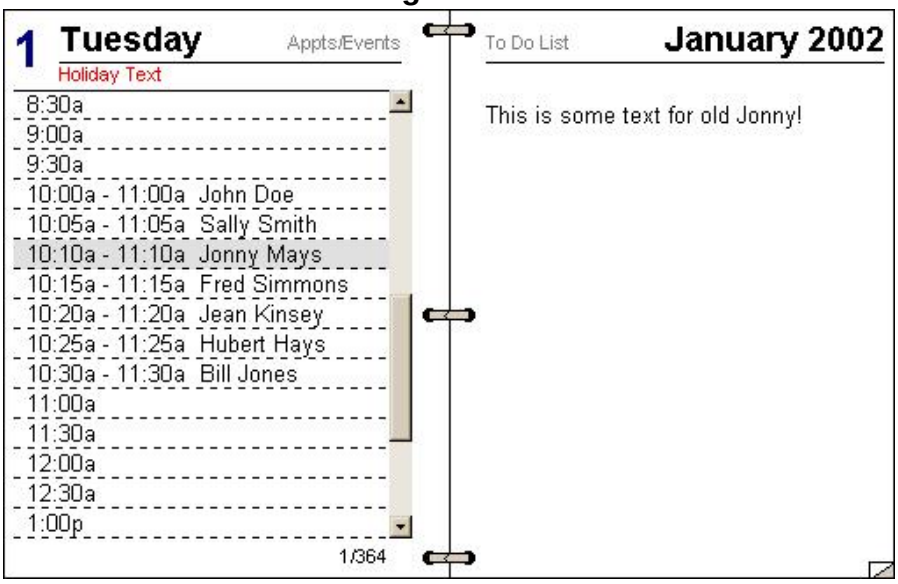
A schedule may also be set to display a month at a time using the “Month” mode setting. Depending on the MinDate and MaxDate settings, a scroll bar may appear to allow for the scrolling through the allowed range of time. Even if the MinDate and MaxDate are the same, the schedule will display one month. There is no way to display only part of a month. The schedule display is formatted as a standard calendar. The range of the schedule is determined by the month and year of the MinDate and MaxDate. The minimum and maximum range is not determined by the days of these two dates. In “Month” mode all months start on the first and end on the last day of the specified month. The schedule property “FirstDayOfWeek” determines the day of the week that is displayed in the first column. The default is Sunday, however it may be set to any day of the week. If your application is being deployed in Europe, you may wish to set this property to Monday for the comfort of users.

Figure 6.5



The third view is “List” mode. This allows the user to view the schedule in a list from top to bottom, with appointment displayed in descending order. You may optionally choose to display appointments in two fashions. All times can be listed with appointments placed at their respective places in the time list or the appointments may be displayed alone with no time list. In “List” mode, the appointment’s notes may be displayed and edited on the right side of the screen if necessary. The notes section may be toggled on and off based on the need to the containing application. When toggled on the schedule is displayed in a two-page format with a dividing binder in the middle. If the notes section has been toggled off, the display is one page with no divider.

Figure 6.6



Quick Tip
Be careful of information that is lost or created when dragging appointments between schedules with differing ViewModes.

Appointments may be dragged from a schedule in any view and dropped on any another schedule of any view. For the most part, there are no surprised or anomalies. There are caveats to keep in mind. If an appointment is created in "Month" or "List" mode it has no Room property. More precisely its room property is set to 0. If the ViewMode is toggled to "Normal" mode and the DayRoomMode is set to "RoomOnly" or "Both" mode then the appointments previously created will not be displayed. This is because those appointments have their Room property set to 0 and this room never exists. So when the schedule is attempting to display appointments it does not find any valid room for these and thus does not draw them. This applies to appointments with any invalid Room value. If an appointment is created in "Both" mode with dates and rooms displayed then the appointment will have a valid Room value. If the schedule is toggled to "Month" mode this value is retained even though there are no Rooms in "Month" mode. It an appointment is moved or copied its Room value stays in tact. If the schedule is toggle back to "Normal" mode then it will still be displayed in the same Room as it was originally created.

WeekNumbers

Quick Tip
European and Asian applications may wish to display the week number in Month mode.

In many countries user will demand to view the week numbers on their schedule or calendar. For instance, in Europe many people are accustomed to specifying days with the week number like so: "Tuesday of the 23 week in 2003". To make your application more suited to these users you may wish to include this functionality on your schedule. Using the AllowWeekMargin and WeekMarginCaption properties, you may customize the look of the schedule. The AllowWeekMargin property will turn on week numbering so that the actual number is visible. The WeekMarginCaption property allows you to ad a small description of what the number is. In calendar mode there is very little room so the description might be simply "w" to indicate week. However in another

language you could change the text it to reflect the local word for week. Currently week number of only displayed in List mode and Calendar mode. All of this adds to the internationalization of your application. Also, if you ever need to determine the week number of date the method “GetWeekNum” method returns this information.

ScheduleIncrement

Quick Tip
The time resolution of the schedule may be any divisor of 60 minutes.

Every schedule will have different resolution needs. This property will allow the schedule resolution to be configured in relation to time increments. The property defines the smallest time block visible on the schedule. The displayed time must be carved into some sized pieces. This property provides the time slice size. Each one the valid values for this property is a factor of 60. This is the number of minutes in an hour and the hour must be sliced evenly for a coherent display. Its possible values are 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, and 60. For example, if the ScheduleIncrement is set to 20 minutes, every hour will be displayed across three rows (or columns). The smaller the ScheduleIncrement value, the more detailed each hour becomes. Figure 6.4 is displayed with a ScheduleIncrement of 30. The time is displayed with two rows per hour.

AllowColumnResizing

Quick Tip
To allow the user to resize columns, set the AllowColumnResizing property to True

This property, when set to true, will allow the user to dynamically resize columns. This allows the column widths to be locked at a predetermined size or it allows the user to modify their size will. If true, the user may move the mouse pointer over the column break, until the mouse pointer changes to a resize icon. At this time, the user may click the left mouse button and drag the mouse, until the column resizes to the preferred width.

AllowRowResizing

Quick Tip
To allow the user to resize rows, set the AllowRowResizing property to True

This property, when set to true, will allow the user to dynamically resize rows. As with columns, the row size may be set to some predetermined size or the user may have access to modify it. Again, the user must move the mouse pointer over the row break until the mouse pointer changes to a resize icon. The user may then drag the mouse until the row size is the preferred height.

TimeFormat

In different countries, the time is displayed in different formats. There could be a display problem if the schedule is forced to always be display the same way. GbSchedule allows for the time to be displayed in 12-hour or 24-hour format, based on the setting of this property. In 12-hour format, the hours 1-12 are used as well as an “AM/PM” symbol. In the 24-hour format, the time is displayed with the hours 0-23. The latter format is the

one preferred by much of Europe and will receive a warmer reception there than the American 12-hour format.

HeaderDateFormat

This is a most useful property. When deploying a scheduling application around the world, each country insists on its own date format. This property determines the format of the date displayed in the header of each day. This property is also useful to customize the date display, even if the application is never deployed over seas. Line breaks may be specified as well to display any type of custom format.

ImageList and IconAlign

The schedule has a property “ImageList” that may be set to a valid reference of a Microsoft Common Controls ImageList. This ImageList may contain icons that you wish to associate with an appointment. First add the icons you wish to use and make sure that each icon has a its key property value set. The images look best if they are 16x16 and the ImageList’s “UseMaskColor” property is set to true. Each appointment has an “Icons” collection that may be used to display custom icons with an appointment. To specify icons just call the collection’s “Add” method with the icon key you wish to use. If the key is not valid it is just skipped when displaying icons on the schedule canvas. No ruin-time error occurs. When drawing an appointment, if the schedule finds a valid ImageList and one of its icons ha a key that matches a key in the Icons collection, the associated icon is drawn on the appointment. You may specify any number of icons for an appointment. Icons that do not fit on the appointment will not be displayed at al. This is a limitation of small appointments. The appointment square is only so big and if you specify 100 icons, all of them will probably not fit. This brings up another property, IconAlign. Icons may be displayed on the left or top of the appointment text. You may decide based on whichever looks the best for your situation.

CategoryBar

Even though Category objects have been defined in the Categories collection, their display may not be needed. The CategoryBar property will allow for the selective display of defined categories so that they may or may not be displayed in the left margin. The associated mapped colors may still be displayed on the appointment’s margin by setting the AppointmentCategoryBar property to true.

Note: If category bar margin on the left is desired to be as small as possible, set the DisplayName of the Categories collection to empty string. The margin is the smaller width of this caption or the summation of the bar widths. Setting the caption to have a zero width ensures that just enough space to display the category bars is used.

CategoryBarWidth

This property determines the width of two different kinds the category bars. The first is the category bar displayed in the left margin of the schedule. If the CategoryBar property is True, the categories will be listed in columns in the left margin. The width of each bar (column) is defined by the CategoryBarWidth property. The second place a category bar may be displayed is on the left edge of each appointment. If an appointment has an associated category, the category's color bar may be displayed on the appointment. The width of this bar is also determined by this property.

ShowProviderAvailableTime

This property provides functionality similar to the CategoryBar property. There may or may not be providers defined in the Providers collection. In any case, the displayed bars may be toggled on and off. When this property is true and the CategoryBar property is true, there will be two tabs on a tab strip in the top, left corner, of the schedule. When the bottom tab is selected, the available times for all of the providers will be displayed. The available times are the times defined for each Provider object's AvailableTimes collection. This collection is defined later under Advanced Functionality.

Note: If this bar margin on the left is desired to be as small as possible, the CaptionAvailable property of the Providers collection may be set to empty string. The margin is the smaller width of this caption or the summation of the bar widths. Setting the caption to have a zero width ensures that just enough space to display the bars is used.

ShowProviderScheduledTime

Each appointment may have an associated provider. When an appointment has one, the provider is scheduled for the duration of the appointment. Again, if the bottom tab on the tab strip in the top, left corner of the screen is selected, the schedule is in provider mode. There will be provider bar displayed in the left margin that indicates the times for which each provider has been scheduled. For each appointment that has an associated provider, the colored, provider bar will be displayed it the left margin of the appointment. The ShowProviderScheduledTime property allows for its selective display.

Note: If this bar margin on the left is desired to be as small as possible, the CaptionAvailable property of the Providers collection may be set to empty string. The margin is the smaller width of this caption or the summation of the bar widths. Setting the caption to have a zero width ensures that just enough space to display the bars is used.

ProviderBarWidth

Similar to the CategoryBarWidth property, this property determines the width of the provider bars. As with categories, provider bars may be in the left margin of the

schedule or in the left margin of an appointment. The width of both is determined by the ProviderBarWidth property.

Chapter 7

Area Availability

On a schedule, it may be important to check for the visibility of items. Also sometimes you may need to make items visible, if they are not already so. There exist a variety of methods that allow the visibility any item of a schedule to be checked and set. An item may be defined as a date, room, time, or appointment. There are methods that may be used to determine the visibility of any date or time. In addition, if one of them is not visible, it may be made visible with another function call. These methods may be used to scroll the viewable window of a schedule to a particular place.

Though the defined methods in this chapter may be used to determine area availability, there is an easier way. The methods give you the power to perform manual checks on arbitrary area. If you wish to merely find the next available slot starting at some location you may use the “GetNextFreeSlot” method. This functionality is defined in the next chapter.

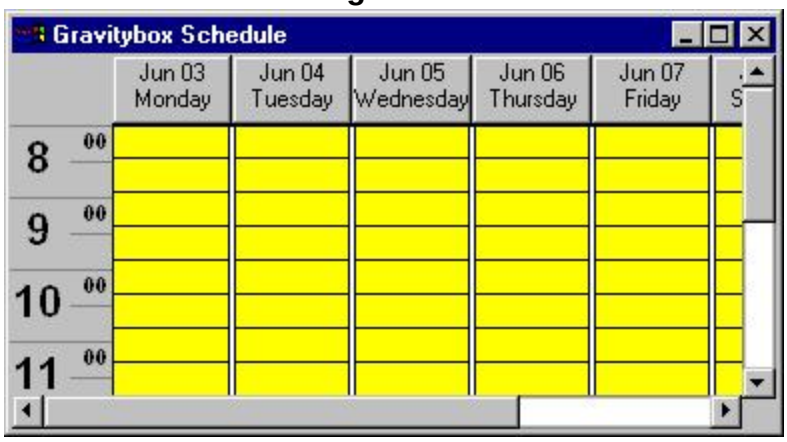
IsDayVisible

Quick Tip
You may check if a day is in the viewable area by using the “IsDayVisible” property.

In the course of building and maintaining a schedule the visibility of particular date will eventually need to be known. This information may be obtained with a simple method call. Given a particular date, this method returns a Boolean value that determines the visibility state of the specified date. If the date is only partially visible, this method returns false. The start edge to the end edge of the date must be in the view port for a true return value.

```
If Not Schedule1.IsDayVisible(#6/3/2002#) Then
  ....
  'What to do if NOT visible
  ....
End If
```

Figure 7.1



In the Figure 7.1, the date Jun 8 is not completely visible. Though the starting edge is visible, its end edge is beyond the view port. If the IsDayVisible function were to be called with this date, it would return false, since the entire room is not visible.

This functionality may be used to add search capability to an application. Let the user specify a date to which to move. On a large schedule this functionality is almost mandatory. An application that requires a user to scroll for over 300 columns (days) to get to a known point is not a very efficient application.

IsRoomVisible

Quick Tip
You may check if a room is in the viewable area by using the "IsRoomVisible" property.

When dealing with many rooms, there may be more rows (or columns) than can be displayed in the view port of the schedule. This IsRoomVisible method allows for the checking of visibility of any Room object. Given a Room object's index or name, the method returns a Boolean value that determines the visibility of the specified Room object. As with the IsDayVisible method, if a room is only partially visible, this method returns false. The start edge to the end edge must be in the view port for a return value of true.

```
If Not Schedule1.IsRoomVisible("Room1") Then
  ....
  'What to do if NOT visible
  ....
End If
```

This functionality may be used to add search capability to an application as well. This will probably not be such an issue, since there will probably not be very many rooms on a schedule. However, the Rooms collection may represent any group of objects. Rooms may be a group of Trucks. The Rooms collection would be labeled "Trucks" and treated as such. This collection allows for the addition of any item. So there may be many items in this collection after all. For this reason, "IsVisible" and "Show" methods are defined for the Rooms collection as well.

IsTimeVisible

Quick Tip
You may check if a time is in the viewable area by using the "IsTimeVisible" property.

Like its two sister methods, the IsTimeVisible method returns with a Boolean value that determines the visibility of a particular time. If the time slot is only partially visible, this method returns false. The start edge to the end edge must be in the view port for a return value of true.

```
If Not Schedule1.IsTimeVisible(#12:00:00 PM#) Then
  ....
  'What to do if NOT visible
  ....
End If
```

ShowDay

The ShowDay method is the complement to the IsDayVisible function. It may be used in conjunction with it to show a date that is not currently visible. The day is not guaranteed to be displayed at any particular spot in the view port. It is only guaranteed that it will be visible after this method is called.

```
If Not Schedule1.IsDayVisible(#6/3/2002#) Then
  Call Schedule1.ShowDay(#6/3/2002#)
End If
```

Search capability for dates may be added to any scheduling application very easily, with a combination of these functions.

ShowRoom

This is the complement method to the IsRoomVisible method. The visibility of a Room object may be determined and displayed within the view port, if necessary.

```
If Not Schedule1.IsRoomVisible("Room2") Then
  Call Schedule1.ShowRoom("Room2")
End If
```

As expressed earlier, this method may be of limited use if displaying a limited number of rooms. If many rooms are present, this method may be very useful. There are only so many rooms that an office may have in reality. A more productive use of this method would be apparent when the Rooms collection defines something other than rooms. It could define people, objects, planes, or any other physical object. There may be many more of these objects and search capability would be needed more.

ShowTime

This is the complement function of the IsTimeVisible method. This method may be used in the same fashion as the ShowDay and ShowRoom methods. Given a valid time, the method will bring the time into the view port. It may not necessarily be the time displayed at the top of the screen. The displayed top time will depend on the length of the total schedule and other factors.

```
If Not Schedule1.IsTimeVisible(#12:00:00 PM#) Then
  Call Schedule1.ShowTime(#12:00:00 PM#)
End If
```

ShowItem

Another useful method that allows for the display of particular sections of a schedule is the ShowItem method. It is perhaps the most useful. If the user has search capability, he will want to display the results of his search. This method is called to make a particular appointment visible. On large schedules, there will be many appointments, most of which will probably not be in the view port window. If the user needs to see a particular appointment, the view port may be scrolled to include the specified appointment.

```
Call Schedule1.ShowItem(32)
```

This code will bring ScheduleItem 32 in the ScheduleItems collection into the view port. If at least 32 appointments do not exist, in the collection, this method will perform no action.

This method may be used to define search capability in the following way. There exist many appointments on the schedule. Each appointment has the client's (patient's, etc) name displayed on it. You may build a screen may be built that loops through the ScheduleItems collection and displays a list of the patient names from every appointment. This will be a visual cue to the user of what each appointment represents. When the user indicates that he wants to edit (view, etc) the item in the X position, this method may be called with the ScheduleItems' index X as a parameter. This will ensure that the specified appointment is in the view port of the schedule.

```
Private Sub List1_DblClick()
  Call Schedule1.ShowItem(List1.ListIndex + 1)
End Sub
```

These methods may be used in conjunction with one another to check the visibility of any place on a schedule. To gain faster performance, the AutoRedraw property of the schedule may be set to false for multiple method calls that affect screen refreshing. The property must be set back to true after the methods calls, to ensure that the screen is

refreshed. This will move the schedule to the desired position before it redraws the screen.

IsEnabledAreaByValues

> **Quick Tip**
> You may check an area's availability by using the IsEnabledAreaByValues method. This will ensure that there are no conflicts for an area.

The "IsEnabledAreaByValues" method will take differing parameters depending on the display options of the schedule and return to you the availability of a particular area. To start, you specify the date, room, and time for which an appointment starts. Also provide as a parameter the length of the area to check. These four parameters define a definite area. The method checks to determine if any part of this area is inside of an existing appointment. It returns a Boolean value indicating this determination. This is an iterative algorithm, so the longer the specified length, then longer it takes to execute. There may be times that you wish to only check the time component of availability. If so the last parameter allows you specify a Boolean value indicating whether you wish to check time only. When this parameter is true all other criteria are ignored. A good place to use this function is in the BeforeMove, BeforeCopy, or even the DragOverScheduleItem schedule. You will probably only use this method in very customized situations since it is built in to the dragging process. When you move or copy an appointment, the Conflicts collection is checked automatically and prompts the user if the ConflictWarn property is true. Remember that the Conflicts collection is only checked and updated automatically if the schedule's ConflictCheck property is true. If you do not care about conflicts, you may turn this feature off for faster execution.

GetScheduleItemFromCor

This method will return the first appointment that it finds under the specified coordinates. Given an X and Y coordinate in pixels, this method will check the ScheduleItems collection to determine the appointment that is scheduled for the date, time, and/or room to which these coordinates map. Keep in mind that there may be more than one. If there is a conflict with two appointments scheduled for the exact same date, time, and room, only the first appointment is returned. The first appointment position is defined by its order in the ScheduleItems collection. If one appointment was the 17th element in the collection and the other conflicting appointment was the 57th element, then the 17th element would be returned.

EnforceTimeLimits

This property ensures that the default dialog will not allow not allow the user to create an appointment not defined by the time area boundaries. The StartTime and DayLength properties define the display times of a schedule. When EnforceTimeLimits is set to true, the user may not set an appointment's StartTime or duration to a value outside of this defined area. When false, the user may set the time to any value. This all is part of the default property dialog for an appointment. In code you may of course set the start time and length of an appointment to any value you wish. This property is provided for

convenience for those developers who use the default dialog as there appointment editor any wish to enforce the time limits set by the schedule.

HitTest

Quick Tip
The “HitTest” property is similar to the Listview’s HitTest method in that it checks for an object under a point.

The HitTest method is provided to give you the ability to determine if an appointment is located at a particular position. Given an X and Y coordinate the method will return a reference to the appointment at that position. If there is no appointment present then the value “Nothing” is returned. This functionality is provided for completeness and probably is not that useful for most developers.

Part IV
Advanced Functionality

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 8

Conflicts

In scheduling, conflicts are a part of life. There will be many times when conflicts are unavoidable. There may even be times when they are desirable. In any case, no schedule would be complete without some sort of conflict handling.

What is a Conflict?

Quick Tip
A conflict occurs when two or more appointments share the same scheduled space.

GbSchedule has a read-only collection named Conflicts. The developer may not add any elements to the collection nor may he remove any. When ScheduleItems are added to a schedule, they may or may not conflict with other appointments on the schedule. After an insert, the newly added object is checked, to verify whether it conflicts with any other objects in the ScheduleItems collection. If it does, it is added to the Conflicts collection. Any time that one of the appointment's properties StartDate, Room, StartTime, or Length is changed, the Conflicts collection is rechecked to determine whether the appointment should be added or removed from the Conflicts collection. Only items that should be in the Conflicts collection will be there. Since a change of any of many display properties (IsEvent, Length, Room, StartDate, and StartTime) on an appointment could cause the appointment to potentially conflict with another appointment, the ScheduleItems collection must be checked for Conflicts after these property changes. If the properties of a particular appointment are changed, the effect might not only influence this appointment but others as well, since it takes at least two appointments to conflict.

The Conflicts collection is located on the ScheduleItems collection object. It has no properties and only a few methods. Its methods are defined in Table 8.1. There is only a minimal amount of functionality needed, for this collection. Even though minimal, this functionality still serves the needed purpose. The collection is actually a collection of conflict groups. Each group object holds a collection of appointments. Each ScheduleItem is in exactly one group. If there are no conflicts there will be one group for each ScheduleItem. If there are N ScheduleItems, there are also N groups in the Conflicts collection, when there are no conflicts. If there are ScheduleItems that conflict with each another, they will be in the same conflict group. A group signifies a block of appointments such that occupy a contiguous block of schedule space. All of the appointments may not conflict with all others, but together as a group there is no free space from the groups starting boundary to its end.

Table 8.1
Conflicts Collection Definition

Count	This method returns the number of objects in the Conflicts collection. This number will be between 0 and N, where N is

	the number of ScheduleItems. If there is one or more appointments there will be at least one Conflict group object present.
FindItem	This method will return the conflict group object that contains the specified ScheduleItem. It takes a ScheduleItem object as a parameter and returns that object's associated Conflict object.
IsConflictByAppt	Given a ScheduleItem object this event will determine if there is a conflict with any other existing ScheduleItem. You may optionally choose to ignore one or more of the ScheduleItems as well by specifying the IgnoreIndexes.
IsConflictByData	Given the information needed to build an appointment (date, time, room, length), this method will return a Boolean value that determines if the specified space if free of appointments.
Item	Given a Conflict index [1..M], this method returns a reference to the object the collection.

The Count method will simply return the number of items in the Conflicts collection. An appointment has a conflict if it is in a group that contains more than one ScheduleItem. The "FindItem" method takes a ScheduleItem as a parameter and returns its associated Conflict group object. You may then use the Conflict object returned to determine which appointments conflict with the specified appointment by accessing the Conflict object's ScheduleItems collection. The code below will display the number of conflicts for the first appointment.

```
Dim oScheduleEl As CScheduleEl
Dim oConflict As CConflictEl
Set oScheduleEl = Schedule1.ScheduleItems(1)
Set oConflict = Schedule1.Conflicts.FindItem(oScheduleEl)
Call MsgBox("Conflict Count: " & oConflict.ScheduleItems.Count, vbInformation)
```

The "IsConflictByAppt" method is given a ScheduleItem object as a parameter. It will then return a Boolean value that determines if the specified appointment conflicts with any other appointment. This method is useful when you wish to check for conflicts but wish to ignore some appointments. You may specify any number of indexes to ignore in the search with the second parameter.

The "IsConflictByData" is particularly useful as well. You may specify the information needed to build an appointment (date, time, room, length) and this method will return a Boolean value that determines if the specified space is free of appointments. You do not need an existing appointment, just the data of a proposed appointment. This method can be used to check for free spaces. Again, the IgnoreIndexes parameter may be used to ignore any number of existing appointments of which you do not care about conflicts. The ignore indexes are numbers separated by a space, comma, colon, or semicolon.

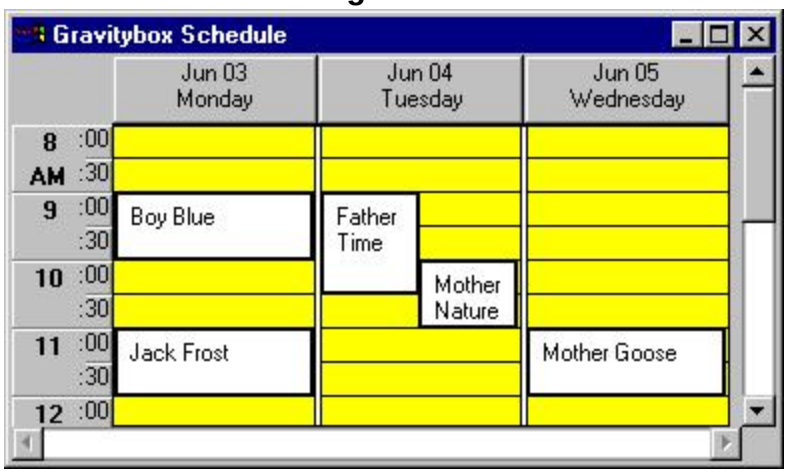
Table 8.2
Conflict Object Definition

Index	This property is the index in the Conflicts collection of the current object.
ScheduleItems	This collection holds references to the appointment in this conflict group.

Conflict Displays

Displayed in Figure 8.1 is an example with five appointments. Two of them, the third and fourth ones, conflict with each other. The ScheduleItems collection will contain five elements and the Conflicts collection will contain four group objects. Each group object will contain one appointment in its ScheduleItems collection except for the third group, which will have two. These are the third and fourth appointments.

Figure 8.1



In this example, the Count method of the Conflicts collection returns the value 4. Following is a loop that determines which elements are in the collection.

```
Dim oConflict As CConflictEl
Dim oAppt As CScheduleEl

  For Each oConflict In Schedule1.Conflicts
    For Each oAppt In oConflict.ScheduleItems
      Debug.Print oAppt.Index
    Next
  Next
```

The output from this code fragment is as follows.

```
3
4
```

This tells us that ScheduleItems(3) and ScheduleItems(4) have conflicts. Since there are only two appointments in the group we know that they conflict with each other.

Next available slot

Quick Tip
You may search for a free space on a schedule by using the "GetNextFreeSlot" method.

Before an appointment is added, you may wish to determine if its addition will cause a conflict. There is a way to do this. The GetNextFreeSlot method of the schedule may be used in a variety of situations that require conflict checks. The method is used in the following manner. Pass the parameters of date, room, start time and length and compare them with the returned results. If the sent parameters match the return parameters, this space it free, otherwise there was some appointment blocking the area. For example, if the parameters Jun 3, 2002 at 10:00AM in Room1 checking for a 60 minute appointment space are sent and the return values are the same date, time and room, there is no other appointment present in this area of the schedule.

There are times when the automated calculation of free appointment slots is necessary. In many cases, the user will drag-and-drop appointments to the desired position. If a schedule is relatively empty, it is easy to spot a free area on which to place the appointment. However finding a place for an appointment may not be so easy, if a schedule is chiefly full. An appointment slot may not be readily visible. In this case, the schedule will perform the grunt work. The GetNextFreeSlot method will take a number of parameters to define exactly where to start the search. The method will then return the first available slot, starting from the specified point, in which the specified appointment of the desired length will fit. The parameters to this method are as follows.

Table 8.3
GetNextFreeSlot Parameters

GroupId	Specify a GroupId when you wish to find the next free slot for a group of appointments. In general, this method if to find a free slot for one appointment. However is may also be used to find the next free slot of the first appointment of a Recurrence group. You can assume that all appointments in the group will have a free slot if all are move the relative distance from the starting appointment in the group to the return value of the method. For example, if the starting date/time for the group's starting appointment is Feb 3, 2004 10:00AM and the method return value is Feb 5, 2004 2:00 PM, you may assume that there is a free slot for all appointment in the group if they are all moved 2 days and 4 hours (3120 minutes). If you do not wish to search by group then leave this parameter blank.
StartDate	This is the date at which the search should begin. If in RoomOnly mode, this parameter is ignored.
StartRoom	This is the room at which the search should begin. If in DayOnly mode, this parameter is ignored.

StartTime	This is the Time at which the search should begin.
ItemLength	This is the length of the desired free slot. The desired appointment length will be used as this parameter.
IgnoreIndexes	This optional parameter specifies indexes in the ScheduleItems collection to ignore in the search. If no appointments should be ignored this value should be set to empty string (optional default). The ignore indexes are numbers separated by a space, comma, colon, or semicolon.

The method needs to know the position to begin its search. This is defined by the first three parameters: StartDate, StartRoom, and StartTime. If any one of these parameters is not applicable, it is simply ignored. For example, in DayOnly mode there is no concept of Rooms, so no matter the StartRoom parameter value it will be ignored.

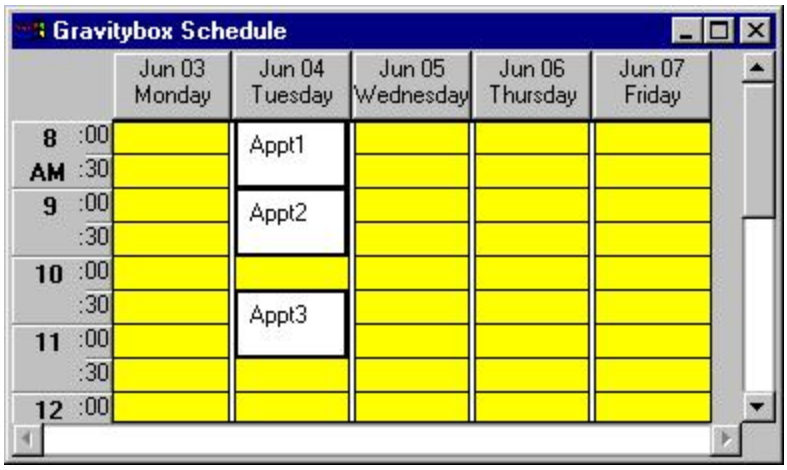
The most curious parameter is the last one, IgnoreIndexes. In most cases, this parameter will not be set. If using Java, or some other language, that does not allow optional parameters, the parameter may be set it to empty string. This value will allow for one or more ScheduleItem objects to be ignored in the search. This is useful if moving an appointment. If moving appointment 57 to the next available position, the StartDate and StartRoom parameters would be set the appointment's StartDate and Room properties respectively. The StartTime parameter would be set to the appointment's StartTime plus one ScheduleIncrement. One increment is added because the appointment already lives where it is and the search is looking for the next slot. In this case the IgnoreIndexes parameter would be set to 57. This action would ignore this ScheduleItem object when performing the search. Since the search is seeking the next slot that contains no conflicts, it does not matter if the specified slot contains this appointment, since it is being moved anyway.

We can create an example schedule with three appointments. All are one hour long. They start at 8AM, 9AM, and 10:30AM respectively. It is obvious that the next available, non-conflicting spot for appointment 2 is for 9:30AM. The appointment may be moved down thirty minutes and no conflict will occur. If a check is performed to find the next available slot without specifying an IgnoreIndexes parameter, the returned StartTime will be 11:30AM. This is because the search will look at 10:30AM as conflicting. Indeed, there is no one-hour slot available, because appointment two lasts from 9:00AM until 10:00AM. This is why the IgnoreIndexes is so important. In this example, it is not important to consider appointment two. In fact, it is important that this appointment not be considered in the search criteria, since this inclusion will cause the search to return erroneous results.

The method returns a CScheduleEl object. This object is NOT part of the ScheduleItems collection. It is returned with its StartDate, Room, StartTime, and Length properties set. All other properties of the object are default values. This object is used because it has all the necessary properties needed to define the next free slot. If there

is no free spot available before the end of the schedule, this method returns a reference to Nothing.

Figure 8.2

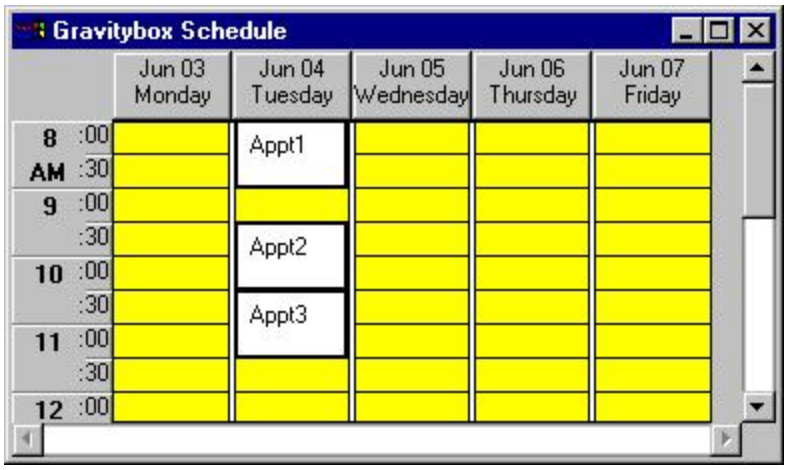


After calling the method with an IgnoreIndexes parameter set to “2”, the GetNextSlot method will return results that may be used to move the specified appointment to the next free slot.

```
Dim oCurrent As CScheduleEl
Dim oItem As CScheduleEl
Dim dtNextTime As Date

  'Appointment to be moved
  Set oCurrent = Schedule1.ScheduleItems(2)

  'Get the next time on the schedule after this
  'selected appointments StartTime
  dtNextTime = DateAdd("n", Schedule1.ScheduleIncrement, _
      oCurrent.StartDate)

  'Search for next appointment
  Set oItem = Schedule1.GetNextFreeSlot( _
          dtNextTime, _
          0, _
          oCurrent.StartTime, oCurrent.Length, _
          oCurrent.Index)

  'Move the appointment to the next
  'free slot if one exists
  If Not (oItem Is Nothing) Then
   oCurrent.StartDate = oItem.StartDate
   oCurrent.StartTime = oItem.StartTime
  End If
```

After executing this code, appointment two will be moved to its new position.

Figure 8.3



Chapter 9

Printing

Printing is the only reason some schedules are even created. Without this functionality a schedule might be all but useless. GbSchedule provides functionality to send a schedule to a printer or an HTML file. As discussed earlier, there are many different display formats and any of these may be printed exactly as they appear on the screen.

GoPrint

Quick Tip
You may print a schedule of any size or complexity using the “GoPrint” method.

The GoPrint method starts a print that creates a hard copy of the schedule. This takes a number of parameters and will send the current schedule to the printer. It is important to remember that the Viewmode and other properties must be set to the desired values before this method is invoked. The schedule will output the screen exactly as it appears on the screen. The only control that the developer wields over the printing is the range of data to be printed.

Table 9.1
GoPrint Parameters

StartDateRoom	This is the starting date or starting room, depending on the value of ViewMode. This specifies the lower value of the range of items to be printed.
EndDateRoom	This is the ending date or ending room, depending on the value of ViewMode. This must be greater than or equal to Start.
StartTimeRoom	This is the starting range value of time for the print. This parameter is the starting room in the viewmodes with no time at all.
EndTimeRoom	This is the ending range value of time for the print. This parameter is the starting room in the viewmodes with no time at all.
PrinterParameters	This object will allow for the setting of many properties of the printer. It includes the properties: Copies, Orientation, PaperBin, PaperSize, PrinterDeviceName, and PrintQuality. Use the SchedulePrinters collection to select a valid system printer.

The Start and End parameters depend on the ViewMode property value. If the value is set to display in a view with no dates, these two parameters must be valid room indexes or names, otherwise they must be valid dates. If dates are specified, they must be in the range MinDate to MaxDate or an error occurs. Also, the starting date must be less than or equal to the ending date. When providing room information, the room index or name

must exist in the Rooms collection or an error will occur. The starting room must be less than or equal to the ending room. The StartTime and EndTime parameters must be valid times and they must be in the range of the schedule. The StartTime may be no less than the schedule's property StartTime and the EndTime parameter may be no more than the value of the schedule's StartTime plus DayLength. DayLength is the length of the day in hours. The starting time must be less than or equal to the ending time. The PrinterDeviceName of the PrinterParameters object is the name of the printer to which the schedule is being sent. The Orientation property of the oPrinterParameters object will determine whether the print job will be performed in portrait or landscape mode. See the printer's documentation for an explanation of this concept.

Note:
Some developers have had problems creating the "CPrinterParameter "object in C++. An object of this type is necessary to call the "GoPrint" method. To help these developers, there is a helper method named "GetPrinterParameter". Given a set of parameters, this method will return a reference to an object of this type. You may use the returned object as the last parameter to the "GoPrint" method. The "CPrinterParameter " object is used to specify all of the necessary information to the print routine, like paper size, copies, etc. So if you cannot call the print method because you cannot create this object, just let the schedule create one for you and use it to print the schedule.

PrintPageInfo

When printing, there is also the option of displaying the page information on each page with the PrintPageInfo property. This property determines if the page positions are printed. A schedule may be several pages across and down. This may cause confusion as to how they are to be pieced together. If the PrintPageInfo property is set, the X and Y positions are printed in the top, left corner of each page to facilitate the reconstruction of large schedules. The format of this information is [X, Y], where X is the horizontal position (1..N) and Y is the vertical position (1..M). There will be a total of M * N pages printed.

During the process of printing, some sort of visual cues may be given to the user, as to what is happening. Several events are provided to render this functionality. These events aid in informing the user about the progress of the print. They inform of the percent complete and determine if a print was canceled.

The PrintStart event is raised immediately after the GoPrint method is invoked and error checking completes. An error may occur if the date or room range is invalid. Another common source of errors is that an end value is less than a start value. For example, the start date is June 3 and the end date is June 1. This is logically impossible, so an error occurs.

The PrintDone event is raised, after the all pages have been sent to the printer. This will be the last event of the sequence assuming that no errors occur and the printing is not

canceled. This event is the place to remove any progress screen that may have been displayed. If the PrintDone event is raised, the PrintCancel is not raised.

The PrintCancel event is raised when the print is canceled. All printing ceases and no more pages are sent to the printer. A print may be canceled from the PrintProgress or the PrintPageDone events. If canceled, the PrintDone event is not raised. These two events are mutually exclusive.

The progress of completed pages maybe monitored, with the PrintPageDone event. This event has several parameters. The Page parameter represents the absolute page that was last printed. The PageX and PageY parameters represent the page in horizontal and vertical coordinates. Finally the Cancel parameter allows for the cancellation of printing, so no more pages are sent to the printer.

Quick Tip
There are several "Print…" events that inform you know of the printing progress.

The event that is used to inform the user of the percent complete is the PrintProgress event. This event returns a Percent parameter that is the a value [0..100] of the percent complete. The Cancel parameter allows for the cancellation of printing. If set to true, the PrintCancel event is raised and the printing will not continue.

Table 9.2
Print Events

PrintCancel	This event is raised if the printing was canceled.
PrintDone	This event is raised when the printing completes normally.
PrintPageDone	This event is raised for every page that was sent to the printer.
PrintProgress	This event is raised when at intervals to give feedback on the progress of printing.
PrintStart	This event is raised when the printing begins.

The printing creates temp files, in the window's temp directory. Needed are several hundred kilobytes per page. The amount varies but it is probably a good idea to have one-half megabyte per page free on the hard disk. Computers running low on disk space should not use the printing functionality. This is not a problem in most cases. However, printing a schedule with a small value for ScheduleIncrement and a very large range of days (i.e. a year) could use quite a bit of temp space. All the temp space is released upon the printing completing or being canceled.

To aid in printer selection, there is a provided collection of printers. This collection is named SchedulePrinters. It is a list of all valid printer names that can be used with the DeviceName parameter of the GoPrint method.

Table 9.3
SchedulePrinters Collection Definition

Count	This method returns the total number of printer in the collection. This is the number of printers installed on the current system.

Item	Given a SchedulePrinter object's index or name, this method will return a SchedulePrinter object.

Each SchedulePrinter object has just one property. This is its DeviceName property. This property is used as a parameter to the GoPrint method, to define a valid system printer.

If you wished to print a particular part of a schedule, you could do so from code. You must first setup the PrinterParameter object. Most of the properties are defaulted to the most common values. However for completeness I have included some of the settings here.

```
Dim oPrinterParameters As CPrinterParameter

  Set oPrinterParameters = New CPrinterParameter
  oPrinterParameters.PrinterDeviceName = Printer.DeviceName
  oPrinterParameters.Orientation = 1
  oPrinterParameters.Copies = 1
  Set oPrinterParameters = Nothing
```

Afterwards you may perform the actual printing. In this example, I assume that you wish to print the schedule from August 1 through the 31 of 2002. I also assume that you wish to print from 8 AM to 6 PM. These parameters are valid if the schedule is displaying dates and times. If the schedule is displaying rooms and times then the first two parameters should be valid room names or indexes.

```
Call oChild.Schedule1.GoPrint(#8/1/2002#, #8/31/2002#, #8:00:00
AM#, #6:00:00 PM#, oPrinterParameters)
```

Chapter 10

Displaying Schedules on the Web

There are times that you may need to display your schedule on the web. You might create schedules that effect many people and you need a way to distribute this information without the GbSchedule ActiveX component. You can perform this by using the schedule's web publishing functionality.

Web Schedules Defined

Quick Tip
You may export some schedule types to a web page by using the "ExportHTML" method.

A web schedule is defined as a schedule displayed in a browser that does not need the GbSchedule component. This is very nice for wide distribution of published schedules. The browser does not need to load any ActiveX. This ensures that the schedule may be viewed in any browser. In other words they are browser-independent. The schedule is actually generated into standard HTML. It is nothing more than a big table. This makes it convenient for distribution, with a caveat: all schedules are read-only. Also a schedule cannot be generated if there are conflicts on the schedule. These are the two limitations on web schedule generation at this time. There are also different publishing options. A schedule may be published on one page or into frames. The frame publishing allows for the custom header and footer to be visible at all times. If a large schedule is created on a single page then the header and footer are at the far left of the screen. This guarantees that is visible when first viewed; however if the schedule is scrolled they disappear of course. A web publish is accomplished with a single function call to ExportHTML.

ExportHTML

In addition to sending a schedule to a printer, the schedule may also be sent to an HTML file. There are times when a schedule needs to be published to an HTML page, so that users may view it. A schedule can be published every day (hour, week, month, etc…), so users or customers may view it. Keep in mind that this will be a read-only schedule. There is no way for a user to modify this HTML page.

The actual export is easy to use and allows for the addition of custom HTML to create a complete page, if necessary. The ExportHTML method may be used to perform this process. The method is called with a parameter object of type CHTMLParameters. The properties may be set to build the page as desired.

Table 10.1
GoExport Parameter Object

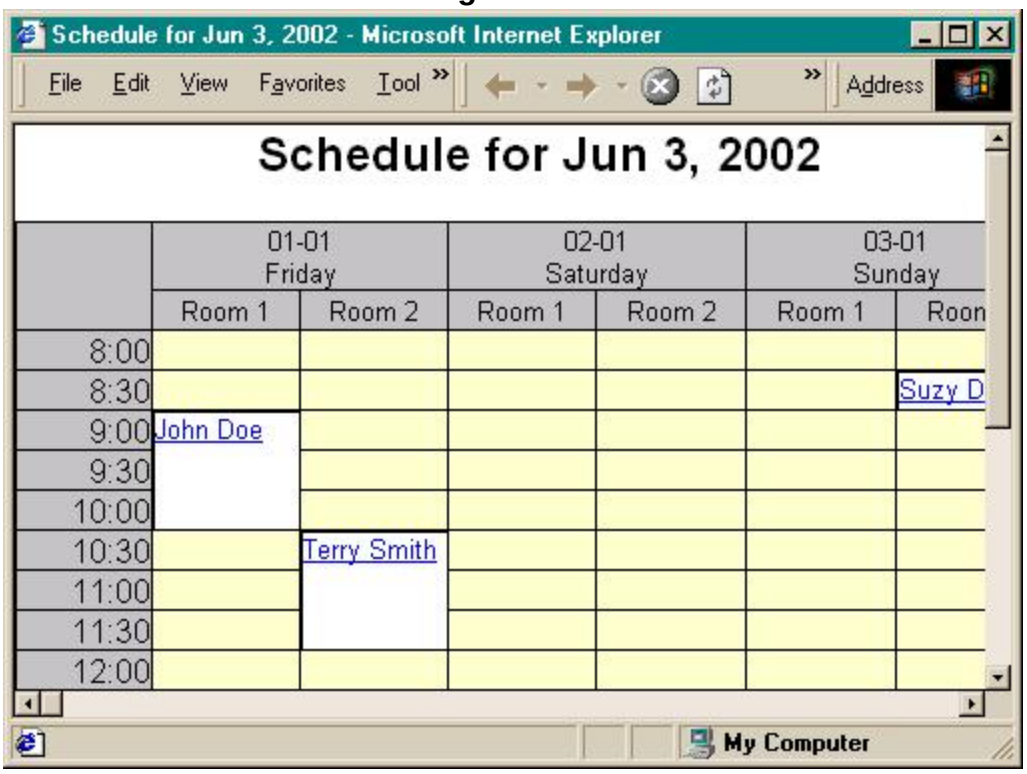
FileName	This is the file to which is being written.
Overwrite	This property determines if this file is overwritten, if it already

	exists.
PageTitle	This is the HTML page name. This is the name the browser will display.
TableOnly	This property determines if only the HTML inside the table tag is written. If this is true, there will be no head tags, page tags, of any other tags in the HTML file. The file will start with “<Table>” and end with “</Table>”.
HTMLHeader	This property is a string value that inserts raw HTML directly above the schedule.
HTMLFooter	This property is a string value that inserts raw HTML directly below the schedule.

These properties are used to build the page to specification. A parameter object was used for future expansion. In the future, the functionality for exporting may be augmented. When this happens the object will add more parameters but the calls the ExportHTML function will not change.

```
Dim oHTMLParameter As New CHTMLParameters


  oHTMLParameter.FileName = "c:\test_html.htm"
  oHTMLParameter.Overwrite = True
  oHTMLParameter.PageTitle = "Schedule for Jun 3, 2002"
  oHTMLParameter.TableOnly = False
  oHTMLParameter.HTMLHeader = ””
  oHTMLParameter.HTMLFooter = ””
  Call Schedule1.ExportHTML(oHTMLParameter)
```

Figure 10.1



Quick Tip
Add custom headers and footer to a schedule webs page by using the "HTMLHeaderText" and "HTMLFooterText" properties on the parameter object of the "ExportHTML" method.

A very nice feature of this method is that extra raw HTML may be inserted into the page, to make it more complete. A schedule is displayed as a table in the created HTML file. An HTML template file may be used to create the schedule, at a particular place inside an HTML file. Squeeze a schedule between two sets of raw HTML code, by using the HTMLHeaderText and HTMLFooterText properties. When an HTML file with a particular layout is required (i.e. Text-Schedule-Text), the HTML above the schedule is placed in the HTMLHeaderText parameter. The additional text, displayed at the bottom of the page, is copied into the HTMLFooterText parameter. Using these two parameters, of the ExportToHTML method, a full HTML page with all necessary tags may be created.

In addition, the TableOnly property will determine if only the table is written or an entire page is written. If this property is false, the HTMLHeaderText and HTMLFooterText properties are used. As an example, an entire page is created with the following format.

```
<html>
<head>
<title>PAGETITLE</title>
</head>
<body bgcolor="#FFFFFF" text="#000000">
<dd align="left"><p align="center"><strong><big><big>
PAGETITLE
```

```
</big></big></strong><br></p>
HTMLHEADERTEXT
<table>
......
</table>
HTMLFOOTERTEXT
</div>
</body>
</html>
```

This HTML sample shows that an entire page is indeed created with page and table tags, along with any other additional, raw HTML above and below the schedule table. If TableOnly is true, only the HTML table representing the schedule is saved. The HTML file would have the following format.

```
<table>
......
</table>
```

As can be seen in this example, this file has no page tags. This format could be used to retrieve the displayed table and insert it into many HTML pages, by copying the table contents only.

Chapter 11

Recurring Appointments

> **Quick Tip**
> A Recurrence is a grouping of two or more appointments.

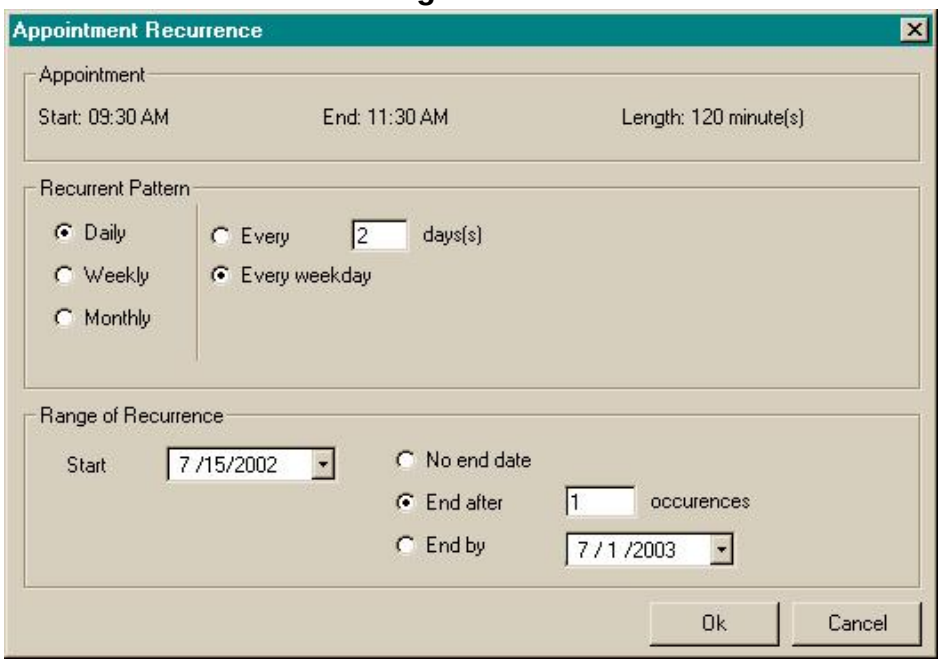
In addition to scheduling single ungrouped appointments, a group of them may also be added. Appointments in a group are related in that they have the same GroupId, but the properties of each may be manipulated separately. When an appointment is created, it is assigned a unique GroupId. To determine the number of appointments in any group, you may use the GroupCount method of the ScheduleItems collection. Given a GroupId, it returns the number of appointments with a matching GroupId. You may programmatically add recurrences with the AddRecurrence method of the ScheduleItems collection. Given an existing appointment and a “Recurrence” object. The proper number of recurring appointments will be added to the ScheduleItems collection. In addition to adding recurrences with code there is a provided UI. The default UI uses the “ScheduleRecurrence” control. You may wish to not use the default dialog and instead use this control to create your own dialog for this functionality.

A recurrence object provides all of the necessary information to build a recurrence pattern. A recurrence pattern may be organized in many different ways. This object has a StartDate that defines the starting date of the recurring pattern. In addition the pattern may end in any of three ways. (1) The recurrence may never end, (2) end after a certain number of occurrences, or (3) end at a specified date. The recurrence interval may be daily, weekly, or monthly. Each recurrence interval has a specified object that stores the parameters for its particular setup.

RecurrenceDay

If the recurrence is daily, the Recurrence object’s “RecurrenceDay” object will contain the needed information to create the recurrence pattern. This object has two properties: DayInterval and RecurrenceMode. The day interval is the number of days to skip before creating another occurrence. For example, if you wish to create a recurring appointment every day then this property would be set to “1”. This indicates that each appointment is one day from the last one. The RecurrenceMode property may be set to one of two property values. The above example used the “DayInterval” setting, where you specify the interval to separate the days. The other property value is “Weekdays”. This will create the recurrence every weekday.

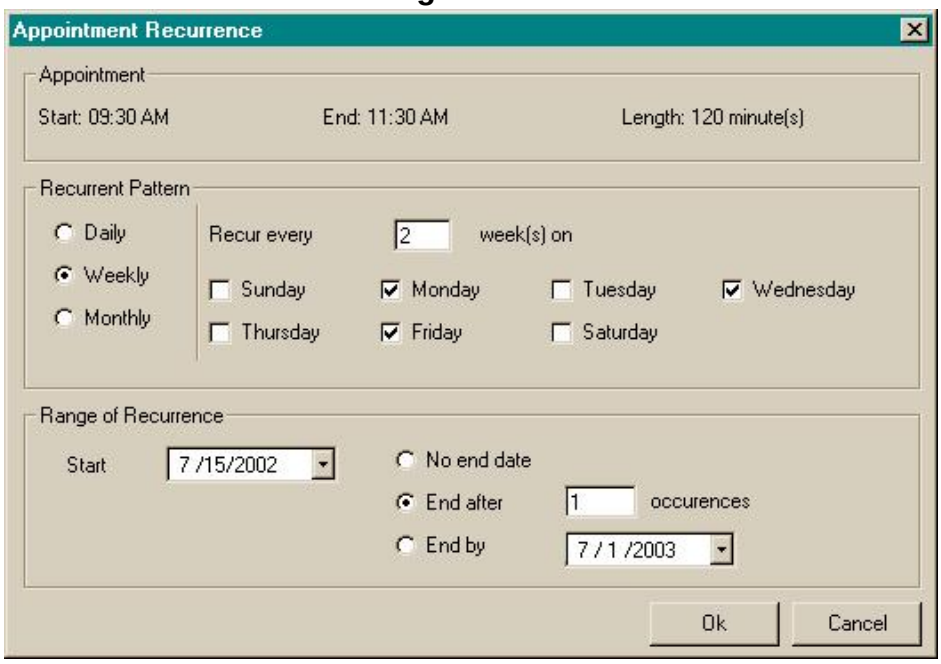
Figure 11.1



RecurrenceWeek

The recurrence interval may also be set to week. The Recurrence object has a “RecurrenceWeek” object that specifies the needed information. There are seven properties of this object that specify which days of the week are to be included. These properties are “UseSun” through “UseSat”; one for each day of the week. The “WeekInterval” property specifies the gap between weeks. To schedule an appointment for every week, set this property to “1”.

Figure 11.2

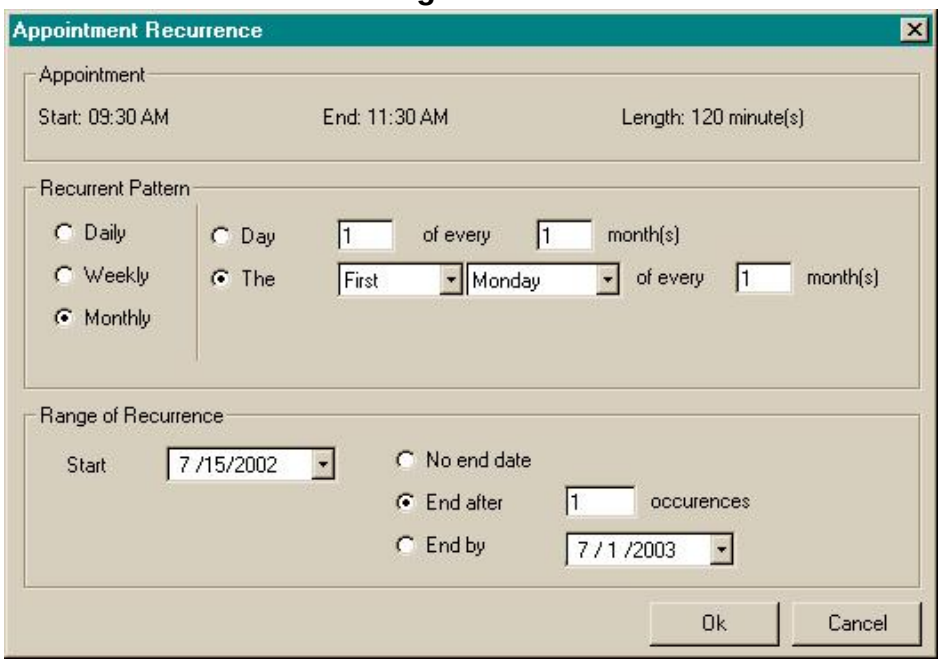


RecurrenceMonth

Perhaps the most complex recurrence pattern is for a month. Each month has a difference number of days and starts on a different day of the week. This can make scheduling quite complex. The Recurrence object uses its “RecurrenceMonth” object to define information to construct this recurrence pattern. The object’s “RecurrenceMode” may be set to “MonthInterval” or “MonthOrdinal”. When it is set to “MonthInterval” the recurrence will happen on the same day number in the month each time. For example, you use this property to define a recurrence on the fourth (4) day of each month. Thus the “DayInterval” property defines which day of the month to use. When RecurrenceMonth object’s RecurrenceMode is set to the “MonthOrdinal” property value, the appointment is scheduled at a certain position in the month though not necessarily on the same day number. For example, you could schedule an appointment on the second Monday of each month. You may not care the day of the appointment (1, 2, etc) just that it is on the second Monday.

The MonthInterval property defines the skip between months. If you wish to schedule an appointment every month, this property would be set to “1”. This allows you to define a pattern that only occurs on the months that you specify.

Figure 11.3



Quick Tip
The appointments in a Recurrence pattern do not have to share any attributes other than the same "GroupId" property if desired.

The Recurrence object itself has properties that constrain the number and range of the specified recurrence pattern. The StartDate defines the starting point of the recurrence pattern. The default is the date of the appointment used as the template for the pattern. The termination of the pattern may be issued in any one of three ways. If there is "No end date" specified, the recurrence pattern will continue until the MaxDate property of the schedule is reached. An end date may be specified so that an appointment may be created on this date but never after this date. The pattern may also end after a certain number of occurrences. You may wish for only five appointments to be made. If this termination method is specified, only that number of appointments or less will be created. If the MaxDate of the schedule is reached before the specified number of appointments is created, there will be fewer appointments created than specified.

```
Call Schedule1.ScheduleItems.Add("", #1/7/2002#, "", #10:00:00
AM#, 60, "Weekly Meeting")
oRecurrence.RecurrenceInterval = ricWeekly
oRecurrence.EndType = recNoEnd
oRecurrence.StartDate = #1/7/2002#
oRecurrence.RecurrenceWeek.UseMon = True
oRecurrence.RecurrenceWeek.WeekInterval = 1
Call Schedule1.ScheduleItems.AddRecurrence(1, oRecurrence)
```

The code above will add an appointment January 7, 2002. This appointment will then be used as a template to create a recurring pattern. All other appointments will use this template appointment's start time and length as their own. The interval will be weekly with no ending point. The recurrence pattern will begin on January 7, 2002 and continue

until the MaxDate of the schedule. All appointments will be created on a Monday. Since the WeekInterval is set to “1”, an appointment will be created every week.

Now that recurring appointments can be created, there is a question, “How can we remove the collection of them?” As always there are two ways to remove appointments: in code or with user interaction. To remove a group of appointments in code, the “RemoveRecurrence” method of the ScheduleItems collection may be used. This method receives two parameters. The first is an existing appointment object, or its key, index, or GroupId. Any one of these may be used to determine the GroupId. The second parameter determines if the specified appointment is removed along with all of its recurrences. If the first parameter is a GroupId the second parameter is ignored. However if the first parameter is an object, index, or key; the second parameter play a significant role. If you wish to remove all occurrences of an appointment but not the original appointment itself then set the second parameter to false; otherwise set it to true.

Another way to remove an appointment’s recurrences is to use the provided GUI. Under normal circumstances when the user highlights an appointment and presses the <Delete> key, he is prompted to remove the selected appointment. If the appointment is part of a group, the prompt changes and asks the user if he wishes to remove the entire series of appointments or the single, selected one. This assumes that the “AllowRecurrences” property is set. If it is not set, the schedule does not recognize groups. Even when there are multiple appointments with the same GroupId the delete prompt will not change since the schedule is configured to ignore groups completely.

Quick Tip
Dump the recurrence information to an XML string or file for later use with the object’s “ExportXML” method.

The Recurrence object’s state may be saved if need be. It contains the methods “ImportXML” and “ExportXML”. The ImportXML method takes an XML, string parameter and loads the object from it. The ExportXML will return the XML state of the object. You may use these methods to load and store the recurrence pattern from and to a string. This may be useful in the certain situations. The ScheduleRecurrence control has a Recurrence object on it. When the control is loaded, its object has the default state. You may wish to load a previous state. An example of the usefulness of these methods is illustrated as follows. Put a ScheduleRecurrence control on a form. When the form is load the user can set each property to some value. Now the user unloads the form and the control along with it of course. If the user loads this form again, he will probably expect his settings from the last time to be restored. This does not happen “auto-magically”. In the form unload event you can store the string from the ScheduleRecurrence control’s ExportXML method in a string somewhere. On the load event of the form, you can use the ImportXML method to restore the previous state of the control that the user set. This is exactly what happens in the background on the ScheduleProperties control. When the user presses the “Recurrences” button on this control, a form is loaded with a ScheduleRecurrence control. Each time the button is pressed the previous state of the recurrence pattern is displayed.

Clusters

There is a way to group appointments without creating a recurrence pattern. When two or more appointments have the same GroupId they are displayed with a small recurrence icon next to each of them. This defines the group. However there are times when you may wish to logically group or cluster appointments without showing any icons relating to the cluster. To provide this functionality, each appointment has a “ClusterId” property that has an initially unique value. You may change this to any value you wish. If you give the same key to more than one appointment, these appointments are said to be in a cluster. The ScheduleItems collection has a “RemoveCluster” method that may be invoked to remove a cluster of appointments at the same time. This eliminates the cumbersome operation of looping through the ScheduleItems collection and removing each individually. Also a Schedule object has a “GetClustered” method that will return a collection of appointment indexes of appointments with the specified ClusterId. This is useful to get a collection of indexes that may be used to access the individual appointment objects in the ScheduleItems collection.

Chapter 12

Advanced Functionality

The schedule also contains some advanced functionality. This functionality allows you to create routines that perform some complex actions. There may be times when you may need to blackout appointments or zoom in on areas of a schedule. Perhaps you need to associate people (Providers) with appointments or relate additional categories. No matter your need, the schedule component probably provides the required functionality with a minimal amount of code.

AllowInterWindowDrop

Quick Tip
Appointments may be dragged and dropped to other schedules if the "AllowInterWindowDrop" property is set to True.

Depending on the properties AllowMove and AllowCopy, the schedule may or may not allow their related actions. If the schedule does allow them, the user may move (or copy) appointments too any valid area of the schedule. There may be times that two or more schedules may be open and information needs to be shared between them. This is possible, if the AllowInterWindowDrop schedule property is true. Two windows or even two separate applications each with a GbSchedule, may share information. Just drag an appointment and move the mouse to the destination window. There is no NoDrop pointer because the move is perfectly valid. When the user drops the appointment, it will disappear from the source window and reappear on the destination window. If this is a copy, the source appointment will not disappear.

Activities and Events

Activities and events are non-traditional appointments. Instead of having a start time and length that keeps an appointment confined to a column or row these objects have differing ways of displaying information. First an event is an appointment with no start time and no length. It takes place on a day or in a room. If you try to access its StartTime property you will get 12:00 AM or the default time. Also its length will be 0. However its IsEvent property will be set and this is what makes it an event. Events are displayed at the top of the screen and are only displayed when days (or and days and rooms) are displayed on the top of the schedule and time is displayed on the left. The appointments mark an event for the day such as a birthday. The event does not happen at a specific time nor does it have a specific length. All that is known is that is occurs on an explicit date.

Quick Tip
To allow the user to create multi-day appointments, set the AllowActivities property to True.

Unlike an event, an activity does have a specific time and length. What differentiates it is that these two properties combined cause the appointment to over lap a day boundary. The appointment takes up 2 or more columns and for this reason is actually displayed at the top of the screen in the event header. You can make a distinction

between the two because an activity has a clock icon on its left and right side, while an event does not. The AllowActivities property must be set to modify appointments to have this condition and also for the display of such appointments. Being even more restrictive in its presentation than an event, an activity will only display if the schedule's Viewmode is set to day on top and time on the left.

Effects

There is a way to display the appearance and disappearance of appointments in a fancy fashion. The AllowEffects property may be used to make appointment shrink and expand. When this property is true and an appointment is added, it will expand from a point to the appointment's specified size. This gives the user the illusion of an appointment being created. Conversely, when the user removes an appointment, it will seem to shrink down into nothing. It will collapse back down into a point and disappear. This is enhanced functionality and may not be appropriate for a your application. If the schedule is running on a slower computer, the repaint may not look very nice. The graphics of the GbSchedule are rather intensive sometimes and not all monitors can handle the extra graphic calls.

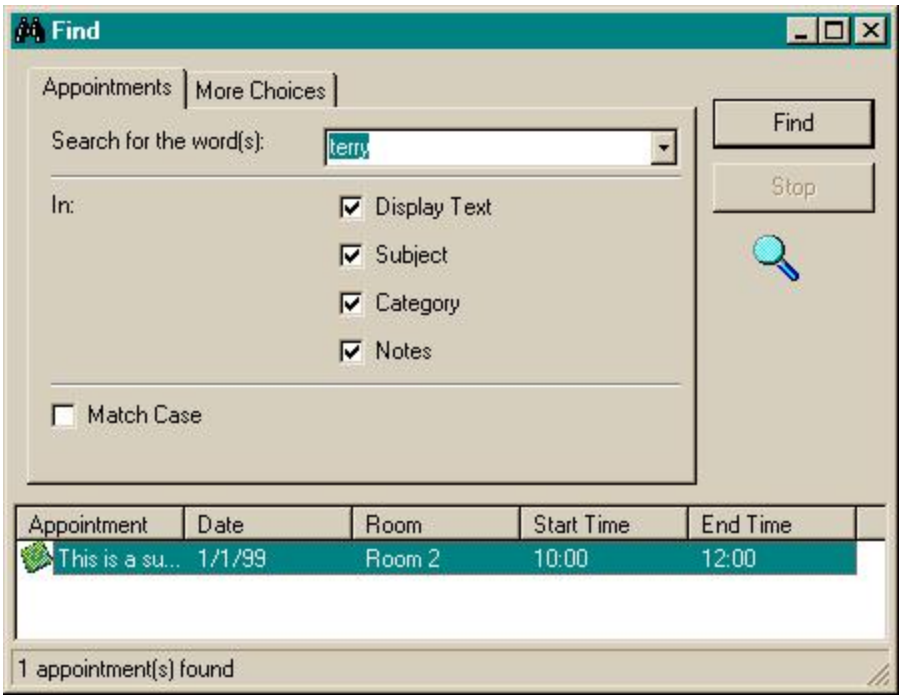
BlackOuts

Quick Tip
Appointments may be blacked-out to reserve space but without revealing any appointment information.

Occasionally an appointment will need to be displayed so that the user should not be able to edit, move, or view. Each ScheduleItem has a BlackOut property. When set to true, this appointment is nothing more than a placeholder. It will be displayed on the schedule as a block of color defined the BlackOutColor property. This color is by default black, but it may be set to any color desired. This functionality allows for the definition of an area on the schedule that is an appointment but the user has no access to modify or view.

Find

The user may also perform a find of appointments if necessary. You may construct your own find appointments dialog window is you wish. However there is one built in. When the AllowFind property is set to true, the user press the <CTRL>-F to display the "Find" window. This window allows you to search for text inside of the Category, Subject, DisplayText, and Notes fields. You may also constrict your search criteria based on date range, start time, or end time. This allows for quite complex search criteria to be created if need be.



DisplayDragTip

This property determines if a small label is displayed, with additional information when needed. This label is displayed when dragging an appointment. It reveals the coordinates of the appointment as it moves. When resizing an appointment, the label is shown to inform the user of the new size of the appointments as it is being resized. In addition, the label is displayed when scrolling the schedule with scroll bars to inform the user of the scrollbar position.

DynamicScroll

The user may click the mouse on a scroll bar and drags it to a new position. After release, the schedule will always have its view port position set accordingly. However, setting this property specifies that the schedule is to be scrolled as the user drags a scroll bar. When this property is false, the schedule will not repaint, until the mouse is released. If true however, each time the user moves the mouse while clicked on the scroll bar, the schedule will repaint itself. This could potentially slow the schedule on slower computers and make it seem very sluggish. This property is false by default, because there is really no advantage to repainting the screen so often, while dragging the scrollbar.

UseUniCode

If need be you may specify that Unicode be used instead of standard ASCII text. Some languages, like Chinese, will not display properly on the schedule. Only half of the characters are displayed. This is because the set of characters that this language uses

consists of 2 bytes per character. The printing code needs to know that this is a double-byte language. When the UseUniCode property is set, all text is assumed to be double byte and is displayed as such. Keep in mind that this only works on Windows NT and Windows 2000 machines. On Windows 95/98 machines the double byte text will still not completely be displayed.

OutsideAreas

Quick Tip
You may mark areas a special color with the OutsideAreas collection.

In addition to the NoDropAreas collection coloring the schedule in a special way, you may also define OutsideAreas. Objects in this collection are defined just as those in the NoDropAreas collection. The difference is that the user may still drag and drop appointment to these areas. Sometimes it may be necessary to define and area as special. It is still valid so the user may move appointments there, but you just wish to mark it a different color for some reason. In this situation, you can use this collection to define these areas. An example may be that you wish to display an entire day 12:00 AM for 24 hours. However you may wish that 12:00 AM to 8:00 AM and 5:00 PM to 12:00 AM be marked another color. This could signify that these times are not normal business hours. You may not care if they schedule appointments there. You just want to make sure that the user knows this is outside normal business hours.

End of day overlap

The DayLength property may not make the day display past 12:00 AM. If the StartTime is 8:00 AM then the maximum DayLength is 16. Since 17 hours would make the schedule display until 1:00 AM the next day. Since days are displayed in columns 1:00 AM the next morning should not be in that day's column since it would really be the next day. Because of the way the schedule handles appointments this is a limitation at this time. If you wish to have a 2-day appointment, you will not be able to display it on the schedule. At this time, the maximum allowed length for an appointment is 24 hours.

End of schedule overlap

Appointments may not overlap the end of a schedule either. Even if the schedule ends well before the 12:00 AM. If the schedule ends at 6:00 PM then an appointment starting at 5:00 PM may not last 2 hours, since there would be no place to display it. When the user drags an appointment to the edges of a schedule, the appointment will stop when its bottom edge (or left edge) touches the edge of the schedule. This ensures that a drag can move all the way to an edge, but not over. If dragging a 2-hour appointment and the schedule end at 6:00 PM the appointment, the user cannot move the appointment any further than 4:00 PM. This is the last valid time that this appointment will fit.

Zoom

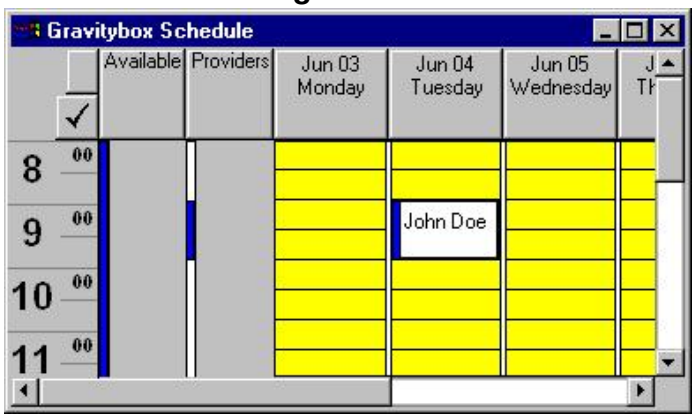
In the course of viewing a schedule, there may come a time when the big picture must be viewed. Many schedules are too big to view all at once and some may be too large to ever view at once. A better perspective may be ascertained, if the user is allowed to shrink the schedule for viewing. GbSchedule provides a Zoom property that allows this. The Zoom may be a number between 25 and 200. This will allow for the shrinking or expanding the schedule to see details that are not readily visible in the regular 100% zoom mode. All the functionality is still available including moving, copying and resizing appointments.

Provider AvailableTimes

We have discussed the basic functionality of the Providers collection. There is however some other behavior not covered. When there exists one or more providers, ScheduleItems may be assigned a Provider that will display as a colored bar on the appointment's left margin. When dealing with people (Providers), it may be advantages to include the times that they are available as well. In a large schedule, there may be Providers that work everyday, some Monday/Wednesday/Friday, some only Tuesdays, etc.

There exists an AvailableTimes collection on each Provider object. This allows for the addition of times that define when a Provider is available. Each one's available bar will display in the left margin of the schedule. This will allow the user to view the available times for a Provider and not schedule an appointment for the Provider, if he is not available.

Figure 12.1



ScheduleItem Categories

Quick Tip
Categories provide a way to color-coding appointments.

There is a Categories collection on the schedule as discussed. It allows for the display of a colored bar on the left margin of each appointment. There is also a Categories collection on the ScheduleItems collection. This allows other user-defined categories to

be displayed, with an appointment. If a schedule has more than one category type, the Categories collection of the schedule will not provide enough functionality. Categories may be added to the ScheduleItems Categories collection. These would be category names. These categories would be assigned to an appointment using the Categories collection of each ScheduleItem.

This allows for as many categories as needed. A color-coded bar will not be displayed on each appointment for these categories. There could potentially be many of them and there is no room on an appointment for an arbitrary number of bars. The bars however will show up on the left margin of the schedule. This is accomplished by using the AllowOtherCategories property. Set this property to true and all of the user-defined categories will be displayed in the left margin of the schedule.

The following code adds two custom categories. The first is a “Shoe” category and it has three items in it. The second is a “Music” category. It also has three items in it.

```
Dim oCategory As CCategoryCol

  'Add a category
  Set oCategory =
Schedule1.ScheduleItems.Categories.Add("Shoes")
  'Add its items
  Call oCategory.Add("Nike", vbBlue)
  Call oCategory.Add("Doc Martin", vbYellow)
  Call oCategory.Add("Addidas", vbRed)

  'Add a category
  Set oCategory =
Schedule1.ScheduleItems.Categories.Add("Music")
  'Add its items
  Call oCategory.Add("Classical", vbBlue)
  Call oCategory.Add("Rock", vbYellow)
  Call oCategory.Add("New Age", vbRed)

  Set oCategory = Nothing
```

This would setup the category information. Now some values may be assigned to the ScheduleItems.

```
'Set the category values for the Appointment 1
Schedule1.ScheduleItems(1).Categories(1).Name = "Nike"
Schedule1.ScheduleItems(1).Categories(2).Name = "New Age"

'Set the category values for the Appointment 2
Schedule1.ScheduleItems(1).Categories(1).Name = "Addidas"
Schedule1.ScheduleItems(1).Categories(2).Name = "Rock"
```

This code will assign category values to the first two appointments (lets assume that there are at least two, Ok). The category 1 is the "Shoes" category and category 2 is the "Music" category. Elements may not be added or removed from an appointment's Categories collection. The number of elements in the ScheduleItems' Categories collection determines the number of elements in the Categories collection, of each ScheduleItem object. As seen here, elements are not added or removed from the Categories collection of each ScheduleItem, but only modified.

AppointmentShape

Quick Tip
If desired, appointments may have different shapes.

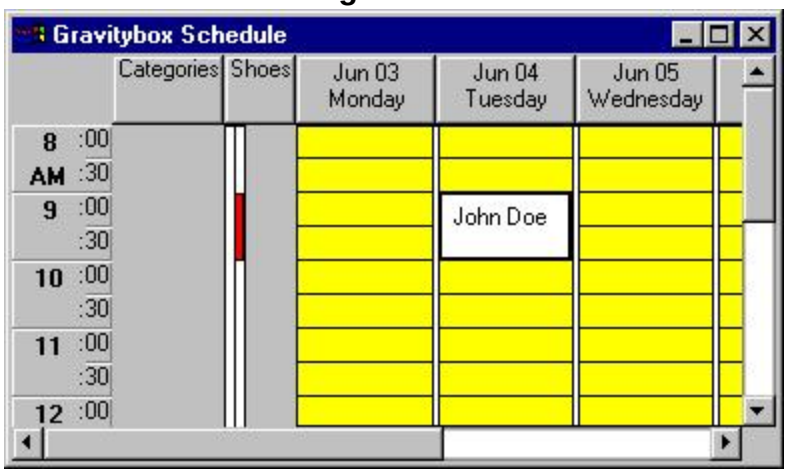
Modifying the appointments' shape may also change the display of a schedule. The default shape for an appointment is square. This will cover the entire area of the appointment. If desired, the schedule's AppointmentShape property may be used to change the appointments' shape. There may be times when you may wish to change the appointment shape, but it is more natural to display the appointments as a square shape. The valid settings for this property are normal, folded, and rounded. The normal is square. The folded settings draws the appointments with the right, bottom edge of the appointment folded up. The rounded setting will draw the all four edges of each appointment rounded.

Redrawing

Quick Tip
When loading many appointments, toggle the "AutoRedraw" property to False. When complete, set it to True. This action will temporarily turn off screen repaints for quicker loading.

When an appointment's properties are changed, the schedule is immediately updated. This only applies to the properties that are displayed of course. The Notes or Id properties would not refresh the screen because they do not affect display, however Length does. This actually may become very time consuming, if many items are set. When loading or clearing a schedule, it is advised to use the schedule property AutoRedraw. Set this property to false before loading or clearing, so that the screen is not refreshed, for every change that is made. The value must be set back to true when done or the schedule will never be redrawn and the user will be very confused as to what is wrong, with the "broke" schedule.

Figure 12.2



Rooms Collection

The Rooms collection was designed to display rooms that might be scheduled in a office. This functionality may be expanded to include other functionality. The Rooms collection defines objects that may be displayed by themselves or that may be displayed under each date. This actually may be used to display anything. Perhaps ”Trucks” need to be scheduled. The trucks may be defined, in the Rooms collection.

```
Schedule1.Rooms.DisplayName = ”Trucks”
Call Schedule1.Rooms.Add(”Truck1”)
…..
Call Schedule1.Rooms.Add(”TruckN”)
```

Now the caption on the schedule will be Trucks, not rooms. Each Room object has a caption of ”Track_i”.

This analogy may by used to expand the collection to represent anything that the developer desires. The word ”Rooms” is not displayed anywhere on the schedule, unless the developer wishes it to be. The Rooms collection’s DisplayName property will allow the ”name” of the collection to be set to anything.

Part V
Other Controls

Not chaos-like together crush'd and bruis'd,
But as the world, harmoniously confus'd,
Where order in variety we see,
And where, though all things differ, all agree.

-Alexander Pope

Chapter 13 Other Controls

Chapter 13

Other Controls

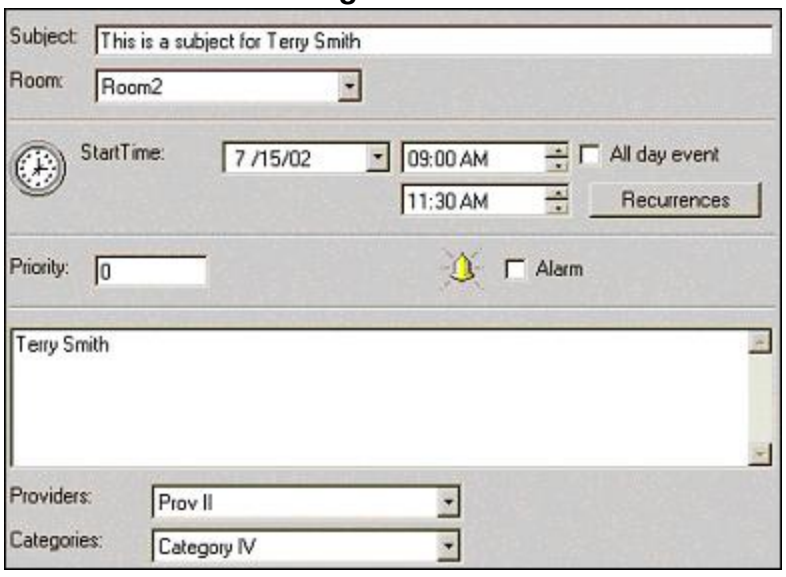
In addition to containing the main schedule control, the GbSchedule ActiveX also contains some other, additional controls. They are schedule related in that you may define headers, contacts, and task lists with them. There is also a properties control that may be used to construct custom property windows for appointments in your programs. All of these complement each other in the creation of robust scheduling applications.

ScheduleProperties Control

Quick Tip
Construct a custom properties screen using the ScheduleProperties control

The ScheduleProperties Control allows for the construction of configurable property windows. In most cases this control may be used to display customized information to the user. In special cases where the application needs to display customized property windows, this control may not be suitable. The control will display the Subject, Room, StartTime, EndTime (StartTime+Length), Priority, Reminder, DisplayText, Provider, and Category. Any of these may be selectively toggled on/off. The control needs to have its Rooms and Categories collections set to the Schedule Rooms and Categories collections. These collections are needed to populate the Rooms and Categories combo boxes on the control. Also the ScheduleItem property must be set. This is the appointment to which all of the control's properties are set. Figure 13.1 shows the control on with all of the available Schedule properties displayed.

Figure 13.1



The ScheduleProperties control has many properties that determine which of the appointment properties are visible. Table 13.1 lists these visibility properties.

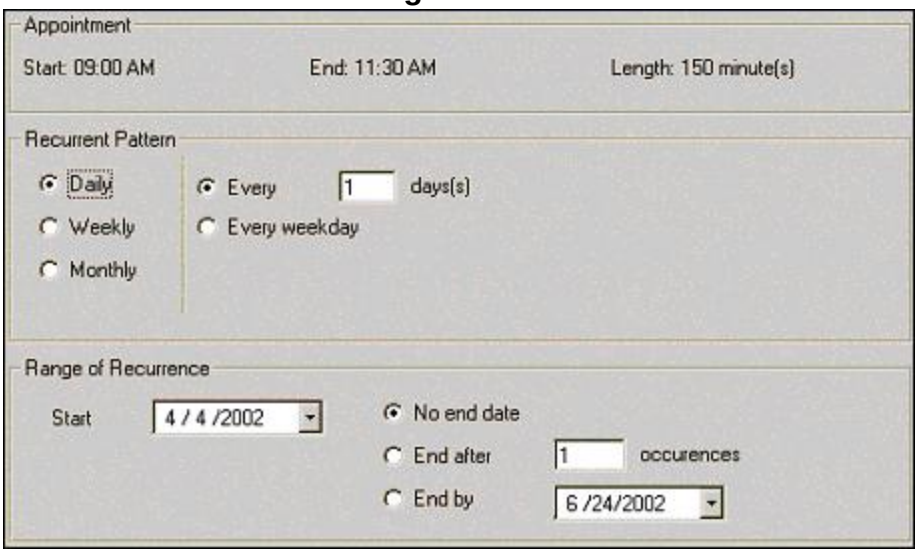
Table 13.1
ScheduleProperties Control Properties

AllowAlarm	Determines visibility of ScheduleItem's Reminder property.
AllowCategory	Determines the visibility of the ScheduleItem's Category property. If the Categories collection is not specified the category combo box is disabled to disallow any user interaction.
AllowDisplayText	Determines visibility of ScheduleItem's DisplayText property.
AllowEventHeader	Determines if the control will allow the user to specify that this appointment is an event. If True a checkbox labeled "All Day Event" will be displayed. The user may use this to make the appointment an event.
AllowPriority	Determines visibility of ScheduleItem's Priority property.
AllowProvider	Determines the visibility of the ScheduleItem's Provider property. If the Providers collection is not specified the provider combo box is disabled to disallow any user interaction.
AllowRecurrences	This property determines if the "Recurrences" button is displayed on the control. When this property is True, the user may press the button and get the recurrence screen. This allows the user to setup recurring appointments.
AllowRoom	Determines the visibility of the ScheduleItem's Room property. If the Rooms collection is not specified the room combo box is disabled to disallow any user interaction.
AllowSubject	Determines visibility of ScheduleItem's Subject property.
AllowTime	Determines visibility of ScheduleItem's StartTime and Length properties. The displayed end time is the appointment's StartTime + Length.
AllowWarning	Determines if the warning message is displayed. You may use this attract the users attention if need be.
ReadOnly	Determines if the user may edit the control.
TimeFormat	Determines the displayed TimeFormat, either 12 or 24 hours format.
WarningMessage	If the AllowWarning property is True, this WarningMessage text will be displayed in the warning label.

ScheduleRecurrence Control

The ScheduleRecurrence control may be used to construct a custom screen for recurring appointments. You may use the object model to add recurring appointments, but this control and the default dialog of the schedule control allows for the addition of this functionality to any application without any code. This functionality has been broken out so that it may be used to build custom screen that may be displayed at some non-standard place in an application.

Figure 13.2



The control has a ”CRecurrence” object as a property. You may use this to update the any item of the GUI. The control also has a ScheduleItem property, which must be set to an existing appointment for the control to display properly. The appointment information at the top of the screen is the information provided by this ScheduleItem. The bottom frame of the control that contains the range information for the recurrence is set by properties of the Recurrence object.

The middle frame sets the actual information about the recurrence. This frame is actually the settings of a collection of three different objects. The Recurrence object has three sub object: RecurrenceDay, RecurrenceWeek, and RecurrenceMonth. The middle frame of the control is the Recurrent Pattern settings. The settings displayed on the right side of the frame determine in which radio button is selected: Daily, Weekly, or Monthly. The properties of the underlying object are displayed when appropriate. For example if the Daily radio button is selected, the right side will display two radio buttons. One will allow the user to set the recurring pattern based on a day interval like every second day. The other radio button will set the appointment to recur every weekday. The week and month settings are displayed when their respective radio buttons are selected.

There are various customizations that may be performed on the control. For example, a warning message may be displayed to the user to indicate some type of condition. The IconWarning may be set to any icon desired and the WarningMessage property may be

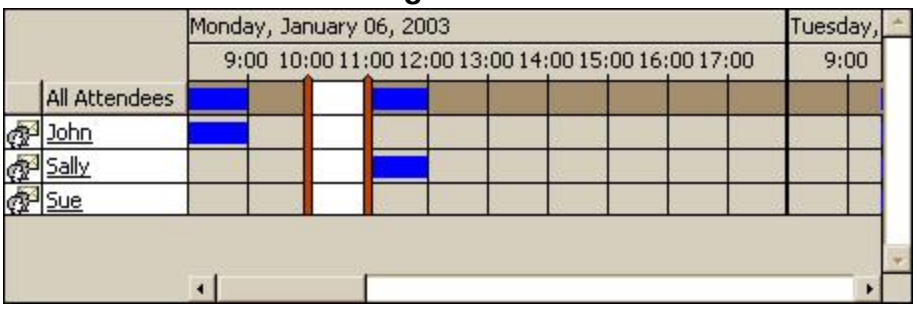
set to some text. When the control is used to display properties for a group of recurrent appointments, you may use the IconRecurrence icon to customize the dialog's look. The control is made language independent by exposing its text properties to the developer. The rooms and category text properties is pulled from their respective collections. However texts like the word "Priority" can be made to display any text using the "PriorityText" property.

ScheduleSummary Control

Quick Tip
View all appointments by provider at a glance using the ScheduleSummary control.

The ScheduleSummary control allows you to build a coordination screen to complement appointment addition. There are times when a user will need to view current appointment information before making additional appointments. An example is creating an appointment in which many people are present. The Schedule control has a Providers collection in which people of an organization may be added. Each appointment has a Provider property that may associate the specified appointment to one of the Providers in the defined Providers collection. If you wish to create an appointment and associate it with only one provider, you may look at the left margin of the schedule to find a time that specified provider is available. However if you wish to coordinate with two or more providers, it may be cumbersome to search for a free time by manually searching each provider's time scale. An easier and more intuitive technique is to display all providers in the left margin and time on the top with non-available times filled with a colored bar to indicate that the specified time is not accessible.

Figure 13.3



In Figure 13.3, a ScheduleSummary control is shown with three people added to the providers collection: John, Sally, and Sue. Each appointment that is as assigned to a provider is displayed on the row that starts with the provider's name. If appointments conflict there is no difference is display. The areas that have assigned appointments are colored. A colored area may have one or more appointments assigned to it, you cannot know by viewing this control. All that is obvious is that is at least one appointment covering the defined area. The color of the bars may be all the same defined by the "DefaultBarColor" property when "UseDefaultBarColor" is set to true. If the "UseDefaultBarColor" property is false, the color defined for the associated Provider object is used to paint its colored bars. The icon to the left of each provider is configurable with the "ResourceIcon" property. There is no way to set an individual icon

for a provider. If no icon is specified this area is blank and painted with the “LeftMarginColor” color.

The top row is a summary. It shows the times taken by appointments regardless of the appointment’s associated provider. The “SummaryBackColor” and “SummaryBarColor” properties define its background and foreground colors respectively. This row is labeled “All Attendees” and the row may not be removed; however the text may be customized with the “AllAttendeeText” property. This makes the control configurable into any language, since this is the only text displayed.

Table 13.2
ScheduleSummary Control Properties

AllAttendeeText	This property defines the text that is displayed as the caption of the top, totals row.
AllowUnassigned	This property defines whether an additional row is appended to the end of the rows after the providers collection has been displayed, which displays all appointments not assigned to a valid provider.
BackColor	The color of the area to the right of the left margin, below the top margin and extending as far down of the last row.
DateFormat	A format string that defines the layout of dates.
DefaultBarColor	Used in conjunction with the UseDefaultBarColor property, this property defines the color used for all time bars not on the totals row.
ForeColor	The color of all text, including the header text and provider text.
LeftMarginColor	The color of the background on which the providers are displayed.
Locked	This property determines if the user may interact with the control. When set to true, the user may not move the select bars to a new position and the scroll bars are disabled.
MajorColumnWidth	Specifies the width in pixels of the time increment defined by the ScheduleIncrement property.
MarginColor	The color that fills any area that is not in the header, left margin, or the provider time grid.
MinorColumnWidth	(Read-only) This is the width of the time slot defined by the SubIncrement property. It is calculated by taken the MajorColumnWidth and dividing it by the number of times that the SubIncrement property will divide into the ScheduleIncrement property. For this reason, the SubIncrement must be a divisor of ScheduleIncrement. For example if the ScheduleIncrement property is 60, the SubIncrement property is 30, and the

	MajorColumnWidth property is 100 pixels, the MinorColumnWidth is 50 pixels since there are 2 sub-increments for each major increment.
ResourceIcon	An icon that is displayed next to each provider name in the left margin.
ScheduleIncrement	The increments of time that are displayed in the header. For example, if this property is set to 30 minutes, the times “9:00”, “9:30”, “10:00”, etc are displayed in the header.
ScheduleObject	A reference a Schedule control. This property must be set for the ScheduleSummary control to display properly. The Providers collection, as well as all associated appointments are taken from this control.
SelectBackColor	This is the background color of the area between the left and right select bars.
SelectBarColor	The color of the vertical select bars.
SelectBarDate	The date where the left select bar resides.
SelectBarLength	The length of time defined as the difference between the left and right select bar position.
SelectBarTime	The time where the left select bar resides.
SubIncrement	This is the smallest time slice by which the select bars may be moved (in minutes). This value must be less than or equal to the ScheduleIncrement.
SummaryBackColor	This is the background color of the of the totals row.
SummaryBarColor	This is the bar color used for all bars in the totals row.
TimeFormat	A format string that defines the layout of times.
UseDefaultBarColor	Determines if the DefaultBarColor is used to paint the provider time bars or if the provider’s defined color is used to draw the bar.

A ScheduleSummary control also may display information about unassigned appointments. If an appointment has not been assigned to a provider, it cannot be displayed in a provider row. If you wish to display these appointments despite not being assigned to a provider you may set the “AllowUnassigned” property to true and an extra row will be displayed with the text “<Unassigned>”. Any appointment not assigned to a provider will be displayed in this row and the appointment’s position will also be present in the top, totals row.

Quick Tip
The SubIncrement property value must be a divisor of the ScheduleIncrement property value.

The parent Schedule control dictates much of the display. A ScheduleSummary control cannot be shown with a Schedule control to which to attach. It determines the format of the dates and times that are displayed. It also defines the start time, day length, min date, and max date. The parent Schedule control defines the Providers collection as

well as the ScheduleItems collection, the appointments. The resolution of the select bar scrolling is defined by the “SubIncrement” property. This must be less than or equal to the ScheduleIncrement property. This is the smallest increment by which the select bars will be moved when dragging. For example, if this value is set to 30, the user would find that the defined area is rounded to 30 minute increments when dragging. As the user moves the left select bar to the right, the SelectBarTime would move from “9:00” to “9:30”, “10:00”, etc. The “ScheduleIncrement” property defines the time resolution displayed on the top margin. If this value is set to 60 minutes then only hours will be displayed in the top margin (“9:00”, “10:00”, etc) but with the SubIncrement set to 30 the user may define an area of one hour from “9:30” and “10:30” with the select bars though these times are not displayed on the top margin.

The most useful functionality of the control is to define an area to for appointment creation. This is performed interactively with the user moving the select bars to define a desired area, effectively to define a time block. The select bars identify a time slot that is delineated by the “SelectBarDate”, “SelectBarTime”, and “SelectBarLength” properties. These properties may be used by an application to create a default appointment on the schedule control. This is not done automatically. You as the programmer have the information needed to create an appointment and you must add the code to do so. Since are as many add scenarios as there are software developers the ScheduleSummary offers no predefined way to create an appointment. You may utilize the user-defined information to generate an appointment, creation routine.

Quick Tip
Automatically search for an available appointment slot with the “SearchNextFreeSlot” method.

When the user moves the mouse over either of the select bars, the mouse pointer will turn to a vertical scroll icon. This informs the user that he may grab a bar to resize the defined area. As the user scrolls the select bars left or right, changing the defined time area, the “SelectionChanged” event is raised to inform the calling a container of this state change. You may use this information to update screen information if necessary or not use it at all if your application has no use of it. Instead of the user scrolling to define an appointment area, you may provide functionality to the user that searches for the next available time slot. This means that there are no appointments defined for any displayed provider. You may use this functionality with the “SearchNextFreeSlot” method. When called this method searches for the next available slot taking into consideration all appointments on the schedule. If one is found, the selection bars are updated, the view is scrolled to bring the defined area in viewing range, and the method returns true. If an available area is not found before the end of the schedule (the maximum defined date) then the control’s display does not change and the method returns false. A limitation with using this control is that an increment that spans multiple days will not be found using the “SearchNextFreeSlot” method. You may define a multi-day increment using the select bars but this method will not find a “next” slot using the current select bar information.

You may also capture clicks that occur when the user clicks on a provider’s name with which you may wish to perform some action. When the user clicks on a provider’s name, the “ProviderClick” event is raised. The index of the provider in the Providers

collection is returned as a parameter to the event. If the index is zero, the “Unassigned” row was clicked, which is not a real provider. Other parameters to the event are “Button” and “Shift”, which allow you to know the mouse button clicked and whether the <ALT>, <CTRL>, or <SHIFT> key was pressed at the time of the click.

TaskList Control

Quick Tip
Add MS-Outlook type tasks to an application with the Tasks control

The TaskList control may be used to list a number of tasks for the user. The columns are configurable. Any number of columns may be added with any of the following types: Date, Text, Time, and NoEdit. These are self-explanatory except for the NoEdit. This setting will display the text but will not allow the user to modify its contents. The top portion of the control may be used to add new tasks to the list. This may be toggled on/off depending on the application’s specific needs. Each Task may be checked or unchecked in the second column. The checked tasks have their PercentComplete property set to 100%. When unchecked, this property may be any whole number from 0 to 99. This allows users to assign completion percents to each task if desired.

Figure 13.4

Quick Tip
You must add columns first before you can add task items.

The TaskList control starts with no columns. Any needed columns must be added manually. This allows the control to be totally configured by the developer’s desires. The following code will clear any columns that are already present and add four new columns. Each column has its own data type.

```
Call TaskList1.Columns.Clear
Call TaskList1.Columns.Add("Subject", ctText, , True)
Call TaskList1.Columns.Add("Date", ctDate, , True)
Call TaskList1.Columns.Add("Time", ctTime, , True)
Call TaskList1.Columns.Add("Length", ctText, , True)
```

When columns as added or removed, each Task of the TaskItems collection is modified to have the same number of elements as the Columns collection. This ensures that each Task will have the same number of TaskItems as columns are present. The TaskItems collection is used to set the value that is to be displayed in the appropriate column of the control. The code below adds a new Task. Since there exists 4 columns on the control now, there are exactly 4 TaskItems for each Task object. So we set the appropriate values for each TaskItem.

```
Set oTask = TaskList1.Tasks.Add
oTask.TaskItems(1).Text = "My Subject"
oTask.TaskItems(2).Text = "12/31/2002"
oTask.TaskItems(3).Text = "11:00 AM"
oTask.TaskItems(4).Text = "60"
Set oTask = Nothing
```

As you can see, we did not add any TaskItems, we just edited the existing ones. In fact you cannot add any of these objects since the collection has not add method. The control is also drag-drop enabled and may accept appointments from the Schedule control. This process if not automatic. When a ScheduleItem is dropped on the TaskList, the TaskList's "DragDropScheduleItem" event is raised. Code may be placed here that will add the dropped item to the TaskList. This is necessary because the TaskList does not have any predefined columns. The developer adds all columns and there is no guarantee that there will be a Date, Time, or Room column present when the appointment is dropped. In addition the dropped appointment does not know which of its properties map to which column. For instance in Germany, the ScheduleItem's "Room" property will not map to a Room column since the developer will not use the English word "Room". For these reasons the developer must add code to map the appointment's properties to the appropriate columns.

```
Private Sub TaskList1_DragDropScheduleItem(ByVal ScheduleItem
As Scheduler.CScheduleEl)

Dim oTask As CTaskEl

  Set oTask = TaskList1.Tasks.Add
  oTask.TaskItems(1).Text = ScheduleItem.Subject
  oTask.TaskItems(2).Text = ScheduleItem.StartDate
  oTask.TaskItems(3).Text = ScheduleItem.StartTime
  oTask.TaskItems(4).Text = ScheduleItem.Length
  Set oTask = Nothing

End Sub
```

This code is executed when a ScheduleItem is dropped on a TaskList. This TaskList has four columns that we added earlier. Therefore each Task will already have 4 TaskItems present. The code maps column 1 to the Subject property, column 2 to StartDate, column 3 to StartTime, and column 4 to the Length property. After this event, the control will append a Task with the appointment's information.

Table 13.3
TaskList Control Properties

AddText	The text that is displayed in the add portion of the control when it does not have focus. This should be an instruction on what is the defined area.

AllowAdd	This property determines if the add portion of the control is displayed. When false, the user may not add text interactively from the control.
AllowBubbleTips	Determines if the tool tips that are displayed are bubbles as the new Win2000 tool tips are or if there are standard Win95 tool tips.
AllowColumnResizing	Determines if the user may grab the edge of a column and resize it width.
AllowContactDrops	Determines if a Contact object from a Contacts control may be dropped on the TaskList. When true and an object if dropped, it will raise the “DragDropContact” event as notification.
AllowCopy	Determines of a contact may be copied by pressing the <CTRL> key and dragging a Task object to another position.
AllowDelete	Determines of the user may press the <DELETE> key to remove a Task from the list.
AllowDragFromFile	Determines if a Task object file may be and dropped on the TaskList to create a new Task object.
AllowDragToFile	Determines if a Task object may be dragged from the TaskList and dropped on a container that supports file drops to create a Task object file.
AllowInterWindowDrop	Determines if a Task objects may be dragged from one TaskList to another TaskList control. This applies to copy or moves.
AllowMove	Determines if a Task object may be moved. Since tasks may not be reordered, this only applies to moving it from one TaskList to another.
AllowOtherDrops	Determines if object from other sources outside of the Schedule control (appointments, contacts, or tasks) may be dropped on the TaskList to raise the “DragDropOtherObject” event.
AllowScheduleDrops	Determines if appointment objects may be dropped on the TaskList.
BackColor	Determines the background color of the TaskList.
BorderColor	Determines the color of the outside border that surrounds the control.
Columns	This is an object collection where the column names are displayed. This collection must have at least one object in it for any kind of user interaction to occur I the TaskList since without any columns nothing may be displayed.
DateFormat	This is the format of dates displayed in columns of type date. Each column has an associated type.

ForeColor	Determines the color of all text.
GridColorEmpty	Determines the color of the grid lines that do not have tasks displayed in them.
GridColorFull	Determines the color of the grid lines that have tasks displayed in them.
RightToLeft	Determines if the text is to be displayed right-to-left in certain, supported languages.
RowHeight	(Read-Only) The height of each row in pixels.
SelectedItem	The Task object that is currently selected. If the add portion of the screen has focus then this property is set to nothing, since no Task is selected.
Tasks	This is the collection of Task objects that are displayed in the TaskList.
TimeFormat	Determines the hour-format of displayed time in columns marked as type time. The possible values are 12-hour and 24-hour.
TimeFormatString	Determines the time format of text displayed in columns marked as type time. This is a full format string including hour, minute, and seconds.
ToolTipTextBackColor	Determines the background color of the bubble tips.
ToolTipTextForeColor	Determines the text color of the bubble tips.
TopRow	This is the top row displayed in the view port.
UseUnicode	Determines if a 2-byte language is supported.
VisibleRows	(Read-Only) This is the number of rows that are visible with the current height of the TaskList.

Much like the Schedule control, the TaskList may also save its state to a file. The ExportXML and ImportXML properties may be used to store the column configuration and the associated tasks to and from a file. The functions take an object of type CXMLTaskParameters. This object stores the necessary settings to save and load the file. The following routines are examples of saving and load a TaskList file.

```
Private Sub SaveFile()

Dim oXMLParameters As New CXMLTaskParameters

  oXMLParameters.FileName = "c:\tasks.xml"
  oXMLParameters.Overwrite = True
  oXMLParameters.SaveDefaults = False
  oXMLParameters.UseColumns = True
  Call TaskList1.ExportXML(oXMLParameters)
  Call MsgBox("The file was saved to '" & _
          oXMLParameters.FileName & "'", vbInformation)
```

```
  Set oXMLParameters = Nothing

End Sub

Private Sub LoadFile()

Dim oXMLParameters As New CXMLTaskParameters

  Call TaskList1.Columns.Clear
  Call TaskList1.Tasks.Clear

  oXMLParameters.UseColumns = True
  oXMLParameters.FileName = "c:\tasks.xml"
  Call TaskList1.ImportXML(oXMLParameters)
  Set oXMLParameters = Nothing
  Call TaskList1.Refresh

End Sub
```

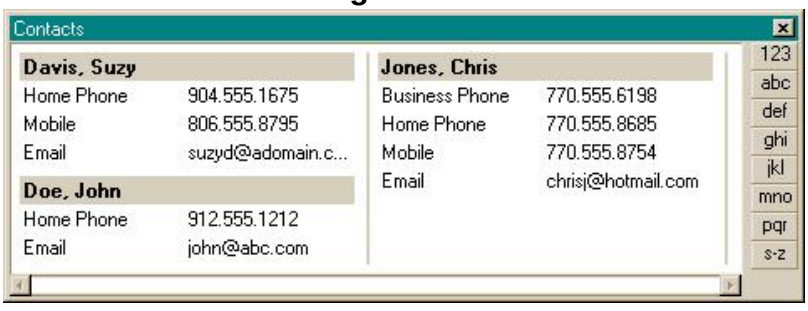
Other functionality includes the controlling of each task item by using the ForeColor property of each task object. In addition you may store any number of extra information using a task object's "ExtraProperties" collection. This is a collection of name / value pairs that be used to store anything. You may address the collection by name or index to retrieve its associated value setting. You may not need this functionality but if you have a large amount of data to store with a task this collection comes in quite handy.

Contacts Control

Quick Tip
Add MS-Outlook type contacts to an application with the Contacts control

If ever you have needed a control to save people, this is the one for you. The control is modeled after the contacts module of MS-Outlook. It allows you to store many attributes associated with a person. Some properties that a contact has are name, home phone, mobile, and business phone. Many of there are displayed on the contact control is populated with data.

Figure 13.5



The component has a Contacts collection. Each object in the collection has many properties that may be set. When one of these properties is populated with data, it is displayed on the screen. In Figure 13.5, the three contacts have 3, 2, and 4 pieces of information populated. The properties that are not set are not displayed on the screen. All colors are configurable. There is at most one selected item on the screen. It has its own color scheme defined. Every other contact uses the inactive color scheme. There is also a focus rectangle drawn around the active contact, if one exists.

Table 13.4
Contact Control Properties

AllowAdd	Determines if the user may double-click on the background to create a new contact object.
AllowBrowse	Determines if the user may browse through the collection of contacts from the default properties screen. When true, two buttons, up and down, are shown on this screen.
AllowColumnResizing	Determines if the user may mouse place the mouse pointer between Contact objects and vertically resize the standard width of all of the objects.
AllowCopy	Determines if the user may hold the <CTRL> key while dragging a contact to create an exact copy of the contact.
AllowDelete	Determines if the user may press the <DELETE> key remove a contact.
AllowDragFromFile	Determines if a Contact object file may be and dropped on the Contacts control to create a new Contact object.
AllowDragToFile	Determines if a Contact object may be dragged from the Contacts control and dropped on a container that supports file drops to create a Task object file.
AllowEdit	Determines if the default property screen is displayed when the user double-clicks a contact or presses the <ENTER> key whilela contact is selected. When true, either action will display a pre-defined property window.
AllowInterWindowDrop	Determines of Contact object may be moved/copied to/from other Contact controls.
AllowOtherDrops	Determines if object from other sources outside of the Schedule control (appointments, contacts, or tasks) may be dropped on the Contact control to raise the “DragDropOtherObject” event.

AllowScheduleDrops	Determines if appointment objects may be dropped on the Contact control.
AllowTabs	Determines if the select tabs are displayed in the right margin. These buttons may be used as a shortcut for users to move to the first contact that starts with a particular letter.
AllowTaskDrops	Determines if Tasks objects may be dropped on the Contact control.
BackColor	Determines the color of the background area outside of the displayed Contact objects.
Categories	This is a collection of categories with which Contact object may be associated.
ColumnWidth	Determines if the width of the each Contact object.
Contacts	This is the collection that holds the actual list of Contact objects.
DefaultDialogFloater	Determines if the default property window is an “on-top” window. When true the window will float above all other windows in the host application. This property only applies when the “DefaultDialogModal” property is true.
DefaultDialogModal	Determines if the default dialog is displayed modally. When true, the default property window must be closed before any other interaction is possible with the host application.
DefaultPhoneMask	Determines the mask to use when displaying phone numbers of Contact objects. This ensures that any phone number mask in any country may be specified to customize the control for local use.
ItemBackColor	Determines the background color of each Contact object.
ItemForeColor	Determines the text color of each Contact object.
ItemHeaderActiveBackColor	Determines the color of the background of the header for the Contact object with the focus only.
ItemHeaderActiveForeColor	Determines the color of the header text for the Contact object with the focus only.
ItemHeaderInactiveBackColor	Determines the color of the background of the header for all Contact objects that do not have the focus.
ItemHeaderInactiveForeColor	Determines the color of the header text for

	the all Contact objects without the focus.
LineColor	Determines the color of the container lines drawn around each Contact object.
RightToLeft	Determines if the text is to be displayed right-to-left in certain, supported languages.
SelectedItem	Determines the Contact object that has the focus. If no object has the focus, this property is set to “Nothing”.
TabAllText	Sets/Returns the text displayed on the selection button, if and only if there is one select button displayed. If the control’s height permits many select buttons are displayed in the right margin. The number of buttons displayed is a function of the control’s height. If the height is small enough only one button is displayed. This property determines the text of this button.
TabAlphaText	This property is the set of characters to be displayed on the select button I th right margin. By default, this value is the 26 characters of the English language. However if need be this text may be changed to coordinate with some other language.
TabNumericText	After the alpha characters are displayed on the select button there is an extra button displayed to indicate contacts that start with a number. By default this value is “123”, but may be customized to display any text desired.

The AllowEdit property gives you the power to control user edits. When true, the user may click on some piece of information to edit it. The contact will become selected. The data item inside of the contact will have a focus rectangle as well. An editable text box will appear and you may enter the data. Clicking elsewhere on the screen or pressing the <Enter> key will save the data. To cancel the edit simply press the <ESC> key and no saving will occur.

```
Dim oContact As CContactEl

  Set oContact = Contacts1.Contacts.Add("John", "", "Doe")
  oContact.Email = "john@abc.com"
  oContact.PhoneHome = "770.555.1212"
  Set oContact = Nothing
```

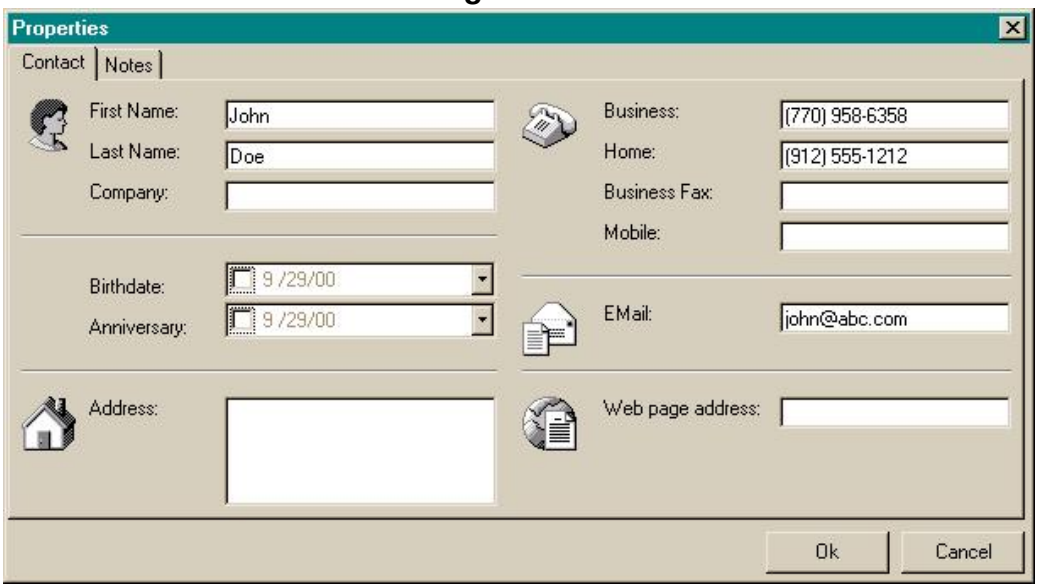
The tabs on the right side of the control may be toggled on/off at your preference. By default these are shown and allow the user an easy way to move to a desired position. When the user presses a button, the first contact that starts with the button's starting letter is shown and selected. For example, if there is a tab with the caption "pqr", the first contact that starts with "P" will be selected. If there is no contact whose name starts with the letter "P", the first contact with a name greater than "P" will be selected. If the next one in sequence is "Herman Zimmerman", he will be selected when though the "pqr" button was pressed.

The user may press the <F2> button to start an edit, if there is a selected item. The events BeforeEditText and AfterEditText are executed in sequence, during this operation. The former allows the user edit to be canceled if desired. The latter event notifies you that the edit was successfully completed.

Quick Tip
You may cancel the default properties screen by setting the "Cancel" parameter to true in the "BeforeEdit" event.

The user may display a property window by double-clicking on the header of a Contact. Keep in mind that the AllowEdit property must be set for this functionality to be present. The property window displays some of the important properties of a Contact. This screen is not configurable and if it does not meet your needs, you will need to construct your own property screen. The phone numbers are auto-formatted to the North American format.

Figure 13.6



Quick Tip
You may define a custom phone mask with the "DefaultPhoneMask" property.

This control also allows for file saving and loading. Just like to TaskList and Schedule controls, it too has "ImportXML" and "ExportXML" methods. They may be used instead of a database to load and save the information in the contacts collection. When saving only those properties with data will be saved. This saves space in the file. Since it would be loaded empty string if saved empty string, there is no need to waste space with placeholders in the XML file.

You may also drag Contacts to the desktop or explorer. When the AllowDragToFile property is set to true, the user may click on the header of a Contact and drag it outside of the control and drop it. A file will be created on the destination that contains the Contact's information. If the AllowDragFromFile property is set, the user may drag a previously saved file back onto the Contact control and drop it to add the Contact to the collection.

All of this combines to create a very useful Contacts control. It is Windows standard adhering to the Microsoft look-and-feel, so you users will already be familiar with it. The collection of contacts is easy to navigate and lends itself to easy file and database manipulation. As with the task object above each contact object has an ExtraProperties collection as well to store additional information about each contact.

Header Control

Quick Tip
You may add a professional header to any screen by using the Header control

This control may be used to display an "MS-Outlook" header on a screen. This will give the MS Outlook look-and-feel. The control has an associated icon that may be displayed large or small. Also it may also be displayed on the left or right side of the header. The displayed text is on the opposite side of the control from the icon. The backcolor and forecolor are configurable as well. This allows for the developer to have total control over the display. The control is read-only with no user interaction. It is for display only.

Figure 13.7

This control adheres to the Windows look-and-feel standards to make users feel more comfortable. Remember, the more standard an application looks, the less support calls you will get for ignorance questions. The main reason for using standard looking controls is to prevent costly user support and training issues. If a user is already familiar with a software paradigm, let him keep using it so that you will not have to shell out money for technical support, which more people are expecting for free these days!

Table 13.5
Header Control Properties

AutoSize	This property determines if the control's height is automatically sized according to the height of the caption.
BackColor	The color of the background of the control. This is only seen if the "HasSpacer" property value is true. This is the color outside of the actual header.
BorderStyle	Determines if there is a border drawn around

	the control.
Caption	The text to be displayed on the control.
Font	The font to be used to display the text.
HasSpacer	Determines if there is a small space around the control color by the BackColor.
HeaderBackColor	The color of the background of the area where the text is displayed. This is the entire control if the “HasSpacer” property value is false.
HeaderForeColor	The color of the caption text.
Icon	The icon to be displayed on the control.
IconAlignment	The alignment position of the icon: left or right.
IconSize	The size of the displayed icon. This is 16x16 or 32x32 pixels.
MouseIcon	The icon to use for the mouse pointer.
MousePointer	An integer value that specifies the mouse pointer type.
ToolTipText	The text to be displayed as a popup window when the mouse hovers over the control.

TitleBar Control

Quick Tip
Define a custom looking screen with the TitleBar control

The TitleBar control may be used to create custom looking frames and forms. It is a control container that may be placed on a screen to group one or more controls. It may display a colored border if desired. Also a TitleBar may display a button in the right corner that allows user interaction. Most of the colors are user defined. You may customize the color of the border and background and well as the text.

Figure 13.8

The most common use of this component is to contain one or more child controls. Placed on a form, this grouping gives a customized look and feel. When the button is pressed you may hide the TitleBar and resize other controls on the your form to compensate for the removal of the specified TitleBar’s disappearance. This allows you to give addition functionality to the user to customize a form by hiding optional parts. If

you do not wish for the user to interact with the TitleBar, set its “AllowCloseButton” property to false and no button will be displayed.

Table 13.6
TitleBar Control Properties

Align	Determines if the control is aligned to one of the four sides of the screen.
AllowALTF4	Determines if the <ALT>-<F4> key combination will cause the “ButtonClick” event to be raised. This is useful on a form with no caption or control box and the “AllowAsCaption” property value is true. This enables the control to act as the real title bar of a window.
AllowAsCaption	Determines if the control is to replace the default TitleBar of a window. A screen can be setup to have no caption or control box. If so, the TItleBar may assume this functionality by setting this property value to true. The user may drag the TitleBar to move the window as if it were the real window title.
AllowBody	Determines if the control is a header only or has a body on which to position child controls. When true, the TitleBar will be vertically resizable and may accept constituent controls. When false the control will have a fixed height, but will be resizable horizontally.
AllowCloseButton	Determines if the close button should be displayed in the top right margin. When pressed it raises the “ButtonClick” event.
Appearance	This property determines if the control has a 3D border. When this 'property value is true, the control has a 3D shadowed border. Otherwise it 'has a single line border of the color defined by the ''BorderColor'' property.
BackColor	Determines the background color of the non-header area. The header color is not determined by this property value.
BorderColor	Determines the color of the border drawn around the control. If no border is desired then set this property to the same color as the background color.
Caption	This is the text to be displayed on the header of the control.
ClientHeight	This is the height of the body area where child controls may be placed.
ClientLeft	This is the left edge of the body area where child controls may be placed.
ClientTop	This is the top edge of the body area where child controls may be placed.
ClientWidth	This is the width of the body area where child controls

	may be placed.
Enabled	Determines if the control may accept user interaction.
Font	Determines the font used to display the caption.
ForeColor	Determines the text color of the caption.
HeaderBackColor	Determines the color of the background of the header portion of the control only.

When the control is resized you may wish to resize its contained controls in the “Resize” event. This task is made easier by the four properties “ClientLeft”, “ClientTop”, “ClientWidth”, and “ClientHeight”. These properties return the coordinates of the client area where constituent controls are to be displayed.

Quick Tip
A user may move a screen with a TitleBar by dragging its caption, if the “AllowAsCaption” property is true.

A TitleBar may be used as a screen as well, while its header will double as the screen’s caption if necessary. When placed on a form with no controlbox and no caption the TitleBar’s header can be made to accept mouse clicks and move the form as a normal form caption would. When its property “AllowAsCaption” is set to true, the user may click in the header portion and the entire form will move as if the form’s title bar was clicked and dragged. When using the control in this fashion you may catch the “ALT-F4” key sequence as well to close a form. If the “AllowALTF4” property is set to true, this key combination will raise the “ButtonClick” event just as if the button was pressed. This allows your form to act just as a real window would. There is a piece of code that must be added for this functionality to perform properly. When the user clicks the header to move the form the event “GetContainerhWnd” is raised. The parameter “lHwnd” must be set to the hWnd property of the window that is to be moved. If this parameter in this event is not set, the window will not move as expected.

Notes Control

The Notes control allows you add the ability to store arbitrary text strings in your application. This control may display in three distinct ways so that you may customize the look and feel of your application. The note objects are just text that is displayed with a note icon. This is modeled after the Notes section of MS-Outlook.

Figure 13.9

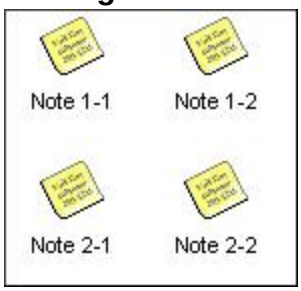


Table 13.7
Notes Control Properties

AllowCopy	Determines if the user may copy objects by holding the <CTRL> button when dragging.
AllowDelete	Determines if the user may remove objects by pressing the <DELETE> key.
AllowDragFromFile	Determines if a Note object file may be and dropped on the Notes control to create a new Note object.
AllowDragToFile	Determines if a Note object may be dragged from the Notes control and dropped on a container that supports file drops to create a Note object file.
AllowEdit	Determines if the user may edit an item by double-clicking it or pressing the <Enter> key.
AllowInterWindowDrop	Determines if Note objects may be dragged to another Note window.
AllowOtherDrops	Determines if non-Gravitybox objects may be dropped on the Note control.
BackColor	Determines the back color of the control when enabled.
DefaultDialogFloater	Determines if the default property window is an “on-top” window. When true the window will float above all other windows in the host application. This property only applies when the “DefaultDialogModal” property is true.
DefaultDialogModal	Determines if the default dialog is displayed modally. When true, the default property window must be closed before any other interaction is possible with the host application.
Enabled	Determines if the user may interact with the control.
ForeColor	Determines the text color of the control when enabled.
Notes	The collection that holds the individual Note objects.
SelectedItem	The Note object currently selected.
View	Determines the way in which the control is displayed. There are three different view modes.

Part VI
Examples

God gives every bird his worm, but he does not throw it into the nest.

-Swedish Proverb

Chapter 14

Scheduling Program

It should be self-explanatory how a day schedule works in normal mode. You just drop a schedule on a form, set the MinDate and MaxDate and you are done. This is not much of an example from which to learn. So I have chosen to make our first real application a bit more complex. We will create an MDI interface and open a new child window for each day. Instead of having just one column for the day, we will schedule rooms. This example assumes that we are building an application for a doctor's office. In this fictional office we are required to schedule patients in different rooms of the office. This is important because in an office you may have 10 patients scheduled at 10:00 AM; however they will be in different rooms. This is a real-world example and one that many developers have had to build.

We may construct a simple scheduling application that still presents much of the functionality of the schedule component. We will be able to create a new schedule. Also we will be able to open an existing schedule. Finally, we will be able to save and print schedules. The first thing to do is to create the needed forms. We will need an MDI and a child form. On the child form we will place a GbSchedule object. To ensure that the schedule is the proper size we will need to add a little code to the resize event.

```
Private Sub Form_Resize()

On Error Resume Next

  Schedule1.Move 0, 0, Me.ScaleWidth, Me.ScaleHeight

End Sub
```

This code will not cause any error if the form is minimized because it skips all errors. The only errors that could occur are resizing errors and there is nothing to be done about those anyway, so we will ignore them.

Next we will add some rooms to the schedule. You may, of course, add a configuration screen to setup rooms. This is the obvious thing to do especially it you are deploying your application to many clients and do not know their room configuration at design-time. However, for the simplicity of this example we will just hard-wire the rooms. I have created a routine to add rooms to the schedule and it is called from the Form_Load event. This routine will add four rooms to the schedule.

```
Private Sub LoadRooms()

  'We will now load the rooms for this day.
```

```
  'Every day has these rooms since every day
  'we have the same number of rooms in our office
  Call Schedule1.Rooms.Clear
  Call Schedule1.Rooms.Add("Operatory I")
  Call Schedule1.Rooms.Add("Operatory II")
  Call Schedule1.Rooms.Add("Exam I")
  Call Schedule1.Rooms.Add("Exam II")

End Sub
```

Now that we know the rooms will be added, we need to write a routine to open and save files. This application will not use any database access. It will use the built-in XML file routines. The OpenFile and SaveFile routines will load and save schedules to XML files.

```
Public Function OpenFile(ByVal dtDate As Date) As Boolean

Dim oXMLParameters As CXMLParameters
Dim sFileName As String

  'Save this screen's date
  MyDate = dtDate

  'This will load a schedule from file (if need be)
  sFileName = AppPath & GetFileName(Me.MyDate)
  If FileExists(sFileName) Then
    Set oXMLParameters = New CXMLParameters
    oXMLParameters.FileName = sFileName
    oXMLParameters.EmployGMT = False
    oXMLParameters.PropertyAll = False
    Call oXMLParameters.UseAllCollections(False)
    oXMLParameters.VerifyOnly = False
    Call Schedule1.ScheduleItems.Clear
    Call Schedule1.ImportXML(oXMLParameters)
  End If

  'Setup the caption for this screen
  Me.Caption = Format(MyDate, "dddd mmm dd, yyyy")

  'There is nothing dirty
  Changed = False

EndSub:
  Set oXMLParameters = Nothing

End Function

Public Sub SaveFile()
```

```
Dim oXMLParameters As CXMLParameters
Dim sFileName As String

  'This will commit the schedule to file
  Set oXMLParameters = New CXMLParameters
  sFileName = AppPath & GetFileName(Me.MyDate)
  oXMLParameters.FileName = sFileName
  oXMLParameters.Overwrite = True
  oXMLParameters.EmployGMT = False
  oXMLParameters.PropertyAll = False
  Call oXMLParameters.UseAllCollections(False)
  oXMLParameters.VerifyOnly = False
  Call Schedule1.ExportXML(oXMLParameters)

  'There is nothing dirty anymore
  Changed = False

EndSub:
  Set oXMLParameters = Nothing

End Sub
```

Notice that the methods make sure to only save and load the ScheduleItems collection. The “UseAllCollections” method has been sent a parameter of false. This will ensure that the Rooms, Categories, and other collections are not saved or loaded to/from file.

The child form has a property called MyDate. This is a property used to identify the date associated with the form. We will use it later to make sure that we do not load a date more than once. Notice that this property is set in the OpenFile method. We will make sure that anytime a form is loaded this method is called. If the file does not exist, we will load simply load a blank day. If it does exist, we will load it form file. Either way, this property is set during the method call.

A final method for the child form is the print method. It will print the entire existing schedule. First we must declare the print parameter object. This stores all the information about the printing process such as copies, orientation, etc. We set its device name to the default printer’s device name. This will setup print to work with this printer. The GoPrint method is flexible. Depending on the configuration of the schedule it takes differing parameters. Since we are printing a schedule that has been displayed in RoomOnly mode, the first two parameters are the start room and end room. These are set to the first and last rooms. The start time is set to the schedule’s start time and the end time to the schedule’s last displayed time. This will print the entire schedule.

```
Public Function PrintFile() As Boolean

Dim oPrinterParameter As New CPrinterParameter
```

```
  oPrinterParameter.PrinterDeviceName = Printer.DeviceName
  PrintFile = Schedule1.GoPrint(1, _
                  Schedule1.Rooms.Count, _
                  Schedule1.StartTime, _
                  DateAdd("h", Schedule1.DayLength, _
                  Schedule1.StartTime), _
                  oPrinterParameter)
  Set oPrinterParameter = Nothing

End Function
```

Next we need to a property to the child form called “Changed”. This is a Boolean property that will keep up with the changed state of the schedule. When true, it means that the schedule needs to be saved to file. After loading and saving this property is set to false, because there is nothing that has changed since the last save. In the schedule’s events that signify change we will set this variable to true as in the following code.

```
Private Sub Schedule1_AfterAdd(ByVal NewIndex As Long)
 Changed = True
End Sub
```

The events that need this line of code include: AfterAdd, AfterCopy, AfterDelete, AfterDragFromFile, AfterEdit, AfterEditNotes, AfterEditText, and AfterMove. If any one of these events is raised the schedule will need to be saved.

If the user presses the child window close button, how will the schedule know to save? In the Form_QueryUnload event we will add code to catch this situation.

```
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As
Integer)

  If Changed Then
    Select Case MsgBox("This schedule has been modified. Do you
wish to save?", vbYesNoCancel + vbQuestion, "Save?")
      Case vbYes: Call SaveFile
      Case vbNo: 'Do Nothing
      Case vbCancel: Cancel = True
    End Select
  End If

End Sub
```

If the “Changed” variable is set we will prompt the user to save the form. The user may then choose “Yes” to save and unload, “No” to not save and unload, or “Cancel” to not save and cancel the unload of the window all together.

Now that the child form has been constructed we need a way to open these forms as well as a container in which to contain them. So we need to build an MDI parent form. We add one to our project and start by declaring the menus. Open the menu editor and add a top-level menu. We will set the caption to “File” and name it “mnuFile”. Under this menu will add the following menus: Open, Close, Save, Print, and
Exit. This will provide us with the needed functionality to use this program.

The open menu item will prompt the user for a date and if the date has not been loaded previously, it will load a child form and have it open the saved file. If the file does not exist it will load a blank schedule, remember? The GetDate function is defined later; just assume that it will return a date to open.

```
Private Sub mnuFileOpen_Click()

Dim F As frmChild
Dim dtDate As Date

  'Get a date from the user If valid then
  'open a child window and load this date
  If GetDate(dtDate) Then
    If IsDateLoaded(dtDate) Then
      Call MsgBox("This date is already loaded!", bExclamation)
    Else
      Set F = New frmChild
      Call F.OpenFile(dtDate)
      Call F.Show
      Call UpdateMenu
    End If
  End If

End Sub
```

The Close menu item will do nothing more than close the active form.

```
Private Sub mnuFileClose_Click()

  Unload Me.ActiveForm
  Call UpdateMenu

End Sub
```

The Save menu item will call the SaveFile method of the child form. Each child knows how to save itself.

```
Private Sub mnuFileSave_Click()
  Call Me.ActiveForm.SaveFile
```

```
End Sub
```

The Print menu item will tell the active form to print itself. Every child knows how to print itself as well.

```
Private Sub mnuFilePrint_Click()

  If MsgBox("Do you wish to print the active schedule?", _
         vbYesNo + vbQuestion, "Print?") = vbYes Then
    Call Me.ActiveForm.PrintFile
 End If

End Sub
```

In order to prompt the user for dates we will need to construct a form for this action. We only need to create a form with a textbox and two command buttons. The textbox will be used to type in the date and the command buttons will be the “Ok” and “Cancel” buttons. In the Ok button’s code we will need to check that the entered date is valid with the VBA “IsDate” function. To ensure that we do not load a date twice we will need to check the loaded child forms and verify that their dates are different than the target date. The IsDateLoaded method is on the MDI parent form and is used for this purpose.

```
Private Function IsDateLoaded(ByVal dtDate As Date) As Boolean

Dim F As Form

  For Each F In Forms
    If F.Name = "frmChild" Then
      If F.MyDate = dtDate Then
        IsDateLoaded = True
        GoTo EndSub
      End If
    End If
  Next

EndSub:

End Function
'This function will ensure that the Date has no round-off error
Public Function GetDate(ByVal dtDate As Date) As Date
  GetDate = DateSerial(Year(dtDate), _
                       Month(dtDate), Day(dtDate))

End Function
```

To make sure that our menus are always up to date, we will need a method that enables the menu system when required. If there is an active child form, the save and print menus should be enabled. However, if there is no active child form these should be disabled, since it make no since to save or print nothing.

```
Private Sub UpdateMenu()

Dim bHasChild As Boolean

  bHasChild = Not (Me.ActiveForm Is Nothing)

  mnuFileClose.Enabled = bHasChild
  mnuFileSave.Enabled = bHasChild
  mnuFilePrint.Enabled = bHasChild

End Sub
```

All of this code will actually make a working application that can save and load files. It can prompt the user for a date to open and error check to make sure that the date is not loaded. It will make sure that the user cannot accidentally unload a schedule without saving it. Further, it will print the schedule as well. In all, this is surprisingly little code to accomplish all of this functionality.

Chapter 15

OtherDrop Example

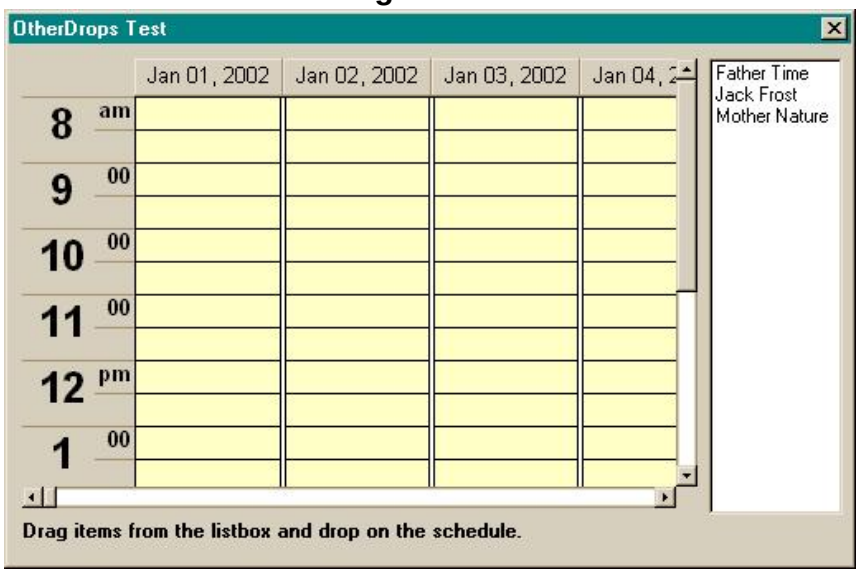
> **Quick Tip**
> You may drop a non-schedule object on a schedule if needed to create an appointment if necessary.

The next example is deceptively simple. It is just over 20 lines of code but still merits the definition of an application that performs a complicated action. There are times when you will want to add appointments from some external source. You may not want the user to click to add appointments at all. This example has one form with a schedule and a list box. The list box is populated with people, clients, trucks, or whatever. You want to drag an item from the list and have it appear on the schedule with the proper text displayed. For this example I have hard-coded the list items. You could of course load it from anywhere, preferably your database of patients. I have three items in it. In order to drag an item and drop it on a schedule you will need to add a little code.

```
Option Explicit

Private Sub Form_Load()

  Call List1.AddItem("Father Time")
  Call List1.AddItem("Jack Frost")
  Call List1.AddItem("Mother Nature")

End Sub

Private Sub List1_MouseDown(Button As Integer, Shift As
Integer, X As Single, Y As Single)
  Call List1.OLEDrag
End Sub

Private Sub List1_OLESetData(Data As DataObject, _
                      DataFormat As Integer)
  Call Data.SetData(Schedule1.CreateByteArray("90"), _
                      Schedule1.OLEDragFormat)
End Sub

Private Sub List1_OLEStartDrag(Data As DataObject, _
                       AllowedEffects As Long)
  Call Data.SetData(, Schedule1.OLEDragFormat)
  AllowedEffects = vbDropEffectCopy
End Sub

Private Sub Schedule1_AfterAdd(ByVal NewIndex As Long)
  Schedule1.ScheduleItems(NewIndex).DisplayText = List1.Text
End Sub
```

The first thing that needs to be done is to inform the list box that it needs to be set for OLE dragging. In its MouseDown event, call its OLEStartDrag method. This will begin a drag. So that target destination knows what type of drag formats the list box supports, we set the drag format to the schedule drag format. This is a special constant defined by the schedule in its OLEDragFormat property. If a target drop destination actually wants the data, it is set through its OLESetData event. In this event you should set the data the schedule expects. The only data that the schedule wants is the appointment length. If you do not set any data the length will be set to the default ScheduleIncrement. If you have a desired length for the new appointment set it by creating a byte array using the schedule's CreateByteArray method. Send in the appointment length and an array of bytes that represent this value is returned. This value is used to populate the Data object. The outline of the appointment as the user is dragging over the schedule will be this length.

Figure 15.1



The final event used is the schedule's AfterAdd event. This will allow us to set the text displayed inside of the appointment. After the drop the new appointment's index in the ScheduleItems collection is returned as a parameter. You may use this to access the newly added object and set it DisplayText property. The appointment then will have been added and the text will have been set.

This is entire application. It will allow you to add appointments by dragging them from an external object. Keep in mind that the schedule's AllowOtherDrops property must be set to true. This allows you to configure whether the schedule actually allows this type of appointment adding in the first place.

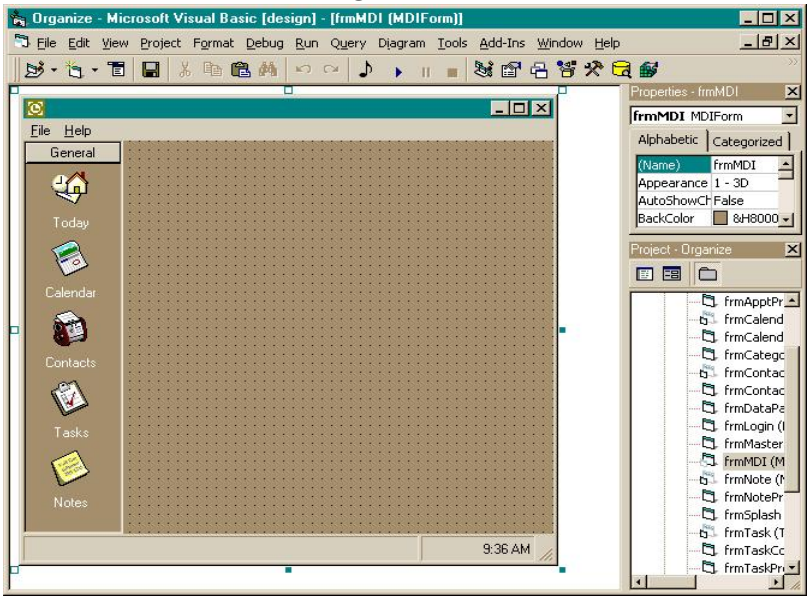
Chapter 16

GbOrganize Application

Our final example is the most complex. We will attempt to integrate the schedule into a real, usable application. GbOrganize is a mock up MS-Outlook. It looks similar and has many of its features. The most glaring absence is the email functionality. However since we are attempting to display the features of GbSchedule, I do not feel that this is a problem. The application can be thought of as four applets. These include calendar, task, note, and contact programs. The last three are added to create a bona fide application. The calendar, which uses the GbSchedule, will be added last. We will create the functionality for it last.

We will use the Gravitybox Listbar, Schedule, and Directory Browser in this application. All may be downloaded free of charge from the Gravitybox website. We begin by creating an MDI parent form with a Listbar and adding five items to it.

Figure 16.1



We now need to add code behind the Listbar to load the forms that we will create later in the demonstration.

```
Private Sub ListBar1_ItemClick(ByVal oTab As GbListBar.CTabEl,
ByVal oItem As GbListBar.CItemEl)

  Screen.MousePointer = vbHourglass

  If Not CloseForms Then GoTo EndSub
```

```
  Select Case oItem.Name

    Case "Today": 'Today
      frmToday.Show

    Case "Calendar": 'Calendar
      frmCalendar.Show

    Case "Contacts": 'Contacts
      frmContact.Show

    Case "Tasks": 'Tasks
      Load frmTask
      Set frmTask.TaskColumns = TaskColumns
      Call frmTask.RefreshForm
      frmTask.Show

    Case "Notes": 'Notes
      frmNote.Show

  End Select

End Sub
```

The code for this method merely loads the appropriate form, when the user presses an item on the Listbar.

The entire application rides on top of the OrganizeAPI DLL file. In the code displayed in this section, you will see references to objects that you have never seen before and no database access. Though this application uses a database to save and load data, only the API actually touches the database. The GUI that we are building simply calls the API. By constructing an object-oriented API, other application may be built that use the API to manipulate the objects. For example, another application could be made that does some type of data backup once a week. Another may be built that cleans up old appointments or creates an archive of old appointments or task items.

Throughout this demonstration, the loading and saving of objects will be done using a global object named “ThisUser”. This object is defined from the “CUserPreference” class in the API DLL. It has properties and methods that control user maintenance, saving, loading, etc. All load/save methods take this object as a parameter to properly perform their operations. The format was chosen to ensure that the same user load and saves the objects. This solves certain consistency problems. This user object also contains the needed database connection information.

Notes

We will start by adding notes, since this is the simplest thing to do. A note object is exactly what it sounds like, apiece of text. The notes form is an MDI child and loads the notes from the API when the form loads.

```
Private Sub Form_Load()

Dim oNote As CNoteEl

  frmMDI.mnuNote.Visible = True
  Call frmMDI.SetStatus(oNotes.Count & " Items")

  Call oNotes.Load(ThisUser)
  For Each oNote In oNotes
    Call lvwNote.ListItems.Add(, oNote.UniqueKey, _
                      oNote.Text, "Note")
  Next

End Sub
```

The code above uses the object “ThisUser” to load the oNotes object. It then loops through the collection of notes and inserts each into the notes ListView. When the user selects a displayed note on the screen by double-clicking or pressing <Enter>, a note property window will appear that allows the user to modify the note text. After the user modifies the text, he may press the window’s close button. In the QueryUnload event the text box’s value will be used to set the note object’s text value. This will ensure that the next time the notes collection is saved this value will be committed to file.

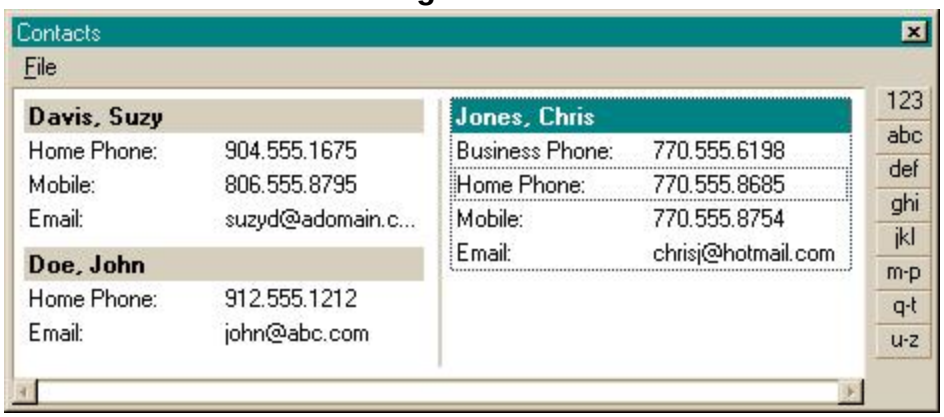
```
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As
Integer)

  Note.Text = txtNote.Text

End Sub
```

This is essentially the entire functionality of the notes portion.

Contacts

The next part of the program is to manage contacts. This is only a little more complicated than the notes. A note has one field, Text that defines an object. A contact object has many fields. A contact is defined a set of information that describes a person.

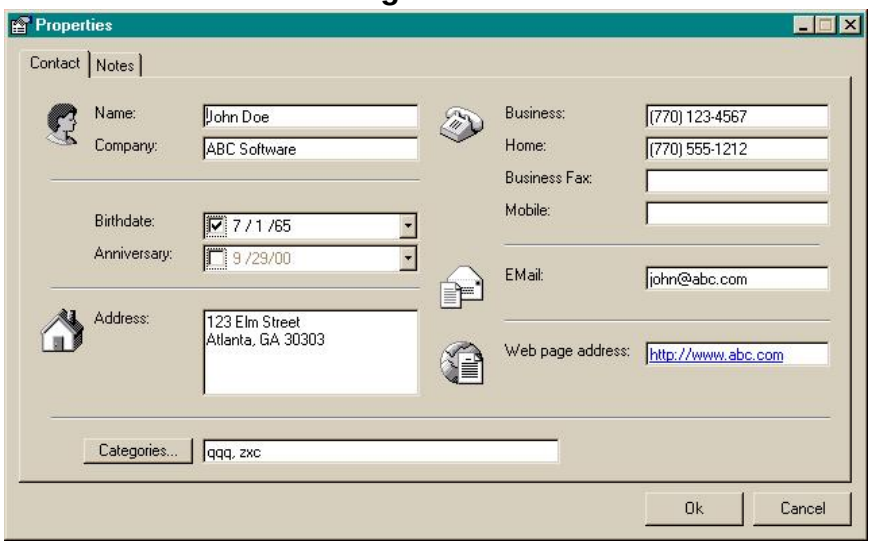
Figure 16.2




```
Private Sub LoadForm()

Dim oContact As CContactEl
Dim oListItem As ListItem

  Call oContacts.Load(ThisUser)
  Call frmMDI.SetStatus(oContacts.Count & " Items")
  Call lvwContact.ListItems.Clear
  For Each oContact In oContacts
    Set oListItem = lvwContact.ListItems.Add(, _
             oContact.UniqueKey, oContact.Name, _
             , "Contact")
    Call FormatListItem(oListItem, oContact)
  Next

End Sub
```

The LoadForm method of this screen will load the contacts collection and populate the Contacts control. The user may select a contact by double-clicking or pressing the <Enter> key. When this occurs the contact's property window is displayed. This allows the user to modify information about the contact.

Figure 16.3



A contact has much associated information. After the user modifies the properties on the screen, he may press the Ok button to save the information. In the save routine the text boxes on the screen are used to set the contact object's properties with the new values.

```
Private Sub cmdButton_Click(Index As Integer)

  ReDim ParOut(1)
  Select Case Index

    Case 0: 'Ok
      Contact.Address = txtAddress.Text
      Contact.Anniversary = IIf(IsNull(dtAnniversary.Value), _
                            0, dtAnniversary.Value)
      Contact.Birthday = IIf(IsNull(dtBirthDate.Value), 0, _
                            dtBirthDate.Value)
      Contact.Company = txtCompany.Text
      Contact.Email = txtEMail.Text
      Contact.HomePageAddress = txtHomePageAddress.Text
      Contact.Name = txtName.Text
      Contact.Notes = txtNotes.Text
      Contact.PhoneBusiness = txtPhoneBusiness.Text
      Contact.PhoneBusinessFax = txtPhoneBusinessFax.Text
      Contact.PhoneHome = txtPhoneHome.Text
      Contact.PhoneMobile = txtPhoneMobile.Text
      Contact.Category = txtCategories.Text
      ParOut(0) = True

    Case 1: 'Cancel
      ParOut(0) = False
```

```
  End Select
  Unload Me

End Sub
```

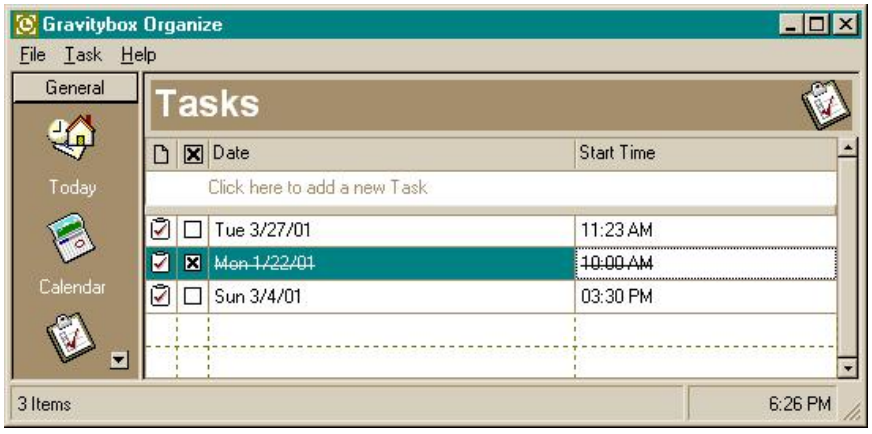
On the main contact form the Contacts collection must be saved when the user unloads the form. This happens by clicking on another function like notes, tasks, or calendar; or by closing the application all together. The QueryUnload event is used to save the contact collection and deallocate the object.

```
Private Sub Form_QueryUnload(Cancel As Integer, _
                             UnloadMode As Integer)

  Call oContacts.Save(ThisUser)
  Set oContacts = Nothing

End Sub
```

Tasks

The next operation uses a control from the GbSchedule module. The TaskList control is used to display any number of columns and the information in them. The tasks in this program have been setup to display none or more of the following items: date, start time, subject, notes, and priority. These are properties associated with appointment; however these tasks are independent from the calendar screen.

Figure 16.4



A TaskList has multiple sections. First there is the add portion at the top of the control. This may be selectively toggled on/off with the “AllowAdd” property. In Figure 16.4, the property value is true. The user may click inside this area to start editing a new appointment. When the <Enter> key is pressed, the task is added to the list. If the <Esc> key is pressed in this section, the new add is canceled and all information typed into this section is removed.

The TaskList has a checked column. This corresponds to the task's "PercentComplete" property. It may have a value of 0 to 100. If the value is equal to 100, the checkbox is marked to indicate that the task is complete. If the user clicks the check box, the "PercentComplete" property value is automatically set to 100, not matter its previous value. As the user pressed the up, down, left, and right keys the selected row and column will change. In Figure 16.4, the second row is selected. In it the second column is selected as well. The user may press the <F2> at any time to edit the selected cell. Cells may have different property types as well. The valid values for cell types are date, time, text, and noedit. The date and time will verify that the user only enters valid dates and times. The text value is a catchall. Any value is a valid string. The noedit value is used when you do not wish to allow the user to edit the cell.

Loading the TaskList is a bit complicated. Because it is a general-purpose control, there are no predefined columns. Before you may load any data, you will need to load the columns. The control has a column's collection and may be loaded as follows.

```
Call TaskList1.Columns.Clear
Call TaskList1.Columns.Add("Date", ctDate)
Call TaskList1.Columns.Add("Start Time", ctTime)
Call TaskList1.Columns.Add("Subject", ctText)
Call TaskList1.Columns.Add("Notes", ctText)
```

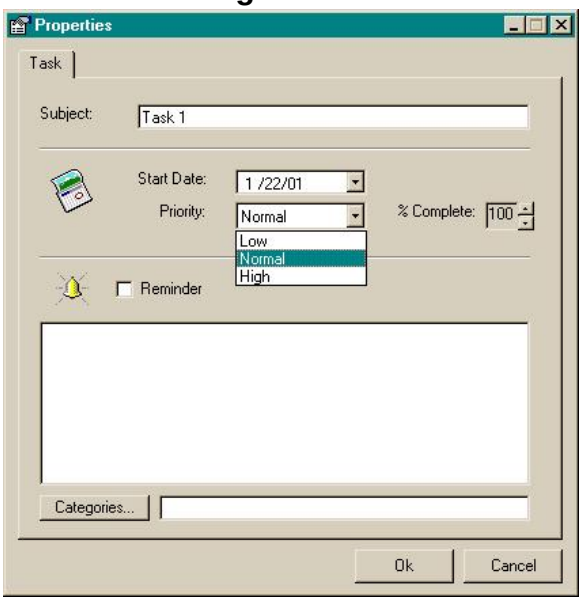
This code will clear any columns that exist. It will then add four columns with their appropriate types. After this addition there will be four columns displayed on the TaskList. We may then start to add data items.

The first thing you will notice when adding data items is that you do not specify any data! There is a reason for this. Since there are no predefined columns there can be no properties associated with these non-existent columns. You must append a data object to the collection and then set its sub-items. Since there are 4 columns, we each Task will have a TaskItems collection with 4 items in it.

```
Dim oTask As CTaskEl

  Set oTask = TaskList1.Tasks.Add
  oTask.TaskItems(1).Text = "12/31/2002"
  oTask.TaskItems(2).Text = "11:00 AM"
  oTask.TaskItems(3).Text = "My Subject"
  oTask.TaskItems(4).Text = "Some Notes"
  Set oTask = Nothing
```

We can use this object hierarchy to load and save. The OrgranizeAPI has a Tasks collection. This may be loaded as the notes and contacts collection objects were previously.

Figure 16.5



Once added, the user may double-click on a task to bring up its property box. It allows the user complete control over all aspect of the task. In the GbOrganize example, columns may be hidden. This means that not all information is necessarily visible. The property box gives the user a way to view all properties at one time.

The property box is not a built-in form. You must build this screen. If you wish to include it in your application, it must be built. The form has text boxes and such on it, but how is it loaded? Simple, we define a new property of the form called Task of type CTaskEl. We then set the property before we show the form. All of the controls on the form are setup by the property set method.

```
Option Explicit

Dim m_Task As OrganizeAPI.CTaskEl

Public Property Get Task() As OrganizeAPI.CTaskEl
  Set Task = m_Task
End Property

Public Property Set Task(ByVal Value As OrganizeAPI.CTaskEl)

  Set m_Task = Value

  'Setup the screen
  txtSubject.Text = Value.Subject
  If Value.StartDate = 0 Then
    datStartDate.Value = Now
  Else
    datStartDate.Value = Value.StartDate
```

```
  End If

  If Value.Priority = pcLow Then cboPriority.ListIndex = 0
  If Value.Priority = pcNormal Then cboPriority.ListIndex = 1
  If Value.Priority = pcHigh Then cboPriority.ListIndex = 2

  VScroll.Value = VScroll.Max
  VScroll.Value = VScroll.Min
  VScroll.Value = 100 - Value.PercentComplete
  txtCategories.Text = Value.Category
  chkReminder.Value = IIf(Value.Reminder, vbChecked, _
                                          vbUnchecked)
  txtNotes.Text = Value.Notes

End Property
```

This means that to call the form all we need to do from the parent is load the form and set its Task object

```
Load frmTaskProperties
Set frmTaskProperties.Task = Tasks(oTask.Index)
FrmTaskProperties.Show vbModal
```

Now the form will loaded, it will load all of its own child controls and then it will be shown as a modal form. We put this code in the “TaskDblClick” event of the TaskList. When the user double-clicks on a task, this code is executed.

Calendar

Now it is time for the crowning jewel of the entire example. This application is used to show off the schedule, so let start to build something with the schedule on it. The calendar functionality of the GbOrganize example illustrates how to build a nice, simple schedule. For this, we will need the same thing the other parts of this application require a main form and a property box.

The main form is an MDI child form with a schedule placed on it. It will be capable of displaying in four different modes: day, work week, week, and month. In day mode the schedule will display one day only. There will be one column (the day) and the times will be displayed on the left. Its MinDate and MaxDate will be the same. In work week mode, the schedule will display one week Monday to Friday. There will be no weekends. In week mode, the schedule will display Sunday through Saturday of a particular week. Finally in month mode there will be an entire month visible at one time. All displays are controlled by a calendar on the right side of the screen.

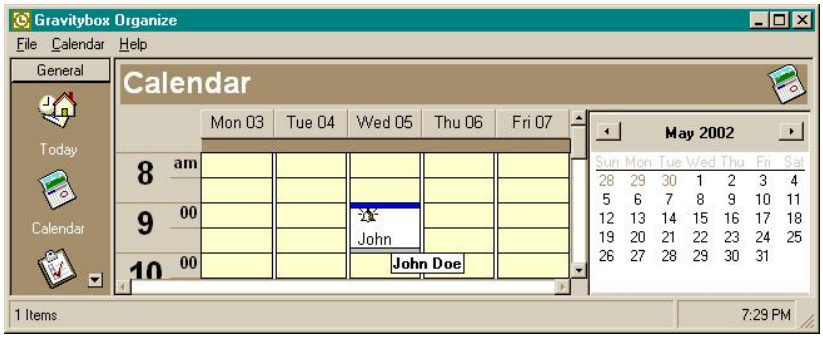
The MonthView control on the right side of the screen defines the current date. In day mode this is the date that is displayed. In any other mode this date is belongs to the

date range displayed. For example, if we are in month mode and the selected date was February 16, the displayed Month would be February.

```
Schedule1.AutoRedraw = False
Select Case DisplayMode

  Case cdcDay:
    Call Schedule1.SetMinMaxDate( _
                  MonthView1.Value, MonthView1.Value)

  Case cdcWorkWeek:
    Call Schedule1.SetMinMaxDate( _
                  FirstDayOfWeek(MonthView1.Value, vbMonday), _
                  LastDayOfWeek(MonthView1.Value, vbFriday))

  Case cdcWeek:
  Call Schedule1.SetMinMaxDate( _
                  FirstDayOfWeek(MonthView1.Value), _
                  LastDayOfWeek(MonthView1.Value))

  Case cdcMonth:
    'Do Nothing Yet

End Select
Call LoadForm
Call RefreshHeaderFormat
Call frmMDI.SetStatus(oAppointments.Count & " Items")
Schedule1.AutoRedraw = True
```

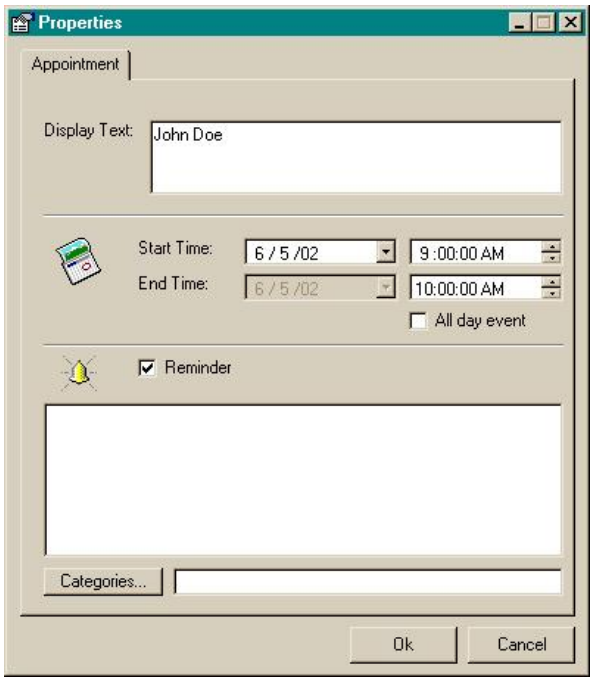
This code will set the MinDate and MaxDate to the appropriate values depending on the MonthView value and the display mode. Notice that the AutoRedraw was set to false while loading occurred. This will speed up the loading significantly. The schedule will normally redraw itself after every appointment addition. We will toggle this feature off only during load.

Figure 16.6



When there are appointments actually on the schedule, the user may double-click on each to display its property box. Each appointment has many properties including date, start time, and reminder. Most of these are self-explanatory. An interesting one is the reminder property. It will set the schedule to raise an event when the specified item comes due. For example, if we make an appointment for June 5, 2002 at 9:00 AM, the "ScheduleItemStart" event would be raised at this time to signify that the appointment has come due. In the GbOrganize example, the property box is displayed.

Figure 16.7



This informs the user that an appointment has come due and gives him access to the appointment's properties. In this example we have loaded the properties screen and shown it non-modally. The reason for this is that if the user is away or this is an automated application then another appointment may come due while the first screen is still displayed. In this example there may be any number of property boxes displayed on the screen, waiting for user attention.

```
Private Sub Schedule1_ScheduleItemStart(ByVal Index As Long)

  Call ShowApptDue(Schedule1.ScheduleItems(Index),
oAppointments)

End Sub

Public Sub ShowApptDue(ByVal oScheduleItem As CScheduleEl, _
                       ByVal Appointments As CAppointmentCol, _
                       Optional ByVal lSnoozeCount As Long = 1)
```

```
Dim F As frmAppointmentDue

  Set F = New frmAppointmentDue
  Set F.ScreenAppt = oScheduleItem
  Set F.Appointment = Appointments.GetObject( _
                      oScheduleItem.UniqueKey, osTag)
  Call F.Show
  Set F = Nothing

End Sub
```

You may be wondering what happens if you remove an appointment that has its property box displayed. Since they are non-modal this is a very real situation. In the schedule's AfterDelete event, a search is performed to determine if an appointment property box exists for the removed appointment. Since every appointment has a unique identifier, a routine may be constructed that loops and searches for the UniqueKey property. If found, the window is unloaded.

```
Private Sub UnloadAppointmentDialog(ByVal sUniqueKey As String)

Dim oForm As Form

  For Each oForm In Forms
    If StrComp(oForm.Name, "frmApptProperties", vbTextCompare)
= 0 Then
      If StrComp(oForm.ScreenAppt.UniqueKey, sUniqueKey, _
                 vbTextCompare) = 0 Then
        Unload oForm
      End If
    End If
  Next

End Sub
```

This concludes the example. Of course the creation of a full-fledged program is more complicated than what I have described. I do not find it necessary to fill volumes with source code that you are neither going to read. All of the code is available on-line and in digital format so that you may actually compile and use it. There are many supporting routines and an API that are used in the GbOrganize example. In its current incarnation there are about 8500 lines of code. This is not much considering that all of the functionality that it provides. Since the entire source is provided for the example, you may feel free to extend it and add any useful functionality that you may dream-up.

Part VII
Appendix

Every closed eye is not sleeping; and every open eye is not seeing.

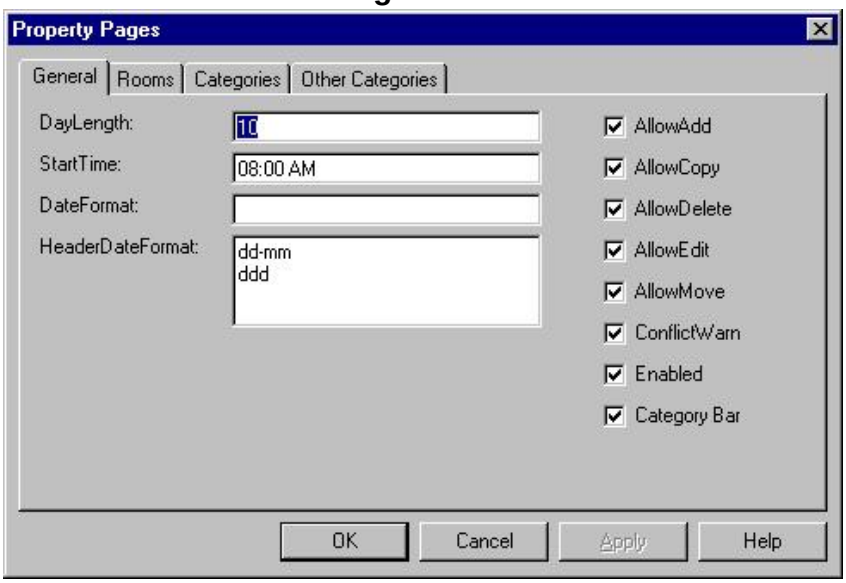
-Unknown

Property Pages

Property pages allow design-time properties to be edited with a graphical interface. In addition, they provide a way to set and store some of the collections at design-time.

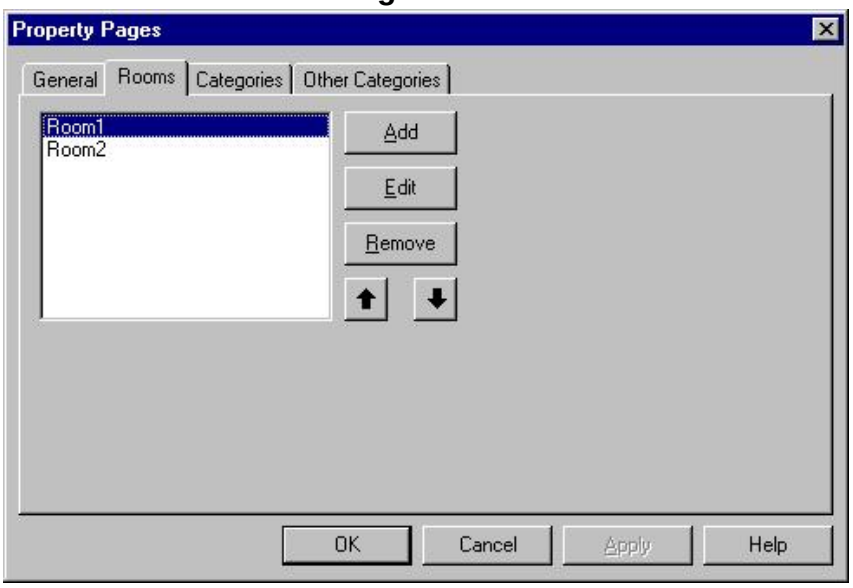
The “General” Property page simply allows for the setting of some of the important schedule properties that are already available in the property box in Visual Basic. Included properties are some of the “Allow…” properties and the text formatting properties. The HeaderDateFormat format may be formatted to display on multiple lines. This text box allows for the addition of a complex header format that may not be readily set in the property box. The StartTime and DayLength may also be set. These will generate corrections if the StartTime plus the DayLength try to make the schedule display past 12 midnight.

Figure A.1



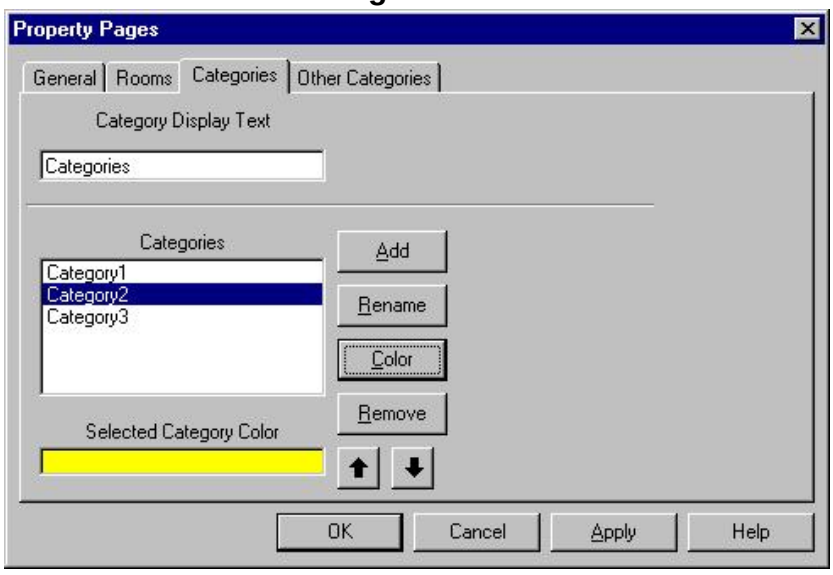
The “Room” property page will allow for the setup of the Rooms collection at design-time. This page may be useful, especially if no dynamic or run-time modifications of the Rooms collections are required. Room objects may be added, edited, or removed from the collection on the page. The position in the collection of each object may be modified as well, using the arrow keys.

Figure A.2



The “Categories” property page allows for the setup of the Categories collection, much like the Rooms page allows for the setup of the Rooms collection. Any Category object may be added, edited, or removed from the collection. The order of objects in the collection may be changed as well, with the arrow keys. A difference from the previous property page is that there is a color button present. After selecting a Category object, its color may be set. This is its associated color that will be used in its displayed category bar.

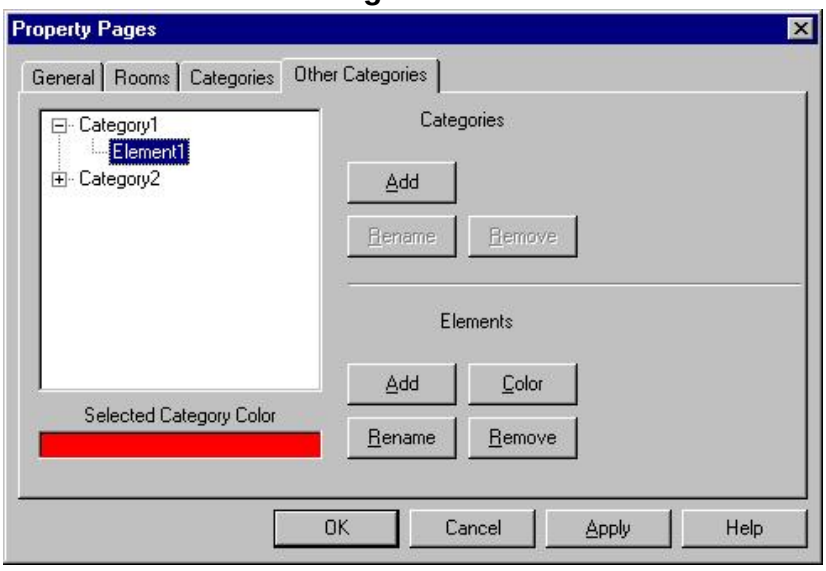
Figure A.3



The final property page is the “Other Categories” page. This will allow for the setup of the Categories collection that resides on the ScheduleItems object. These additional user defined categories may be used to associate additional information with each appointment. Since this is a collection of categories, a Category collection must be

defined first. After selecting an existing Category collection, elements may be added to it. Each ScheduleItem will have a Categories collection with X elements. The X is the number of collections (not elements) defined in this screen. Each element in the Categories collection of a ScheduleItem object is one of the elements in that category. This functionality is defined in the ScheduleItem Categories section under the Advanced Functionality section.

Figure A.4



Properties

AllowActivities	`Property AllowActivities() As Boolean` This property determines if Activities are displayed on a schedule. An Activity is an appointment that is not and all day event and its start time and length make it overlap a day boundary. Activities only exist when the viewmode supports days on top and time on the left. If the viewmode has any other value, no activities are displayed. When this property is set the default property dialog allows the user to set the end date. When false, the default property dialog lists only start date, start time, and end time, but no end date. This ensures that no appointment may overlap a day boundary. Activities are displayed in the event header at the top of the screen.
AllowAdd	`Property AllowAdd() As Boolean` AllowAdd determines if the user can double-click on the background and create an appointment. If this property is True, then the appointment is made and the edit window is evoked. This allows the user to modify any of the item's properties.
AllowAddDialog	`Property AllowAddDialog() As Boolean` Determines if the Add dialog box appears automatically when a ScheduleItem is added. If false, the item is drawn on the screen blank or you may use the AfterAdd event to populate the Item before it is drawn on the screen. If the "AllowAddDialog" property is set to false then the BeforeEdit and AfterEdit events will NOT be raised. So if you want display a custom edit dialog you must do it in the AfterAdd event. This event is called after the appointment has been added to the ScheduleItems collection so the appointment can now be accessed with the "NewIndex" parameter of the event.
AllowAddDrag	`Property AllowAddDrag() As Boolean` AllowAddDrag determines if the user can press the left mouse button on the background and drag the length of a

	desired appointment and it will be created. While the mouse is being dragged, a boxed outline is displayed over the area that will become the new appointment. The user can press the <Esc> key to cancel the create, before the mouse button is released. If this property is true, the appointment is made and the edit window is evoked. This allows the user to modify any of the item's properties.
AllowBrowse	`Property AllowBrowse() As Boolean` This property determines if the up/down arrow keys are enabled on the default property screen. When this property value is true the arrow keys allow to user to navigate between appointments without unloading the screen.
AllowBubbleTips	`Property AllowBubbleTips() As Boolean` This property determines if the tooltips are displayed as standard square windows or the new Windows 2000 bubble windows. When true, the tooltips are drawn in a round window with a pointer to the specified area. When false, the tooltip window is a normal square.
AllowColumnResizing	`Property AllowColumnResizing() As Boolean` This property determines if the user can grab the edge of a column and resize it. If this property is false then the user can not resize columns at all. All columns are the same width, so resizing one applies to all columns.
AllowCopy	`Property AllowCopy() As Boolean` AllowCopy lets the user copy an appointment instead of moving it. If the user drags the appointment with the <CTRL> key pressed then the mouse will have a plus on it to show that this is a copy. When it is dropped, the original appointment will NOT be removed. Note: If the AllowMove property is false, then this property is also false.
AllowDblClickAdd	`Property AllowDblClickAdd() As Boolean`

	This property determines if the user may double click on the background of the schedule to create an appointment. The double click must take place on the background area, not an existing appointment. This property is tied to the AllowAdd property. If AllowAdd is False, this property will also be False, since no adds are allowed.
AllowDelete	`Property AllowDelete() As Boolean` AllowDelete determines if a highlighted appointment can be removed using the <Delete> key. This action will remove the selected ScheduleItem from the ScheduleItems collection. This can be performed programmatically in any case, but can only be performed by the user if this property is set.
AllowDragFromFile	`Public Property AllowDragFromFile() As Boolean` This property determines if the user is allowed to drag a previously saved file to the schedule. When the AllowDragToFile is set, the user may drag an appointment and create a file on the desktop or Explorer. Later the user may drag the saved file back to a schedule window to add the appointment saved in the file to the target schedule.
AllowDragToFile	`Public Property AllowDragToFile() As Boolean` This property determines if the user is allowed to drag an appointment and drop it on the desktop or Explorer. When this property set, the user may drag an appointment and drop it to create a file that contains the information for the specified appointment.
AllowEdit	`Property AllowEdit() As Boolean` AllowEdit determines if the user can double-click on a scheduled item and get an edit window for the modification of a scheduled item's information. If AllowEdit is false then the Edit window is read-only. It will still appear but you will not be able to save any changes. If you wish to cancel any dialog from displaying then cancel it in the BeforeEdit event.

AllowEffects	`Property AllowEffects() As Boolean` This property will allow you to make appointments appear and disappear from and to a point on the screen. When adding an appointment it will expand from a point on the screen into its fully specified size. Conversely, when removing and appointment, it will shrink down into a point.
AllowEventDrag	`Property AllowEventDrag() As Boolean` This property determines if an appointment may be dragged to the event header or an event may be dragged off the event header to become a normal appointment. When this property is false you may view all types of appointments but you may not cross the event header boundary. All events and activities cannot become normal appointments and normal appointments cannot be dragged to the event header.
AllowEventHeader	`Property AllowEventHeader() As Boolean` This property will allow for events to be displayed. Each ScheduleItem has an "IsEvent" property. This property maps to the “All day event” in the default property dialog. If this property is True then the Appointment has no start time or length in reality. The appointment may be thought of as an all day appointment. If the "IsEvent" property is true, the appointment will not be visible inside the main schedule area. The main schedule area is for appointments that have a start time and a specific length. An event by definition does not have these properties set. If the Schedule's property AllowEventHeader is True then all events will be visible above the main schedule area, in the event area. In this area are also displayed Activities. An activity has a start time and length, but these properties cause it to overlap a day boundary thus occupying two or more days.
AllowExtraFields	`Property AllowExtraFields() As Boolean` Added for future use. This property currently has no functionality.
AllowFiles	`Property AllowFiles() As Boolean`

	This property determines if the "files" button is available on the default property screen. When this propety value is true, this button allows to user to add or remove file associations to an appointment.
AllowFind	`Property AllowFind() As Boolean` This property, when set to true, allows the user to press the CTRL-F key combination to display the default "Find Appointments" dialog. This dialog allows the user to search define criteria that are used to find appointments. The results are displayed in the bottom portion of the screen after pressing the "Find" button. The user may double-click on an appointment to edit it if the schedule's AllowEdit property is true. The user may also remove appointments from the list by pressing the <Delete> key if the schedule's AllowDelete property is true.
AllowInPlaceEdit	`Property AllowInPlaceEdit() As Boolean` This property determines if an edit box will appear when the user clicks on an appointment. When the property is True, it allows the user to edit either the Subject or DisplayText of an appointment, depending on the TextDisplay property. AllowInPlaceEdit is tied to the AllowEdit property. If AllowEdit Is False, this property will also be False.
AllowInterWindowDrop	`Property AllowInterWindowDrop() As Boolean` This property determines if appointments dragged from other windows are accepted. If this property is true then the user can move an appointment from another window into the current one. If it is false then the mouse cursor will turn to a NoDrop sign if another appointment is dragged over this schedule. Note: If the AllowMove property is false, then this property is also false.
AllowItemResizing	`Property AllowItemResizing() As ItemResizingConstants` The property determines if the user can resize an appointment. There are 4 modes that can be set. None, TopLeft, BottomRight, or All. If the property is set to None then there is no item resizing by the user. When this

	property is set to TopLeft then only the StartTime of the appointment can be changed by moving its Top/Left edge depending on if the value of the Viewmode property. When BottomRight is set then only the user can set the length of the appointment. He will be able to drag the bottom or right edge of the appointment depending again on the value of Viewmode. In All mode then the user can drag the Top or Bottom of an appointment (Left or Right if time is displayed on top)
AllowMove	`Property AllowMove() As Boolean` AllowMove determines if the user can move a scheduled item with the mouse by dragging it. When this property is set the user can automatically drag a ScheduleItem and drop it elsewhere in the schedule. If this property is False, then no automatic action is taken when the user tries to drag a scheduled item. ScheduleItems can also be dragged across window to another schedule.
AllowOtherDrops	`Property AllowOtherDrops() As Boolean` This allows any object to be dragged onto the schedule. If it is a ScheduleItem from the same or another schedule window, then the operation is a move. If the object is NOT a ScheduleItem then this is an appointment creation. When the object is dropped a new appointment is created at that position. This is useful for having users drop object from other program to create appointments for a schedule
AllowRecurrences	`Property AllowRecurrences() As Boolean` This property defines the functionality of the schedule for recurring appointments. When false, no user interface is displayed to add or remove recurring appointments. When this property is true, the default property window will have a "Recurrences" button. The user may setup a recurrence pattern by using the screen display with this button. When a pattern is applied, each of the newly created appointments is part of a group. They all have the same GroupId property. The GroupId is read-only and is assigned in the background. When the user highlights a recurring appointment and presses the <Delete> key, he will be prompted to remove the single appointment or the entire series (group) of appointments.

AllowRowColTips	`Property AllowRowColTips As Boolean` This property determines if a tooltip is displayed when the mouse hovers over the row or column header. The tooltip is the name displayed in the header.
AllowRowResizing	`Property AllowRowResizing() As Boolean` This property determines if the user can grab the edge of a row and resize it. If this property is false then the user can not resize rows at all. All rows are the same width, so resizing one applies to all rows.
AllowScrollBars	`Property AllowScrollBars() As Boolean` This property determines if the control will display scrollbars. When true, scrollbars are displayed when needed for the user to move about the schedule canvas. When this property is false, no scrollbars are displayed even when needed.
AllowSelector	`Property AllowSelector() As Boolean` The AllowSelector property toggles a selector that highlights an area of a schedule. The user may use the keyboard to navigate and highlight an area of the schedule on which to create an appointment upon pressing the <ENTER> key.
AllowTimeSelector	`Property Allow TimeSelector () As Boolean` This property determines if an arrow is displayed next to the time that is currently selected. When the user moves uses the arrow keys or clicks the mouse the current position is moved. This position is highlighted and the selected may have an arrow next to it to indicate the current time position.
AllowWarning	`Property AllowWarning() As Boolean` This property determines if a warning message is displayed on the SchedulePropertiesControl. The message in is defined by the WarningMessage property.
AllowWeekMargin	`Property AllowWeekMargin() As Boolean`

	This property is used to toggle week numbers on and off. In some locals, such as Europe, week numbers are used in addition to day number on calendars. This property allows you to display the week numbers is desired.
AppointmentBorderWidth	`Property AppointmentBorderWidth() As Long` This property determines the border between appointments. When a conflict exists between 2 appointments they are displayed side by side. The width is determined by the AppointmentBorderWidth property.
AppointmentCategoryBar	`Property AppointmentCategoryBar() As Boolean` This property will determine if the appointments have a colored bar on their left side when their Category property is set. Each ScheduleItem has a Category property that may or may not have a value. If the value matches the name or index of a category in the Categories collection, the appointment has a category. Each category has an associated color. If you wish a colored bat to appear next to the appointment to identify its associated category, set this property to true and it will appear. The Categories collection may also be displayed in the schedule's left margin by using the CategoryBar property.
AppointmentsPlural	`Property AppointmentsPlural() As String` This is the default text used to describe appointments in the plural. The default English text is "Appointments". This is used as a header in many places. If you wish to call your appointments some other name, you may change this property to do so.
AppointmentsSingular	`Property AppointmentsSingular() As String` This is the default text used to describe appointments in the singular. The default English text is "Appointment". This is used as a header in many places. If you wish to call your appointments some other name, you may change this property to do so.
AutoColumnFit	`Property AutoColumnFit() As Boolean`

	This property determines if the user can grab the edge of a row and resize it. If this property is false then the user can not resize rows at all. All rows are the same width, so resizing one applies to all rows.
AutoFocus	`Property AutoFocus() As Boolean` When a schedule is on a screen that does not have focus, there may be times when you want the schedule to get focus if the mouse moves over a ScheduleItem. If this property is True the schedule screen will receive focus every time the mouse moves over any ScheduleItem.
AutoHilite	`Property AutoHilite() As Boolean` This property determines if appointments are highlighted when the mouse moves over them. When true the appointment under the mouse pointer will be set to the SelectedItem. It will be highlighted by drawing the highlight bars on the top/bottom or left/right sides of the appointment.
AutoRedraw	`Property AutoRedraw() As Boolean` This property is a toggle for screen refreshes. The screen is repainted when any action is taken that affects the screen display. Such as a ScheduleItem, Room, Provider, or Category being added or removed. When the AutoRedraw property is False, no screen repaint is performed. To repaint the screen, you must set this property back to True when you wish the screen to be repainted. Turning this property off while adding objects can make loading occur many times faster.
AutoRowFit	`Property AutoRowFit() As Boolean` This property allows for the RowHeight to be updated automatically when the control is resized. The RowHeight will be set to a value that allows all rows to fit exactly in the view port. If you wish for the rows to be evenly spaced, set this property to True. If the number of rows requires the RowHeight to be less than the minimum row height then a scrollbar will appear since it is impossible to fit all the rows on the screen.
BackColor	`Property BackColor() As OLE_COLOR`

	This is the BackColor of the displayed area for the schedule. The display area is defined by the total length down that it takes to display the number of hours for the property DayLength and the total width across that it takes to display the number of days specified by the MinDate and MaxDate Properties.
Background	`Property Background() As StdPicture` This property allows you to specify a background picture to display instead of the Backcolor. If this property is set the backcolor property will be ignored and the specified picture will be tiled across the back of the schedule under the gridlines and appointments.
BlackOutColor	`Property BlackOutColor() As OLE_COLOR` This property allows you to set the color that blacked-out ScheduleItems are displayed. These items cannot be moved or edited. They are displayed on the schedule to let the user know that there is something in that spot, but it can not be modified. This specified color is the color the displayed block. Note: Each ScheduleItem ha a BlackedOut property. This property determines if the BlackOutColor is used. When BlackedOut is set to true, the appointment is colored and can not be moved by the user.
CategoryBackColor	`Property CategoryBackcolor() As OLE_COLOR` This color will fill the category background area. If there are Categories defined and the CategoryBar property is set, the schedule's Categories collection is displayed on the left side of the schedule. Every appointment may have an associated category, if so desired.
CategoryBar	`Property CategoryBar() As Boolean` This determines if a summation of the ScheduleItem's category information is displayed. There are N bars for N categories displayed in the order the categories are in the Categories collection. There is no way to tell if ScheduleItems have conflicting categories.

	If the ShowProviderScheduledTime or ShowProviderAvailableTime properties are set then there are two tabs displayed at the top of the screen Note: This property only applies if the time is displayed in the top margin.
CategoryBarWidth	`Property CategoryBarWidth() As Long` This property defines the width in pixels of each Category bar on the left margin of the schedule. It also defines the width of the Category bar on each appointment's left side.
ColumnWidth	`Property ColumnWidth() As Long` This determines the width of the columns for the schedule only. There is a minimum value determined by the specified font. This value is specified in pixels.
ConflictCheck	`Property ConflictCheck() As Boolean` This property toggles the conflict checking on/off. On schedules with many appointments (>200), there may be a time lag when adding and removing appointments. This happens when it is necessary to calculate the conflicts between appointments. If no conflict checking is necessary, you may turn this feature off to save time. If conflict resolution if necessary but you still wish to save time, you can turn this feature off and when you need to display or perform conflict checking, turn it back on. For example, if you need to display a screen with conflicts turn this feature off until the user selects to display this screen. You can then set ConflictCheck to True and display the conflicts. It is best to turn this feature off if not needed. If your schedule allows conflicts and you do not ever use the Conflicts collection, set this property to false to save time. When this property is false the ConflictWarn property is also false. There can be no warning of conflicts, if there is no checking.
ConflictWarn	`Property ConflictWarn() As Boolean` Determines if the user receives a warning prompt when

	he is moving or saving a scheduled item that overlaps one or more other items.
CustomIconAlarm	`Property CustomIconRecurrence() As StdPicture` This icon may be used to define a custom icon for the alarm. Each appointment object has an Alarm property. When set to true an icon is displayed next to the appointment. This icon defines the icon for this purpose. If this property is set to "Nothing", then the default icon is used.
CustomIconFlag	`Property CustomIconFlag() As StdPicture` This icon may be used to define a custom icon for the flag. Each appointment object has an IsFlagged property. When set to true an icon is displayed next to the appointment. This icon defines the icon for this purpose. If this property is set to "Nothing", then the default icon is used.
CustomIconRecurrence	`Property CustomIconRecurrence() As StdPicture` This icon may be used to define a custom icon for appointments that are in a recurrence pattern. Each appointment object has a GroupId property. When more than one appointment has the same GroupId, the appointments are said to be grouped in a recurrence pattern. This allows you to create and maintain a set of appointments. This icon defines the icon for this purpose. If this property is set to "Nothing", then the default icon is used.
DateHeaderAlign	`Property DateHeaderAlign() As AlignmentConstants` This property determines the alignment of the header text for dates in the top of left margin.
DayLength	`Property DayLength() As Long` This is the total number of hours in a day. The valid range is 1 through 24. The control starts to display the day from the StartTime and progresses for DayLength hours.

	Note: The DayLength can not cause the schedule to overlap into the next day. This causes an error. If the StartTime is 8:00 PM, then the maximum value for DayLength is 4, since more than 4 hours would be the next day.

DayRoomAppearance, DayRoomBackColor, DayRoomFont, DayRoomForeColor	`Property DayRoomAppearance() As AppearanceConstants`
	`Property DayRoomBackcolor() As OLE_COLOR`
	`Property DayRoomFont() As StdFont`
	`Property DayRoomForeColor() As OLE_COLOR`
	
	These properties will determine the display of the day and room headers of a schedule. The font and colors of this section are controlled by these properties. They allow you to set the back and fore colors as well set a 3D or flat appearance. The font may be set as well to customize the look and feel of this section.

DefaultDialogFloater	`Property DefaultDialogFloater() As Boolean` This property determines if the default property dialog is a floating window. When True, the property dialog will "float" on top of all other windows. This means that the schedule control will always be under the dialog. If this property is False then the dialog can be behind the schedule window. If the user clicks inside the schedule, the default property dialog will move behind the schedule window.
DefaultDialogModal	`Property DefaultDialogModal() As Boolean` The default property window may be displayed in a modal or non-modal fashion. If the programming language that you are using supports non-modal dialogs and this property is set to False, the default property window will be non-modal. Keep in mind that since it is non-modal, more than one window may be opened for a schedule and the user may minimize the window as well. Two property windows for a single appointment cannot be opened, but two windows may be opened for two different appointments. The property windows will dynamically be updated if the user changed properties on the main schedule. Also if the user changes properties on the property window and presses the "Apply" button the

	changes will be cascaded to the main schedule. If this property is set to True, only one window ma be opened at a time and it may not be minimized. This property only applies to the default property window. If you cancel the default edits and display your own property window, this property has no effect.
DefaultDialogSubjectCombo	`This property allows you to specify a list of items separated by semicolon (;) that will be listed in a drop-down combo on the appointment default property screen. This will allow you to display a fixed list for the subject of an appointment in which the user may not enter free-from text.`
DisplayDragTip	`Property DisplayDragTip() As Boolean` This property determines if the screen displays a ToolTip when dragging an appointment. If the user is moving an appointment, this ToolTip displays the Date/Room/Time of the position of the mouse. You can also use the DragOverScheduleItem event to display your own message.
DisplayMinutes	`Property DisplayMinutes As GBDisplayMinuteConstants` This property allows for the control of the display of minutes. There are times when you may wish to only display the "00" minutes of an hour. If so, you may set this property to "mdcFirstOnly", otherwise set it to "mdcShowAll".

DragFormatDate
DragFormatTime
	`Property DragFormatDate() As String`
	`Property DragFormatTime() As String`
	
	These properties control the look of the drag tip. When an appointment is being moved or copied to a new position a tooltip window is displayed to inform the use of the current position of the appointment if the mouse is released. These two properties control the format of the date and time portions of the text respectively.

DynamicScroll	`Property DynamicScroll() As Boolean` This property allows you to make the schedule refresh

	when you are dragging a scrollbar. When false, a ToolTip window is displayed to let the user know what day/room/time the schedule will display when the mouse is released. When true, the schedule will redraw during the drag. This maybe considerably slower, since the screen is refreshed for every position on the scrollbar. In most instances, this property should be false. If the target machine is fast and the schedules are small, it may be appropriate to show real-time drag refreshes. Otherwise, avoid using this feature, especially on slower machines.
EnforceTimeLimits	`Property EnforceTimeLimits() As Boolean` This property ensures that the default dialog will not allow not allow the user to create an appointment not defined by the time area boundaries. The StartTime and DayLength properties define the display times of a schedule. When EnforceTimeLimits is set to true, the user may not set an appointment's StartTime or duration to a value outside of this defined area. When false, the user may set the time to any value.
EventsFrozen	`Property EventsFrozen() As Boolean` This property determines if any events will be raised. When this property is True, none of the schedule's events will be raised to the parent container.
EventHeaderColor	`Property EventHeaderColor() As OLE_COLOR` This property will determine the color of the event header background. The event header is an area portion above the main schedule area that displays events. Events are appointments that have been marked as all day appointments.
FindDialogFloater	`Property FindDialogFloater() As Boolean` This property determines if the "Find" dialog is a floating window. When True, the Find dialog will "float" on top of all other windows. This means that the schedule control will always be under the Find dialog. If this property is False then the dialog can be behind the schedule window. If the user clicks inside the schedule, the Find dialog will move behind the schedule window.

FindDialogModal	`Property FindDialogModal() As Boolean` This property determines if the Find dialog is displayed in a modal fashion. If this property is True, the find screen will display as a modal screen. No user interaction with the main schedule is possible until the screen is closed. When False the find screen will be displayed non-modal and the user may interact with the schedule while the Find screen is displayed.
FirstDayOfWeek	`Property FirstDayOfWeek() As VbDayOfWeek` This property determines which day of the week is displayed in the first column of the schedule in month view. When a schedule is displayed in month view, it is in standard calendar format. Different locals around the world start the week on different days. This property allows you specify which day of the week is the start day.
ForeColor	`Property ForeColor() As OLE_COLOR` This is the forecolor of the schedule. All grid lines, text, etc will be drawn with this color.
GridLines	`Property GridLines() As Boolean` This property determines if the grid is displayed on the schedule. By default there is a grid. It consists of lines drawn between every time increment and every column, be it a day or room. This enables the user to clearly see where each appointment lies. There may be times when you do not wish to display the grid. If so, set this property to False, otherwise the grid is displayed.
HeaderDateFormat	`Property HeaderDateFormat() As String` This property defines the format of the date at the top of the Day Schedule. The date headers for the Day Schedule will be displayed in this format. This can be a multiple line format string. The Day Schedule displays two lines to accommodate large date formats. Examples: 1) mm/dd/yy 2) YYYY-MM-DD 3) dd/mm/yyyy

	Note: This property has no meaning if the ViewMode property is set to exclude days.
HiliteBarBottom	`Property HiliteBarBottom() As OLE_COLOR` An appointment is highlighted when the mouse moves over it. The highlight is a bar drawn on the top and bottom (or left/right) of the appointment. This property corresponds to the top (or left) bar that is displayed on a highlighted appointment.
HiliteBarTop	`Property HiliteBarTop() As OLE_COLOR` An appointment is highlighted when the mouse moves over it. The highlight is a bar drawn on the top and bottom (or left/right) of the appointment. This property corresponds to the bottom (or right) bar that is displayed on a highlighted appointment.
IconAlign	`Property IconAlign() As AlignIconConstants` This property determines whether appointment icons are displayed to the top or left of the appointment text. Each appointment has an Icons collection that may be used to display custom icons on an appointment. These icons may be displayed on the top or left of the text.

ImageList	`Property ImageList() As Object`
	
	This is a standard ImageList form the Microsoft common controls. This is used as a repository for images that are to be used on appointments. The proper format for pictures is 16x16 with no mask color. Images of this format will show up on the appointment the best. Each image in the ImageList needs to have a unique key specified. This key is used to display the appropriate image on the appointment.
	
	`Dim oAppt As CScheduleEl`
	
	`Set oAppt = F.Schedule1.ScheduleItems.Add("", _ #1/1/2002#, 0, #8:00:00 AM#, _ 150, "Appt 1")`
	`Call oAppt.Icons.Add("caution")`

	The ImageList is used by each appointment's "Icons" collection. You may add the key of any image in the ImageList. Only valid keys will be processed. Invalid ones are ignored. If a key in an appointment's "Icons" collection is found in the ImageList, the associated image is displayed on that appointment. This allows you to add custom icons to any appointment. There is no limit to the number of icons that may be associated with an appointment. All of the icons that can be displayed inside of the defined appointment area will be shown.
IsDirty	`Property IsDirty() As Boolean` This property returns/sets whether the schedule is dirty and needs to be saved. It returns the dirty values of all the properties, ScheduleItems, Rooms, etc. If set to false all the collection's dirty properties will be set to false as well.
ListDisplayDayOfYear	`Property ListDisplayDayOfYear() As Boolean` This property determines if the days of the year are displayed in List mode. The property only applies then the ViewMode is set to "List". At the bottom of the left frame you may wish to display the current date's day position of the year and the total number of days per year. An example would be "2 / 365" this is January second. The "2" is the second day of the year and the "365" is the number of days per year. The display does display the correct number of days in a leap year.
ListDisplayDoublePage	`Property ListDisplayDoublePage() As Boolean` When a schedule has its Viewmode set to "List", this property determines if the displayed view has one or two pages. If the control displays 2 pages, the Notes property of the appointment will be displayed on the right page.
ListDisplayTimeScale	`Property ListDisplayTimeScale() As Boolean` When a schedule has its Viewmode set to "List", this property determines if all times are shown or just with an associated appointment. When true, all times beginning at the control's StartTime and continuing for the

	DayLength or displayed in increments of ScheduleIncrement. For example if the StartTime is 8:00 AM, the DayLength is 10, and the ScheduleIncrement is 30 minutes, there would be 21 times displayed. Starting at 8:00AM, 8:30AM, ..., and finishing at 6:00PM. If this property is false, no times would be displayed unless it was the StartTime of an appointment.
ListHeaderDateColor	`Property ListHeaderDateColor() As OLE_COLOR` When a schedule has its Viewmode set to "List", this property determines the color of DateText drawn in the header portion of the control. The DayText is not to be confused with the DateText. The DayText is the word for the day, for example "Monday". The DateText is the number of the date for example 23.
ListHeaderDayColor	`Property ListHeaderDayColor() As OLE_COLOR` When a schedule has its Viewmode set to "List", this property determines the color of DayText drawn in the header portion of the control. The DayText is not to be confused with the DateText. The DayText is the word for the day, for example "Monday". The DateText is the number of the date 23.
ListHeaderLineColor	`Property ListHeaderLineColor() As OLE_COLOR` When a schedule has its Viewmode set to "List", this property determines the color of the lines that are drawn in the header portion of the control.
ListHeaderSubTitleColor	`Property ListHeaderSubTitleColor() As OLE_COLOR` When a schedule has its Viewmode set to "List", this property determines color of the SubTitle. There is only one subtitle and it is always displayed on the left page directly under the TitleLeft text.
ListHeaderSubTitleText	`Property ListHeaderSubTitleText() As String` When a schedule has its Viewmode set to "List", this

	property determines the text that displayed below the header on the left page. This is generally used to specify a holiday or some special text. The color can be controlled to attract attention to the text.
ListHeaderTitleColor	`Property ListHeaderTitleColor() As OLE_COLOR` When a schedule has its Viewmode set to "List", this property determines the color of the TitleLeft and the TitleRight properties. These are texts displayed at the top of the control on the left and right pages respectively.
ListHeaderTitleLeftText	`Property ListHeaderTitleLeftText() As String` When a schedule has its Viewmode set to "List", this property determines the text to be displayed at the top of the control on the left page.
ListHeaderTitleRightText	`Property ListHeaderTitleRightText() As String` When a schedule has its Viewmode set to "List", this property determines header text that is displayed on the right page. If there is only one page displayed, this text is not shown.
ListHilightBackColor	`Property ListHilightBackColor() As OLE_COLOR` When a schedule has its Viewmode set to "List", this property determines the color of the item that has been selected by the user. Only one item at a time may be selected. The user may select an item by clicking it with the mouse or using the arrow keys.
MaxDate	`Property MaxDate() As Date` This property defines the starting date for the schedule. The range of days displayed on a schedule is between the MinDate and MaxDate. This defines the total scrollable area of the schedule. The MaxDate has to be greater than or equal the MinDate. Note: This property has no meaning if the ViewMode

	property is set to exclude days.
MinDate	`Property MinDate() As Date` This property defines the ending date for the schedule. The range of days displayed on a schedule is between the MinDate and MaxDate. This defines the total scrollable area of the schedule. The MinDate has to be less than or equal the MaxDate. Note: This property has no meaning if the ViewMode property is set to exclude days.
MonthAppointmentEventColor	`Property MonthAppointmentEventColor As OLE_COLOR` When a schedule has its ViewMode set to Month, it displays a month of appointments at a time. This property determines the backcolor of appointments that have been marked as events.
MonthDisplayTime	`Property MonthDisplayTime() As Boolean` This property determines if the start times of appointments are displayed in front of the appointment text when the schedule is in "Month" mode. When False, the appointment text displayed with no time element.
MonthEvenBackColor	`Property MonthEvenBackColor() As OLE_COLOR` When a schedule has its ViewMode set to Month, it displays a month of appointments at a time. This property determines the backcolor of the days for the months that are Even (2,4,…,12).
MonthEvenForeColor	`Property MonthEvenForeColor() As OLE_COLOR` When a schedule has its ViewMode set to Month, it displays a month of appointments at a time. This property determines the forecolor of the days for the months that are Even (2,4,…,12).
MonthOddBackColor	`Property MonthOddBackColor() As OLE_COLOR` When a schedule has its ViewMode set to Month, it displays a month of appointments at a time. This property

	determines the backcolor of the days for the months that are Odd (1,2,…,11).
MonthOddForeColor	`Property MonthOddForeColor() As OLE_COLOR` When a schedule has its ViewMode set to Month, it displays a month of appointments at a time. This property determines the forecolor of the days for the months that are Odd (1,2,…,11).

NoHorzScroll
NoVertScroll
	`Property NoHorzScroll() As Boolean`
	`Property NoVertScroll() As Boolean`
	
	These properties allow the scrolling to be turned off in either direction. There may be times when the schedule will have scrollbars because the entire schedule can not be displayed on the screen. There may also be times that you wish to keep the user in the current view port and not allow scrolling. If this is the case, you may set either of these properties to False to disallow scrolling in the appropriate direction.

OLEDragFormat	`Property OLEDragFormat() As Long` This property returns the number that defines a Gravitybox Schedule appointment drag. Usually an appointment is moved, copied, etc to the same window or another GbSchedule window. There may be times when you wish to drop an appointment on a non-GbSchedule object. In this case you will need to check to make sure that the object being dragged is an appointment. For example, if you define a picturebox where you wish your users to drop appointments you can add code in its OLDDragOver event to check the drag type. You may compare it to the value (OLDDragFormat) to confirm that the object being dragged is a GbSchedule appointment.
OtherAreaBackColor	`Property OtherAreaBackColor As OLE_COLOR` This color will be used to paint all areas not cover by some other color. This is mainly the top, left margin of the control. All other areas are painted with their own colors such as the time, day, category, and provider headers.
PrintPageInfo	`Property PrintPageInfo() As Boolean`

	This property determines if the page numbers are printed on the pages when a schedule is printed. When true, a printed schedule will display its X and Y page coordinates in the top, left corner of the page. This allows a large schedule to be reconstructed after printing.

PropertiesAllowCustomButton
PropertiesCustomButtonText
	`Property PropertiesAllowCustomButton() As Boolean`
	`Property PropertiesCustomButtonText() As String`
	
	The schedule has a default property window. You have the option of displaying a custom button on this window. When the user clicks on this button the "PropertiesCustomButtonClick" event is raised so that you may add additional functionality to this property window. If you have no use for an extra button on the default properties screen, just set the "PropertiesAllowCustomButton" property to false and it will not appear. You may set the text of the button using the "PropertiesCustomButtonText" property of the schedule control.

ProviderAppearance,
ProviderBackColor,
ProviderFont,
ProviderForeColor
	`Property ProviderAppearance() As AppearanceConstants`
	`Property ProviderBackcolor() As OLE_COLOR`
	`Property ProviderFont() As StdFont`
	`Property ProviderForeColor() As OLE_COLOR`
	
	These properties will determine the display of the provider section of a schedule. If the ShowProviderAvailableTime or ShowProviderScheduledTime properties are set then the Providers collection will be displayed in the left margin. The font and colors of this section are controlled by these properties. They allow you to set the back and fore colors as well set a 3D or flat appearance. The font may be set as well to customize the look and feel of this section.

ProviderBarWidth	`Property ProviderBarWidth() As Long` This property defines the width in pixels of each Provider bar on the left margin of the schedule. It also defines the width of the Provider bar on each appointment's left side.
RightToLeft	`Property RightToLeft() As Boolean`

	This property determines if the control will display the text right-to-left. In Arabic and Hebrew the font will display in this manner if this property is set to true. In western languages this property will not make any difference in the display because the font must support this style of writing for the display to be correctly.
RoomHeaderAlign	`Property RoomHeaderAlign() As AlignmentConstants` This property determines the alignment of the header text for rooms in the top of left margin.
RowHeight	`Property RowHeight() As Long` This determines the height of the rows. There is a minimum value determined by the specified font. This value is specified in pixels.

ScheduleIncrement	`Property ScheduleIncrement() As ScheduleIncrementConstants`
	
	This is the increment in minutes that determine how an hour is broken down. The valid values are listed below and correspond to evenly divisible factors of an hour. Any invalid values are mapped to sch30Minute. The default break for an hour is 30 minutes.
	
	`Public Enum ScheduleIncrementConstants`
	`  sch5Minute = 5`
	`  sch10Minute = 10`
	`  sch15Minute = 15`
	`  sch20Minute = 20`
	`  sch30Minute = 30`
	`  sch60Minute = 60`
	`End Enum`

SelectedColor	`Property SelectedColor() As OLE_COLOR` There is a selected area on a schedule in all "normal" Viewmodes. When the user uses the arrow keys to navigate, he is moving the selected area. This area is colored with the SelectedColor property value.
SelectedItem	`Property SelectedItem() As CScheduleEl`

	This is the currently selected ScheduleItem, the item under the mousepointer. The Item is highlighted with a colored bar above and below it. If no Item is selected then this value is "Nothing".
SelectorStartDate	`Property SelectorStartDate() As Date` This property determines the starting date of the highlighted area. An area is defined as a starting date, room, and /or time and a number of cells that are highlighted from this point with the selector. This property has no meaning in the List and Month viewmodes.
SelectorStartRoom	`Property SelectorStartRoom() As Long` This property determines the starting room (if applicable) of the highlighted area. An area is defined as a starting date, room, and /or time and a number of cells that are highlighted from this point with the selector. This property has no meaning in the List, Month, or any viewmodes without rooms.
SelectorStartTime	`Property SelectorStartTime() As Date` The AllowSelector property toggles a selector that highlights an area of a schedule. The user may use the keyboard to navigate and highlight an area of the schedule on which to create an appointment upon pressing the <ENTER> key. This property determines the starting time of the highlighted area. An area is defined as a starting date, room, and /or time and a number of cells that are highlighted from this point with the selector. This property has no meaning in the List and Month viewmodes.
SelectorCellLength	`Property SelectorCellLength() As Long` The AllowSelector property toggles a selector that highlights an area of a schedule. The user may use the keyboard to navigate and highlight an area of the schedule on which to create an appointment upon pressing the <ENTER> key. This property determines the number of cells that are highlighted with the selector. In viewmodes with no time, this property value will always be one. This property has no meaning in the List and

	Month viewmodes.
SelTab	`Property SelTab() As Long` This property is only used if the either the ShowProviderAvailableTime property or the ShowProviderScheduledTime property is set. If either one is set, two tabs will appear at the top of the schedule. The top tab is 1 and the bottom is 2. When these tabs are not visible SelTab = 0. The tabs determine if the schedule is being viewed in Category or Provider mode. A schedule can have so much information associated with it that it can not be displayed on one screen. Therefore, the Schedule control has two modes to view a schedule. Note: This property only applies if the time is displayed in the top margin.
ShowProviderAvailableTime	`Property ShowProviderAvailableTime() As Boolean` This property allows for the display of timeslots available for a provider. Each provider in the Providers collection has an AvailableTimes collection that defines what time slots the provider is available. This could be the times that he/she works or is in the office. Each provider has his/her own color defined by the Provider element's color property. You can also display the times the Provider is scheduled to work as defined by the ScheduleItems that are assigned to him/her. The ShowProviderScheduledTime property determines if this information is displayed. Note: This property only applies if the time is displayed in the top margin.
ShowProviderScheduledTime	`Property ShowProviderScheduledTime() As Boolean` This property determines if the time allotments for the providers, as defined by the ScheduleItems to which they are assigned, are displayed on the left of the screen. Each ScheduleItem has a Provider property. If this property is set to a valid provider then this provider is assigned to it. If the provider bars are displayed then the

	Provider bar will be filled in with the Provider's associated color for the length of the ScheduleItem. This allows for quick viewing of how much time a particular provider is needed for a schedule. Note: This property only applies if the time is displayed in the top margin.
StartTime	`Property StartTime() As Date` This property determines the first displayed time for the schedule. The day begins at this point and progresses down for DayLength hours. Note: The DayLength can not cause the schedule to overlap into the next day. This causes and error. If the StartTime is 8:00 PM, then the maximum value for DayLength is 4, since more than 4 hours would be the next day.
TabOrder	`Property TabOrder() As TabOrderConstants` When the user presses the <Tab> key, an appointment is highlighted if at least one exists. The tab cycles between appointments. If there are two appointments on a schedule, pressing the <Tab> key alternates the highlighting of each. There are two ways to cycle through the appointments: absolute position, and ordered position. The absolute position is the order in which the items were added to the ScheduleItems collection. The order position selection will move though the appointments in the order they appear on the schedule. For example, all the appointments from top to bottom and then left to right would be highlighted in order no matter their absolute position in the collection. This would allow the user to tab until he found the appointment for which he was looking without jumping all over the schedule. If the highlighted appointment is not in the viewable area, the schedule is scrolled so that the appointment is visible.
TabVisible	`Property TabVisible() As Boolean` This property toggles the tab strip in the top, left margin on/off. When this property is set to False no tab strip will be visible. When True, the tabs will be visible, if time is

	displayed in the top margin. The tabs allow the user to toggle between Category and Provider mode. This will display the Category and Provider bars in the left margin respectively.
TextDisplay	`Property TextDisplay() As TextDisplayConstants` An appointment may display its DisplayText or Subject property in its display area. The TextDisplay property determines which one is used.

TimeAppearance, TimeBackColor, TimeFont, TimeForeColor	`Property TimeAppearance() As AppearanceConstants`
	`Property TimeBackcolor() As OLE_COLOR`
	`Property TimeFont() As StdFont`
	`Property TimeForeColor() As OLE_COLOR`
	
	These properties will determine the display of the time headers of a schedule. The font and colors of this section are controlled by these properties. They allow you to set the back and fore colors as well set a 3D or flat appearance. The font may be set as well to customize the look and feel of this section.

TimeFormat	`Property TimeFormat() As ScheduleTimeFormats`
	
	This property determines if the control displays times in 12-hour or 24-hour mode. If in 12-hour mode, then the printed times have the AM/PM marker (ex. 3:00 PM). If this property is set to 24-hour mode then the same time would be displayed 15:00 with no AM/PM marker.
	
	ScheduleTimeFormats Constants:
	`Public Enum ScheduleTimeFormats`
	`  [12Hour] = 1`
	`  [24Hour] = 2`
	`End Enum`

TimeLayout	`Property TimeLayout As TimeLayoutConstants` This property determines the mode in which the time margin is displayed. If the setting is "tlHour", the time margin has a double-size, large font hour display and a normal size minute display. When the property value is

	"tiMinute", each time increment is not broken-up but is displayed as a full time i.e. "9:00 AM".
TimeSelectorColor	`Property TimeSelectorColor() As OLE_COLOR` This property specifies the color of the time selector. This is an arrow the points to a particular time on the schedule.

ToolTipBackColor, ToolTipForeColor	`Property ToolTipTextBackColor() As OLE_COLOR`
	`Property ToolTipTextForeColor() As OLE_COLOR`
	
	The schedule has bubble tips (tool tips) that appear when the mouse moves over certain objects on the schedule canvas. These properties determine the back and fore colors of the pop-up window that displays these bubble tips.

ToolTipTextDisplay	`Property ToolTipTextDisplay() As TextDisplayConstants` An appointment may display its DisplayText or Subject property as its ToolTipText. The ToolTipTextDisplay property determines which one is used.
UseOtherCategories	`Property UseOtherCategories() As Boolean` This property determines if the Categories collection on the ScheduleItems object is used. There is Categories collection on each schedule control. This is used to draw the colored category bar on each ScheduleItem element. It is also displayed in the left margin if CategoryBar is true. The ScheduleItems object also has a Categories collection to which each ScheduleItem element can point. Each ScheduleItem element has a static Categories collection that has elements that can point to the Categories collection of the ScheduleItems collection.
UseUnicode	`Property UseUnicode() As Boolean` If need be you may specify that Unicode be used instead of standard ASCII text. Some languages, like Chinese, will not display properly on the schedule. Only half of the characters are displayed. This is because the set of

	characters that this language uses consists of 2 bytes per character. The printing code needs to know that this is a double-byte language. When the UseUniCode property is set, all text is assumed to be double byte and is displayed as such. Keep in mind that this only works on Windows NT and Windows 2000 machines. On Windows 95/98 machines the double byte text will still not completely be displayed.
ViewMode	`Property ViewMode() As ViewModeConstants` This property determines the mode in which the schedule displays. There are three options: Normal, Calendar, and List. In normal mode the schedule displays days, room, or both with time across either the top or left margin. In Calendar mode, the schedule does not display time at all. It displays a month at a time. Normal appointments and events are all shown together. This view is useful to summarize a month at a time. The List view shows a day at a time but with the times displayed in an appointment book format.
WarningMessage	`Property WarningMessage() As String` This property determines the message displayed in the warning header at the top of the control. The visibility of the header is determined by the AllowWarning property value.
WeekMarginCaption	`Property WeekMarginCaption() As String` When the AllowWeekMargin property is set, week numbers are displayed on the schedule. To inform the user what these numbers mean use the WeekMarginCaption property. It is displayed as a header for week numbers.
Zoom	`Property Zoom() As Long` This property will set the zoom factor of the schedule. The normal zoom value is 100%. This number can be a value between 25 and 200 in increments of 25 (25, 20, 75, etc...). The schedule window will be scaled to this factor. All schedule functionality will remain in tact. The only difference will be the look of the schedule.

Note:
The font will be scaled down if possible. If the font has a smallest value like "MS Sans Serif" (8.25) and the Zoom is set to less than 100%, no text will be displayed. Otherwise, the font will be scaled smaller as necessary, even if it cannot be read.

Methods

CloneItem	`Function CloneItem(OldItem As CScheduleEl, NewName As String, NewDate As Date, NewRoom As String, NewStartTime As Date) As CScheduleEl` This method returns a ScheduleItem that is an exact copy of the one provided with the "OldItem" parameter. Given a new Name, Date, Room and Time this function adds a copy to the ScheduleItems collection and returns a pointer to the new object.
DeleteItem	`Function DeleteItem(ByVal Index As Long) As Boolean` DeleteItem removes an item from the ScheduleItems collection given a valid index. The return value is Boolean, true if the item was removed, false otherwise.
EditItem	`Function EditItem(ByVal Index As Long) As Boolean` The EditItem method brings up the edit window for the specified item index. This screen allows the user to modify the contents of a scheduled item or if AllowEdit is false to view the data. Also, keep in mind that the ScheduleItem has a read-only property as well. If the item being edited is read-only then the dialog will be read-only as well. The return value is Boolean, true if the item was modified, false otherwise. If the user double-clicks an item then the following behavior follows: If AllowEdit is true then the Edit screen is displayed. If AllowEdit is false then the read-only property screen is displayed.
ExportHTML	`Function ExportHTML(oHTMLParameters As CHTMLParameters) As Boolean` An HTML page of a schedule can be created with this method. The parameter object has the following values:

	PageTitle - This is the title given to the page. If generating a full HTML page this is also visible at the top of the page. HTMLHeader - This is any addition HTML that you wish to appear above the table. HTMLFooter - This is any addition HTML that you wish to appear below the table. TableOnly - This will specify if you wish to have an entire page with page headers and footer and page tags or if you wish only the table definition to appear in the file. FileName - This is the filename that will receive the generated HTML. Overwrite - If the Filename exists and Overwrite is true then the old file will be overwritten by the newly generated HTML. If the Filename exists and the Overwrite is false then NO HTML will be generated at all.
ExportXML	`Function ExportXML(oXMLParameters As CXMLParameters) As Boolean` Although you will probably loop through the collections and store the information in a database, there is an alternative. The schedule has its own file format that can be used instead of a database. If you are designing a "light" program and do not wish to distribute a full-blown database like MS-Access, you can use the ImportXML and ExportXML methods to save information without a database. The parameter object allows you to control what will be saved to the file. Seven collections can be saved from this method. The ScheduleItems collection is always saved, which means there are 6 collections that can be conditionally saved. The parameter object has the following properties: Filename - This is the full path of the file to use to which to save the data. Overwrite - If the specified file already exists, this flag will determines if the file is overwritten or the save is canceled. UseCategories - determines the loading/saving state of this collection UseNoDropAreas - determines the loading/saving state

	of this collection UseProviders - determines the loading/saving state of this collection UseRooms - determines the loading/saving state of this collection
GetDayFromCor	`Function GetDayFromCor(X As Long, Y As Long, [bInBounds As Boolean = False]) As Date` Given X/Y coordinates in pixels, this property returns the date that is associated with the position. The InBounds parameter may be used to specify whether you wish the specified coordinates to be inside the client area. When true, the method will only return a valid value if the coordinates are inside the client area where the appointments are displayed. Clicking in the margins will not return a valid value in this case.
GetFirstVisibleDay	`Function GetFirstVisibleDay() As Date` This method returns the first day that is visible in the schedule window. Note: This property has no meaning if the ViewMode property is set to exclude days.
GetFirstVisibleRoom	`Function GetFirstVisibleRoom() As Long` This method returns the first room that is visible in the schedule window. Note: This property has no meaning if the ViewMode property is set to exclude rooms.
GetFirstVisibleTime	`GetFirstVisibleTime() As Date` This method returns the first time that is visible in the schedule window.
GetNextFreeSlot	`Function GetNextFreeSlot(pdtStartDate As Date, plStartRoom As Long, pdtStartTime As Date, plItemLength As Long, [psIgnoreIndexes As String]) As CScheduleEl`

	This method, given a number of parameters, will return the next available slot into which the appointment will fit. Define the parameters start date, start room, start time, and appointment length and the method will determine the next fit on the schedule. The return is an appointment object that is NOT in the ScheduleItems collections. The object is returned populated with its Date, Room, and StartTime. No other properties are set. You can use this information to add another appointment to the ScheduleItems collection with its Add method. You may specify the indexes to ignore when searching. These indexes include appointments that may conflict with the specified appointment space but you do not care about them. The ignore indexes are numbers separated by a space, comma, colon, or semicolon. Note: The time resolution is determined by the ScheduleIncrement property. In other words if the ScheduleIncrement is 15 minutes and you are testing an appointment of 20 minutes then 2-15 minute slots will be needed.
GetRoomFromCor	`Function GetRoomFromCor(X As Long, Y As Long, [bInBounds As Boolean = False]) As Long` Given X/Y coordinates in pixels, this property returns the room that is associated with the position. The InBounds parameter may be used to specify whether you wish the specified coordinates to be inside the client area. When true, the method will only return a valid value if the coordinates are inside the client area where the appointments are displayed. Clicking in the margins will not return a valid value in this case.
GetTimeFromCor	`Function GetTimeFromCor(X As Long, Y As Long, [bInBounds As Boolean = False]) As Date` Given X/Y coordinates in pixels, this property returns the time that is associated with the position. The InBounds parameter may be used to specify whether you wish the specified coordinates to be inside the client area. When true, the method will only return a valid value if the coordinates are inside the client area

	where the appointments are displayed. Clicking in the margins will not return a valid value in this case.
GetVisibleDayCount	`Function GetVisibleDayCount() As Long` This function returns the number of days that are visible in the schedule window.
GetVisibleRoomCount	`Function GetVisibleRoomCount() As Long` This method returns the number of rooms visible in the schedule window.
GetVisibleRowCount	`Function GetVisibleRowCount() As Long` This method returns the number of rows visible in the schedule window.
GetVisibleTimeCount	`Function GetVisibleTimeCount() As Long` This method returns the total number of time intervals visible on the schedule window. This is defined by the StartTime, DayLength, and ScheduleIncrement .
GetWeekNum	`Function GetWeekNum(dtDate As Date, [lFirstDayOfWeek As VbDayOfWeek])` This method returns the week number of the specified date. Week numbers are normally used in some countries and this functionality is provided for developer convenience.
GoPrint	`Function GoPrint(vStartDateRoom, vEndDateRoom, [vStartTimeRoom], [vEndTimeRoom], [oPrinterParamters As CPrinterParameter]) As Boolean` This method prints a specified part of a schedule. Given the start/end date and rooms and the start/end times, this defined section is sent to the printer. This method can print an arbitrarily large schedule, but must have the needed hard drive temp space available. Each printed page takes about a Mb of hard drive space. After the print is completed, this temporary area is released.

	The " vStartDateRoom " and " vEndDateRoom " parameters are the start and end dates or rooms. This parameter will be rooms if ViewMode property is a setting that excludes days, otherwise it is dates. An error is raised if invalid data is specified. The "PrinterDeviceName" parameter is the printer name. This must be a valid printer installed on the system. If the printer does not exist then an error is raised. The "lOrientation" parameter allows you to print Portrait or Landscape. The constants for this are as follows: VbPRORPortrait = 1 vbPRORLandscape = 2
HitTest	`Function HitTest(X As Long, Y As Long) As CScheduleEl` This function takes a set of coordinates and returns the associated appointment under the coordinates. If there is no appointment at this position, “Nothing” is returned.
ImportXML	`Function ImportXML(oXMLParameters As CXMLParameters) As Boolean` Although you will probably loop through the collections and store the information in a database, there is an alternative. The schedule has its own file format that can be used instead of a database. If you are designing a "light" program and do not wish to distribute a full-blown database like MS-Access, you can use the ImportXML and ExportXML methods to save information without a database. The parameter object allows you to control what will be loaded from the file. There are 7 collections that can be loaded with this method. The ScheduleItems collection is always saved, which means there are 6 collections that can be conditionally saved. The parameter object has the following properties: Filename - This is the full path of the file to use from which to load the data. Overwrite - <NOT USED>

	UseCategories - determines the loading/saving state of this collection UseNoDropAreas - determines the loading/saving state of this collection UseProviders - determines the loading/saving state of this collection UseRooms - determines the loading/saving state of this collection
IsDayVisible	`Function IsDayVisible(pdtDate As Date) As Boolean` Given a date, this method will return a Boolean value indicating if that date is in the viewable window. A schedule may be much larger than the viewing window. This method can be used in conjunction with the ShowDay method to check if a particular date is visible and if not bring it into the viewing window.
IsEnabledAreaByValues	`Function IsEnabledAreaByValues(ByVal dtDate As Date, ByVal lRoom As Long, ByVal dtTime As Date, ByVal plLength As Long, ByVal pbCheckTimeOnly As Boolean) As Boolean` This method will return whether an area is enabled. After setting up the NoDropAreas collection, you may need to know if an appointment may be placed at a specific position. This method will take date, room, time, and length information and return True or False whether this position is blocked. No error occurs if an appointment is inserted inside of a NoDropZone through code, but if the user may become confused if he sees appointments in blocked areas.
IsRoomVisible	`Function IsRoomVisible(pvRoom) As Boolean` Given a room, this method will return a Boolean value indicating if that room is in the viewable window. A schedule may be much larger than the viewing window. This method can be used in conjunction with the ShowRoom method to check if a particular room is visible and if not bring it into the viewing window.
IsTimeVisible	`Function IsTimeVisible(pdtTime As Date)`

	`As Boolean` Given a time, this method will return a Boolean value indicating if that time is in the viewable window. A schedule may be much larger than the viewing window. This method can be used in conjunction with the ShowTime method to check if a particular time is visible and if not bring it into the viewing window.
MoveSeries	`Function MoveSeries(sGroupId As String, [lIncDates As Long], [lIncRooms As Long], [lIncTimes As Long]) As Boolean` This method will move an entire series of appointments at one time. You may specify a number of days, rooms, or minutes or move the appointment series. If you wish to move the series back in time, use a negative increment. For example, to move series 2 days back into the past call the method with the "lIncDates" parameter set to -2. The dates are measured in days and the time in minutes. If there method is successful it will return true, otherwise it returns false.
SetMinMaxDate	`Sub SetMinMaxDate(ByVal dtMinDate As Date, ByVal dtMaxDate As Date)` This method let you set the Min-Max pairs for Date without encountering errors. You can set the MinDate-MaxDate pairs at the same time and not get in intermediate error. Example: If the MinDate and MaxDate values are set to 1/1/78 and 2/1/79 respectively and you want to change the MinDate and MaxDate to 6/1/78 and 7/1/78. You would get an error when you set the MinDate to 6/1/78. Because at this time the MaxDate is 2/1/78, the new value of 6/1/78 for MinDate would make the MinDate greater than the MaxDate. This is an error. With the SetMinMaxDate method you could set both MinDate and MaxDate at the same time and avoid this error. Note: These properties (MinDate / MaxDate) have no meaning if the ViewMode property is set to exclude days.

ShowDay	`Function ShowRoom(ByVal Room) As Boolean` If the parameter day is in the valid range to (MinDate..MaxDate), then the schedule scrolls to the specified day. This allows any day to be brought into the display window. Note: This property has no meaning if the ViewMode property is set to exclude days.
ShowItem	`Function ShowItem(Index As Long) As Boolean` Given an index into the ScheduleItems collection on the Schedule, this method makes the item visible on the screen. It scrolls to the date and time of the item. This allows any ScheduleItem to be brought into the display window.
ShowRoom	`Function ShowDay(ByVal NewDate As Date) As Boolean` If the parameter date is in the valid range MinDate to MaxDate, then the schedule scrolls to the specified day. This allows any day to be brought into the display window.
ShowTime	`Function ShowTime(NewTime As Date) As Boolean` If the parameter time is in the valid range StartTime to StartTime + DayLength, then the schedule scrolls to the specified time. This allows any valid time to be brought into the display window.

Events

AfterAdd	`Event AfterAdd(NewIndex As Long)` AfterAdd is raised after a ScheduleItem has been added. A user can add an appointment by double-clicking on the background or in code. The parameter is the index of the new item. AllowAdd dictates this functionality.
AfterColumnResize	`Event AfterColumnResize(NewWidth As Long)` This event is raised after a user resizes the columns. If the AllowColumnResizing property is set, then the user can use the mouse to grab the edge of a column and resize it. All columns are the same width, so resizing one applies to all columns. The "NewWidth" parameter is the new ColumnWidth property.
AfterCopy	`Event AfterCopy(ScheduleItem As CScheduleEl)` AfterCopy is raised after an item is has been copies to a new position on the same schedule screen or a new screen. If a ScheduleItem is being move within the same window, there is no confusion as to where the event is raised, but if you drag the appointment to another schedule window, it is important to remember that this event is raised on the source window. Note: The user copies an appointment by dragging it with the <CTRL> key pressed. If this key is not pressed then it is an appointment move.
AfterDelete	`Event AfterDelete(LastIndex As Long)` AfterDelete is raised after an item is removed with the DeleteItem method. The user can do this by pressing the <Delete> key when highlighted or in code. AllowDelete dictates this functionality.

AfterDragFromFile	`Event AfterDragFromFile(ScheduleItem As CScheduleEl)` This event is raised after the user has dragged a schedule file onto the schedule window. The AllowDragToFile property allows the user to drag an appointment and drop it outside of the schedule to create a file with an appointment's properties. The AllowDragFromFile allows the user drag that previously saved file and drop it on a schedule to recreate the saved appointment. This event is raised after that operation has completed.
AfterEdit	`Event AfterEdit(Index As Long)` AfterEdit is raised after an item is modified with the EditItem method. This method can be called from code or by double-clicking an appointment. AllowEdit dictates this functionality.
AfterEditNotes	`Event AfterEditNotes(Index As Long, Notes As String, Cancel As Boolean)` When the ViewMode of the schedule is List, the user may edit the notes associated with an appointment in on the right displayed page of the schedule. After an edit has completed, this event is raised to inform that the operation was a success.
AfterEditText	`Event AfterEditText(Index As Long, DisplayText As String, Cancel As Boolean)` This event is raised after the user has edited the text of a ScheduleItem. The AllowEdit property has to be true for this to happen. When the user clicks on an appointment an edit box is displayed and the user can edit the appointment's displayed text.
AfterHorizontalScroll	`Event AfterHorizontalScroll()` This event is raised after the user moves the horizontal scroll bar. It also may be raised when calling the ShowItem, ShowDay, or ShowRoom methods, since these methods will scroll the screen as well.

AfterItemResize	`Event AfterItemResize(Index As Integer)` This event is raised after the user resizes a SelectedItem. When the mouse is moved over a ScheduleItem, it becomes highlighted. The SelectedItem has two bars on the top and bottom, which can be dragged to resize an appointment. This is the event raised after each resize. The parameter is the item's index in the ScheduleItems collection.
AfterMove	`Event AfterMove(ScheduleItem As CScheduleEl)` AfterMove is raised after an item is has been moved to a new position on the same schedule screen or a new screen. If a ScheduleItem is being move within the same window, there is no confusion as to where the event is raised, but if you drag the appointment to another schedule window, it is important to remember that this event is raised on the source window. Note: The user copies an appointment by dragging it with the <CTRL> key pressed. If this key is not pressed then it is an appointment move.
AfterRowResize	`Event AfterRowResize(NewHeight As Long)` This event is raised after a user resizes the rows. If the AllowRowResizing property is set, then the user can use the mouse to grab the edge of a row and resize it. All rows are the same height, so resizing one applies to all rows. The "NewHeight" parameter is the new RowHeight property.
AfterVerticalScroll	`Event AfterVerticalScroll()` This event is raised after the user moves the vertical scroll bar. It also may be raised when calling the ShowItem, ShowDay, or ShowRoom methods, since these methods will scroll the screen as well.

BackGroundClick	`Event BackGroundClick(Button As Integer, Shift As Integer, dtDate As Date, lRoom As Long, dtTime As Date)` The event is raised when the user clicks on the schedule background where a scheduled item is not present. This background also does not include the horizontal or vertical headers. The headers have their own click events, DayHeaderClick and TimeMarginClick respectively. Note: The dtDate property has no meaning if the ViewMode property is set to exclude days. In addition, the lRoom property has no meaning if the ViewMode property is set to exclude rooms.
BeforeAdd	`Event BeforeAdd(Cancel As Boolean)` BeforeAdd is raised before a schedule item is Added. The user holding the Ctrl-key and dragging the mouse the length of a proposed schedule item or in code performs this. In addition an add can be initiated by double-clicking the background. The Cancel parameter gives you the chance to cancel the Add before it is actually performed. AllowAdd dictates this functionality.
BeforeColumnResize	`Event BeforeColumnResize(Cancel As Boolean)` This event is raised before a user resizes the columns. If the AllowColumnResizing property is set, then the user can use the mouse to grab the edge of a column and resize it. All columns are the same width, so resizing one applies to all columns. The "Cancel" parameter allows you cancel this operation before it gets started.

BeforeCopy

```
Event BeforeCopy(OldAppt As
CScheduleEl, OldApptIndex As Long,
OldMode As ViewModeConstants, NewDate
As Date, NewRoom As String, NewTime As
Date, NewMode As ViewModeConstants,
DoPrompt As Boolean, Cancel As Boolean)
```

BeforeCopy is raised before the user drags a ScheduleItem to a new location. This happens after the user drops the item on its new location. The event provides the appointment and its index in the ScheduleItems collection. The NewDate, NewRoom, and NewTime define the position of the new appointment. Also there are two "By Reference" parameters that allow you to send information back to the schedule. "DoPrompt" allows you to display your own confirmation by canceling the default prompt. If you do this then the copy will automatically be confirmed, unless the "Cancel " parameter is set to False. "Cancel" provides a way for you to cancel a drag, if desired.

The OldMode and Newmode parameters determine the ViewMode setting of source schedule (OldMode) and the mode of the target schedule (NewMode). This allows you cancel a copy between windows if desired, for example if the modes do not match.

Note:
The "OldApptIndex" parameter will be zero if the user is dragging between schedule windows. This parameter is only valid when a drag is started and completed inside the same schedule window.

BeforeDelete	`Event BeforeDelete(Index As Long, Prompt As String, Cancel As Boolean)` BeforeDelete is raised before the removal of a scheduled item using the DeleteItem method. The DeleteItem method can be called by the user or in code. In addition it is called by pressing the <Delete> key when an appointment is highlighted. The Prompt parameter is the text to be displayed as a delete prompt to the user. If you do not wish for a prompt set this parameter to an empty string. The Boolean Cancel parameter gives you a chance to cancel the operation from code. If you set Cancel to true, the item will not be removed and the AfterDelete event will not be raised. AllowDelete dictates this functionality.
BeforeDrag	`Event BeforeDrag(Index As Long, InitialDragOperation As DragOperationConstants, Cancel As Boolean)` This event is raised before a copy or move. When the user starts to drag an appointment, this event is raised first and gives you a chance to cancel the drag by setting Cancel = true. The "Index" parameter is the index in the ScheduleItems collection of the appointment being dragged. The "InitialDragOperation" parameter is the drag operation that is taking place.
BeforeDragFromFile	`Event BeforeDragFromFile(ScheduleItem As CScheduleEl, Cancel As Boolean)` This event is raised after the user drops a schedule file onto the schedule window. Dragging an appointment and dropping it outside of the schedule create the file. This event is raised before the appointment is added by dragging that saved file back onto the schedule window.

BeforeDragTip	`Event BeforeDragTip(Prompt As String)` This event is raised before the drag tip text is displayed. It provides the option of changing the text with the “Prompt” parameter. This text is displayed when the user is dragging an appointment to another position.
BeforeEdit	`Event BeforeEdit(Index As Long, Cancel As Boolean)` BeforeEdit is raised before the edit of a scheduled item using the EditItem. The Boolean Cancel parameter gives you a chance to cancel the operation from code. If you set Cancel to true the item will not be edited and the AfterEdit event will not be raised. AllowEdit dictates this functionality.
BeforeEditNotes	`Event BeforeEditNotes(Index As Long, Cancel As Boolean)` When the ViewMode of the schedule is List, the user may edit the notes associated with an appointment in on the right displayed page of the schedule. Before an edit starts, this event is raised to give the developer a change to cancel the operation.
BeforeEditText	`Event BeforeEditText(Index As Long, Cancel As Boolean)` This event is raised before the user has a chance to edit an appointment's displayed text. The AllowEdit property has to be true for this to happen. When the user clicks on an appointment an edit box is displayed and the user can edit the appointment's displayed text.
BeforeFind	`Event BeforeFind(Cancel As Boolean)` This event allows you to cancel the find. If the AllowFind property is true, the user may press the CTRL-F key combination to display the "Find Appointments" screen. Before the screen is displayed this event is raised and you may cancel the find by setting the Cancel parameter to true.

BeforeHorizontalScroll	`Event BeforeHorizontalScroll(Cancel As Boolean)` This event is raised before the horizontal scrollbar value changes. When the user scrolls the horizontal scroll bar this event is raised to give you a chance to cancel the scroll. This event may also raised when the methods ShowDay, ShowRoom, or ShowTime are called. In addition, it may be called when an appointment is being dragged and causes and auto scroll when it get too close to one of the schedule edges.
BeforeItemResize	`Event BeforeItemResize(Index As Long, Cancel As Boolean)` When the mouse is moved over a ScheduleItem, it becomes highlighted. The SelectedItem has two bars on the top and bottom, which can be dragged to resize an appointment. This is the event raised before each resize. The parameter is the item's index in the ScheduleItems collection. This event is raised if the user tries to resize a ScheduleItem. You can cancel this operation with the "Cancel" parameter.

BeforeMove	`Event BeforeMove(oAppt As CScheduleEl, lApptIndex As Long, lMode As ViewModeConstants, NewDate As Date, NewRoom As String, NewTime As Date, NewMode As ViewModeConstants, DoPrompt As Boolean, Cancel As Boolean)` BeforeMove is raised before the user drags a ScheduleItem to a new location. This happens after the user drops the item on its new location. The event provides the appointment and its index in the ScheduleItems collection. The NewDate, NewRoom, and NewTime define the position of its new location. Also there are two "By Reference" parameters that allow you to send information back to the schedule. "DoPrompt" allows you to display your own confirmation by canceling the default prompt. If you do this then the move will automatically be confirmed, unless the "Cancel " parameter is set to False. "Cancel" provides a way for you to cancel a drag, if desired. The OldMode and Newmode parameters determine the ViewMode setting of source schedule (OldMode) and the mode of the target schedule (NewMode). This allows you cancel a copy between windows if desired, for example if the modes do not match. Note: The "lApptIndex" parameter will be zero if the user is dragging between schedule windows. This parameter is only valid when a drag is started and completed inside the same schedule window.
BeforeRowResize	`Event BeforeRowResize(Cancel As Boolean)` This event is raised before a user resizes the rows. If the AllowRowResizing property is set, then the user can use the mouse to grab the edge of a row and resize it. All rows are the same width, so resizing one applies to all rows. The "Cancel" parameter allows you cancel this operation before it gets started.

BeforeVerticalScroll	`Event BeforeVerticalScroll(Cancel As Boolean)` This event is raised before the vertical scrollbar value changes. When the user scrolls the vertical scroll bar this event is raised to give you a chance to cancel the scroll. This event may also raised when the methods ShowDay, ShowRoom, or ShowTime are called. In addition, it may be called when an appointment is being dragged and causes and auto scroll when it get too close to one of the schedule edges.
CategoryClick	`Event CategoryClick()` This event is raised when the mouse is clicked on the category area below the headers. There is no way to determine which category was clicked upon.
CategoryHeaderClick	`Event CategoryHeaderClick(Button As Integer, Shift As Integer)` This event is raised when the mouse is clicked on the category headers. There is no way to determine which category header was clicked upon.
DayHeaderClick	`Event DayHeaderClick(Button As Integer, Shift As Integer, X As Single, Y As Single)` The event is raised when the user clicks on the area defined as the header for a day. Note: This event has no meaning if the ViewMode property is set to exclude days.
DefaultDialogCancelClick	`Event DefaultDialogCancelClick(Index As Long)` This event is raised when the default dialog is used for editing and the "Cancel " button is pressed. If the "Ok" button of the dialog is pressed the "AfterEdit" event is fired because an appointment was edited.

DragDropScheduleItem

Contacts Control

```
Event DragDropScheduleItem(ScheduleItem
As CScheduleEl, DragOperation As
DragOperationConstants, X As Long, Y As
Long, Effect As Long, Cancel As
Boolean)
```

This event is raised when the user drops a ScheduleItem from a Schedule control onto a Contacts control. A copy of the ScheduleItem object is passed as a parameter.

Schedule Control

```
Event DragDropScheduleItem(ApptDate As
Date, ApptRoom As Long, ApptTime As
Date, ApptLength As Long, DragOperation
As DragOperationConstants, X As Long, Y
As Long, Effect As Long, Cancel As
Boolean)
```

This event is raised when an appointment is dropped on a schedule window. This can be another OLEDrag drag (i.e. from Windows Explorer) if the AllowInterWindowDrop and AllowOtherDrops properties are enabled. The date/room/time/length are sent in as parameters so that any conflict checking or other processing can be performed. The DragOperation parameter is the operation being performed which is none, copy, move, or add. If the AllowOtherDrops is True and the user drags a non-appointment item (like files or text or anything else) then this parameter will always be set to "Add" since the user is adding a new appointment with a drag. When dragging an appointment to another position, in the same window or across windows, this parameter is set to "Move". Alternately, if the <CTRL> key is pressed it is a "Copy". The Effect parameter is the standard OLE Effect.

TaskList Control

```
Event DragDropScheduleItem(ScheduleItem
As CScheduleEl, DragOperation As
DragOperationConstants, X As Long, Y As
Long, Effect As Long, Cancel As
Boolean)
```

This event is raised when the user drops a ScheduleItem from a Schedule control onto a TaskList. A copy of the ScheduleItem object is passed as a parameter. In this event you may add the ScheduleItem as a task if you wish. Since there are no predefined columns on a TaskList, an

DragEnterSchedule	`Event DragEnterSchedule(DragOperation As DragOperationConstants)` Event DragEnterSchedule(DragOperation As DragOperationConstants) This event is raised when a drag enters the a schedule window with a drag. If you are adding/moving/copying from outside the schedule, this event will let you know when the mouse moves over the schedule area for the first time when dragging. The DragOperation parameter is the effect that is used and can be one of three values Copy, Move, or Add.
DragExitSchedule	`Event DragExitSchedule()` This event is raised when a drag exit the a schedule window with a drag. If you are adding/moving/copying from outside the schedule, this event will let you know when the mouse moves over the schedule area for the last time before leaving the window during a drag operation.

DragOverScheduleItem

Contacts Control

```
Event
DragOverScheduleItem(DragOperation As
DragOperationConstants, X As Long, Y As
Long, Effect As Long)
```

When the user is dragging a ScheduleItems from a Schedule control over a Contacts control this event is raised. The coordinates of the mouse are specified as parameters.

Schedule Control

```
Event DragOverScheduleItem(ApptDate As
Date, ApptRoom As Long, ApptTime As
Date, ApptLength As Long, DragOperation
As DragOperationConstants, X As Long, Y
As Long, Effect As Long)
```

This event is raised when an appointment is being dragged over the schedule to be moved to another position. The date/room/time/length are sent in as parameters so that any conflict checking or other processing can be performed. These parameters correspond to the position under the current mouse position. The DragOperation parameter is the operation being performed which is none, copy, move, or add. If the AllowOtherDrops is True and the user drags a non-appointment item (like files or text or anything else) then this parameter will always be set to "Add" since the user is adding a new appointment with a drag. When dragging an appointment to another position, in the same window or across windows, this parameter is set to "Move". Alternately, if the <CTRL> key is pressed it is a "Copy". The Effect parameter is the standard OLE Effect.

TaskList Control

```
Event DragDropScheduleItem(ScheduleItem
As CScheduleEl, DragOperation As
DragOperationConstants, X As Long, Y As
Long, Effect As Long, Cancel As
Boolean)
```

When the user is dragging a ScheduleItems from a Schedule control over a TaskList control this event is raised. The coordinates of the mouse are specified as parameters.

HTMLDone	`Event HTMLDone()` An HTML page of a schedule can be created with the method ExportHTML. This event is raised when the HTML creation process is completed.
HTMLPercentDone	`Event HTMLPercentDone(lPercent As Long)` An HTML page of a schedule can be created with the method ExportHTML. This event is called periodically throughout the export to alert you of the progress made in exporting. The parameter is the percent progress made from 0 to 100.
HTMLStart	`Event HTMLStart()` An HTML page of a schedule can be created with the method ExportHTML. This event is called when the process of HTML creation begins.
MonthDateDblClick	`Event MonthDateDblClick(MonthDate As Date, Button As Integer, Shift As Integer)` This event only applies when the schedule's ViewMode is set to month view. If the user double clicks on the day portion of the day grid then this event is raised. Each day of the month has two parts, a top thin portion that displays the day number only and a bottom portion that lists each appointment for that day. This event is raised if the top portion is double clicked.
MonthDateHeaderClick	`Event MonthDateHeaderClick(MonthDate As Date, Button As Integer, Shift As Integer, X As Long, Y As Long)` This event only applies when the schedule's ViewMode is set to month view. If the user clicks on a day header this event is raised. The month has 7 day headers at the top of the screen, one for each day of the week.

MonthDateHeaderDblClick	`Event MonthDateHeaderDblClick(MonthDate As Date, Button As Integer, Shift As Integer)` This event only applies when the schedule's ViewMode is set to month view. If the user double clicks on a day header this event is raised. The month has 7 day headers at the top of the screen, one for each day of the week.
PrintCancel	`Event PrintCancel()` If the user cancels the printing by setting the Cancel parameter in any of the print events to true, the printing will stop. This event is called to notify you that the printing was stopped prematurely.
PrintDone	`Event PrintDone()` Using the GoPrint method, you can print any part of a schedule. When the printing process is complete this event is raised.
PrintPageDone	`Event PrintPageDone(Page As Long, PageX As Long, PageY As Long, Cancel As Boolean)` This event is raised when a page has completed being formatted for printing. The parameter "Page" is the absolute page number that has completed. The "PageX" and "PageY" define the horizontal [1..N] and vertical [1..M] page that has completed. A schedule is made up of a grid of pages [N x M] that are pieced together to form an entire schedule.
PrintProgress	`Event PrintProgress(Percent As Long, Cancel As Boolean)` Using the GoPrint method, you can print any part of a schedule. This event is raised continually to show the printing progress. The "Percent" parameter is the percent done 0..100. Use this event to build a wait progress dialog for the user to view during printing. The Cancel parameter can be set to true to cancel the printing.

PrintStart	`Event PrintStart(Cancel As Boolean)` This event is raised when the GoPrint method is invoked. You can stop the print before anything goes to the printer by setting the Cancel parameter to true.
PropertiesCustomButtonClick	`Event PropertiesCustomButtonClick()` The schedule has a default property window. You have the option of displaying a custom button on this window. When the user clicks on this button the "PropertiesCustomButtonClick" event is raised so that you may add additional functionality to this property window. If you have no use for an extra button on the default properties screen, just set the "PropertiesAllowCustomButton" property to false and it will not appear. You may set the text of the button using the "PropertiesCustomButtonText" property of the schedule control.
ProviderAvailableClick	`Event ProviderAvailableClick(oProvider As CProviderEl, Button As Integer, Shift As Integer)` This event is raised when the available times of a provider is clicked with the mouse. The ShowProviderAvailableTime property must be set for the times to be displayed.
ProviderAvailableHeaderClick	`Event ProviderAvailableHeaderClick(Button As Integer, Shift As Integer)` This event is raised when the header of the available times of a provider is clicked with the mouse. The header is the text that heads the column. The ShowProviderAvailableTime property must be set for the times to be displayed.

ProviderScheduledClick	`Event ProviderScheduledClick(oProvider As CProviderEl, Button As Integer, Shift As Integer)` This event is raised when the scheduled times of a provider is clicked with the mouse. The ShowProviderScheduledTime property must be set for the times to be displayed.
ProviderScheduledHeaderClick	`Event ProviderScheduledHeaderClick(Button As Integer, Shift As Integer)` This event is raised when the header of the scheduled times of a provider is clicked with the mouse. The header is the text that heads the column. The ShowProviderScheduledTime property must be set for the times to be displayed.
RecurrenceFailed	`Event RecurrenceFailed()` This event is raised when the user presses the Cancel button on the default Recurrence screen. This screen is displayed by pressing the Recurrence button on the toolbar of the ScheduleProperties control.
RecurrenceClick	`Event RecurrenceClick(Cancel As Boolean)` This event is raised from the ScheduleProperties control when the Recurrence button on the toolbar is pressed. This action will display the default recurrence screen, allowing the user to setup a recurrence pattern. You may cancel the display of the recurrence screen by setting the Cancel parameter to true.

RoomHeaderClick	`Event RoomHeaderClick(RoomIndex As Long, RoomName As String, Button As Integer, Shift As Integer, X As Single, Y As Single)` This event is raised when the mouse is clicked on a Room Header. This is the area above each room with the room name. The parameter RoomIndex is the Room (1..N) which was clicked upon.
ScheduleItemClick	`Event ScheduleItemClick(ItemIndex As Long, Button As Integer, Shift As Integer)` The event is raised when the user clicks on a scheduled item. The index parameter is the index of the selected item in the ScheduleItems collection.
ScheduleItemDblClick	`Event ScheduleItemDblClick(ItemIndex As Long, Button As Integer, Shift As Integer)` The event is raised when the user double-clicks on a scheduled item. The index parameter is the index of the selected item in the ScheduleItems collection.
SelectedDateChanged	`Event SelectedDateChanged(NewDate As Date)` This event is raised after the schedule window's date has changed. If the user. This event is only raised if the schedule is in Month or List view mode. When the user moves to a new day or month in the Month view this event is raised. In List mode the user needs only to move to a different day by using the left or right arrow keys.

ScheduleItemStart	`Event ScheduleItemStart(Index As Integer)` This event is raised when it is time for a ScheduleItem to begin. Each Item has a StartTime property. On a Schedule, when the current day matches an Item's "StartDate" and the current time matches the Item's StartTime, then this event is raised with the "Index" parameter being this particular Item's index in the ScheduleItems collection. All rooms for a particular day are checked for a time match. This event allows you to build applications that trigger other processes at predetermined times.
SelectedItemChange	`Event SelectedItemChange()` This event is raised when the property SelectedItem is changed. This happens any time that the Highlight bars are drawn on an appointment. There are several ways this event may be raised. The developer may select an appointment by setting the SelectedItem property. The user may select an appointment by moving the mouse over an appointment. Alternately, the <Tab> key may be pressed to the move through the ScheduleItems collection while selecting each one in turn.

TabClick	`Event TabClick(TabIndex As Long, Button As Integer, Shift As Integer, X As Single, Y As Single)` This event is raised if the a Tab at the top of the screen is clicked. The tabs only appear if either the ShowProviderAvailableTime or howProviderScheduledTime properties are set. If either one is then two tabs will appear at the top of the schedule. The top tab is 1 and the bottom is 2. When these tabs are not visible SelTab = 0. The tabs determine if the schedule is being viewed in Category or Provider mode. A schedule can have so much information associated with it that it can not be displayed on one screen. So the Schedule control has two modes to view a schedule. Note: The tabs are displayed only applies if the time is displayed in the top margin.
TimeMarginClick	`Event TimeMarginClick(Button As Integer, Shift As Integer, X As Single, Y As Single)` The event is raised when the user clicks in the area that displays the time.
WhileItemResize	`Event WhileItemResize(Index As Long, NewStartTime As Date, NewLength As Long, Cancel As Boolean)` This event is called when the user is resizing an appointment. The NewStartTime and NewLength parameters are the new values of the SelectedItem that will be changed after this event. You may check for conflicts or do other processing and cancel the resize by setting the Cancel parameter to true if necessary.

ValidateAdd	`Event ValidateAdd(Index As Long)` This event is raised after the BeforeAdd event but before the AfterAdd event. It may be used to set information for appointments before the properties dialog is displayed.
ValidateAppointment	`Event ValidateAppointment(StartDate As Date, Room As Long, StartTime As Date, ApptLength As Long, DisplayText As String, Subject As String, Priority As Long, Alarm As Boolean, Category As String, Provider As String, GroupId As String, UniqueKey As Long, Cancel As Boolean)` This event is raised from the ScheduleProperties control just before a save is performed. It allows you to verify and change information about the appointment before this information is saved to the main schedule. You may cancel the save by setting the Cancel parameter to true.
ValidateDelete	`Event ValidateDelete(Index As Long)` This event is raised after the BeforeDelete event but before the AfterDelete event. It may be used to remove appointments from a database if necessary. The BeforeDelete event is raised to allow you to change the prompting text if necessary. You do not want to remove appointment from your database here, since the user may choose NOT to remove the appointment when prompted. After the user is prompt and he chooses to remove the selected appointment the ValidateDelete event is raised with the Index of the appointment in the ScheduleItems collection. You may not cancel the delete at this point. This event is raised so that clean-up code can be added. The item to be removed is still in the ScheduleItems collection at this point. When the AfterDelete event is raised the item is no longer in the collection.

ViewModeChange	`Event ViewModeChange(ViewMode As ViewModeConstants)` This event is raised when the value of the ViewMode property is changed.
WarningClick	`Event WarningClick()` This event is raised from the ScheduleProperties control when the mouse is clicked on the warning header at the top of the control.
XMLExportDone	`Event XMLExportDone()` This event is raised when an export to XML is completed after calling the ExportXML method.
XMLExportStart	`Event XMLExportStart()` This event is raised when calling the ExportXML method starts an export to XML.
XMLImportDone	`Event XMLImportDone()` This event is raised when an import to XML is completed after calling the ImportXML method.
XMLImportStart	`Event XMLImportStart()` This event is raised when an import to XML is started after calling the ImportXML method.