Understanding Telerik's Scheduler Recurrence

Published: 05 Feb 2009
By: Brian Mains

Brian Mains continues to explore the Telerik's Scheduler control.

Contents [hide]

The scheduler is a component that allows users to schedule appointments similar to Outlook. Although this functionality is similar to the Outlook windows application, this component is a web component that can be used in a web application. The recurrence concept can be a difficult one to master.

The scheduler works with Appointment objects that represent an appointment on the calendar, whether it's a meeting, reminder, or general appointment. These appointments can be setup as a recurring appointment, a means to duplicate the appointment on an hourly, daily, weekly, monthly, or even yearly basis. The process of recurring appointments can be a little bit of a mystery.

The scheduler control itself supports multiple approaches to binding data. The first approach is through a custom provider. The base scheduler provider class defines all the necessary method to add, update, and delete the data from the underlying data source. It's often the easy way to work against your data, especially if you have a layered architecture. The custom provider can simply instantiate your business or data layer, and execute the appropriate code. Below is a skeleton of a custom provider shell; I didn't implement anything because this is not the route I'm going to use for databinding.

Listing 1: A Custom Scheduler Provider

```
01. public class AspNetSchedulerProvider : SchedulerProviderBase
02. {
03.     public override void Delete(RadScheduler owner, Appointment
           appointmentToDelete)
04.     {
05.         throw new NotImplementedException();
06.     }
07.     public override System.Collections.Generic.IEnumerable<Appointment>
08.           GetAppointments(RadScheduler owner)
09.     {
10.         throw new NotImplementedException();
11.     }
12.     public override System.Collections.Generic.IEnumerable<ResourceType>
13.           GetResourceTypes(RadScheduler owner)
14.     {
15.         throw new NotImplementedException();
16.     }
17.     public override System.Collections.Generic.IEnumerable<Resource>
```

```
18.             GetResourcesByType(RadScheduler owner, string resourceType)
19.     {
20.         throw new NotImplementedException();
21.     }
22.     public override void Insert(RadScheduler owner, Appointment
           appointmentToInsert)
23.     {
24.         throw new NotImplementedException();
25.     }
26.     public override void Update(RadScheduler owner, Appointment
           appointmentToUpdate)
27.     {
28.         throw new NotImplementedException();
29.     }
30. }
```

The custom scheduler provider uses the Telerik Appointment object to pass along inserts, updates, or deletes. There are some other important objects to know about, when working with the Telerik framework, to handle setting up recurrence:

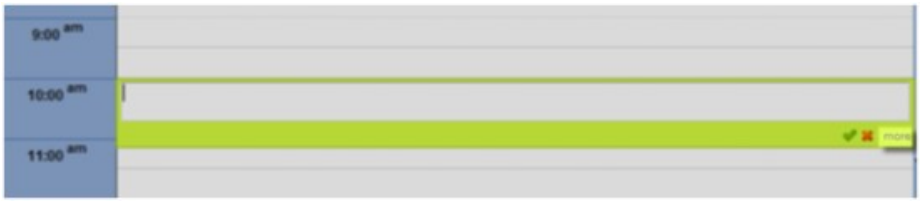
- RecurrenceRule - This is the rule that determines how an appointment recurs. This rule determines whether it's an hourly, daily, weekly, or other basis. The `ToString` method converts the recurrence rule into an equivalent text that can be stored in a database.
- RecurrenceRange - This object specifies the range that the recurring appointment lasts until. For instance, a user can say that an appointment's recurrence ends on "1/1/2010". This is a specific date; however, an alternative is to say that the appointment can recur for ten occurrences.
- RecurrencePattern - This is the pattern that the appointment recurs by, whether it's on an hourly, daily, etc. basis (frequency), the day(s) of the week to recur on (DayOfWeekMask), the number of times to separate recurrence appointments by (every 2 weeks, every 5 days, etc.,which is the Interval), and several more factors.

These objects affect how an appointment and is the basis for generating recurring appointments. The scheduler supports recurring on an hourly basis, a daily basis (every X number of days), a weekly basis (every Monday/Wednesday/Friday of every X week), a monthly basis (either the first Monday of every X months or the Xth day of every X months), or even a yearly basis.

Scheduler Execution

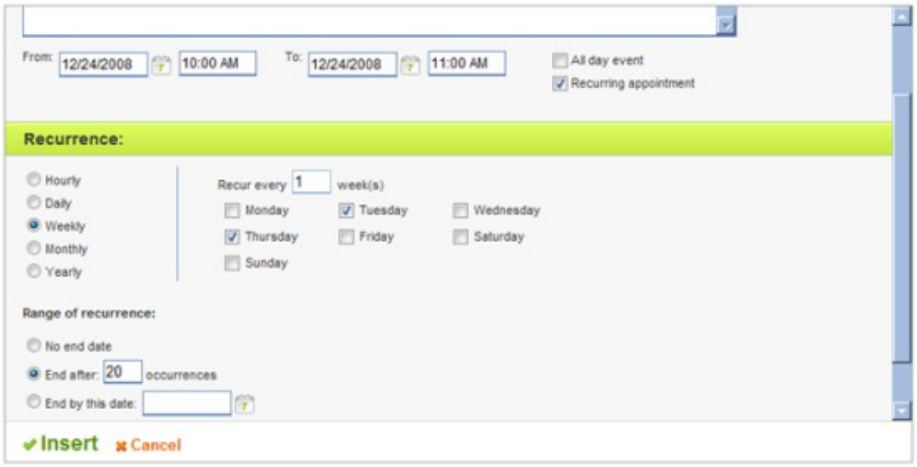
So now that we have an understanding of how the recurrence works, how does the scheduler make use of it? Let's first walk through the setting up of a recurring appointment, and how this recurring appointment can be altered in pretty unique ways. To create an appointment, double-click in the scheduler and create an appointment like in Figure 1.

Figure 1: Creating an appointment



Select the *more* link (highlighted and emphasized above). This generates the advanced view; in this view, the following appointment will recur on Tuesdays and Thursdays, every week, for twenty weeks.

Figure 2: Creating recurring appointment

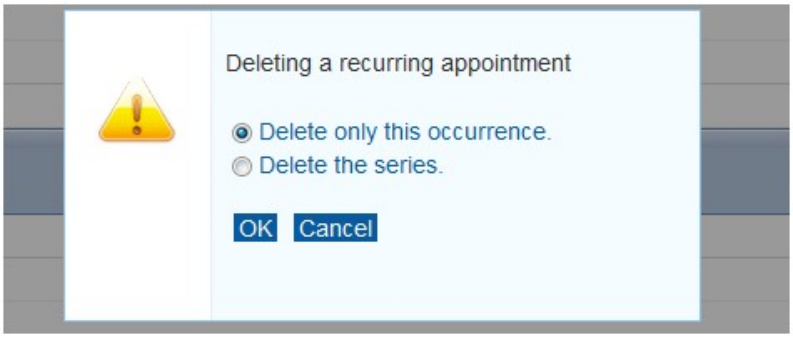


Clicking insert creates the appointment. Figure 3 shows the appointment created on Dec. 30th. Notice the red "X" in the upper right corner.

Figure 3: Selecting an appointment to delete

Clicking this button prompts the user to perform an action, as shown below. The entire appointment can be deleted (all occurrences), or only this single occurrence can be deleted. Deleting a single occurrence causes some special handling, which will be discussed later. Clicking on the red "X" pops up the deletion form. Clicking the OK button deletes a single occurrence as shown in Figure 4.

Figure 4: Deleting a single instance

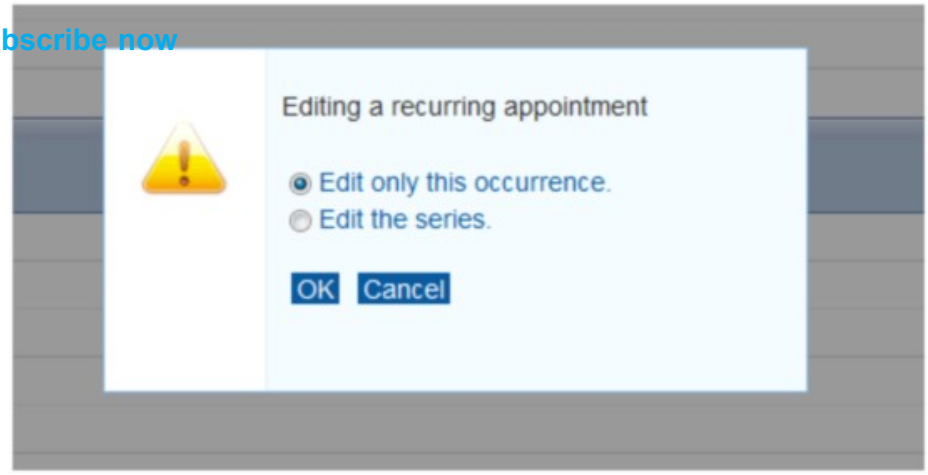


Deleting the entire series deletes any appointment created using the recurrence rules, but deleting a single occurrence leaves all of the other appointments in existence, but deletes just this single appointment only, as you can do in Outlook.

The next type of modification that can be made is by holding the mouse down while clicking on an appointment. It can be dragged (creating a transparent background effect) to a new time slot or even a new day. This action updates the appointment. If the appointment is a recurring appointment, only this single occurrence is deleted.

Figure 5: Updating an appointment by moving it

If the user didn't drag and drop the appointment, but rather double-clicked it and edit the information directly, the following prompt shows, as in Figure 6.



Editing the series changes all of the appointments in the recurrence, but selecting to edit only the current occurrence edits that one single appointment. So what logic does the scheduler use to track all of these changes? Here's a brief overview, and we'll look at a sample application I setup to track the various types of recurring appointments.

Storing Recurring Appointments

Recurring appointments are stored as one record in the underlying data system (database or XML file). The string used to store a recurring appointment looks like the following.

```
DTSTART:20081218T180000Z
DTEND:20081218T190000Z RRULE:FREQ=WEEKLY;COUNT=10;INTERVAL=5;BYDAY=TH
DTSTART:20081224T100000Z DTEND:20081224T110000Z
RRULE:FREQ=WEEKLY;COUNT=20;INTERVAL=1;BYDAY=TU,TH
EXDATE:20081230T100000Z,20090101T100000Z
```

This string gets stored in the database and is the sole basis for making up the recurrence capabilities. As the appointments get loaded into the system, this recurrence string is used to create a RecurrenceRule through the RecurrenceRule object's TryParse static method. This method essentially uses the RecurrencePattern and RecurrenceRange objects to generate the appointments (all of this occurs internally).

Notice the string has some extra attributes, like EXDATE. This is an important attribute for the next section; it stores dates that are considered exceptions.

Deleting an Appointment Occurrence

When any standalone appointment or the appointments for an entire series are deleted programmatically or using the red "X" in the upper-right corner, the scheduler's `AppointmentDelete` event fires, and any delete processing logic occurs (in the case where you bind using `DataSource` property, you have to call the delete on the backend). But for recurring appointments, deleting a single occurrence triggers an update and calls the `AppointmentUpdate` event.

Single appointment deletions, within a group of recurring appointments, are excluded in a special way; the recurrence rule stores a list of exception dates that should be excluded from view. This deleted appointment is now in that list, and excluded from view. This is a similar process for updating. The Scheduler handles these capabilities for you; you do not necessarily need to do anything about this, as all the information is available in the event argument.

Changing an Appointment Occurrence

In the two scenarios above, where the appointment's start/end date are edited either by using the drag/drop or edit form capabilities, what actually occurs is an insert and an update. The appointment that was edited has that date stored in the exceptions list, and a new appointment created to represent that current time.

If you think about it, editing the start dates where it doesn't fit within the occurrence isn't easy to store within a recurrence. The original date gets excluded from the list (by storing the date in the recurrence rule's list of exception dates). A new appointment is created with the modified information, while creating a link between the original (master record in a sense) and this new record. So now our recurring appointment has two records: the master and the newly modified appointment.

Handling Events

Do you really need to handle events in the scheduler if it does all of this? Yes, this is still necessary, because you may need to process data in a certain way, or append information not provided in the appointment form. There is a reason to tap into these events if you do not bind using the appointment's provider structure or a data source control. If you bind manually using the `DataSource` property, you may need to do extra work. Take a look at the following example:

Listing 2: Attaching to Scheduler Events

```
protected void rsAppointments_AppointmentDelete(object sender,
   SchedulerCancelEventArgs e)
{
    _manager.DeleteByKey((int)e.Appointment.ID);
    this.OutputList();
}
protected void rsAppointments_AppointmentInsert(object sender,
    SchedulerCancelEventArgs e)
{
    _manager.AddItem(new SchedulerItem
                        {
                            Key = _manager.GetNextKey(),
                            StartDate = e.Appointment.Start,
                            EndDate = e.Appointment.End,
                            Subject = e.Appointment.Subject,
                            RecurrenceRule =
                                            e.Appointment.RecurrenceRule,
                            RecurrenceParentKey =
                                            (int?)e.Appointment.RecurrenceParentID
                        });
    this.OutputList();
}
protected void rsAppointments_AppointmentUpdate(object sender,
    AppointmentUpdateEventArgs e)
{
    _manager.UpdateItem(new SchedulerItem
    {
        Key = (int)e.ModifiedAppointment.ID,
        StartDate = e.ModifiedAppointment.Start,
        EndDate = e.ModifiedAppointment.End,
        Subject = e.ModifiedAppointment.Subject,
        RecurrenceRule = e.ModifiedAppointment.RecurrenceRule,
        RecurrenceParentKey = (int?)e.ModifiedAppointment.RecurrenceParentID
    });
    this.OutputList();
}
```

A custom component peforms an insert or update, using its method to modify the underlying data store.

Conclusion

The scheduler is a capable control, which includes the ability to schedule recurring appointments. This article demystifies the recurring process and provides some more insight in handling recurring appointments and the underlying philosophy.

<< Previous Article

Continue reading and see our next or previous articles

Next Article >>

About Brian Mains

Brian Mains is an application developer consultant with Computer Aid Inc. He formerly worked with the Department of Public Welfare. In both places of business, he developed both windows and web applications, small and large, using the latest .NET technologies. In addition, he had spent many hou...

This author has published **73** articles on DotNetSlackers. View other articles or the complete profile **here**.

Other articles in this category

Code First Approach using Entity Framework 4.1, Inversion of Control, Unity Framework, Repository and Unit of Work Patterns, and MVC3 Razor View