Scheduling Tasks from VFP

Doug Hennig

Automatically executing a task on a regular schedule isn't something every application needs to do, but if you do need it, this month's article provides the code you need to do this from VFP.

Some applications need to execute certain tasks at regular intervals. For example, accounting systems often need to do day-end and month-end processing. An application may need to download and process a file every night. Rather than having a user manually start these processes, it would be handy to start them automatically. One way to do that is to leave the application running and have a timer fire at a regular interval. When it's the desired date and time, the timer starts the process. There are at least a couple of problems with that approach: if the application is always running, it (and possibly its data files) can't be backed up, and the application is using system resources even when it isn't doing anything.

A better approach is to use the built-in Windows Task Scheduler. Its job is to wait until a designated start time and then do something, such as running an application. Typically, an application designed to be run from the Windows Task Scheduler has a few characteristics:

- It accepts one or more command-line parameters telling it to perform a certain task. If no parameter is passed, such as when the user runs the application normally, it does the usual thing: displaying its user interface and waiting for user action. If the parameter is passed, which is what the scheduled task is configured to do, the application performs that task and then shuts down.
- If it's running in task mode (that is, the parameter was passed), it doesn't display any user interface. That means no login dialog (the application needs to log in automatically), no main form or _SCREEN visible, no error or warning messages displayed (they should be logged instead), and no prompts of any kind. The reason for this is that there's no guarantee any user is present to respond to any prompts or dialogs.

Although you can create scheduled tasks using the Windows Task Scheduler's command line interface, that's clumsy. A better approach is to use a COM interface. There are a couple of interfaces depending on which operating system you're using. I've created wrappers for these interfaces, making them easy to work with in VFP applications. The wrappers are contained in TaskScheduler.PRG, included in the download for this article.

TaskScheduler

TaskScheduler.PRG has three classes: TaskScheduler, the parent class for the other two classes and not intended to be used directly; XPTaskScheduler, the class to be used with Windows XP; and VistaTaskScheduler, the class to be used with Windows Vista and later. The reason for the operating system-specific classes is that Windows XP and earlier uses Task Scheduler 1.0, which is difficult to use with VFP. In fact, that's why the download for this article includes a DLL file, TaskScheduler.DLL, that's only used by XPTaskScheduler. This freeware DLL, written by Mark Pryor, contains a COM class named Scheduler.SchAgent that provides an interface to Task Scheduler 1.0. If you're using Windows XP, register this DLL using REGSVR32. Starting with Vista, Windows now uses Task Scheduler 2.0, which has a completely different interface, one that fortunately is easy to use from VFP. The COM component for Task Scheduler 2.0 comes with Windows so it doesn't require anything extra to be installed or registered; the class to instantiate is named Schedule.Service.

So you don't have to worry about the differences between Task Scheduler 1.0 and 2.0, the TaskScheduler parent class has a common interface you'll use to work with scheduled tasks. **Table 1** shows the properties of this class and its subclasses. TaskScheduler only has one public method, CreateTask, which creates a task using the properties of the class.

One thing I'd like to support but haven't figured out a way is creating folders. If you've used the Windows Task Scheduler, you know you can create folders and organize scheduled tasks into them. However, creating a folder requires

passing a security descriptor to a COM method and I haven't figured out how to do that. So, for now, TaskScheduler requires that the folder specified in FolderName property already exists.

Because the Task Scheduler 2.0 API is easier to use from VFP, VistaTaskScheduler has four additional methods which are basically just wrappers for the API methods (in all of these methods, if tcFolder isn't specified, the root folder is used):

- GetFolders(@taFolders [, tcFolder]): fills the specified array with all subfolders of the specified folder. The first column of the array is the folder name and the second is a TaskFolder COM object.
- GetTasks(@taTasks [, tcFolder]): fills the specified array with all tasks in the specified folder. The first column of the array is the task name and the second is a RegisteredTask COM object.
- GetTask(tcName [, tcFolder]): returns a RegisteredTask object for the specified task name in the specified folder or NULL if the task doesn't exist.
- DeleteTask(tcName [, tcFolder]): deletes the specified task in the specified folder.

RegisteredTask is one of the Task Scheduler 2.0 objects. (MSDN has detailed documentation on Task Scheduler 2.0 COM objects, starting at http://tinyurl.com/ncekpgg). You don't have to work with it directly to create a scheduled task, but may wish to if you want to provide task management features such as the ones we'll discuss later.

Table 1. The properties of the TaskScheduler class.

Property	Description
ErrorMessage	The text of any error message
ErrorReason	The reason for the error
ErrorCode	An error code; can be used to localize error messages: 1 = scheduler class not found 2 = scheduler failed to instantiate 3 = a new task could not be created 4 = the task properties could not be set 5 = the scheduler settings could not be saved 6 = the task could not be registered 7 = tasks/folders could not be enumerated 8 = the task could not be retrieved 9 = the task could not be deleted
TaskName	The name of the task
Description	The description of the task
ScheduleType	The type of schedule: 1 = daily 2 = weekly 3 = monthly 4 = monthly day-of-week
StartTime	The starting date and time for the task
Interval	The frequency for a daily or weekly schedule: 1 = every day or week, 2 = every second day or week, and so on
EXEName	The name and path of the program to run
EXEParameters	Parameters to pass to the program
AuthorName	The name of the task author
UserName	The name of the user to log in as
Password	The password for the user
StartWhenAvailable	.T. to start the task as soon as possible after a scheduled start is missed; only supported in Vista and above
FolderName	The name of the folder the task goes in; currently the folder must exist

RunOnLast	.T. to run on the last week or last day of the month; only supported in Vista and above
DaysOfWeek[7]	Set the appropriate element to .T. to execute the task on that day for a weekly task; Sunday is day 1 and Saturday is day 7
DaysOfMonth[31]	Set the appropriate element to .T. to execute the task on that day for a monthly task
MonthsOfYear[12]	Set the appropriate element to .T. to execute the task in that month for a monthly task
WeeksOfMonth[4]	Set the appropriate element to .T. to execute the task in that week for a monthly day-of-week task

Scheduling a task

To create a scheduled task, instantiate the appropriate class (either XPTaskScheduler or VistaTaskScheduler), set its properties as desired, and call CreateTask. If an error occurs, check the ErrorMessage, ErrorReason, and ErrorCode properties for an explanation.

Test.PRG, included in the download for this article and shown in **Listing 1**, demonstrates how to create tasks that execute daily, weekly, and monthly. In this case, the task is very simple: launch Notepad.EXE. In real-life, you'll likely set the EXEParameters property to pass parameters to some application. As noted in the code, change the values of the lcUserName and lcPassword variables to valid values for your system.

Listing 1. Test.PRG shows how to create scheduled tasks.

```
* Ensure we use the correct class.

lcClass = iif(os(3) < '6', ;
    'XPTaskScheduler', 'VistaTaskScheduler')

* Change these to the user name and password
* for a Windows account.

lcUserName = 'YourUserName'
lcPassword = 'YourPassword'

* Create a task that runs at 3:00 AM every
* day.

loSchedule = newobject(lcClass, ;
    'TaskScheduler.prg')
with loSchedule
    .TaskName     = 'Run Notepad'
    .UserName     = lcUserName
    .Password     = lcPassword
    .StartTime    = {^2015-07-01 03:00:00}
    .EXEName      = 'Notepad.exe'
    .ScheduleType = 1
    if not .CreateTask()
        messagebox(.ErrorMessage)
    endif not .CreateTask()
endwith

* Create a weekly task that runs at 3:00 AM
* Tues, Thurs, and Sat of every second week.

loSchedule = newobject(lcClass, ;
    'TaskScheduler.prg')
with loSchedule
    .TaskName     = 'Run Notepad'
    .UserName     = lcUserName
    .Password     = lcPassword
    .StartTime    = {^2015-07-01 03:00:00}
    .EXEName      = 'Notepad.exe'
    .ScheduleType = 2
    .Interval     = 2
    store .T. to .DaysOfWeek[3], ;
        .DaysOfWeek[5], .DaysOfWeek[7]
    if not .CreateTask()
        messagebox(.ErrorMessage)
    endif not .CreateTask()
endwith

* Create a monthly task that runs at 3:00 AM
* on the 1st and 15th of every month.

loSchedule = newobject(lcClass, ;
    'TaskScheduler.prg')
with loSchedule
    .TaskName     = 'Run Notepad'
    .UserName     = lcUserName
    .Password     = lcPassword
    .StartTime    = {^2015-07-01 03:00:00}
    .EXEName      = 'Notepad.exe'
    .ScheduleType = 3
    store .T. to .DaysOfMonth[1], ;
        .DaysOfMonth[15]
    .MonthsOfYear = .T.
        && initialize all 12 elements of array
        && to .T.
    if not .CreateTask()
        messagebox(.ErrorMessage)
    endif not .CreateTask()
endwith
```

Note that if you run this program, you'll only find a single scheduled task when you look in the Windows Task Scheduler. The reason is that all three tasks have the same value for the TaskName property. In that case, the second and third calls to CreateTask overwrite the task created by the first call. The lesson here is that you should give your scheduled tasks unique names. Try changing the assignment statements for TaskName in the second and third examples and running the program again; this time, you'll see all three tasks.

If you want your user to specify the settings for a scheduled task, you can create a wizard or other type of dialog to do so. The form shown in **Figure 1** was taken from Stonefield Query, my company's main product, which uses a wizard so the user can schedule report runs.

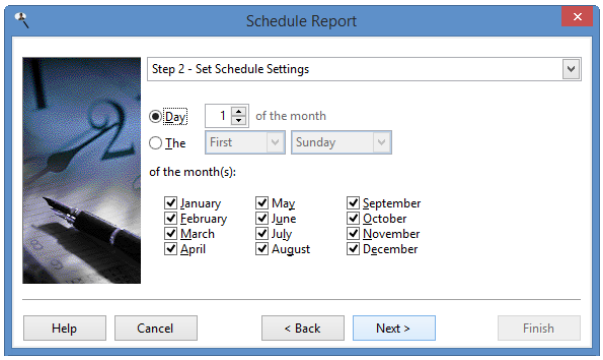

Figure 1. You can create a wizard or other dialog to allow the user to schedule a task.

Managing scheduled tasks

If you want to manage tasks you've scheduled using the TaskScheduler class, you have to use the Windows Task Scheduler, shown in **Figure 2**. While that may be something you can do, it might be a lot to ask of your users, who may not be comfortable working with this dialog.

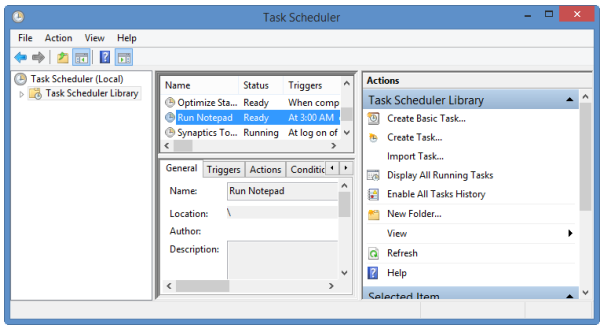

Figure 2. The Windows Task Scheduler allows you to manage scheduled tasks.

Fortunately, you can duplicate much of the functionality of the Windows Task Scheduler using the VistaTaskScheduler class (the XP version doesn't support this), including running or deleting a task or seeing the last time it was run and what the result was. **Figure 3** shows a form class, Schedules in Schedule.VCX, that provides these features:

- It only lists those tasks that execute a certain application, specified in the cEXEName property of the class.
- It displays the status, last run date and time, and last run result for the selected task.
- It allows the user to delete or run the selected task.
- It refreshes the list and information about the selected task when you click Refresh.

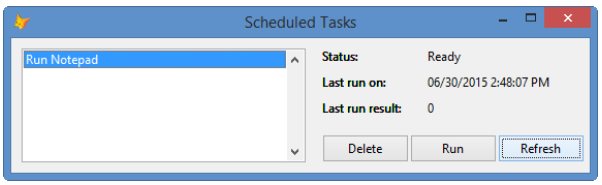

Figure 3. You can use the Schedules class to manage scheduled tasks.

Here's an example of using this class to manage the scheduled tasks created by Test.PRG:

```
loForm = newobject('schedules', ;
    'schedule.vcx')
loForm.cEXEName = 'notepad.exe'
loForm.Show()
```

The GetTasks method, called from Show, uses the GetTasks method of VistaTaskScheduler to fill an array with all tasks, then removes those from the array that don't use the application specified in cEXEName. The array is then used as the RowSource for the listbox. The information about the selected task is retrieved from the State, LastRunTime, and LastTaskResult properties of the RegisteredTask object GetTasks put into the array.

Summary

The TaskScheduler class and subclasses discussed in this article make it easy to create and manage scheduled tasks from a VFP application. Now, if your user asks if it's possible to automate certain tasks at regular intervals, you can say Yes!

Doug Hennig is a partner with Stonefield Software Inc. He is the author of the award-winning Stonefield Database Toolkit (SDT); the award-winning Stonefield Query; the MemberData Editor, Anchor Editor, and CursorAdapter and DataEnvironment builders that come with Microsoft Visual FoxPro; and the My namespace and updated Upsizing Wizard in Sedna.

Doug is co-author of "Making Sense of Sedna and SP2," the "What's New in Visual FoxPro" series (the latest being "What's New in Nine"), "Visual FoxPro Best Practices For The Next Ten Years," and "The Hacker's Guide to Visual FoxPro 7.0." He was the technical editor of "The Hacker's Guide to Visual FoxPro 6.0" and "The Fundamentals." All of these books are from Hentzenwerke Publishing (http://www.hentzenwerke.com). He wrote over 100 articles in 10 years for FoxTalk and has written numerous articles in FoxPro Advisor, Advisor Guide to Visual FoxPro, and CoDe. He currently writes for FoxRockX (http://www.foxrockx.com).

Doug spoke at every Microsoft FoxPro Developers Conference (DevCon) starting in 1997 and at user groups and developer conferences all over the world.

He is one of the organizers of the annual Southwest Fox conference (http://www.swfox.net). He is one of the administrators for the VFPX VFP community extensions Web site (http://vfpx.codeplex.com). He was a Microsoft Most Valuable Professional (MVP) from 1996 to 2011. Doug was awarded the 2006 FoxPro Community Lifetime Achievement Award (http://tinyurl.com/ygnk73h).