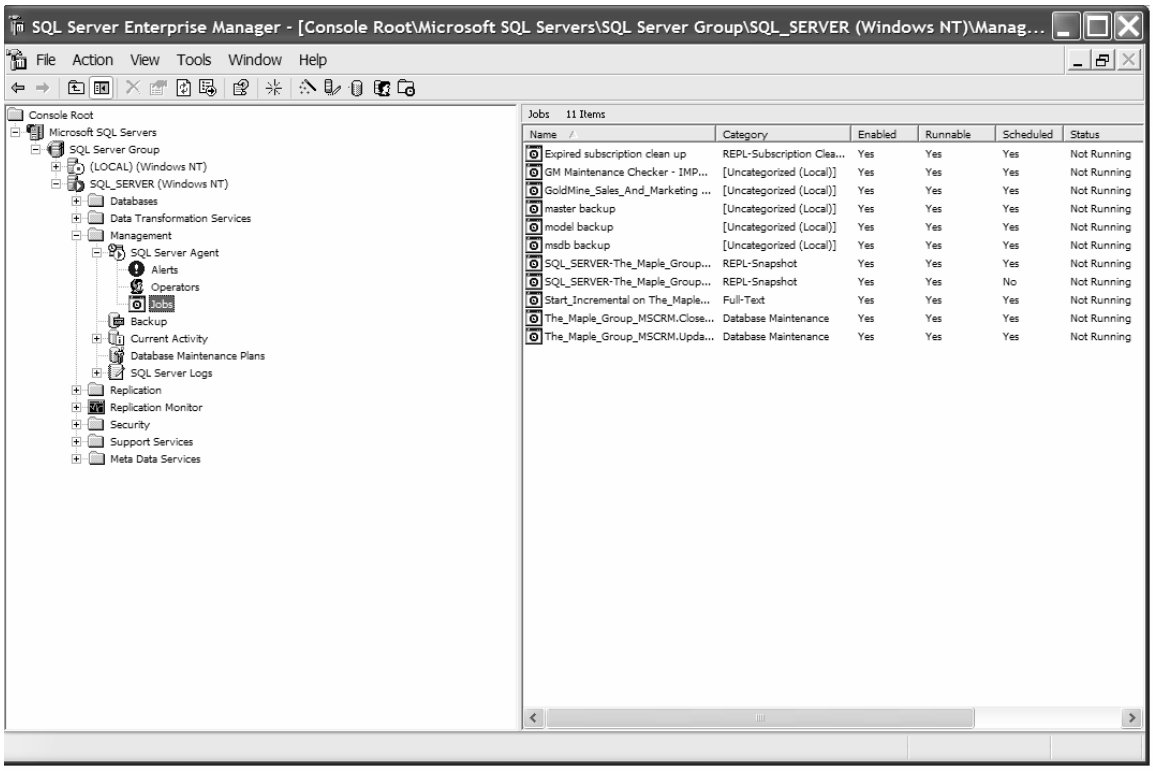
relatia software corporation

its not in the manual

Frustrated with GoldMine's Automated Processes? How about SQL Jobs.

The Back-story:

I've been a long-time fan of GoldMine's Automated Processes feature, however, I've often been frustrated at how difficult it can be to do the simplest tasks, or when I need an audit trail or notification when something goes wrong. Example: the other day, one of my salespeople asked me if it would be possible for them to be notified when one of their customers was up for maintenance renewal. Sounds simple enough; I knew we had the maintenance renewal date for all of our customers in a user-defined field in GoldMine, however, when I looked into the problem further, I realized that GoldMine's AP system may not be the simplest way to accomplish the task. After contemplating the options, I decided that this was the perfect job for SQL Server. SQL Server's job engine is a highly reliable way to schedule recurring events system allowing you to schedule the execution of various system functions at regular intervals (including once). In my case, the ability to run an SQL Script every night sounded perfect.



SQL Server's Job List

Getting Started:

Step one for me is always to outline the project in plain English so, we'll start with some easy-to-read project steps. I know the maintenance renewal dates are stored in our database in a field called UGMMAINT; so:

Step 1: Scan the database for contacts where the date value in UGMMAINT is 61 days from today (I'm assuming the job will run every night, so I don't need to fiddle with < or >).

Step 2: For each record found, create an activity for person x that will show up in GoldMine the next time they login or synchronize.

Pretty straight-forward; should be a piece of cake.

Implementing the solution:

Step 1 – The SQL Script

Now that we have a clear understanding of the objectives, the most logical place to start is writing the SQL code that will do the actual work. Typically, when you write the code for a recurring job, you start by writing a simple script that would

accomplish the goals in a once-in-time situation; then you simply schedule the script to run at some interval. So… here goes.

By the way, I'm assuming you have GoldMine's Stored Procedures API (GMSPROCS) installed and configured (a 5-minute job).

Let's start with the select statement that returns the records that meet our maintenance cutoff requirements:

```
select
      c1.accountno,
      c1.contact,
      c2.ugmmaintdt
from
      contact1 c1
      inner join
      contact2 c2
      on c1.accountno=c2.accountno
where c1.accountno in
      (
      select
            accountno
      from
            contact2
      where
            datediff(day,getdate(),ugmmaintdt)=61
      )
```

Yup, your average, ho-hum join query. The only interesting item here is the datediff function, which is an intrinsic SQL function designed to compare two dates and return the difference between the two.

Okay, now that we know how to get the records out of the database, what can we do with them? In order to take advantage of the automatic update of GoldMine's TLOGS provided by the GMSPROCS API, we typically need to process return sets in a cursor so that GMSPROCS functions can do the updates properly for each record. If follows that we need to wrap our query in an SQL cursor so we can step through the return set one record at a time.

First, Declare some variables we're going to need:

```
Declare

@reccheck int,
@accountno varchar(40),
@contact varchar(40),
@maintdate datetime,
@gmnv varchar(20),
@recid varchar(15),
@today datetime,
```

```
@ref varchar(80)
```

Let's pop today's date into the @today variable:

```
select @today=getdate()
```

Now, it would probably be a good idea to check and see if we have any records to process before we go and send the SQL Server off on a wild goose chase:

```
select @reccheck=count(*) from contact1 where accountno in(select
accountno from contact2 where datediff(day,getdate(),ugmmaintdt)=61)

if @reccheck>0
```

Okay, assuming we have some records to process, create the cursor and pump the return values (one record at a time) into our variables:

```
BEGIN
      declare goldcursor scroll cursor for
      select
            c1.accountno,
            c1.contact,
            c2.ugmmaintdt
      from
            contact1 c1
            inner join
            contact2 c2
            on c1.accountno=c2.accountno
      where
            c1.accountno in
            (
            select
                  accountno
            from
                  contact2
            where
                  datediff(day,getdate(),ugmmaintdt)=61
            )
      open goldcursor
      fetch first from goldcursor into
      @accountno,
      @contact,
      @maintdate
```

Now, process the records:

```
while @@fetch_status=0

      BEGIN
```

As you can see from the above code, each time we loop through the cursor code, we'll have a handle to a contact1 record (via the accountno) so all we need to do is

write an activity record linked to that accountno and assigned to a person (in our case 2 people, Linda and Russell).

To use the GMSPROCS you must first create a container to hold a series of name / value pairs:

```
exec gmw_nv_create @gmnv OUTPUT
```

Once you've created the handle (@gmnv), you can now populate the container with your fields/data:

```
exec GMW_NV_SetValue @gmnv,   'user','master'
exec GMW_NV_SetValue @gmnv,   'accountno',@accountno
exec GMW_NV_SetValue @gmnv,   'rectype','C'
exec GMW_NV_SetValue @gmnv,   'userID','LINDA'
exec GMW_NV_SetValue @gmnv,   'Contact',@contact
exec GMW_NV_SetValue @gmnv,   'OnDate',@today
exec GMW_NV_SetValue @gmnv,   'Ref',@ref
```

Then, you call an API function against the container to write to the database (and update the sync logs properly).

```
exec GMW_WriteSchedule @gmnv
```

Now, delete the container:

```
exec GMW_NV_Delete @gmnv
```

In my case, I wanted to set a second activity for myself just to be sure the job was firing:

```
exec gmw_nv_create @gmnv OUTPUT
exec GMW_NV_SetValue @gmnv,   'user','master
exec GMW_NV_SetValue @gmnv,   'accountno',@accountno
exec GMW_NV_SetValue @gmnv,   'rectype','C'
exec GMW_NV_SetValue @gmnv,   'userID','RUSSELL'
exec GMW_NV_SetValue @gmnv,   'Contact',@contact
exec GMW_NV_SetValue @gmnv,   'OnDate',@today
exec GMW_NV_SetValue @gmnv,   'Ref',@ref
exec GMW_WriteSchedule @gmnv
exec GMW_NV_Delete @gmnv
```

Now fetch the next record in the cursor:

```
fetch next from goldcursor into
@accountno,
@contact,
@maintdate


END
```

Finally, close and deallocate the cursor:

```
close goldcursor
deallocate goldcursor

END
```

And that's all there is to the SQL script (contiguous version below). I'm using literal names here for the GoldMine users who get assigned an activity, however, you could just as easily query the contact record for a KEY field or record ownership to dynamically allocate the activity at runtime.

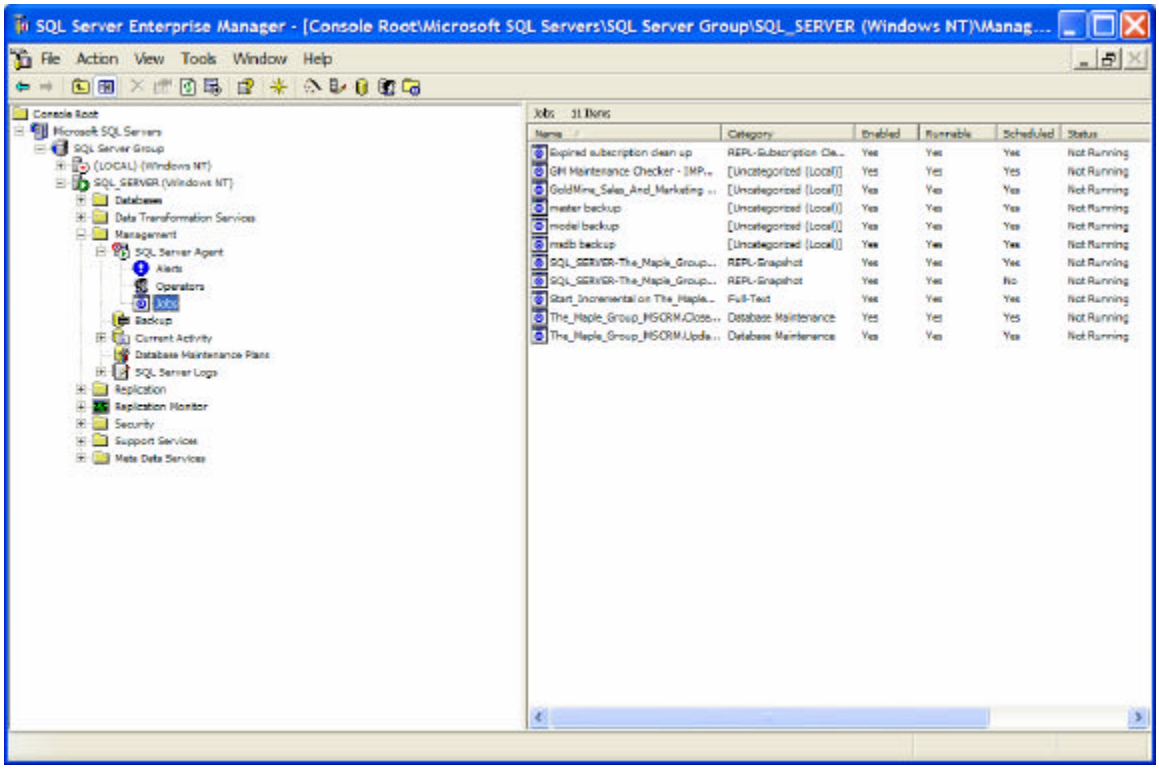
In fact, there's a whole host of additional functionality that could be added to this simple script, however, I chose to keep it simple here for clarity.

Step 2 – Create the SQL Job.

If you simply plugged the script into Query Analyzer and ran it, you would create an activity for any contact records whose maintenance date is 61 days from today. Well, you say, that's great and everything, but It would be slightly more useful if this script were run daily.

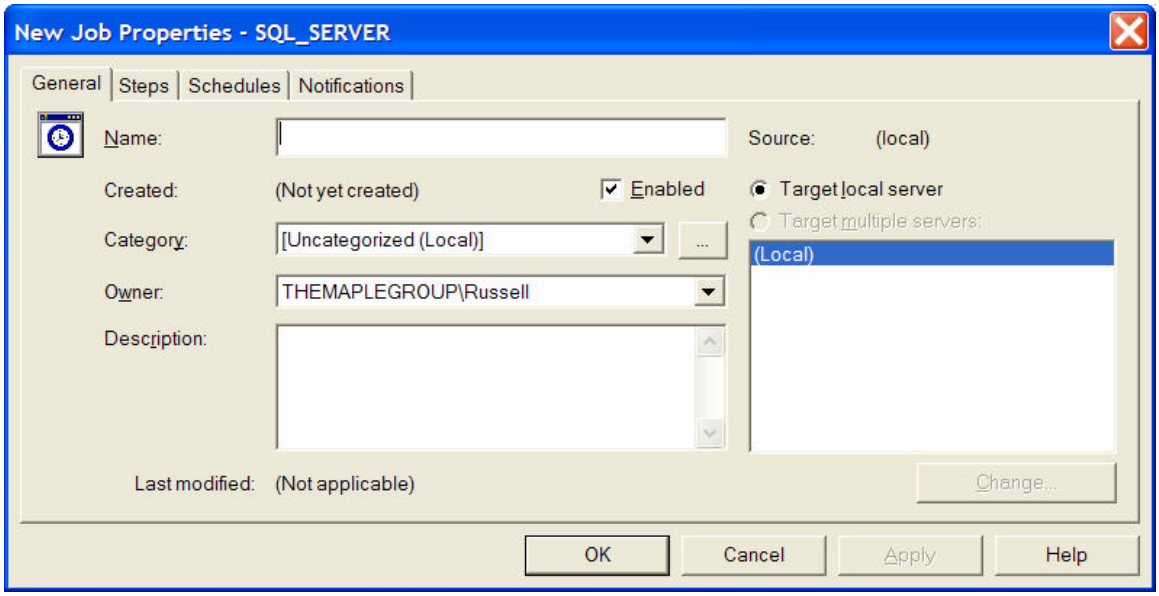
Enter SQL Server's Job engine. Setting this script up to be run at a certain time interval is a snap with SQL Server, you simply create a job, then assign a schedule to it, and, voila!, instant process automation.

To create the job, open SQL Server Enterprise Manager and expand your GoldMine database server. Now expand the "Management" folder, then expand the "SQL Server Agents" folder and highlight the "Jobs" node:



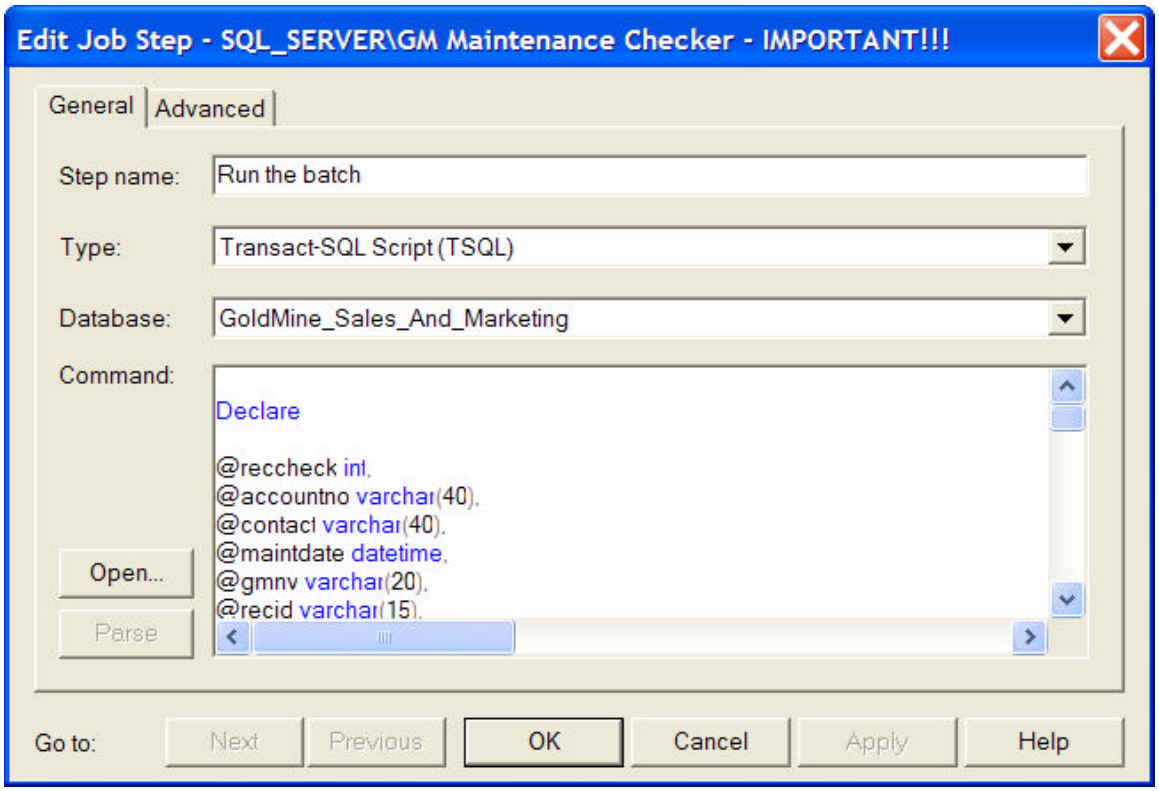
To the right is the list of jobs that have been created on this instance of SQL Server.

Right-click on the jobs node and select "New Job". You should see the following window:



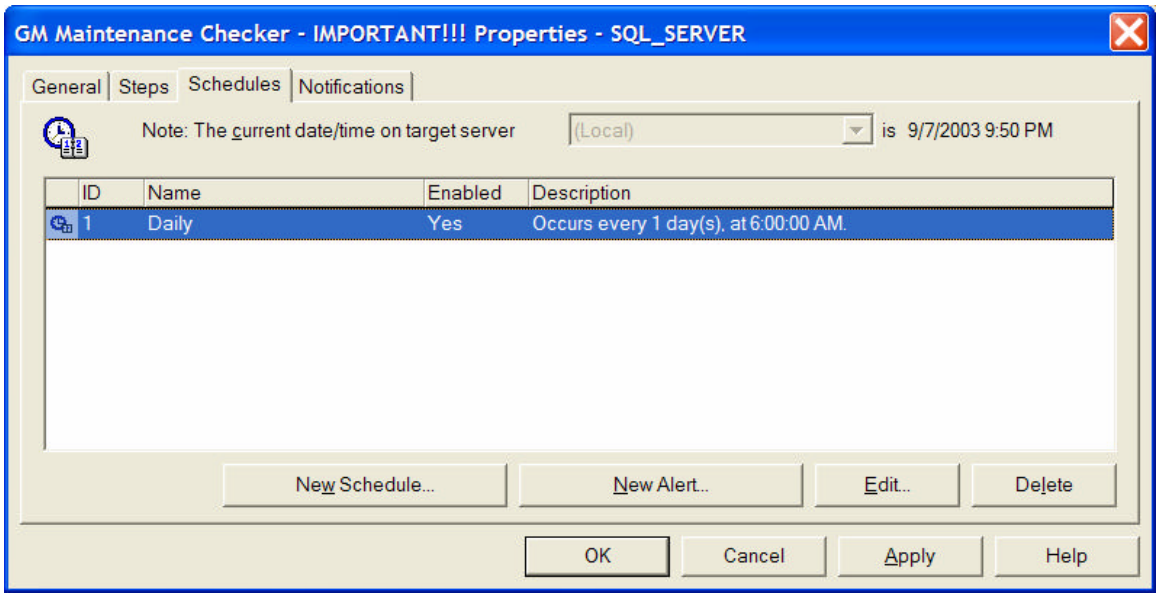
You can call the job whatever you like and leave it uncategorized but be sure to set the owner of the job to a user who has the appropriate rights on the GoldMine database.

Click the Steps Tab, then click “new” to add a new step:

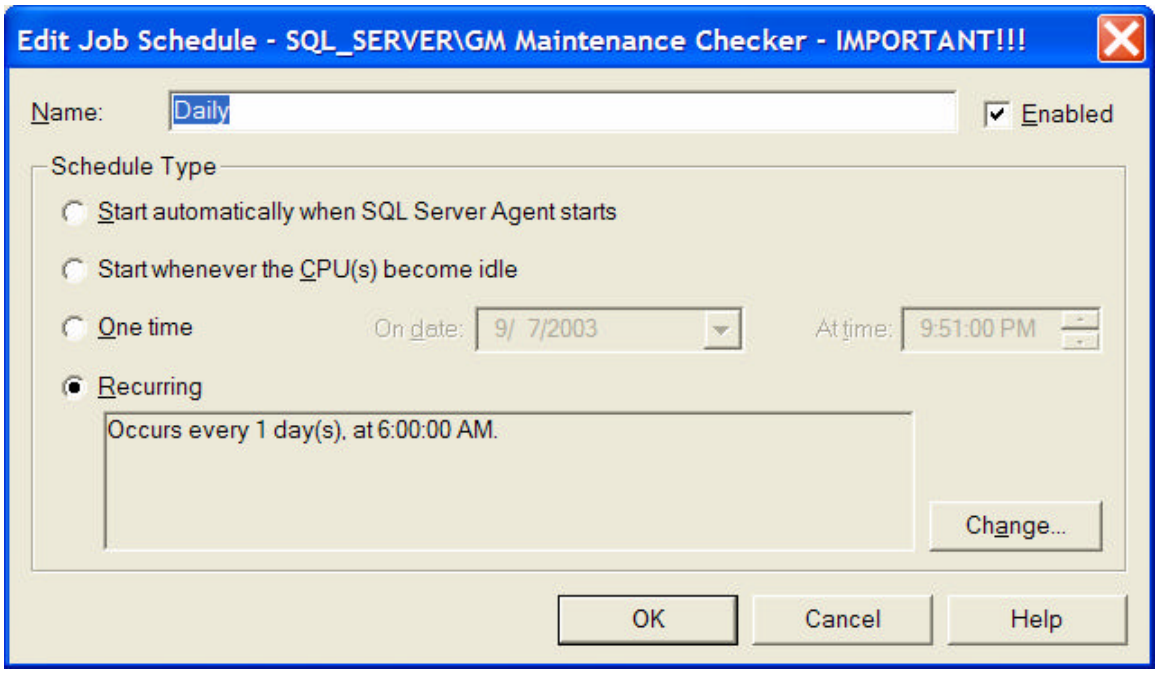


Past the text of your SQL query into the command window, and change the database to your GoldMine database. Make sure the type “Transact-SQL Script(TSQL)” is selected. Click “OK”.

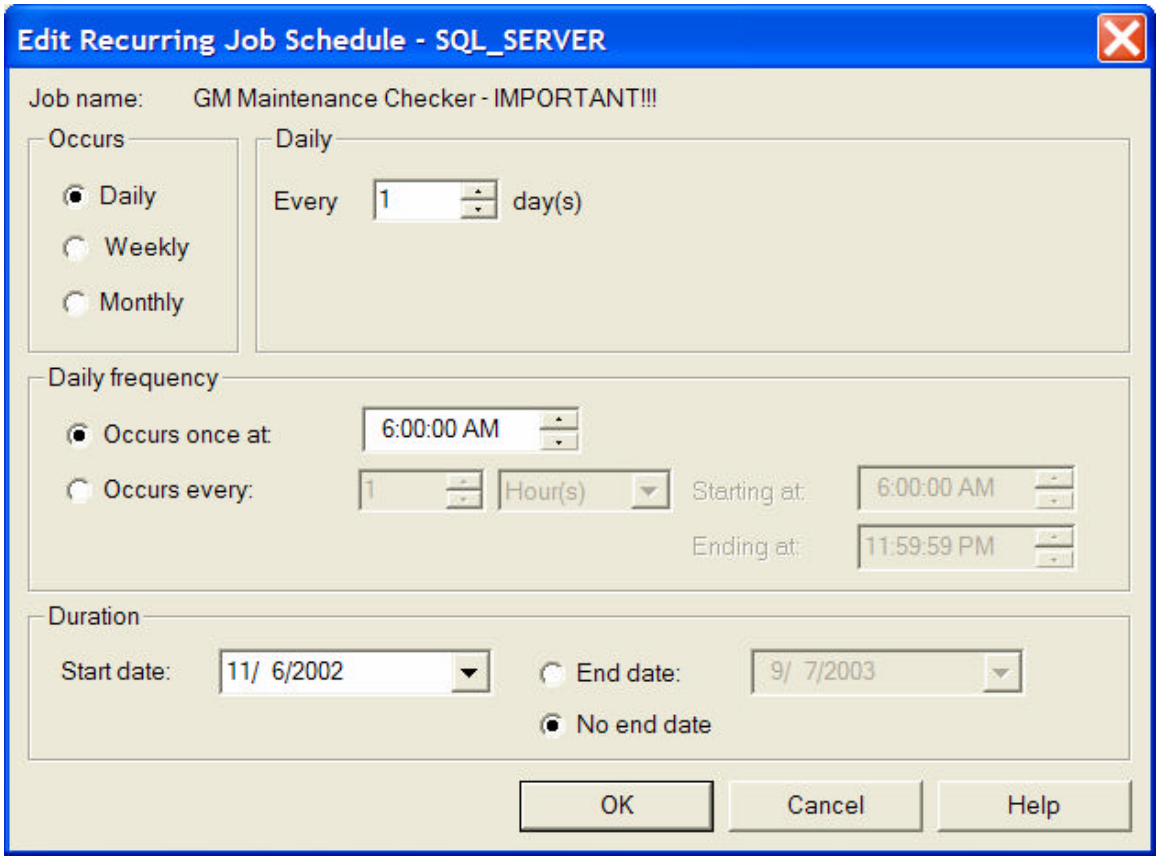
Click on the “Schedules” tab:



Click “New Schedule”:



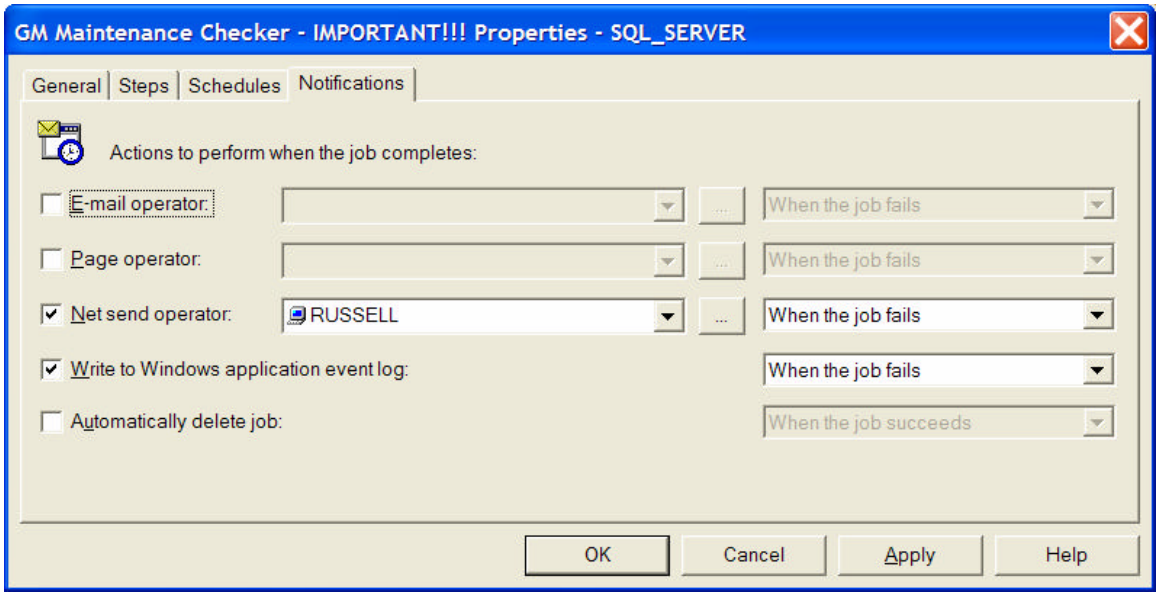
Now give your schedule a name, then click “Change”



Change the frequency, time and start/end dates as necessary and click “OK”.

Now for the really cool part….

Click the “Notifications” tab.



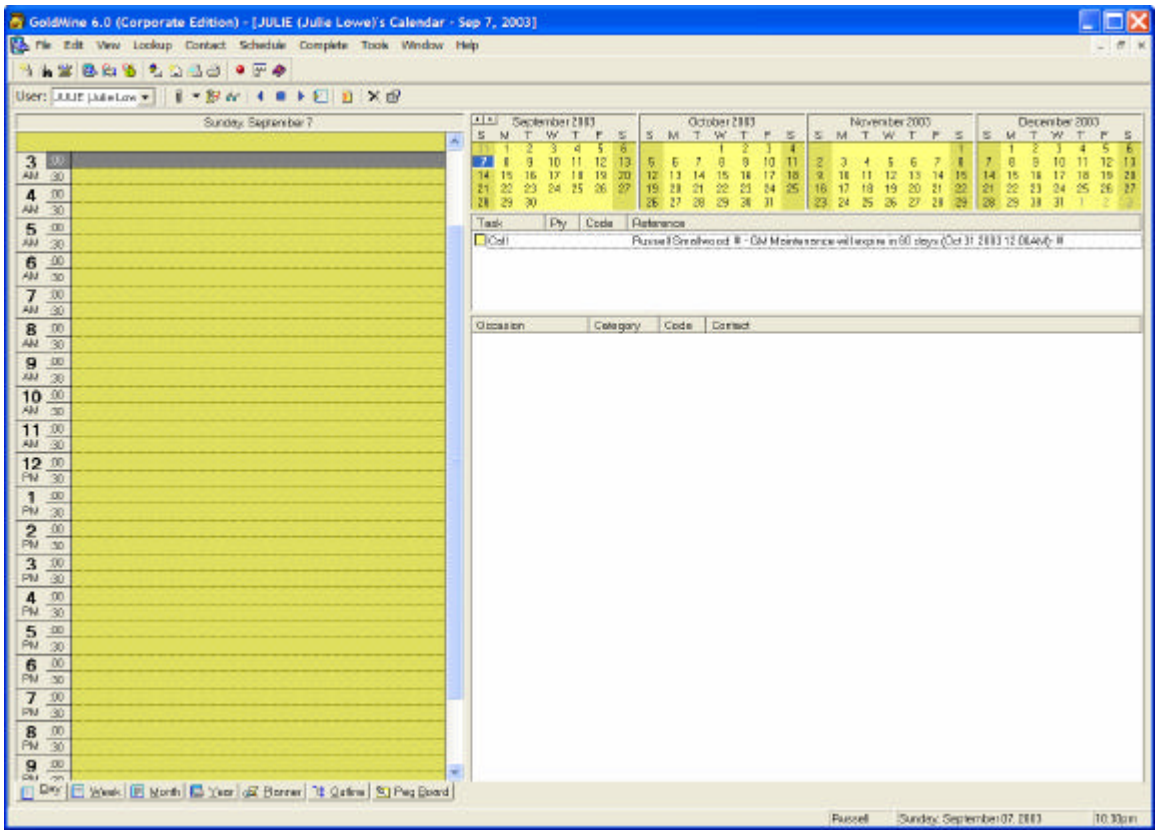
Here's where you can be sure someone is notified when the job fails !! Wow, an actual audit trail with notification. Pretty slick in my book.

Change the options here as desired, then click "OK".

That's it, you've now scheduled your maintenance renewal update job to run to run every day (or whatever you specified) until the end of time, and, with SQL Server, you can be certain that it will run, without fail until you tell it to stop. No further intervention is required on your part.

The Wrap-up:

I chose this very simple example because none of the SQL was very difficult, and the end result was something tangible and easy to understand. As you can see, however, the possibilities are endless. I typically use SQL Jobs for back-end data manipulation that needs to take place on a regular basis, or, in situations where it's critical to my customer that automation of certain processes happens without fail or leaves a clear audit and notification trail when it doesn't.



The result of our job..

Another advantage of using SQL Jobs is the sophistication of database manipulation that can be attained. I once setup an SQL job that scrolled through a GoldMine database, evaluated the number of referrals that were made to that contact and updated a field in the contact2 table to indicate a referral classification. This allowed the customer to run filters against the data in ways that are just not possible using the data in its raw format.

Where SQL jobs are not the best solution is in cases where something needs to happen immediately when some trigger threshold has been reached. Since SQL Jobs will only run on specific schedules or if run manually from the jobs window in SQL Server Enterprise Manager, they aren't a good choice if a moderate level of latency is unacceptable when evaluating and firing a database event. For that, I'd use an SQL Trigger, but that's a conversation for another day.

Russell Smallwood is CEO of Relatia Software Corporation, authors of the popular Relatia Time and Billing GoldMine add-on. Relatia also provides custom programming services to GoldMine VARs across the US.

For more information on how you can help your customer and win more business, call Linda O'Connor at (770) 663-4455 x 305 today.

Addendum A – SQL Script in its entirety

```
Declare

@reccheck int,
@accountno varchar(40),
@contact varchar(40),
@maintdate datetime,
@gmnv varchar(20),
@recid varchar(15),
@today datetime,
@ref varchar(80)


select @reccheck=count(*) from contact1 where accountno in(select
accountno from contact2 where datediff(day,getdate(),ugmmaintdt)=61)
select @today=getdate()
if @reccheck>0

      BEGIN
      declare goldcursor scroll cursor for
      select c1.accountno,c1.contact,c2.ugmmaintdt  from contact1 c1
inner join contact2 c2 on c1.accountno=c2.accountno where c1.accountno
in(select accountno from contact2 where
datediff(day,getdate(),ugmmaintdt)=61)
      open goldcursor
      fetch first from goldcursor into
      @accountno,
      @contact,
      @maintdate
      while @@fetch_status=0

            BEGIN
            select @ref='!!! - GM Maintenance will expire in 60 days ('
+ cast(@maintdate as varchar(40)) +  ')- !!!'

            exec gmw_nv_create @gmnv OUTPUT
            exec GMW_NV_SetValue @gmnv,   'user','master'
            exec GMW_NV_SetValue @gmnv,   'accountno',@accountno
            exec GMW_NV_SetValue @gmnv,   'rectype','C'
            exec GMW_NV_SetValue @gmnv,   'userID','LINDA'
            exec GMW_NV_SetValue @gmnv,   'Contact',@contact
```

```
        exec GMW_NV_SetValue @gmnv,   'OnDate',@today
        exec GMW_NV_SetValue @gmnv,   'Ref',@ref
        exec GMW_WriteSchedule @gmnv
        exec GMW_NV_Delete @gmnv
        --Write an activity for Russell as well

        exec gmw_nv_create @gmnv OUTPUT
        exec GMW_NV_SetValue @gmnv,   'user','master'
        exec GMW_NV_SetValue @gmnv,   'accountno',@accountno
        exec GMW_NV_SetValue @gmnv,   'rectype','C'
        exec GMW_NV_SetValue @gmnv,   'userID','RUSSELL'
        exec GMW_NV_SetValue @gmnv,   'Contact',@contact
        exec GMW_NV_SetValue @gmnv,   'OnDate',@today
        exec GMW_NV_SetValue @gmnv,   'Ref',@ref
        exec GMW_WriteSchedule @gmnv
        exec GMW_NV_Delete @gmnv

        fetch next from goldcursor into
        @accountno,
        @contact,
        @maintdate
        END

close goldcursor
deallocate goldcursor

END
```