White Paper

The GoldMine® Advisor

Editor

DJ Hunt

SQL Server 2008 Maintenance Plan

by

DJ Hunt

Over the years people have asked me for documentation on an accepted SQL Server Maintenance Plan. To date, I have been unable to find one that was written for the GoldMine application/SQL Database. FrontRange utilizes SQL Server 2008 for Workgroups in that it distributes this with GoldMine Premium v9.0.0.102, yet to date, FrontRange will not support the SQL Server application/installation/configuration.

Computerese Incorporated has implemented many SQL Server Maintenance Plans over the years, and although we know how we do it, we really don't know if this is the correct ***FrontRange Approved*** way of do this. I am writing this whitepaper as a way of documenting, for the first time, the Computerese Incorporated SQL Server Maintenance Plan implementation process. As always, if you utilize these steps to implement your own SQL Server Maintenance Plan for your GoldMine Database(s), you do so at you own risk knowing full well that these steps have not received the stamp of approval from FrontRange nor any organization for that matter. Although, since it's original printing in The GoldMine Advisor - August issue, I have heard from many GoldMine Partners who have approved this process with a few exceptions which I will discuss in this whitepaper at the appropriate time.

Step 01

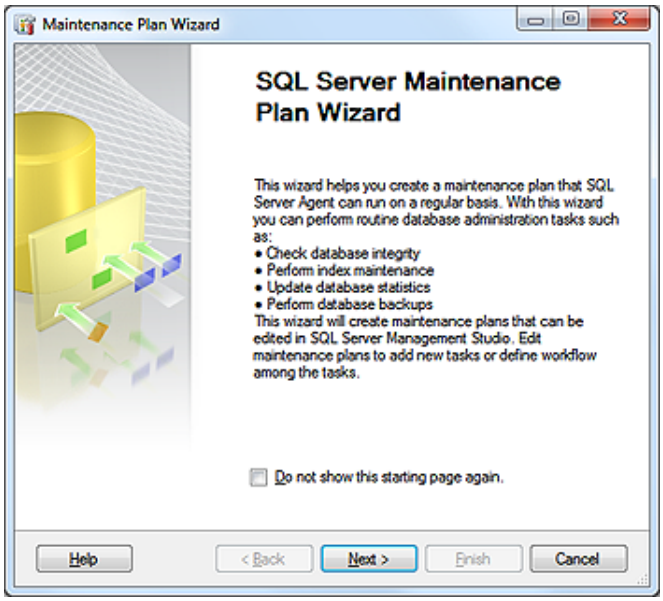
Open SQL Server Management Studio, and **Connect** to your SQL Server.

Step 02

Expand the tree **+Management** by clicking upon the + sign.

Step 03

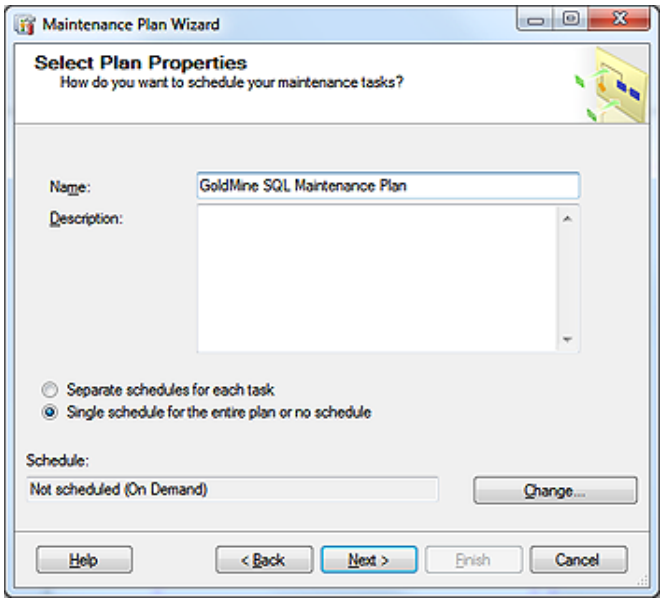
Right-click on **+Maintenance Plans** and select **Maintenance Plan Wizard** from the local menu. Even though we know the steps involved, we always use the Wizard to assure a consistent Maintenance Plan execution.



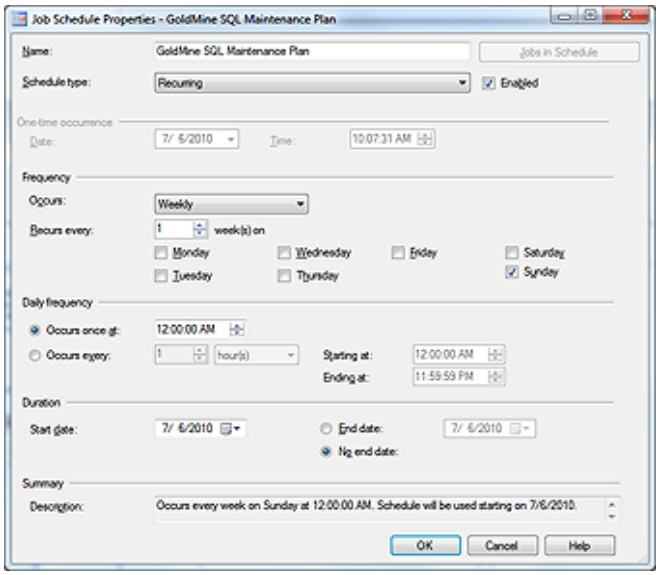
Doing so will produce the dialog form shown on Page 1. I have chosen to not check the ☐ **Do not show this starting page again.** option as I wanted you to see the default screens, however, you could select this option as this splash screen is really superfluous.

Step 04

Clicking on the **Next >** button will bring you to the **Select Plan Properties** dialog form:



It is here that one should supply a unique **Name** for their plan. For this example I have chosen **GoldMine SQL Maintenance Plan.** Now is also the time to decide on the frequency of execution of this plan once created. One does this by clicking in the **Schedule:** area of the dialog form on the **Change** button to produce this dialog form:



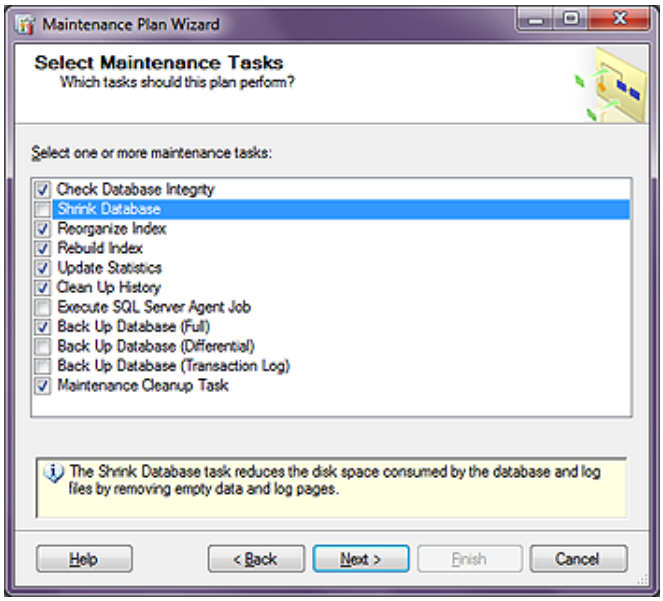
Now, again, this is the default screenshot, however, you must decide how to configure this based on your office backup procedures. The SQL Maintenance Plan, which can include its own Backup Plan, should not be confused with your Office Backup Plan. Redundant, yes, but how important is your GoldMine Database to your office? So important, I would imagine, that one should welcome redundancy.

Regardless, the Computerese Inc **Frequency** is **Daily**, in fact, we set up two of these plans with the first running at 12:15 pm Eastern Time, and the second running at 7:00 pm Eastern Time. We don't mind redundancy, and we don't plan on losing any more than 7 hours worth of work. We have clients that have requested a plan run hourly. The decision should be strictly between you and your IT department. If you plan on just nightly, make sure it executes before your Office Backup Plan so the files that we will be creating will get backed up there as well.

Everything else on this dialog form remains in its default state, and we would now click upon the **OK** button.

Step 05

Clicking on the **Next >** button will bring you to the **Select Maintenance Task** dialog form:



Note

*Readers of the original article in The GoldMine Advisor may remember that the **Shrink Database** option was selected in that article. Since that printing I have received many e-mails pointing me to sites which described why the **Shrink Database** was a bad idea. Once such article was written by one of the developers of the SQL Server application, hence, I am no longer recommending it be utilized. Said article is reprinted in this whitepaper on the next page for your review.*

True, this is not the default state of this form, however, I did want to show you the selections that we make on this screen. I won't try to explain what each of these options represents, however, if that is of interest to you then you will find the SQL Server Help files to be most useful in this matter.

In the original article, and in the past for that matter, I had selected to **Shrink Database** as and option. Since then Jane Oliverio sent me a link that discusses the pros and the cons, actually more cons than pros, for doing this, and I have decided against shrinking my database. Here is the article reprinted from Paul S Randals' Blog in case this site goes dormant:

Why you should not shrink your data files

by

Paul B. Randal

One of my biggest hot-buttons is around shrinking data files. Although I used to own the shrink code while I was at Microsoft, I never had a chance to rewrite it so that data file shrink is a more palatable operation. I really don't like shrink.

Now, don't confuse shrinking the transaction log with shrinking data files. Shrinking the log is necessary if your log has grown out of control, or as part of a process to remove excessive VLF fragmentation. However, shrinking the log should be a rare operation and should not be part of any regular maintenance you perform.

Shrinking of data files should be performed even more rarely, if at all. Here's why - data file shrink causes *massive* index fragmentation. Let me demonstrate with a simple script you can run. The script below will create a data file, create a 10MB 'filler' table at the start of the data file, create a 10MB 'production' clustered index, drop the 'filler' table and then run a shrink to reclaim the space.

```
USE MASTER;
GO

IF DATABASEPROPERTYEX ('DBMaint2008', 'Version') > 0
 DROP DATABASE DBMaint2008;

CREATE DATABASE DBMaint2008;
GO
USE DBMaint2008;
GO

SET NOCOUNT ON;
GO

-- Create the 10MB filler table at the 'front' of the data file
CREATE TABLE FillerTable (c1 INT IDENTITY,  c2 CHAR (8000) DEFAULT 'filler');
GO

-- Fill up the filler table
INSERT INTO FillerTable DEFAULT VALUES;
GO 1280

-- Create the production table, which will be 'after' the filler table in the data file
CREATE TABLE ProdTable (c1 INT IDENTITY,  c2 CHAR (8000) DEFAULT 'production');
CREATE CLUSTERED INDEX prod_cl ON ProdTable (c1);
GO

INSERT INTO ProdTable DEFAULT VALUES;
GO 1280

-- check the fragmentation of the production table
SELECT [avg_fragmentation_in_percent] FROM sys.dm_db_index_physical_stats (
   DB_ID ('DBMaint2008'), OBJECT_ID ('ProdTable'), 1, NULL, 'LIMITED');
GO

-- drop the filler table, creating 10MB of free space at the 'front' of the data file
DROP TABLE FillerTable;
GO

-- shrink the database
DBCC SHRINKDATABASE (DBMaint2008);
GO
```

```
-- check the index fragmentation again
SELECT [avg_fragmentation_in_percent] FROM sys.dm_db_index_physical_stats (
   DB_ID ('DBMaint2008'), OBJECT_ID ('ProdTable'), 1, NULL, 'LIMITED');
GO

avg_fragmentation_in_percent
-----------------------------
0.390625

DbId   FileId      CurrentSize MinimumSize UsedPages   EstimatedPages
------ ----------- ----------- ----------- ----------- --------------
6      1           1456        152         1448        1440
6      2           63          63          56          56

DBCC execution completed. If DBCC printed error messages, contact your system administrator.

avg_fragmentation_in_percent
-----------------------------
99.296875
```

Look at the output from the script! The logical fragmentation of the clustered index before the shrink is a near-perfect 0.4%. After the shrink, it's almost 100%. The shrink operation *completely* fragmented the index, removing any chance of efficient range scans on it by ensuring the all range-scan readahead I/Os will be single-page I/Os.

Why does this happen? A data file shrink operation works on a single file at a time, and uses the GAM bitmaps to find the highest page allocated in the file. It then moves it as far towards the front of the file as it can, and so on, and so on. In the case above, it completely reversed the order of the clustered index, taking it from perfectly defragmented to perfectly fragmented.

The same code is used for DBCC SHRINKFILE, DBCC SHRINKDATABASE, and auto-shrink - they're equally as bad. As well as introducing index fragmentation, data file shrink also generates a lot of I/O, uses a lot of CPU, and generates *loads* of transaction log - as everything it does is fully logged.

Data file shrink should never be part of regular maintenance, and you should NEVER, NEVER have auto-shrink enabled. I tried to have it removed from the product for SQL 2005 and SQL 2008 when I was in a position to do so - the only reason it's still there is for backwards compatibility. Don't fall into the trap of having a maintenance plan that rebuilds all indexes and then tries to reclaim the space required to rebuild the indexes by running a shrink - that's a zero-sum game where all you do is generate a log of transaction log for no actual gain in performance.

So what if you *do* need to run a shrink? For instance, if you've deleted a large proportion of a very large database and the database isn't likely to grow, or you need to empty a file before removing it?

The method I like to recommend is as follows:

- Create a new filegroup
- Move all affected tables and indexes into the new filegroup using the CREATE INDEX ... WITH (DROP_EXISTING) ON <filegroup> syntax, to move the tables and remove fragmentation from them at the same time
- Drop the old filegroup that you were going to shrink anyway (or shrink it way down if its the primary filegroup)

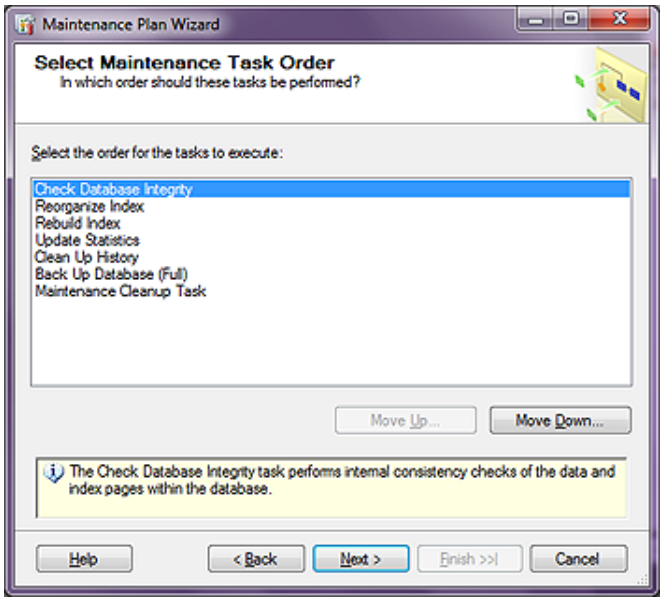
Basically you need to provision some more space before you can shrink the old files, but it's a much cleaner mechanism.

If you absolutely have no choice and have to run a data file shrink operation, be aware that you're going to cause index fragmentation and you should take steps to remove it afterwards if it's going to cause performance problems. The only way to remove index fragmentation without causing data file growth again is to use DBCC INDEXDEFRAG or ALTER INDEX ... REORGANIZE. These commands only require a single 8KB page of extra space, instead of needing to build a whole new index in the case of an index rebuild operation.

Bottom line - try to avoid running data file shrink at all costs!

Step 06

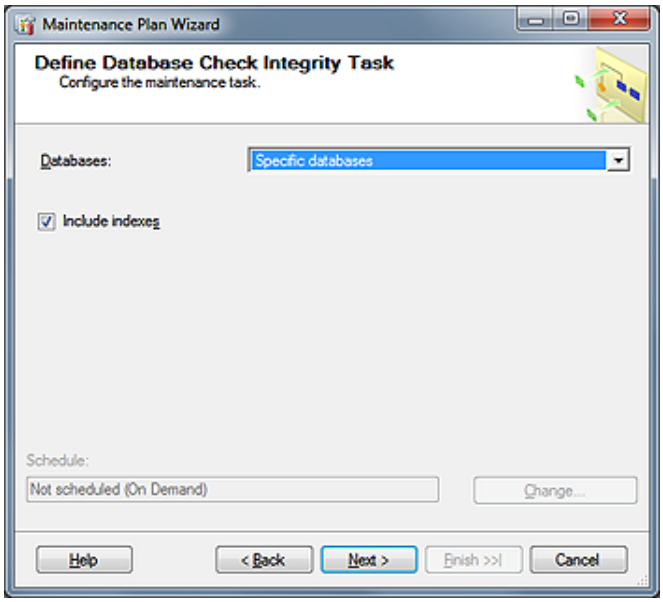
Clicking on the **Next >** button will bring you to the **Select Maintenance Task Order** dialog form.



This dialog form permits you to change the order of execution of the steps which you had selected on the previous dialog form.

Step 07

Clicking on the **Next >** button will bring you to the default screen of the **Define Database Check Integrity Task** dialog form:



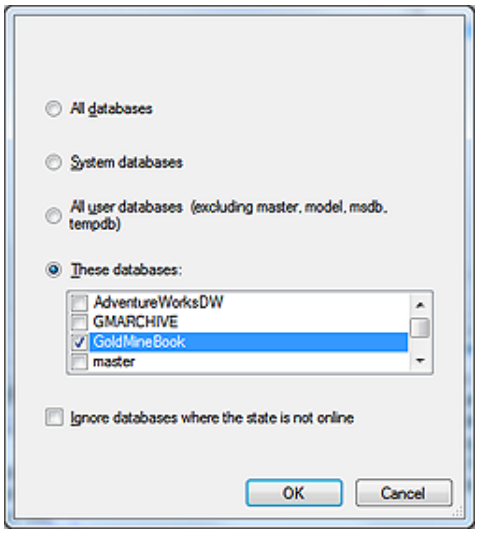
However, in the default state, the **Database:** input field would have stated: **<Select one or more>**, and it would have been up to you to select one or more databases to include in this plan. As this is the same dialog form that is displayed on each of the successive dialog forms, I will show it to you but only this once in this article.

As you can see in the figure at the top of the continuing page which was instantiated by clicking on the down arrow at the end of the **<Select one or more>** option, the default setting is: ◉ **These databases:** where one would actually check the database(s) to include in this plan. As I mentioned previously, this setting **is not** sticky, and you will need to reselect the database(s) on each of the successive dialog forms in this Wizard where you come across the verbiage **Database: <Select one or more>**.

Note

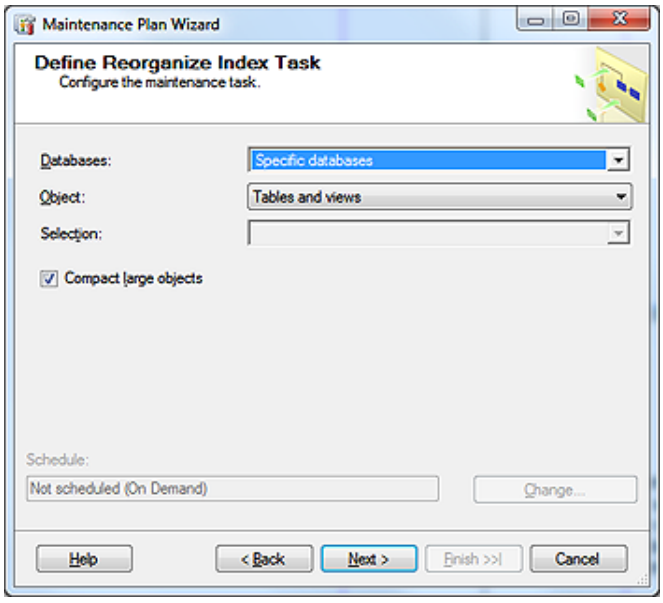
Normally, I would not touch the arrangement on this dialog form, however, Alberto Diaz of 180° Solutions has told me that he prefers to move the **Maintenance Cleanup Task** to run prior to the **Back Up Database (Full)** task. He found, in his tests, that the plan tends to fail less frequently when executed in that order. I must say that I have had a few instances where having the **Maintenance Cleanup Task** as the last task in the sequence order has caused a failure of the Maintenance Plan to produce a **Successful** log entry.

Your milage may vary!



I have only selected a single **GoldMineBook** database, however, you could have selected all of your GoldMine databases if you utilize more than one (not recommended). One would next click on the **OK** button to return to the **Define Database Integrity Check Task** dialog form.

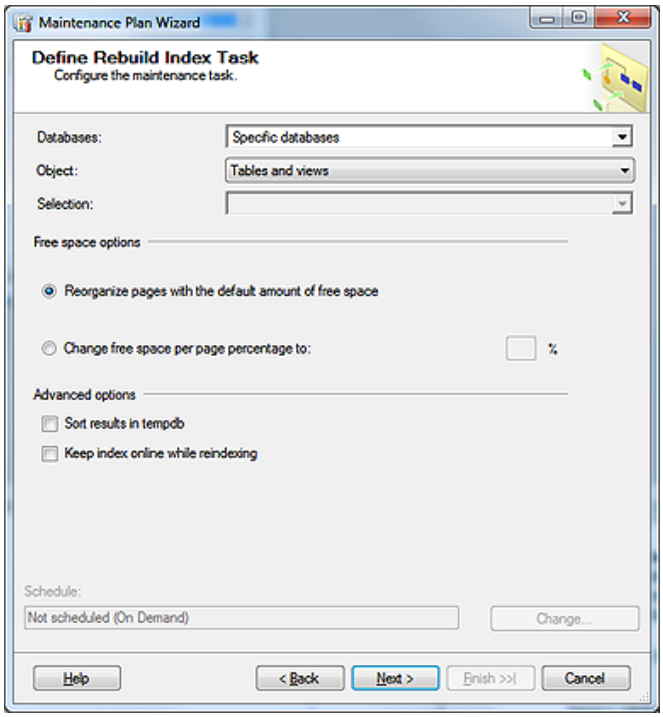
Step 08



Clicking on the **Next >** button will bring you to the **Define Reorganize Index Task** dialog form displayed in the next column. Again, this is not the default dialog form as I have already selected **Specific databases**, although everything else on this dialog form remains in its default state.

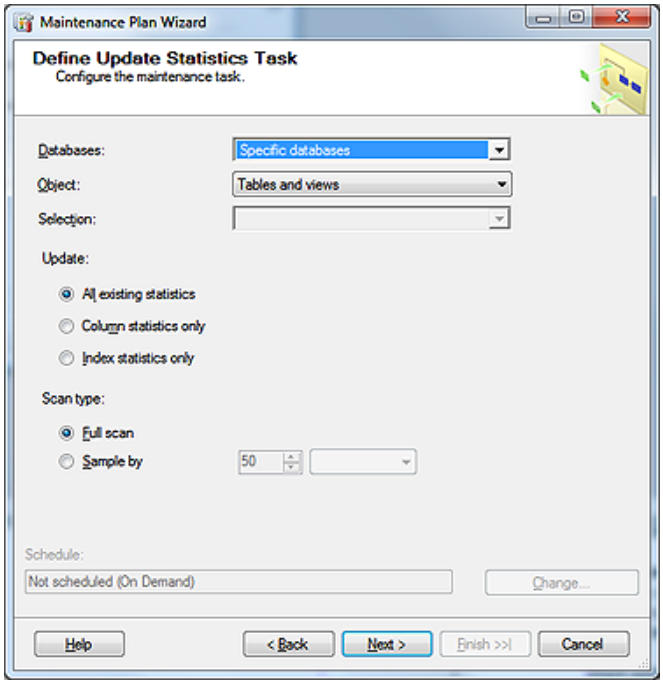
Step 09

Clicking on the **Next >** button will bring you to the **Define Rebuild Index Task** dialog form displayed, diplayed at the top of page 8. Again, this is not the default dialog form as I have already selected **Specific databases**, although everything else on this dialog form remains in its default state.



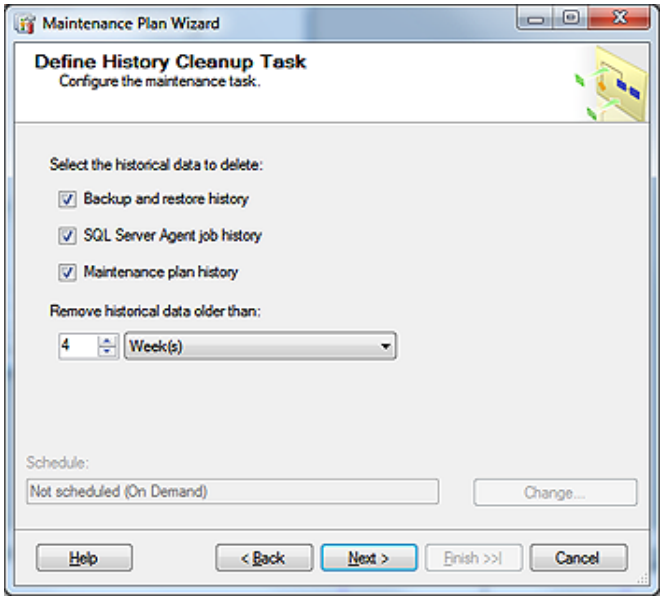
Step 10

Clicking on the **Next >** button will bring you to the **Define Update Statistic Task** dialog form. Again, this is not the default dialog form as I have already selected **Specific databases**, although everything else on this dialog form remains in its default state.



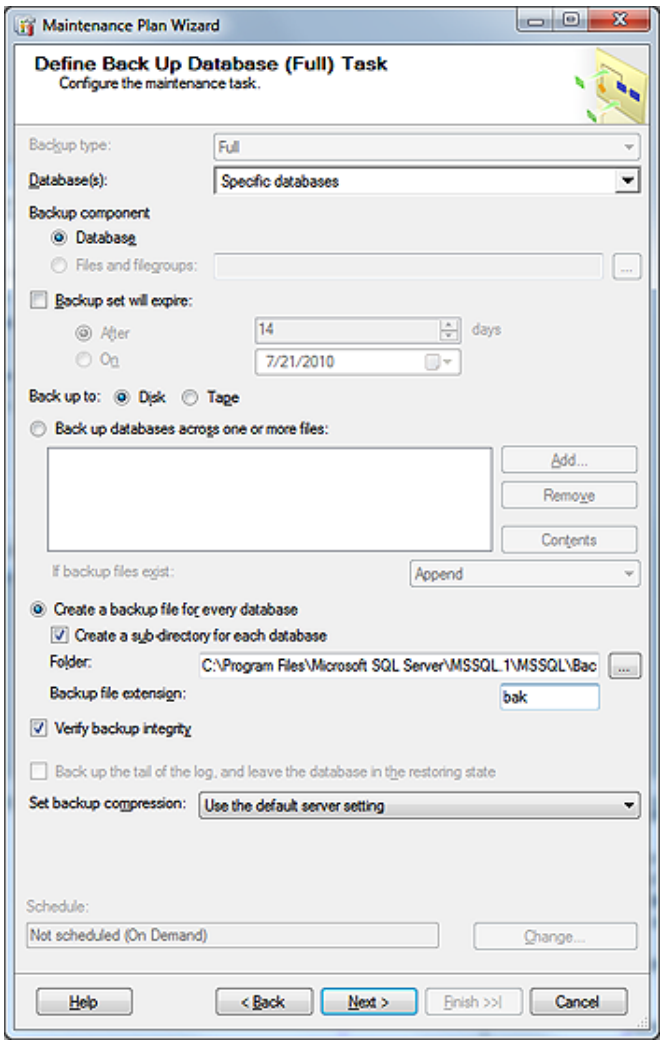
Step 11

Clicking on the **Next >** button will bring you to the **Define History Cleanup Task** dialog form show at the top of page 9. Everything on this dialog form remains in its default state.



Step 12

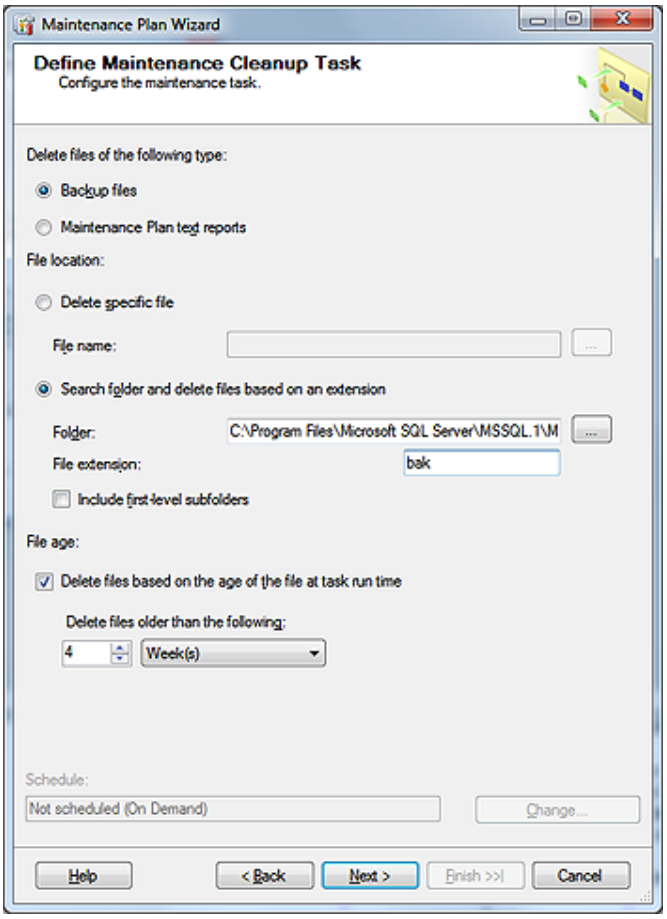
Clicking on the **Next >** button will bring you to the **Define Back Up Database (Full) Task** dialog form displayed in the next column. Again, this is not the default dialog form as I have already selected **Specific databases**. Unlike the other screenshots, I have made changes to the default settings on this dialog form.



For instance, I did select to ☑ **Create a sub-directory for each database**, and, as well, I did select to ☑ **Verify backup integrity**.

Step 13

Clicking on the **Next >** button will bring you to the **Define Maintenance Cleanup Task** dialog form:



In the default state for this dialog form the ⊙ **Search folder and delete files based on an extension** is selected, however, it is up to you to supply the **Folder:** to search in and the **File extension:** for which to search.

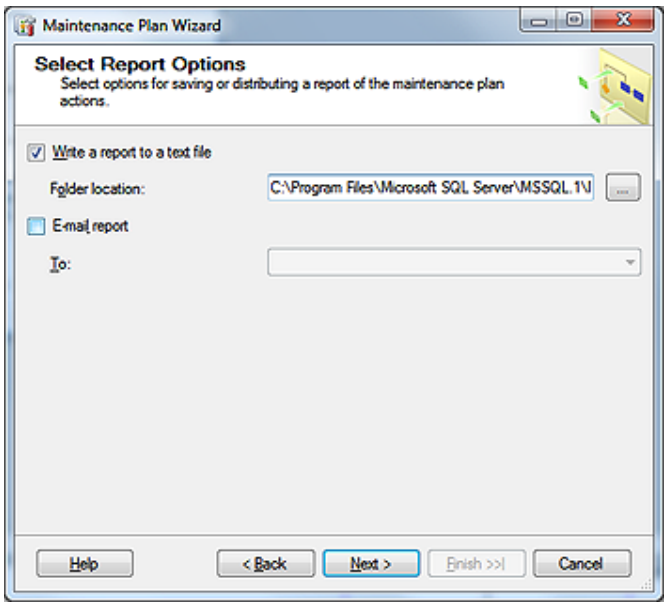
In my particular case I used the ellipsis (**...**) button to browse to my designated backup folder and the selection entered:

C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Backup\SQLGoldMine

I also entered the file extension of **bak** which is the default extension as accepted on the above **Define Back Up Database (Full) Task**.

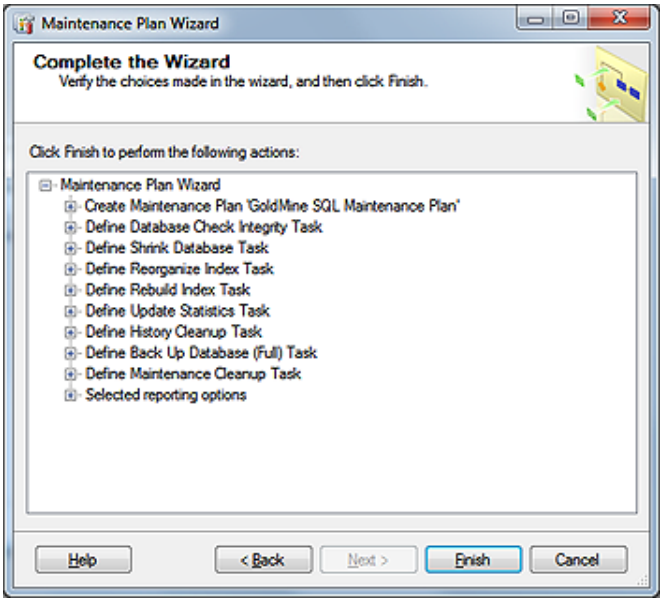
Step 14

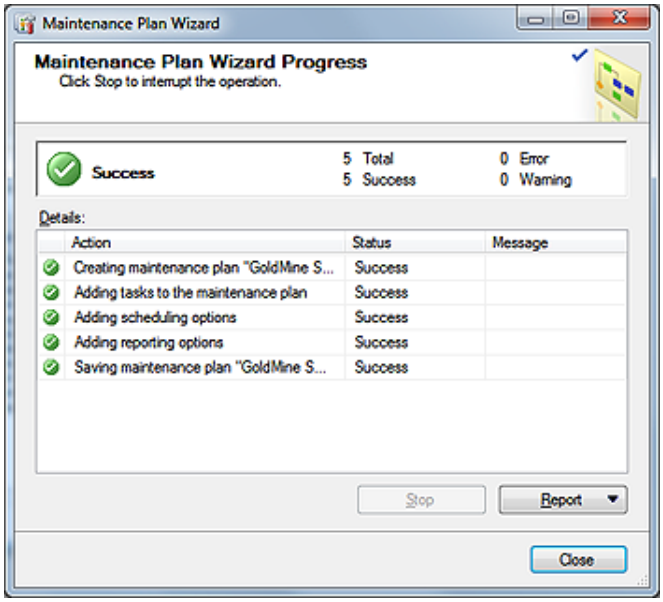
Clicking on the **Next >** button will bring you to the **Select Report Options** dialog form shown at the top of page 11. I usually just accept the defaults here, however, I have had instances where the clients had wanted to also have the report e-mailed to them. In those cases, I would have to select the ☐ **Email report** option while including an e-mail address in the **To:** field.



Step 15

Clicking on the **Next >** button will bring you to the **Complete the Wizard** dialog form:





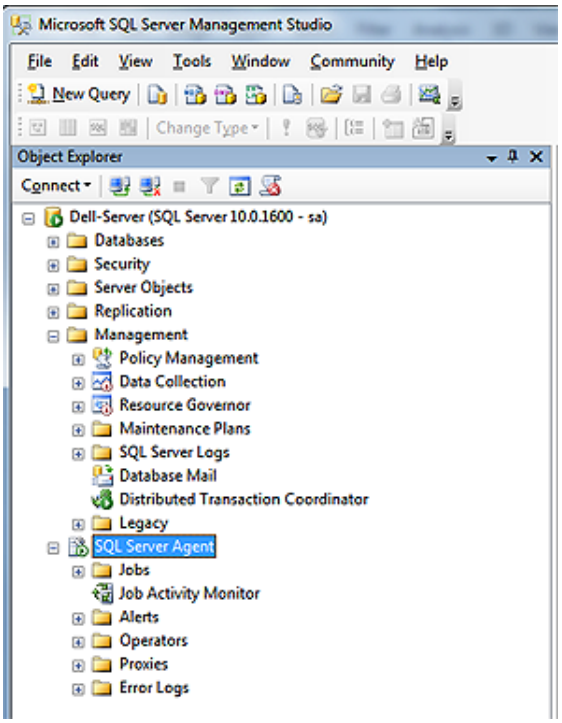
Finally, we have completed the Wizard, and we have little more to do than to click on the **Finish** button which will generate the **Maintenance Plan Wizard Progress** dialog form. Refer to final figure on Page 11. Hopefully, yours will also display the **Success** check mark. Clicking on the **Close** button, and you will have created your GoldMine SQL Maintenance Plan.

Step 16

3 days later you decide to check your **Job Activity Monitor**, and you notice that your GoldMine SQL Maintenance Plan has never run. You look in your SQL Backup folder and there are no backups to be found.

Why do I even bring this up, because just yesterday I had a client call me and tell me that the plan that I had created for them wasn't working, and hadn't worked since June 30th, 2010.

My first thought is to open SQL Server Management Studio to see if there was anything obviously wrong. Sure enough, there it was staring me right in the face. The **SQL Server Agent** had been **Stopped** as often happens when one stops the SQL Service itself. People remember to turn the SQL Service back on, but often forget that the SQL Server Agent, which is automatically stopped when the SQL Service is stopped, must be **Started**.



The dialog form above has the **SQL Server Agent** highlighted, and the icon to the left displays a green right facing arrow indicating that the service is **Started**. It is imperative that this service be started if you want any of your SQL Maintenance Plans to function at all.

Once I restarted the service for my client, the GoldMine Maintenance Plan ran Successfully again.