SQL Server Log Shipping: Replacing the Windows File Server with SQL File Tables for higher availability

Writer: Wolfgang “Rick” Kutschera

Technical Reviewers: Markus Wagner, Biljana Lazic, Hannes Ulrich, Sanjay Mishra, David P. Smith

Technical Writer:

Project Editor:

Originally Published: April 2012

Last Updated: April 2012

Applies to: SQL Server 2012

Summary: This white paper describes a solution of how to increase the availability of LogShipping configurations and protect the log backup chain in this configuration in both High Availabilty and Disaster Recovery manner by replacing the underlying Windows File Server with a SQL Server 2012 file table that is setup for HA/DR using the AlwaysOn Availability Group technology.

The solution can also be used with LogShipping on SQL Servers pre 2012, but the central FileTable component leverages SQL Server 2012 features.

Copyright

Contents

Introduction

LogShipping has been a valuable feature of SQL Server for a long time now, leveraged for increased availability, disaster recovery as well as a simple solution for taking and verifying log backups. The basic setup for logshipping looks like this:

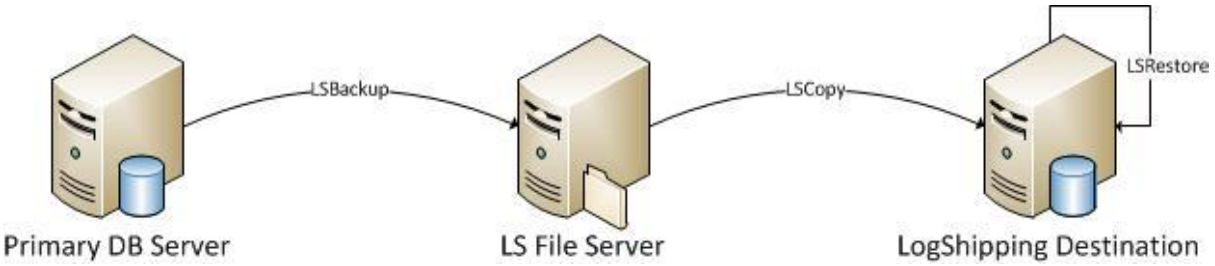

Figure 1: Standard LogShipping configuration

In simple terms the database on the Primary server is set for automatic log backups using the SQL Server Agent and the LSBackup job. Those backups get written to a fileserver. From there the Destination server picks those files up using his own SQL Server Agent and the LSCopy job, transferring the files to his local harddrive. As a last step the files are being applied (Restored) to the destination database using another SQL Server Agent job, the LSRestore. The standard interval for LSBackups is 15 minutes, but it can be set to any interval the user needs.

One interesting thing to note in this constellation is that the system includes its own cleanup and alerting mechanisms. LSBackup has a threshold (normally 24 hours). Any backup file on the LS File Server older than this date will be deleted automatically. The LSRestore has the same mechanism for its local copies, but ensures that those are only deleted after they have been applied successfully. There is also a SQL Server Agent job names LSAlert that runs on both the source and destination machine and monitors if log backups/restores are done regularly. If not the job will fail, raising an alert.

Another neat feature worth mentioning is the restore delay: Logshipping can easily be setup to wait for a defined amount of time (e.g. 60 minutes) before applying the logs to the secondary, giving the ability to step back from a fatal user or application error, like dropping a table. This is a feature that even nowadays has not been made available in any other HA or DR method.

In the time when logshipping was introduced it was a pretty good solution for the HA/DR needs. SQL Server failover clusters were very expensive to build and maintain and most companies still lived in a 8x5 manner, meaning that IT was mainly used during office hours and was not extremely vital to the companies success. Therefore a data loss of 15 minutes was more than acceptable given the extremely low price of the solution. It also proved to be perfect for customers with higher demands that were employing SQL Server failover cluster instances, as cross location clustering pretty much non existant at the time, making this the only viable solution for quick disaster recovery. Unfortunately the time has changed and businesses nowadays run 24x7, relying on IT to very extremes, making data loss an almost unacceptable thing. Technology has coped with this demand, and with the increase of affordable storage subsystems and the wider availability of SQL Server failover cluster instances the demand for

high availability has been covered, pushing the technology of LogShipping into the background, which might be one of the reason the technology has not changed too much in this area.

By now many users of SQL Server have identified the potential of logshipping for another purpose though: The use as a log backup solution. With it's ability to automatically cleanup the fileshare it basically implements the needs of rolling backups out of the box, removing the need for expensive backup solutions like the Microsoft Data Protection Manager, that would also add additional maintanence efford, hardware needs and training for the IT crew. Unfortunately the LogShipping system has a vital flaw as a backup solution. (Which is by the way also true for almost all other backup solutions in this area.) While full and differential backups can just be redone if one is lost, log backups can't. If you log chain misses even a single backup file, it is just not restorable anymore. SQL Server acknowledged this problem by introducing the "WITH MIRROR TO" option in the BACKUP LOG statement, which allows the backup to be written into two locations at once, but LogShipping does not deliver a way of utilizing this. Therefore in our highly available world of today a typical logshipping solution would look somewhat like this.

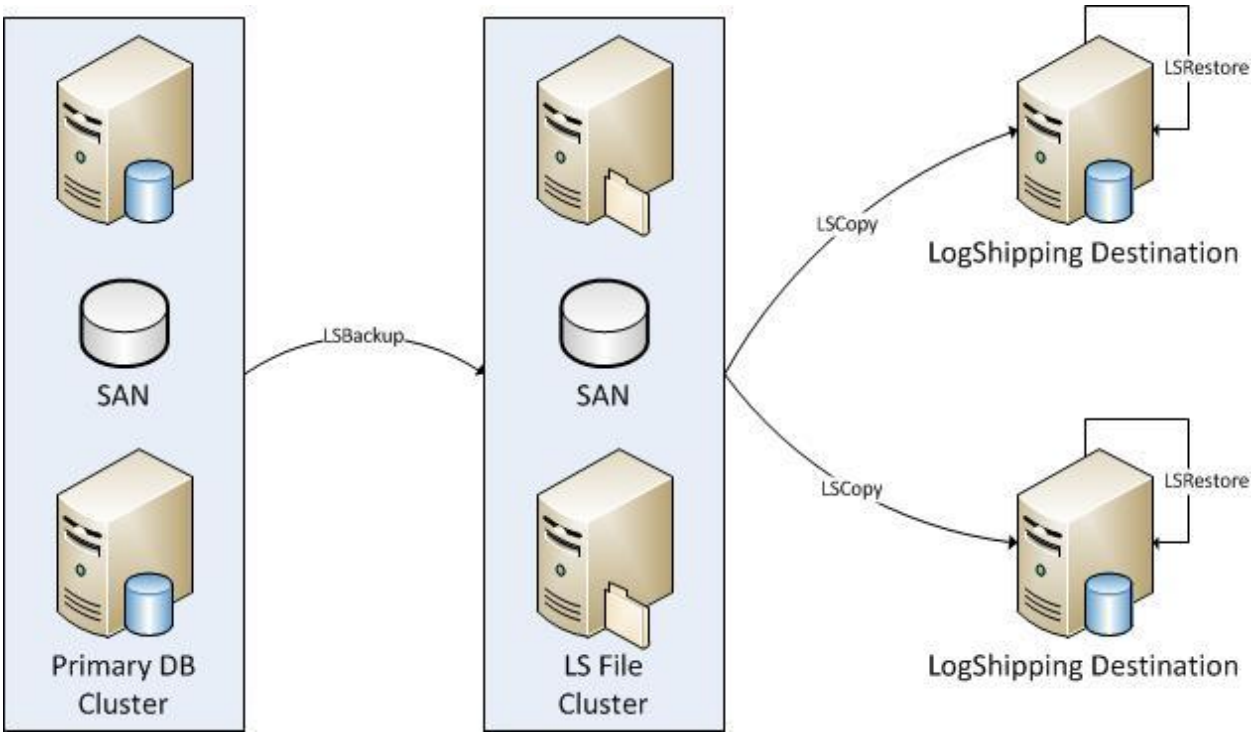

Figure 2: High availability LogShipping

This setup brings high availability to the LogShipping technology, leveraging Windows Server Failover Clustering and a storage area network (SAN). This bears many consequences though. For one, a SAN, although much cheaper these days than in the past, is still a huge investment and a technology that needs increased maintenance. For two, no matter how good your SAN is, even it bears the chance of breaking. And given that many companies only deploy one SAN for cost reasons chances are that if your SAN breaks you loose both the DB data as well as the log backups on the LS File Cluster. And for three, SANs are still mostly used for running servers within the same datacenter. Although technologies like EMCs SRDF or IBMs MetroMirror

provide SAN based cross location data sync capabilities those are normally only affordable for the top brass of customers, and they come with a lot of limitations.

So it comes as no surprise that Microsoft found a way to eliminate the SAN as a single point of failure in their database system and with release of SQL Server 2005 introduced Database Mirroring. Leveraging this technology you can now setup cross location high availatility and disaster recovery without the need of expensive hardware, changing the scenario like this.

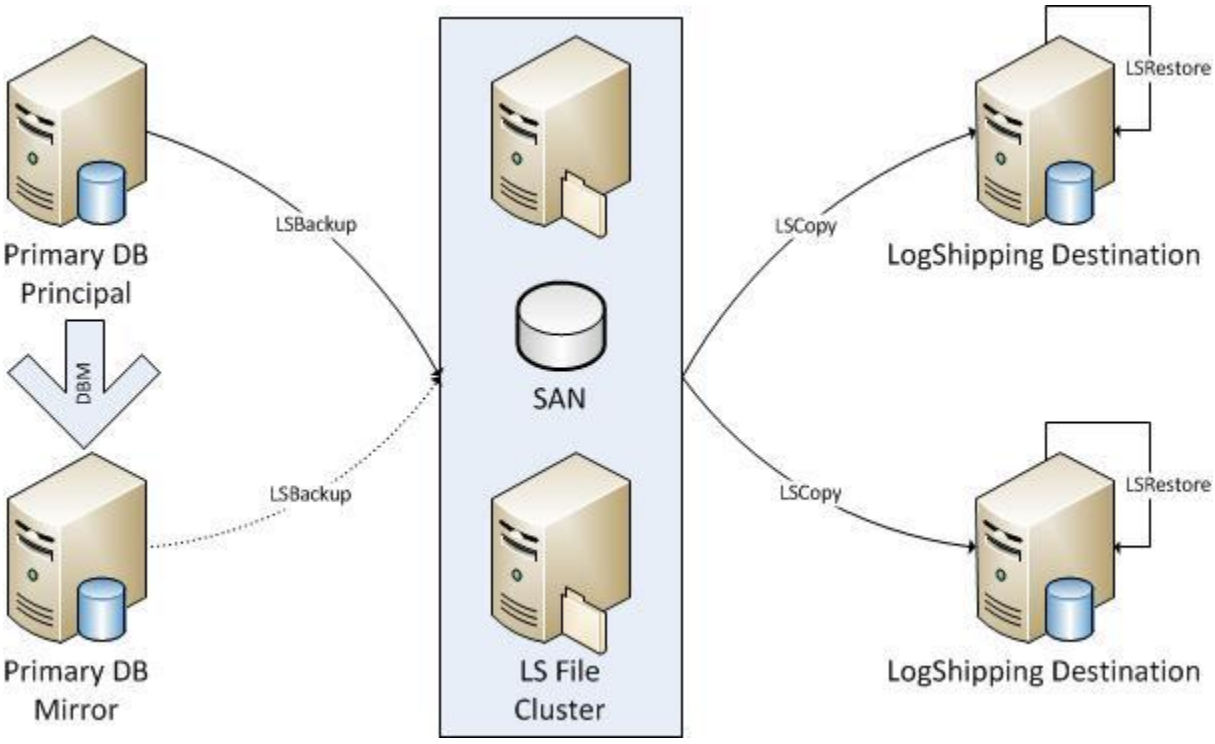

Figure 3: LogShipping with Database Mirroring (DBM)

Now the one component that remains in this picture that is in need of a SAN as central storage, and with it hosts a single point of failure in the SAN and a large cost factor if deployed across multiple locations, is a simple, old fashioned File Server. And this is where SQL Server 2012 comes in handy…

SQL File Tables

A while back, with SQL Server 2008, Microsoft introduced the concept of FileStreams. The basic idea was to provide a BLOB store within the SQL Server database that could be accessed by legacy applications using the standard Win32 method of an SMB share while also allowing easy access to the data using Transact SQL. With SQL Server 2008 FileStreams had two drawbacks though: You could only have one per database, plunshing all your files into the same stream, and, more importantly, you could not have a highly available access to those shares, unless you employed a SQL Server Failover Cluster Instance using a SAN. With the release of SQL Server 2012, SQL File Tables and the Always On Availability Group technologies those

drawbacks have been removed, creating a system complex where FileTables can easily be made highly available without the needs of a SAN and can even be made highly available across multiple datacenters using AlwaysOn Availability Groups on top of a Multi-Subnet Windows Server Failover Cluster.

SQL File Tables and LogShipping

Now, with the use of the new FileTable technology my proposal of a LogShipping setup looks like this.

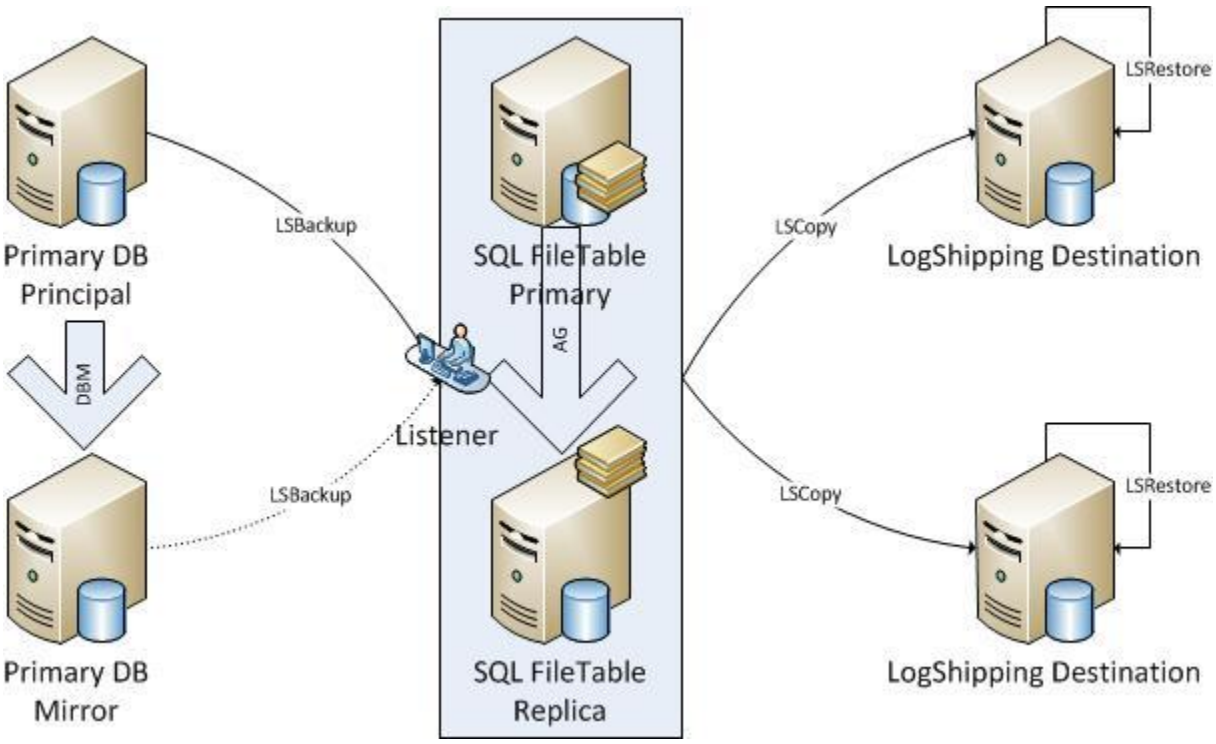

Figure 4: LogShipping with Database Mirroring (DBM)

Looks similar to the one above? Right… But with one major difference: Although the filetable in the middle looks like one big data store to both LSBackup and LSCopy (made possible by the use of the new AlwaysOn Availability Group Listener), it really exists as two copies, one on each node of the FileTable Cluster, leaving the system with no shared components and therefore no single point of failure. And of course (as mentioned earlier) with this solution you are not bound to one datacenter. As there are no shared components in this solution you can easily split it across two, or even more, datacenters.

Setting up the FileTable Availability Group

The picture above might look pretty simple, but looking at it in detail brings a quite different impression unfortunately. First of all, as with all AlwaysOn technologies, you need to deploy a Windows Server Failover Cluster as a base for your system and with it spend a lot of thoughts

around a quorum model and voting strategies that fit your needs. (This article will not go into details on that, there are other great papers out there that cover those areas.) So for the purposes of the document let's assume that you have already deployed WSFC, installed stand alone instances of SQL Server on each of the nodes and enabled AlwaysOn High Availability. Just to make sure that we are on the same page, it's that little button here:

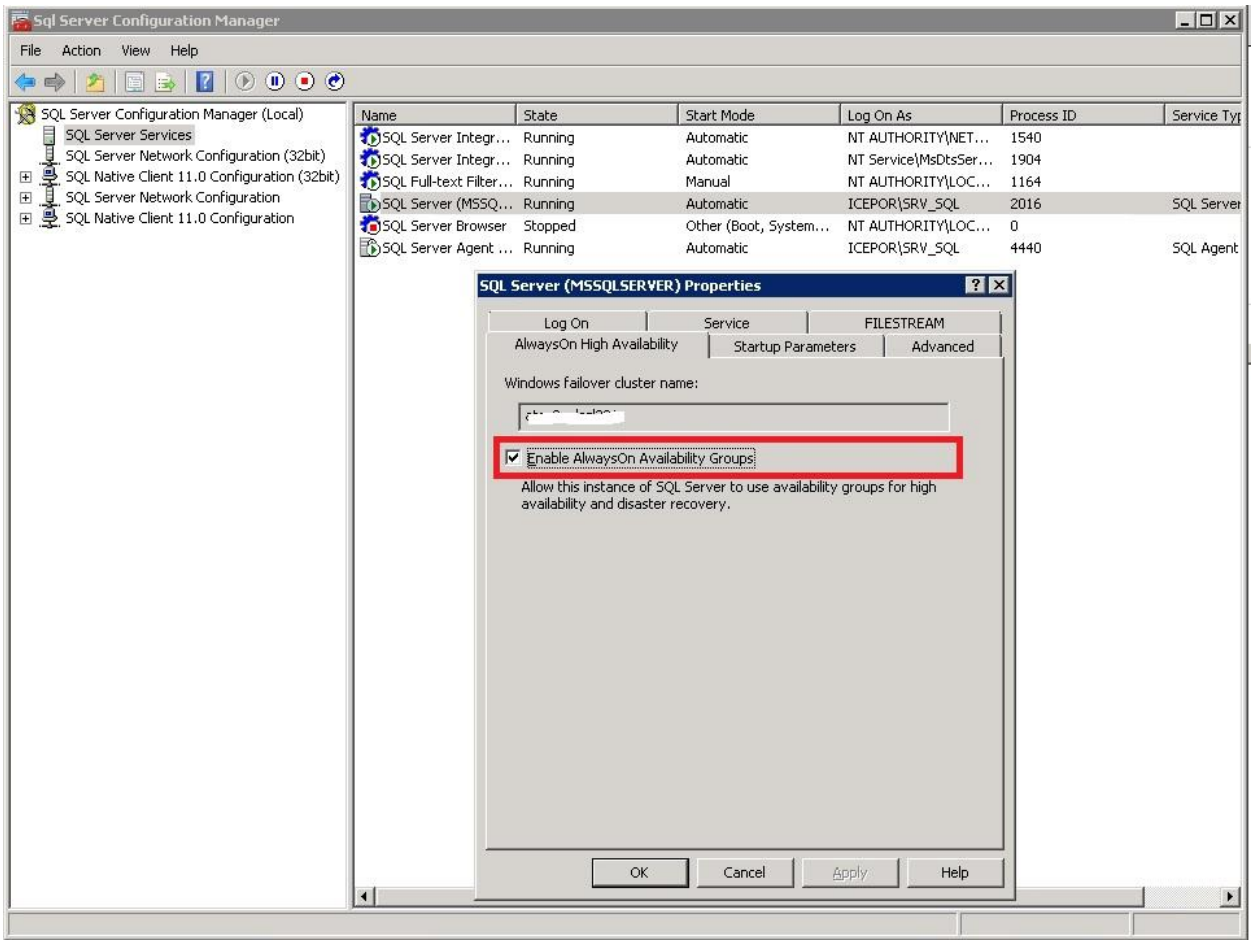

Figure 5: Enabling SQL Server AlwaysOn High Availability

Now, the next thing you need to do is enable FileStream. Look, it's not far from the AlwaysOn switch…

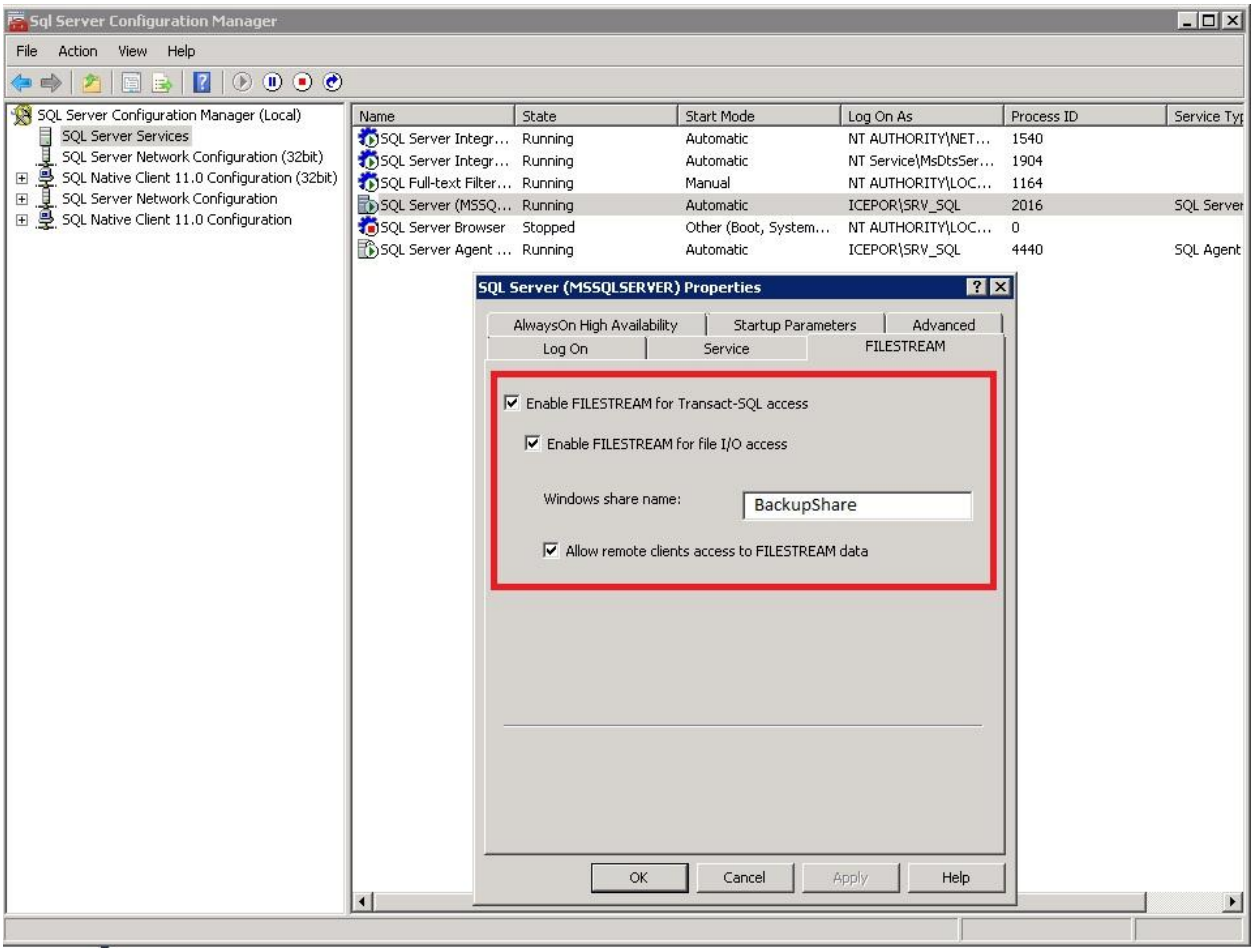

Figure 6: Enabling FileStream

For this scenario there are some important parts to mention at this point. For one, you have to enable all options. The really important one is the last, “Allow remote clients access to FILESTREAM data”, and this one is only available after you enabled the other two. For two, the Windows Share name is important. This is the SMB share that SQL Server will attach the FileTable to. This has to be set to exactly the same value on all instances hosting the FileStream, otherwise the system will not work. And last, after changing those settings you need to restart the SQL Server instance, so you should do this before you start using the server…

To check if your setup step worked you should now be able to access your FileStream on the nodes via SMB. In my sample I could now browse to \\localhost\BackupShare on the server and would find an empty folder there.

Once this is working on all nodes the next step is creating a database. The database needs to run recovery model “Full”, otherwise the AlwaysOn Availability Group will not work.

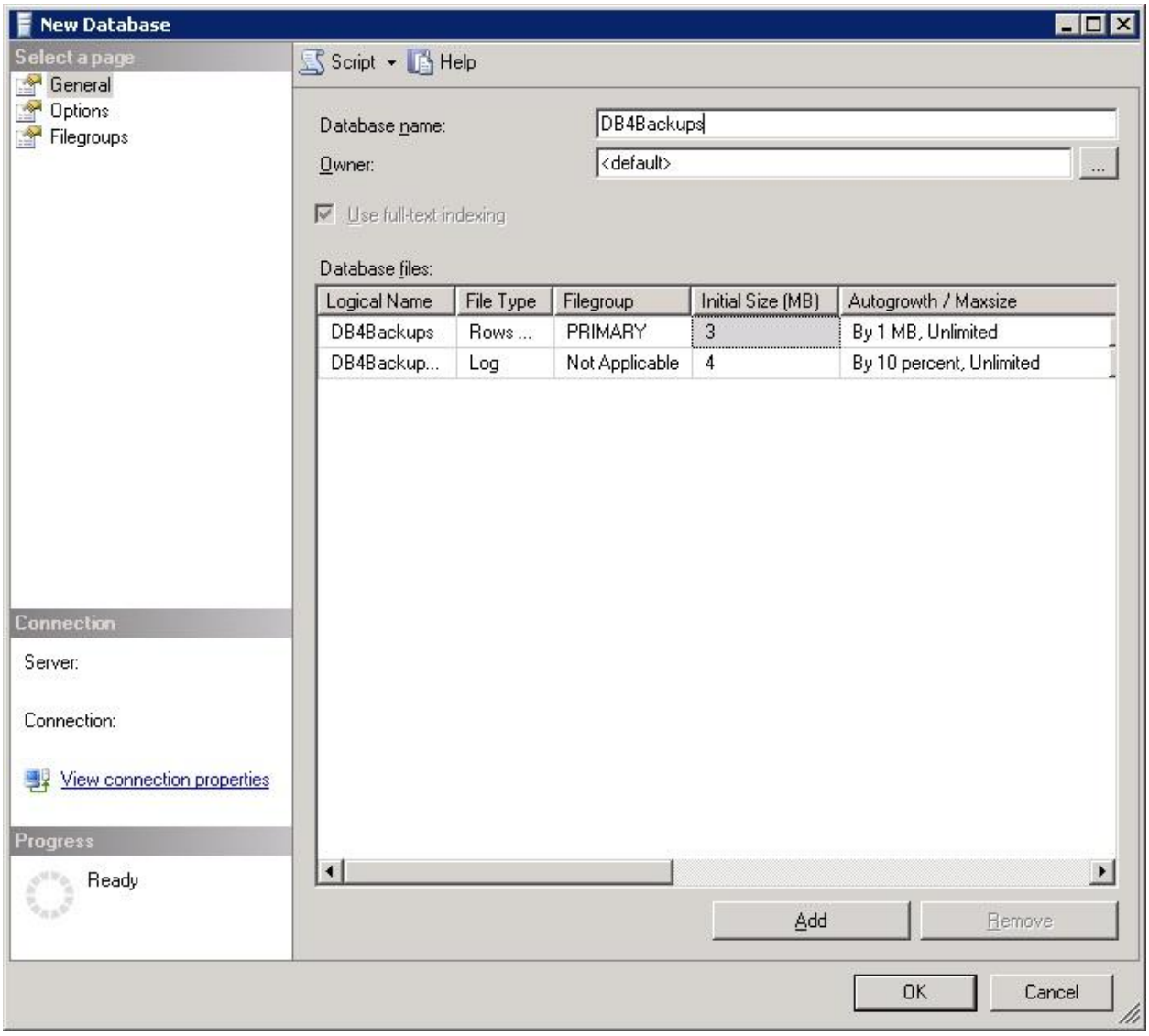

Figure 7: New database

You might of course want to change the file paths and the growth settings, etc. to your standards. Next… Build high availability around the database. Otherwise the whole setup would be pointless. And here is one more feature that is nice in SQL 2012, compared to the older versions: There is actually a really usefull GUI support for doing that… Unfortunately to start the process we still need to run one command manually…

```
BACKUP DATABASE DB4Backups
TO
DISK='C:\Temp\DB4B.bak'
WITH INIT, FORMAT, COMPRESSION
```

Don't ask me why they didn't build this into the Wizard… Well, anyway… After running this we can start with the cool new wizard.

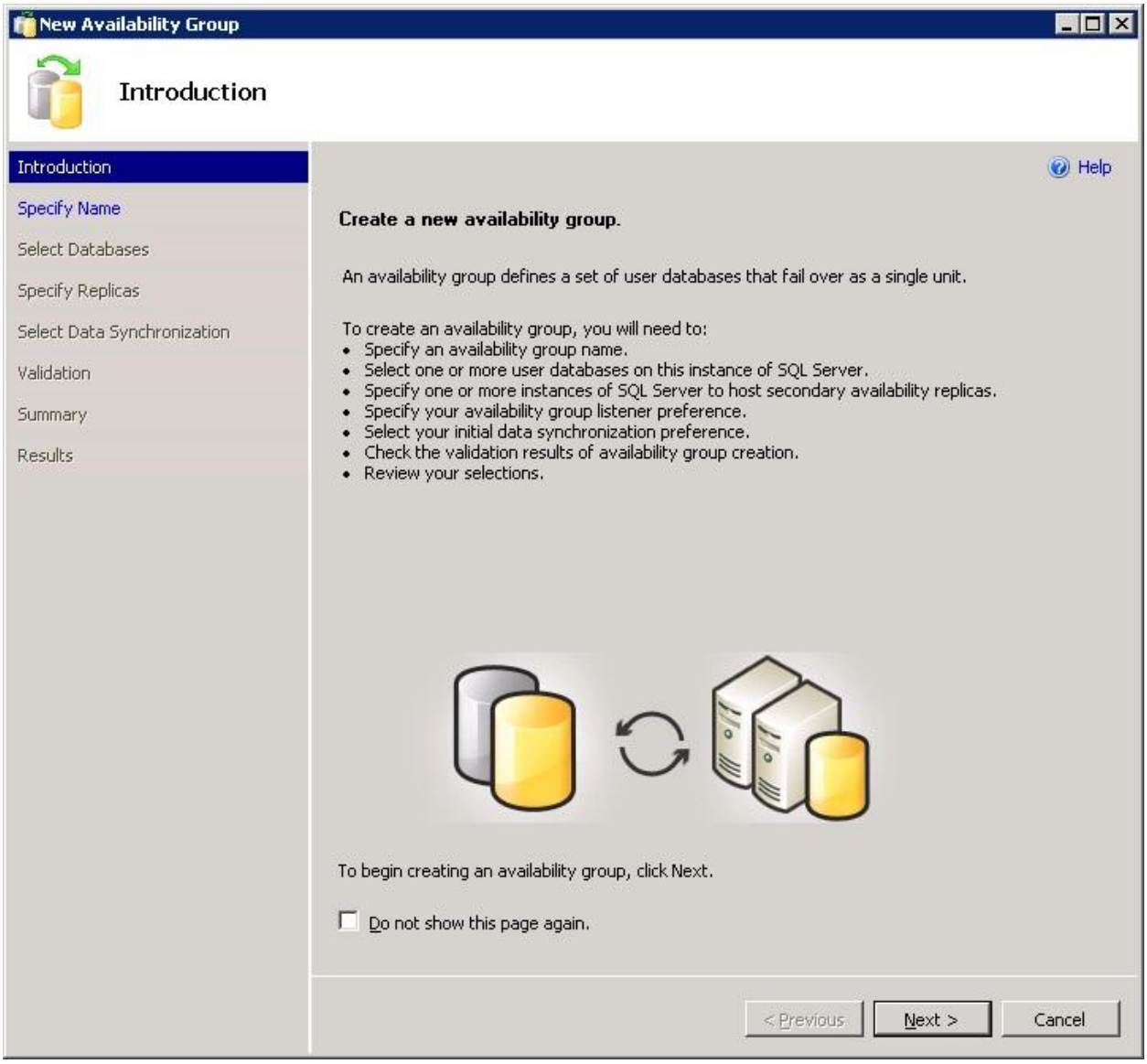

Figure 8: Create Availability Group Wizard

This is not really an AlwaysOn document, but just for the sake of completeness, let's walk through it… After hitting next on the start screen you are asked to specify a name for your AlwaysOn Availability Group. This name is also used as a cluster group name.

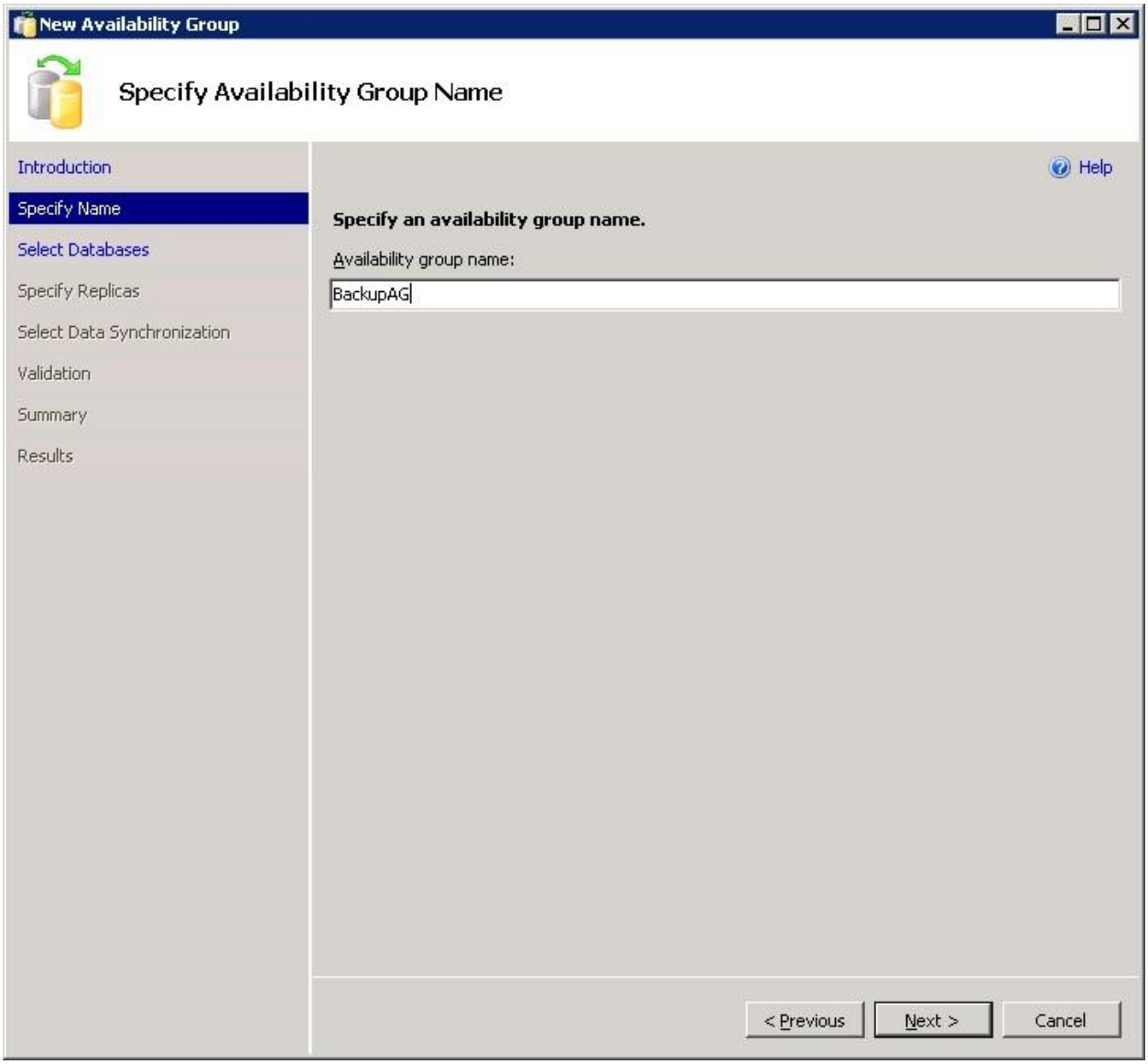

Figure 9: Specify name for availability group

Now next… And we are asked to select all DBs that will participate in the group. In this case it's only one… And look, it really meets the prerequisites…

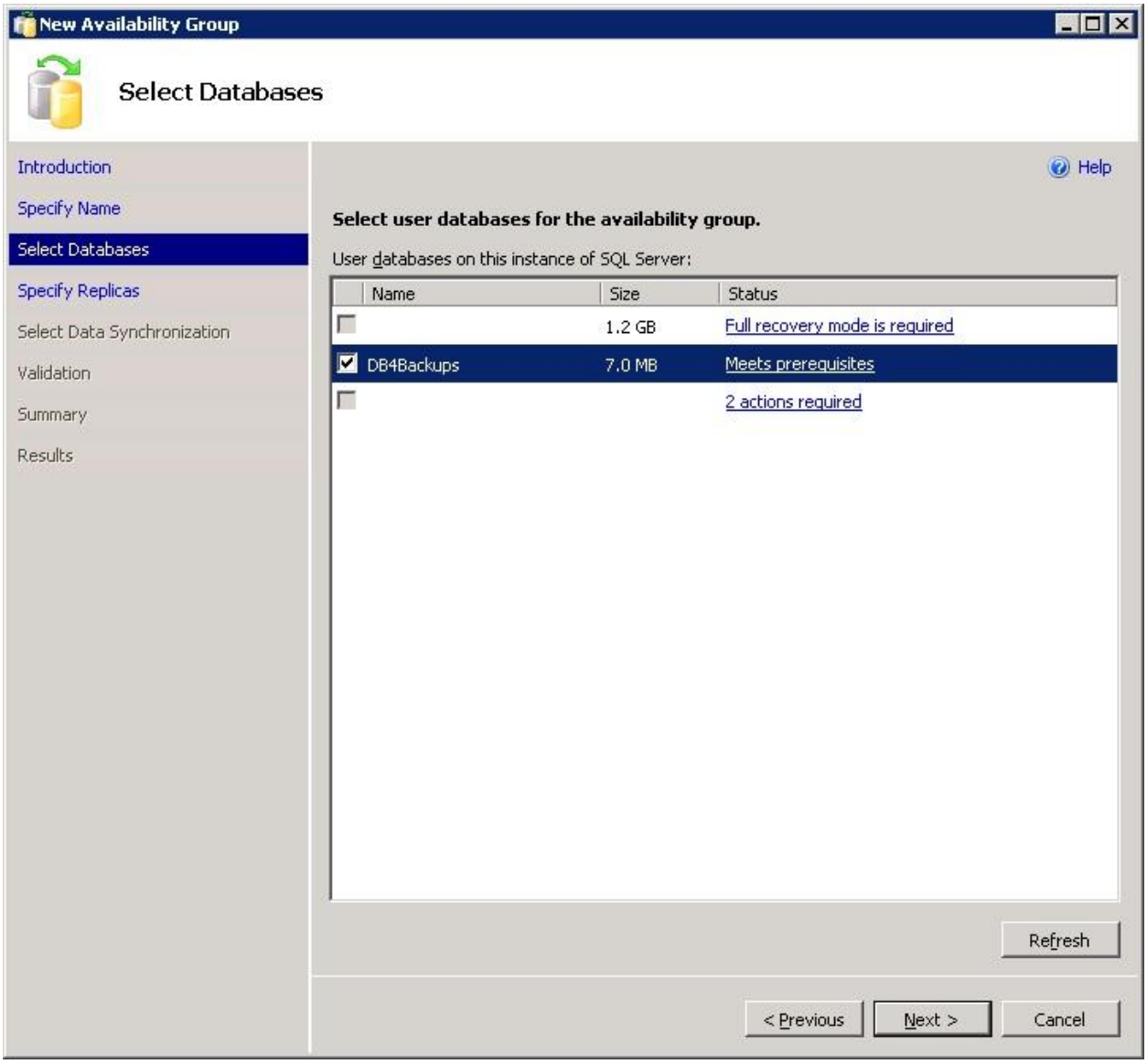

Figure 10: Select databases for Availability Group

In the next step we need to specify all servers that should hold copies of the data. In my picture I showed two, but you can have more than that if you like.

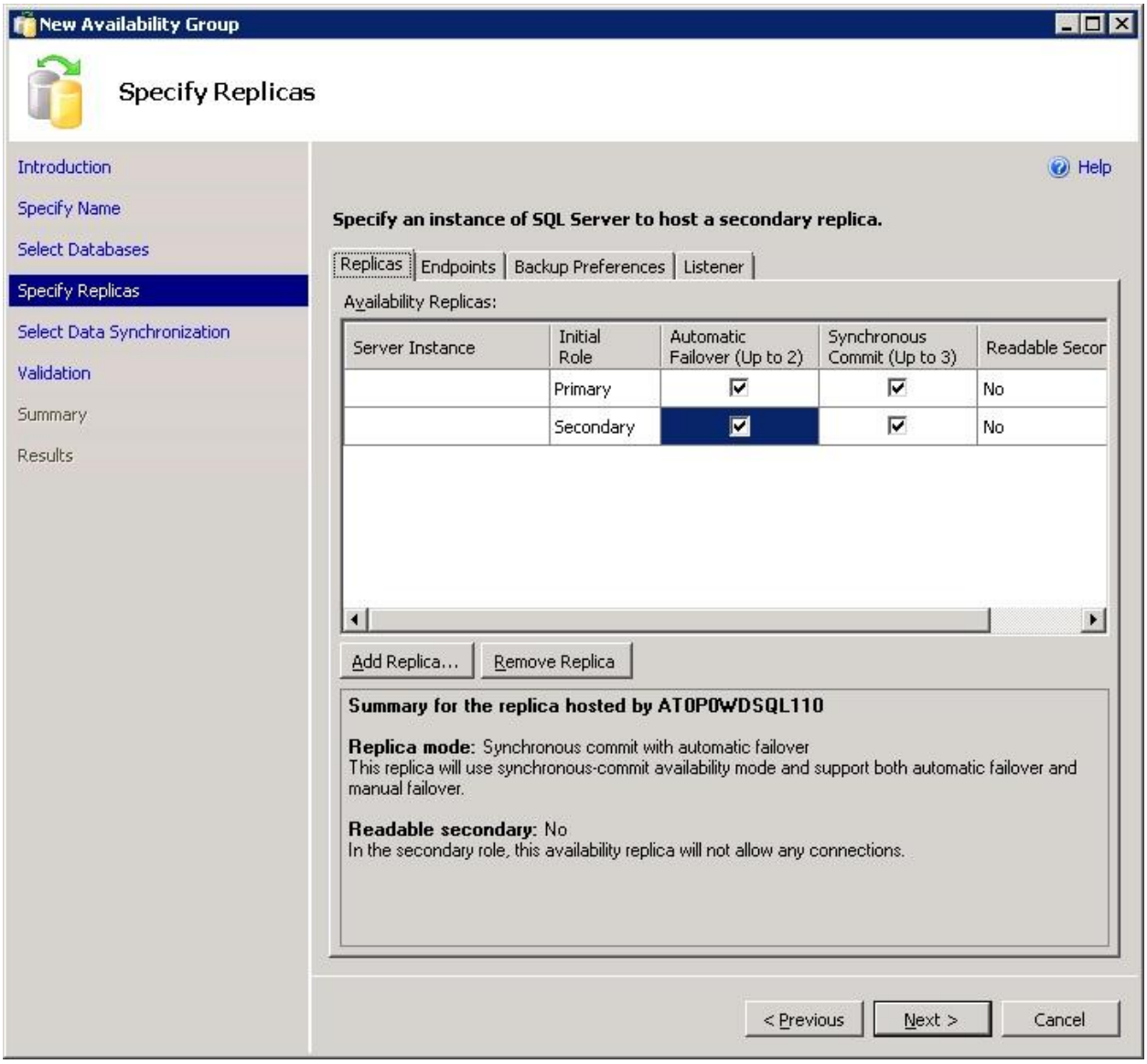

Figure 11: Select replicas for Availability Group

Note that I have specified both replicas to be Synchronous and Automatic Failover. It might seem counter-intuitive that you have to specify Synchornous commit on the primary, but that's what it is. (And I am not going to discuss the details on that, let's leave this for another document.)

If you have permissions to create Active Directory objects in your organization you can now switch to the "Listener" tab and specify the DNS name that your clients will connect to lateron.

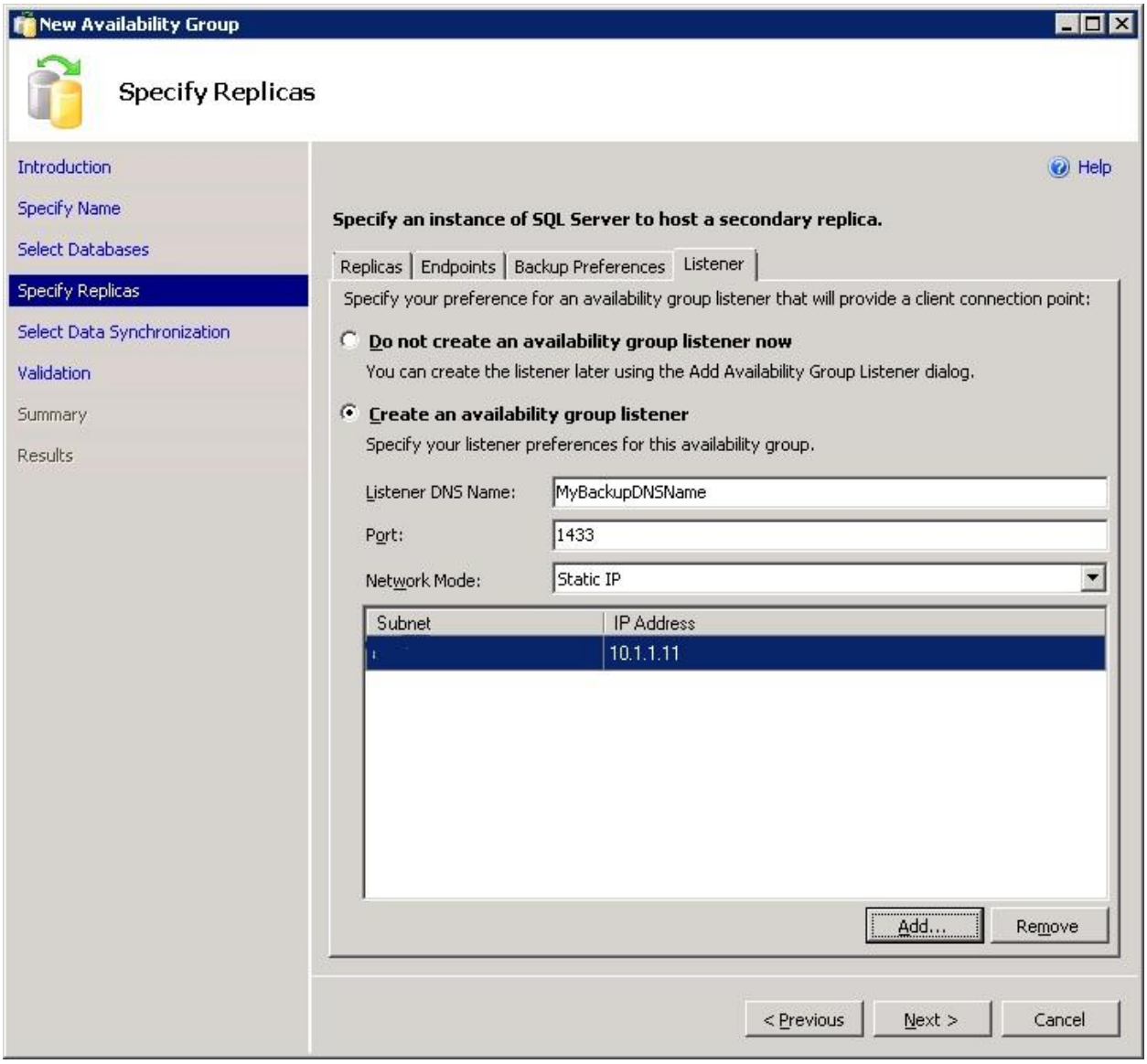

Figure 12: Availability Group Listener

If your company has segregation of duties and you can't do this yourself there is of course a way to do it later.

After hitting next you are asked to specify a network share that all replicas can access. This is needed for the wizard to store a full + transaction log backup from the primary to apply to the replicas before initializing the group. So why did we have to do the Backup before? Don't ask…

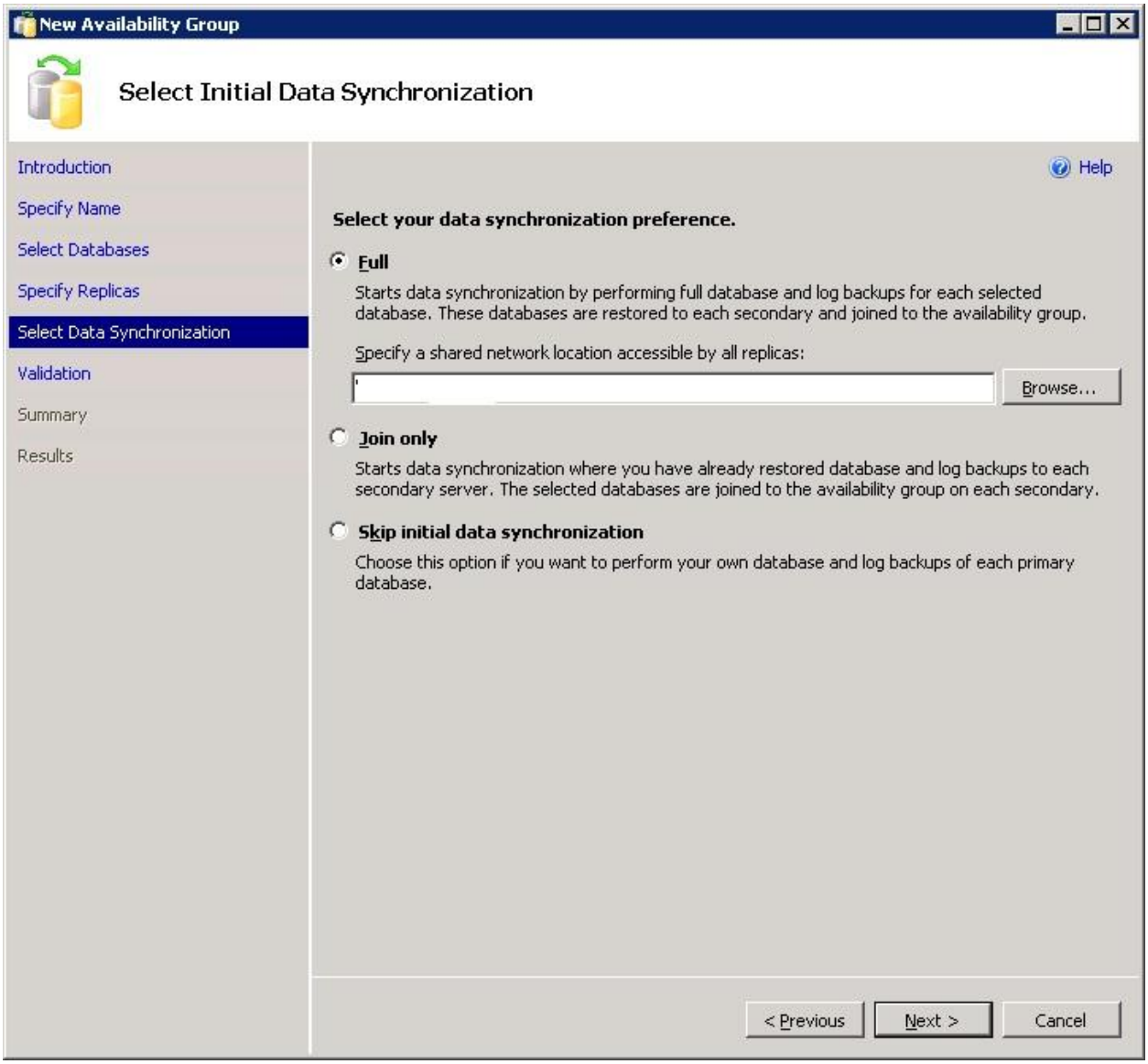

Figure 13: Availability Group Initial sync

OK, almost done. Now the Wizard checks all the settings…

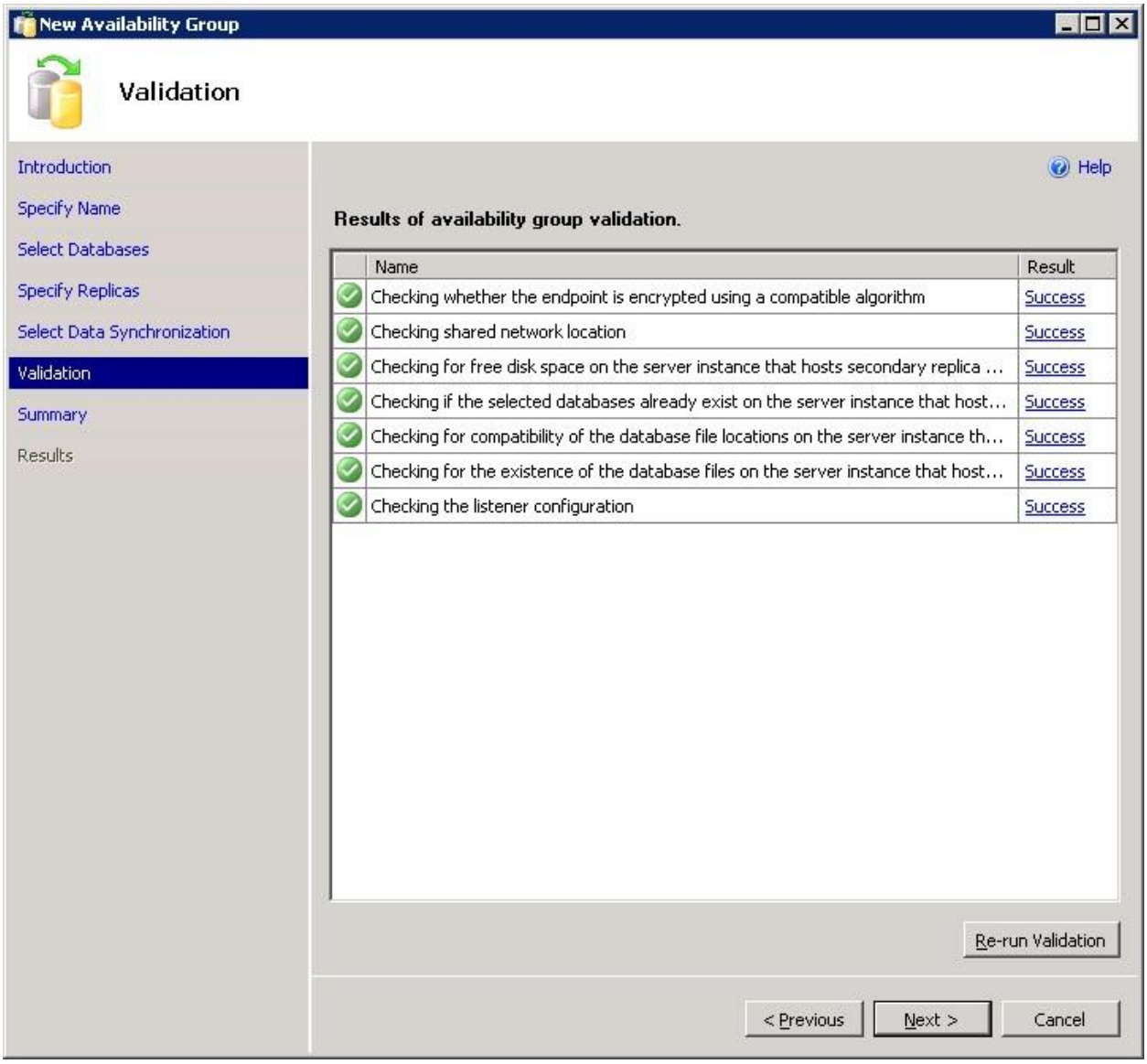

Figure 13: Validate Availability Group settings

And after hitting next another two times your new AlwaysOn Availability Group is created and your databases are in sync. So now we can start with the real configuration…

First you need to create a Filegroup for your stream data.

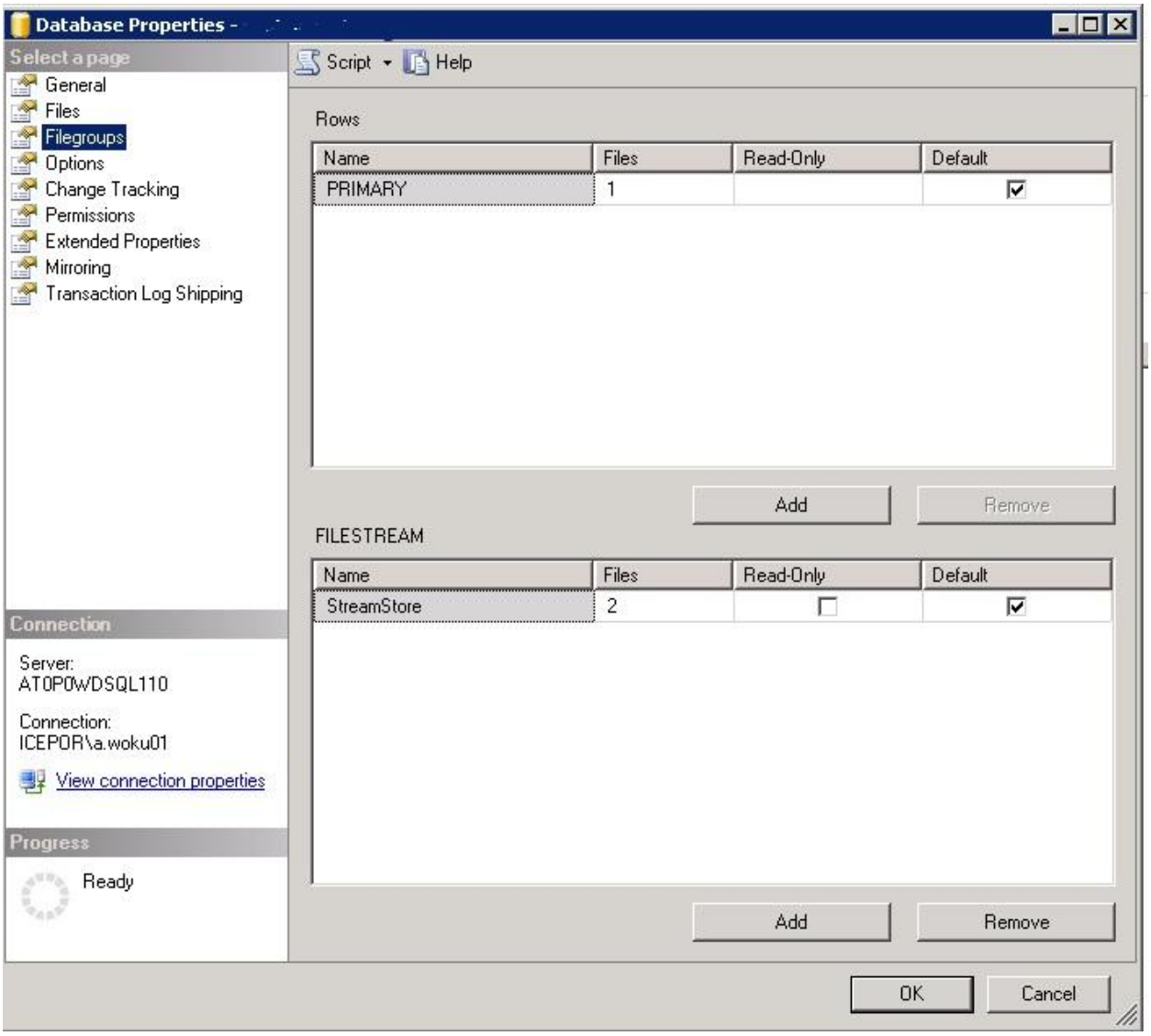

Figure 7: Filegroup for FileStream

Notice that the new filegroup is not in the “Rows” section, but in the “FILESTREAM” section.

Next we switch to the “Files” section and create a new file for the Stream-Data

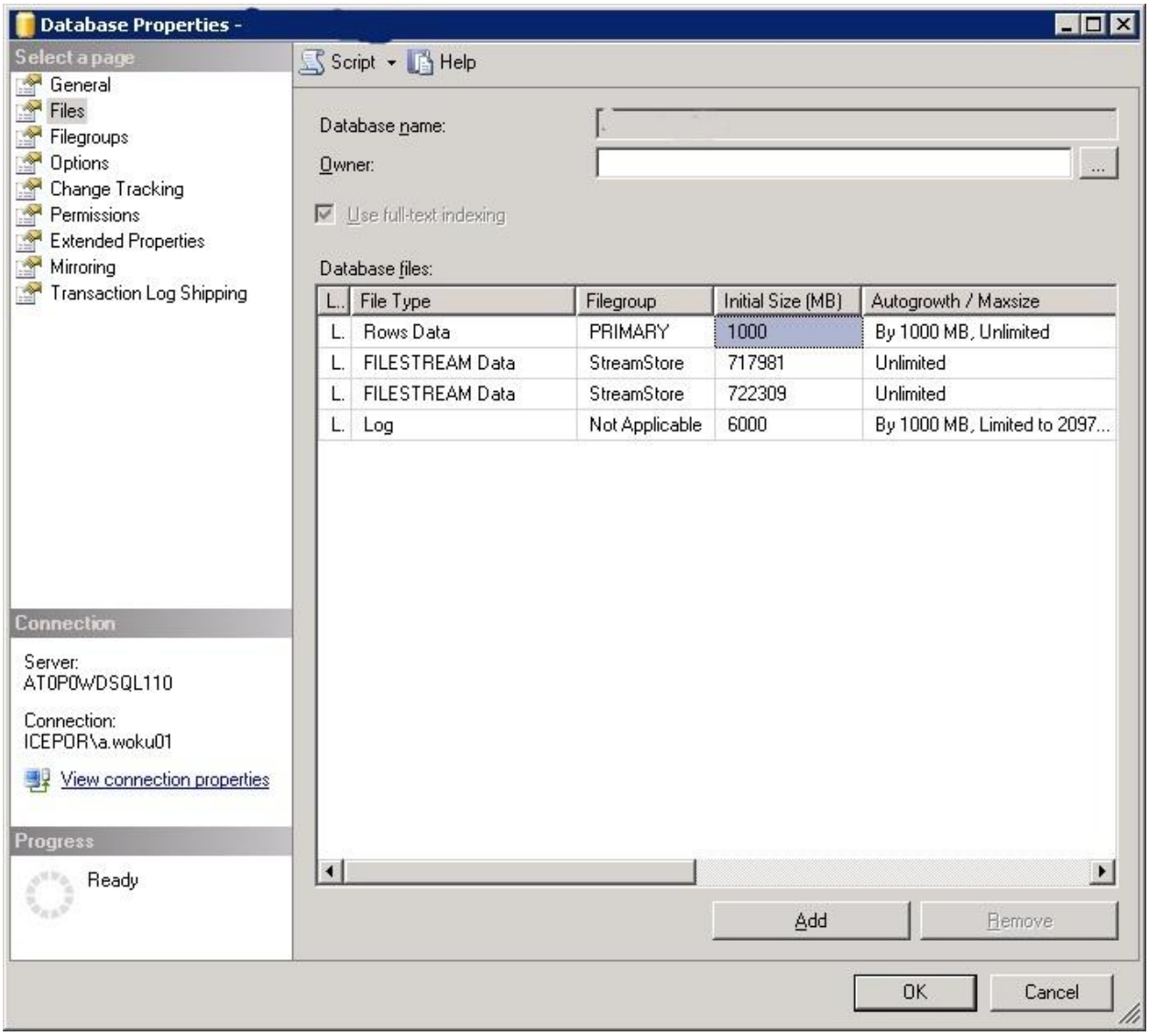

Figure 8: File for FileStream

Notice that I created two FileStream Data files. This was purely for space reasons on the machine I took the screenshots from. One is enough for the general purpose.

Next we need to enable the filestream access for the database.

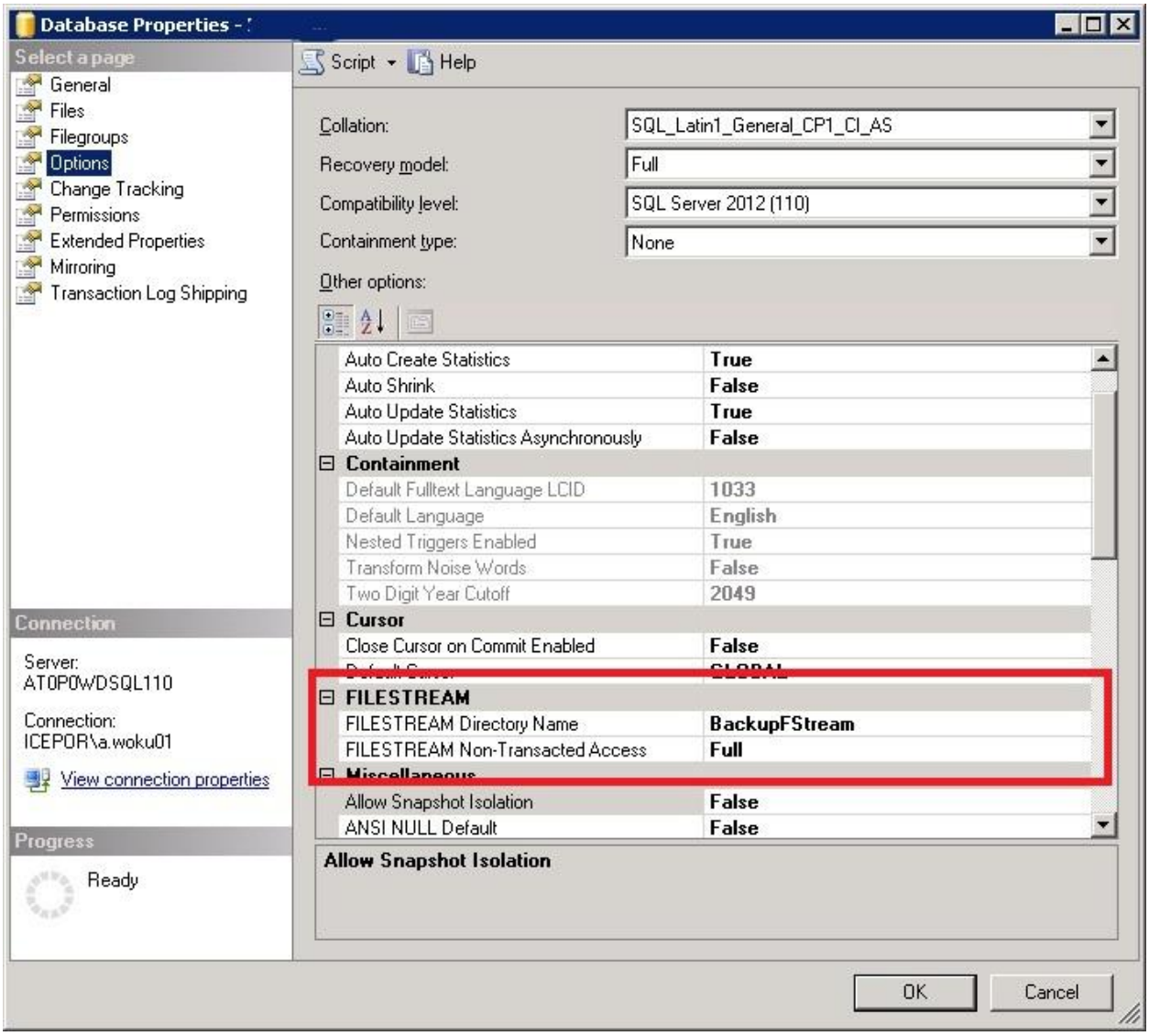

Figure 9: Database Options

You need to specify a directory name of your choice (I would recommend one without blanks…) and importantly set the “FileStream Non-Transacted Access” to Full.

If you have done things right to this point you now have a Stream-Access ready that can be tested. To do that you can now (from any machine) point your Windows Explorer to \\MyBackupDNSName\BackupShare and should see a directory there named “BackupFStream”. Remember… BackupShare is the name that I gave the SQL Instance when configuring FileStream on an instance level. BackupFStream is the subshare for the FileStream in your particular database. And this is available via the VNN of the AlwaysOn Availabilty Group because those two work in common now, recognizing that the database is using both technologies and binding them together.

Now… Last step… Create a File table in the database (Or more, if you want to split your files up logically.) There is no GUI there, here is the code:

```
CREATE TABLE dbo.BackupFileTable AS FILETABLE
  WITH
  (
    FILETABLE_DIRECTORY = 'BackupTableDirectory',
    FILETABLE_COLLATE_FILENAME = database_default
  )
```

You will notice that there are two names there… "BackupFileTable" is the name that is used by Transact SQL commands to read the data from it, BackupTableDirectory is the name that you will see pop up in your BackupShare once you have issued the command.

So… Now you are all set. All that is left to do is point all your LogShipping configurations to \\MyBackupDNSName\BackupShare\BackupFStream\BackupTableDirectory and you are done. I would strongly urge you to either create directories within the FileTable or have one FileTable per LogShipping (I prefer the first.), otherwise you will end up in a mess of files. But this is nothing new, it was the same with the original LogShipping Fileserver.

One very last thing: Where to store the transactionlog backup files of the new database? Well… You could of course build another cluster like this and store them there, honoring the Mythbusters saying: "If it's worth doing, it's worth overdoing.", but I would stop at some point…

Side note for large deployments

While writing the document I ran into a performance problem within a large deployment of that solution. After a while of debugging (with the great help of Microsoft Customer Support Services) we figured out that the problem was not within the FileTable itself but within the NTFS filesystem.

The FileTable operates in a way where files are not written into the typical MDF structure that you are used to from SQL. Instead FileTable uses a special directory where it creates sparse files, and a lot of them with fancy names. Now… per default (in Windows 2008 R2) Windows has a compatibility mode enable on all NTFS volumes that forces it to create a unique 8.3 filename for every single one of those sparse files. And once you hit a certain amount of files this is getting awefully slow… (And even crashes at a certain point when no more 8.3 names are available…)

The workaround/solution for this is to disable 8.3 compatibility on the volumes in question. Simply open a command prompt and issue

```
fsutil 8dot3name set 1
```

This will disable 8.3 on ALL volumes. Alternatively you can do it on one volume only of course.

```
fsutil 8dot3name set <Drive Letter or Mountpoint Path> 1
```

After setting this you need to reboot the server.

Conclusion

This whitepaper shows how to use SQL Server 2012 AlwaysOn Availability Groups and FileTables to achieve high availability of a LogShipping fileserver without the need of special hardware or SAN infrastructure. It outlines the reasons behind the approach and all steps necessary to set it up, except for the setup of AlwaysOn.

For more information:

http://www.microsoft.com/sqlserver/: SQL Server Web site

http://technet.microsoft.com/en-us/sqlserver/: SQL Server TechCenter

http://msdn.microsoft.com/en-us/sqlserver/: SQL Server DevCenter

http://sqlcat.com/ : The SQL Customer Advisory Team

http://sqlblog.dangerous.it/ : Ricks Blog

Should you have any questions or feedback regarding the document or the steps outlined in it, please do not hesitate to contact me via email: rick@dangerous.it

Change History

Date	Description
April 23, 2012	Update of Reviewerlist after Feedbacks
March 24, 2012	Changed FT solution picture to clearer reflect the idea. Also added hint about 8.3 limitations we just stumbled upon
March 20, 2012	Added availability group creation as suggested by a reviewer
March 19, 2012	Draft completed.