

# SQL Storage Compress and the SQL Server IO Reliability Program

Jeffrey Aven

## Table of Contents

1	Executive Summary .....	3
2	SQL Server IO Reliability Program Overview .....	3
3	HyperBac Architecture Overview .....	4
4	SQL Server IO Reliability Program Requirements .....	7
4.1.0	Windows Logo Certification .....	7
4.1.1	Core Windows API Support .....	7
4.1.2	Stable Media .....	7
4.1.3	Forced Unit Access (FUA) and Write-Through .....	8
4.1.4	Asynchronous Capabilities .....	8
4.1.5	Write Ordering .....	8
4.1.6	Torn I/O Protection .....	8
4.1.7	NTFS Support .....	9
4.1.8	Testing .....	10
4.2.2	Transactional Sector/Block Rewrites .....	13
4.2.5	File Streams .....	13
5	References .....	14

## 1 Executive Summary

The SQL Server IO Reliability Program is designed for independent third party vendors to provide specific information on how their solution satisfies core program requirements for reliable, high availability SQL Server storage systems. Specific information on the Always On program can be found at:

<http://www.microsoft.com/sqlserver/en/us/solutions-technologies/mission-critical-operations/high-availability.aspx>

This document describes how the SQL Storage Compress solution by Red Gate Software has been reviewed against and satisfies the program's core requirements.

SQL Storage Compress is built upon the HyperBac™ architecture which consists of a random access file system compression/encryption filter driver which is based upon the Microsoft Installable File System (IFS). SQL Storage Compress and the HyperBac architecture allows administrators to run Microsoft SQL Server database data files on compressed media, reducing read and write disk I/O and recovering storage space.

A key design objective of the HyperBac architecture and the HyperBac based products is providing the same durability, consistency and reliability as native Microsoft SQL Server IO operations.

This document is intended for systems or storage administrators and/or architects involved in the design, implementation and support of high availability solutions incorporating Microsoft SQL Server and HyperBac technologies including SQL Storage Compress. This document assumes knowledge of Microsoft SQL Server architecture and familiarity or knowledge with storage design and architectural principles.

Although this document primarily documents the HyperBac architecture in the context of SQL Storage Compress, all other solutions based upon the HyperBac architecture including the SQL HyperBac and SQL Virtual Restore products satisfy the same program requirements.

## 2 SQL Server IO Reliability Program Overview

*What the program is:*

The SQL Server I/O Reliability Program is a partner program that enables providers which handle SQL Server I/O to highlight their solutions and configurations via the SQL Server I/O Reliability labeling which they have successfully reviewed against core functional Microsoft SQL Server storage requirements.

The core requirements defined on this page must be met for reliable systems which interact with SQL Server I/O. The program does not define or review the performance characteristics of storage solutions.

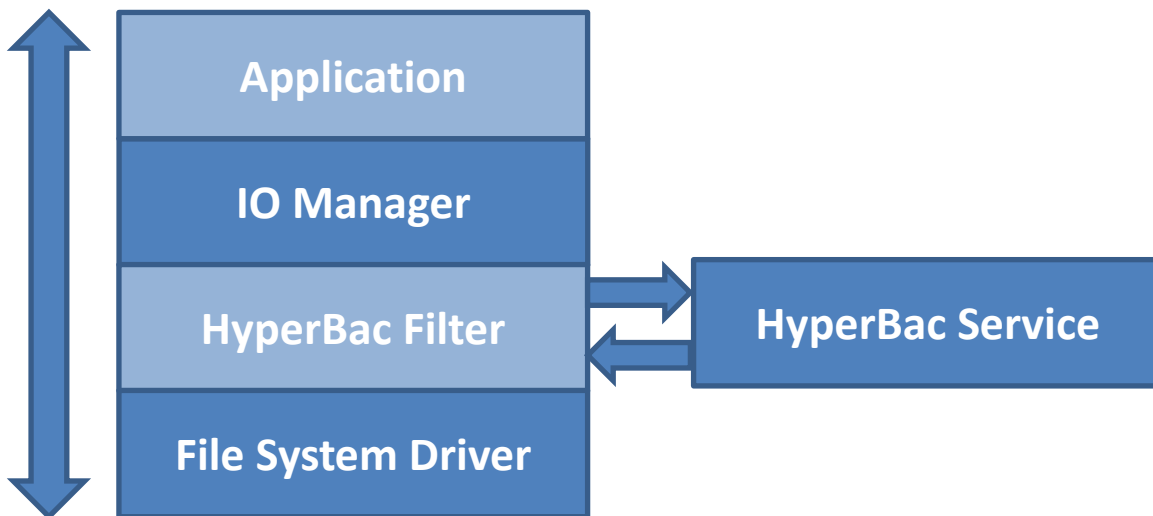
*What the program is not:*

The SQL Server I/O Reliability program is not a Microsoft certification, qualification, or logo program. Microsoft makes no warranties or representations with regard to third-party storage solutions or any third party solutions in general.

### 3 HyperBac Architecture Overview

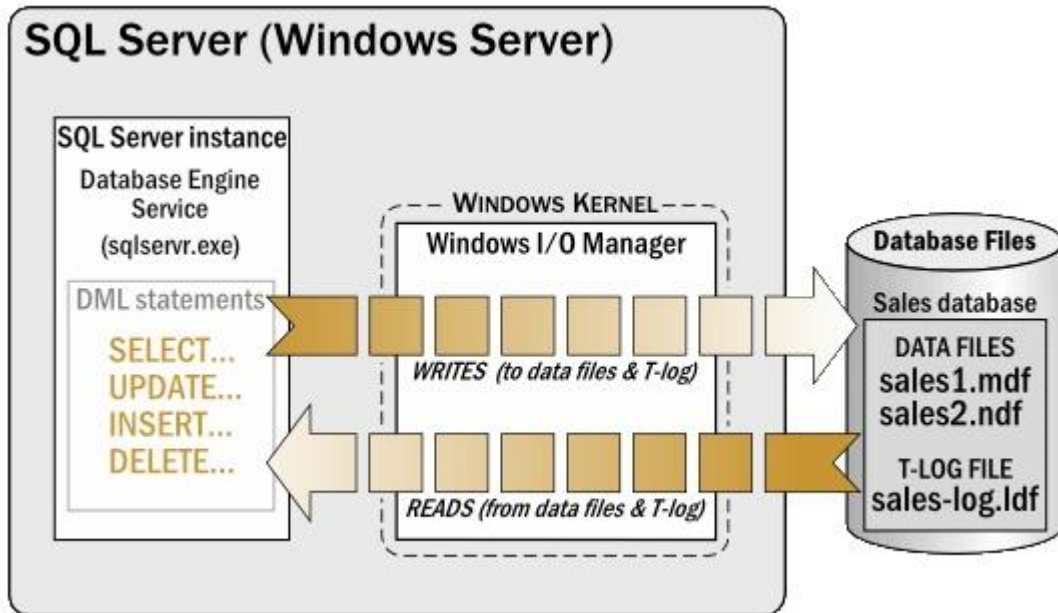
The HyperBac core architecture consists of a file system filter driver (the HyperBac driver) and a user mode service (the HyperBac Control Service), both components running at the Windows operating system level.

As an application requests to create, read to or write from a specific file, if the file meets a specified criteria (being a specific operating system process (eg sqlservr.exe) and a defined file name, path or file extension), this request is passed by the HyperBac driver to the HyperBac Control Service where action is taken as configured by the user (to compress/decompress, encrypt/decrypt). This operation is transparent to the application, which only receives uncompressed and decrypted data.



**Figure 1: Simplified HyperBac Architectural Stack Diagram**

In the specific context of SQL Server, SQL Server does not directly write to or read from disk. SQL Server passes its file I/O requests for database data and log files to the Windows I/O Manager, which runs in the Windows Operating System kernel. At this point, the Windows I/O manager passes the I/O request to a device driver, and eventually the data is read from, or written to disk.



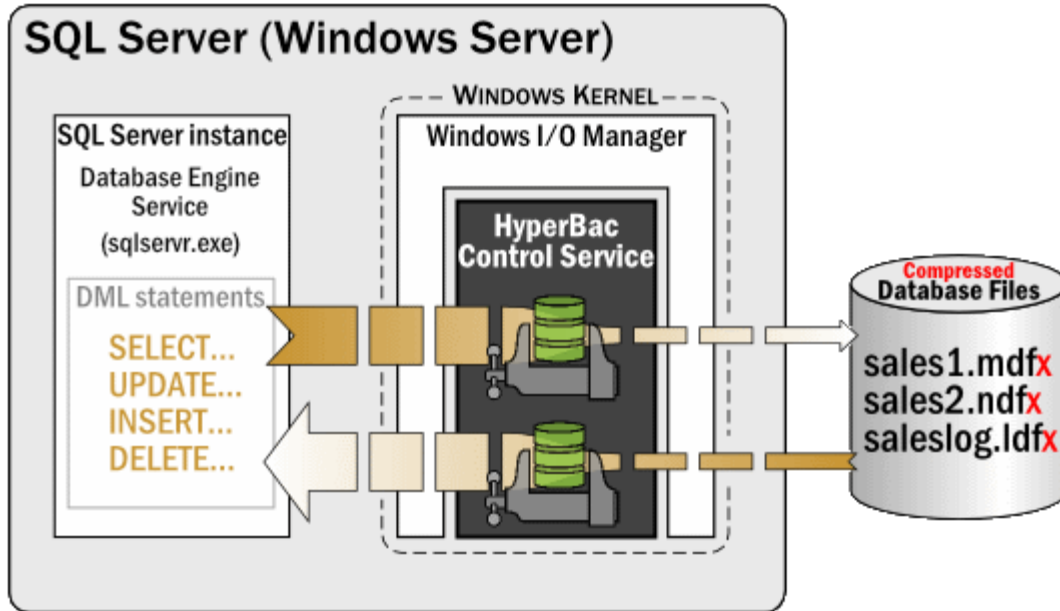
**Figure 2: A high-level overview of how SQL Server interacts with the OS.**

For example, in figure 2 above, any SELECT or DML statement executed within the database engine that requires data to be moved to or from the SQL Server Buffer cache to disk is passed through the Windows I/O manager to the MDF, NDF or LDF files, as appropriate.

When SQL Storage Compress is installed and configured and the HyperBac Control Service is running, the Windows I/O manager passes specified requests to the HyperBac driver which then passes these requests to the HyperBac Control Service where they are either compressed and/or encrypted in the case of WRITE operations or conversely decompressed and/or decrypted in the case of READ operations.

SQL Server itself is unaware that the requests are being compressed and decompressed as these requested are proxied under normal circumstances through the Windows I/O manager in any case. SQL Server works as it normally would under any other circumstances.

The default configuration of SQL Storage Compress is to use specific file extensions for compressed data and log files, these are .MDFX and .NDFX for primary and secondary data files and .LDFX for transaction log files.



**Figure 3: Overview of SQL Server I/O using SQL Storage Compress**

For example, in figure 3 above, any SELECT or DML statement executed within the database engine that requires data to be moved to or from the SQL Server Buffer cache to disk is passed through the Windows I/O manager to the MDF, NDF or LDF files, as appropriate.

These requests are then passed through the HyperBac Control Service to provide compression and encryption services.

## 4 SQL Server IO Reliability Program Requirements

### 4.1.0 Windows Logo Certification

Microsoft Windows logo certification helps ensure the safety of Microsoft SQL Server data by testing various aspects. To be compliant with the SQL Server I/O Reliability Program Review Program, solutions must pass and maintain the latest certifications for Windows logos.

The HyperBac driver has been successfully reviewed by the Windows Quality (WinQual) program, establishing that it meets the specified Microsoft WinQual criteria for driver installation and stability.

More information on the HyperBac submission can be found at <https://winqual.microsoft.com>

Submission Number: 1423826

### 4.1.1 Core Windows API Support

The HyperBac architecture is based on the Windows Installable File System Kit from Microsoft. HyperBac uses native Microsoft API's in its file system filter driver. Moreover, the HyperBac filter driver implements or passes all file system IRP requests as per normal to the appropriate file during its operation.

This means that all Core Windows API calls including but not limited to [CreateFile](#), [DeviceIoControl](#), [FlushFileBuffers](#), [GetVolumePathName](#), [GetVolumeInformation](#), [GetVolumeNameForVolumeMountPoint](#), [WriteFile](#), [WriteFileGather](#), [ReadFile](#) and [ReadFileScatter](#) are maintained with the same options, flags and definitions as the initial SQL Server request.

### 4.1.2 Stable Media

SQL Server relies on the Write-Ahead Logging (WAL) protocol to maintain the Atomicity, Consistency, Isolation, and Durability (ACID) properties of the database and to ensure data integrity. WAL relies on stable media capabilities. SQL Storage Compress and the HyperBac architecture fully comply with this stable media intention.

HyperBac maintains Write Ahead Logging (WAL) by never overwriting existing data blocks until such time as a new block or data has been written to disk. At this point the old block is flagged as "free", meaning that at any time if the new data is not written to disk due to some I/O error condition (for instance a sudden power outage or DASD failure) the previous data is completely preserved and unchanged.

HyperBac compressed media is successfully tested under 'Pull the Plug' test circumstances. Under this test HyperBac media comes back in the same consistent state as native SQL Server does without HyperBac running. For detailed information, see the 'Power Outage Testing – Pull The Plug' section in Microsoft SQL Server I/O Basics Chapter 2.

### **4.1.3 Forced Unit Access (FUA) and Write-Through**

To support Write-Ahead Logging (WAL), SQL Server uses FILE\_FLAG\_WRITETHROUGH and FlushFileBuffers to open files. Both of these options must be supported by storage solutions.

SQL Storage Compress writes out all I/O requests to disk without any cache delay, so when the HyperBac filter driver returns to the SQL Server process that a write operation has occurred, the data will have already been written prior to this point. The data at this point in time is "hard written" to the underlying hardware/disk I/O subsystem.

### **4.1.4 Asynchronous Capabilities**

HyperBac performs all IRP operations as presented to the HyperBac filter by the application or process. HyperBac fully supports asynchronous as well as synchronous operations.

All HyperBac IRP operations are performed asynchronously if they are passed to the HyperBac filter driver as asynchronous operations.

### **4.1.5 Write Ordering**

Write Ordering is completely preserved by SQL Storage Compress. As SQL Storage Compress does not schedule I/O operations to cache, all write ordering is maintained. Due to the write-through nature of the HyperBac filter driver, the write order of all I/O operations is maintained exactly as SQL Server presents these. This is a paramount design goal of the SQL Storage Compress product to preserve data integrity.

### **4.1.6 Torn I/O Protection**

The SQL Storage Compress solution employs a similar technique to SQL Server itself with respect to sector alignment and sizing to detect and prevent torn I/O conditions.

SQL Storage Compress maintains a write count on each 512 byte boundary of each open file. If a block is read which does not have matching write counts then a torn I/O

condition is detected. This is especially relevant after such events as sudden power outages to determine which blocks have been written and which blocks were not written completely.

SQL Storage Compress satisfies this core requirement and protects against torn I/O or torn page conditions automatically without any specific product configuration.

## **4.1.7 NTFS Support**

SQL Storage Compress fully supports or is compatible with NTFS capabilities. Support or compatibility for specific NTFS properties is detailed below:

### **Sparse files**

Sparse files are not applicable to SQL Storage Compress, as the SQL Storage Compress devices are stored compressed so blank spaces do not exist in the file.

### **Encryption**

The HyperBac filter driver resides above the EFS layer, so if NTFS encryption is used the data is encrypted/decrypted as would normally be expected. With that said, the SQL Storage Compress product supports AES-256, AES-192 and AES-128-bit encryption at the filter driver level which can be used as well, advantages are that the file can be copied or moved to a different volume and remain in an encrypted state.

### **Compression**

SQL Storage Compress implements compression at the file system filter driver level, so you would not normally use a SQL Storage Compress device on an NTFS compressed volume, however because the HyperBac filter driver sits above the NTFS compression layer, this would be possible.

### **Security**

The HyperBac filter driver passes all security related IRP calls to the SQL Storage Compress device, so NTFS security is completely preserved.

## 4.1.8 Testing

The [SQLIOSim](#) utility was used to perform stress tests and simulate SQL Server activity against compressed disk devices running on SQL Storage Compress.

Data durability and integrity tests were run against the following system configuration:

<b>Operating System</b>	Windows Server 2008 R2 Enterprise x64
<b>CPU</b>	Intel Core 2 Duo P8800 @ 2.67 GHz
<b>Physical Memory</b>	6 GB
<b>SQL Server Version</b>	SQL Server 2008 R2 Enterprise Edition x64

The following configuration file (AlwaysOn.SQLIOSim.cfg.ini) was used to perform the SQLIOSim durability tests against the data and log devices in 48 30min cycles a total of 224 hours as per the SQL Server IO Reliability Requirements Specifications.

```

;=====
; FEB 2008 - SQL Server Always On Database I/O Test configuration
;
; TESTER: Alter the [File] sections to include appropriate paths
;         for stressing the solution components.
;
;         - SQL Server is not required, this is an independent test.
;         - Do not point this as actual SQL files or the data will be lost.
;
; User Count:  -1 (Default based on CPU count)
;              0 (Disabled)
;              ## (Absolute value)
;
; Missing sections or values - SQLIOSim default(s) used
;=====
[CONFIG]
StopOnError=TRUE
ErrorFile=sqliosim.log.xml
Affinity=0
IOAffinity=0

; Default
; CPUCount=2
; MaxMemoryMB=600

;=====
; 30min cycles for 24 hours
;=====
TestCycles=48
TestCycleDuration=1800

CacheHitRatio=1000
NoBuffering=TRUE
WriteThrough=TRUE
MaxOutstandingIO=0
TargetIODuration=100

```

```
; Allow bursts to push I/O depth to max areas and high resource usage
```

```
AllowIOBursts=TRUE  
UseScatterGather=TRUE  
ForceReadAhead=TRUE  
DeleteFilesAtStartup=TRUE  
DeleteFilesAtShutdown=FALSE  
StampFiles=FALSE
```

```
[RandomUser]  
; Default based on number of CPUs formula  
UserCount=-1  
JumpToNewRegionPercentage=500  
MinIOChainLength=50  
MaxIOChainLength=100  
RandomUserReadWriteRatio=9000  
MinLogPerBuffer=64  
MaxLogPerBuffer=8192  
RollbackChance=100  
SleepAfter=5  
YieldPercentage=0  
CPUSimulation=FALSE  
CPUCyclesMin=0  
CPUCyclesMax=0
```

```
[AuditUser]  
UserCount=4  
BuffersValidated=64  
DelayAfterCycles=2  
AuditDelay=200
```

```
[ReadAheadUser]  
UserCount=10  
BuffersRAMin=32  
BuffersRAMax=1024  
DelayAfterCycles=2  
RADelay=200
```

```
[BulkUpdateUser]  
UserCount=-1  
BuffersBUMin=64  
BuffersBUMax=1024  
DelayAfterCycles=2  
BUDelay=10
```

```
[ShrinkUser]  
MinShrinkInterval=120  
MaxShrinkInterval=600  
MinExtends=1  
MaxExtends=2048
```

```
[File1]  
FileName=C:\MSSQL\Data\SQLIOSim.mdfx  
InitialSize=4096  
MaxSize=8192  
Increment=10  
Shrinkable=TRUE  
LogFile=FALSE  
Sparse=FALSE
```

```
[File2]  
FileName=C:\MSSQL\Data\SQLIOSim.ldfx  
InitialSize=50  
MaxSize=50  
Increment=0
```

Shrinkable=FALSE  
LogFile=TRUE  
Sparse=FALSE

SQL Storage Compress has also been successfully reviewed for compatibility with:

- x86 with 3GB Enabled
- x86 with PAE Enabled
- WOW64 running x86 in x64
- Low paged and non-paged pool conditions
- Excessive outstanding request boundaries
- Memory requests are not forced to a specific memory location

## 4.2.2 Transactional Sector/Block Rewrites

SQL Storage Compress provides transactional safety while maintaining asynchronous capabilities by not overwriting blocks or parts thereof for previously existing virtual blocks, therefore any failure of the data before the completed request or partially completed request will not affect the existing information. Transactional recovery of the appropriate block is achieved through the use of the unique write count at the beginning of each sector.

## 4.2.5 File Streams

FILESTREAM operations through SQL Server are fully supported using the HyperBac filter and SQL Storage Compress solution. The HyperBac filter is above the NTFS layer and supports all FILESTREAM writes and reads from SQL Server, providing streaming compression and/or encryption to these operations as well.

## 5 References

Microsoft SQL Server 2000 I/O Basics –

<http://www.microsoft.com/technet/prodtechnol/sql/2000/maintain/sqlIObasics.mspx>

SQLIOSim –

<http://support.microsoft.com/kb/231619>

SQL Server 2008 R2 Best Practice Analyzer-

[http://download.microsoft.com/download/9/3/C/93CCF4F6-B1DC-4636-BE83-05478125E98E/1033/X86/SQL2008R2BPA\\_Setup32.msi](http://download.microsoft.com/download/9/3/C/93CCF4F6-B1DC-4636-BE83-05478125E98E/1033/X86/SQL2008R2BPA_Setup32.msi)