

Remote BLOB Storage

SQL Server White Paper

**Author:** Russ Houberg, KnowledgeLake

**Technical Reviewers:** Pradeep Madhavarapu, Srini Acharya, Michael Warmington

**Published:** February 2011

**Applies to:** SQL Server 2008 R2

**Summary:**

This document contains extensive information about Microsoft SQL Server 2008 R2 Remote BLOB Storage and the FILESTREAM RBS Provider. It also discusses how the FILESTREAM RBS Provider can be implemented in Microsoft SharePoint Server 2010.

Copyright

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This white paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in, or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2011 Microsoft Corporation. All rights reserved.

Microsoft, <plus, in alphabetical order, all Microsoft trademarks used in your white paper> are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

Contents

[Introduction: What is Remote BLOB Storage? 6](#_Toc283621627)

[The Case for RBS 6](#_Toc283621628)

[Large Database of Mostly Binary Data 6](#_Toc283621629)

[Digital Asset Management Databases 7](#_Toc283621630)

[When Storage Tiers Need to be Implemented 7](#_Toc283621631)

[When Storage Needs to be Optimized 7](#_Toc283621632)

[Installing, Configuring, and Administering RBS 8](#_Toc283621633)

[Component Overview 8](#_Toc283621634)

[RBS Feature Pack 8](#_Toc283621635)

[Important RBS Concepts 11](#_Toc283621636)

[Entity Relationships 12](#_Toc283621637)

[Differences between LOCAL and REMOTE FILESTREAM Providers 12](#_Toc283621638)

[High Availability Considerations 14](#_Toc283621639)

[Process Flow 15](#_Toc283621640)

[BLOB Insert 15](#_Toc283621641)

[BLOB Retrieval 16](#_Toc283621642)

[BLOB Delete 17](#_Toc283621643)

[BLOB Delete Propagation 17](#_Toc283621644)

[Pre-requisites and Required Resources 18](#_Toc283621645)

[Enabling FILESTREAM for file I/O Streaming Access 18](#_Toc283621646)

[Setting the filestream\_access\_level Parameter 20](#_Toc283621647)

[Storage Architecture Considerations 20](#_Toc283621648)

[Implementation Guidance 21](#_Toc283621649)

[Installing the RBS Feature Pack 21](#_Toc283621650)

[Database Preparation 21](#_Toc283621651)

[Executing the RBS Feature Pack Installer 21](#_Toc283621652)

[Installing on Multiple Servers 32](#_Toc283621653)

[Configuring RBS 33](#_Toc283621654)

[BLOB Store Core Configuration Settings 33](#_Toc283621655)

[BLOB Store Extended Configuration Settings 33](#_Toc283621656)

[RBS Configuration Settings 34](#_Toc283621657)

[Uninstalling RBS (and the FILESTREAM Provider) 36](#_Toc283621658)

[Remove BLOB Store Database and Provider Configurations, Uninstall RBS 36](#_Toc283621659)

[Maintenance and Administration 37](#_Toc283621660)

[The Maintainer application 37](#_Toc283621661)

[Monitoring the BLOB Store 38](#_Toc283621662)

[Performance Counters 39](#_Toc283621663)

[Database Resources 39](#_Toc283621664)

[Backup and Restore 40](#_Toc283621665)

[Prescriptive Guidance 41](#_Toc283621666)

[Synchronizing RBS Configuration with Backup Schedule 42](#_Toc283621667)

[RBS Operational Issues 42](#_Toc283621668)

[SharePoint 2010 and RBS 43](#_Toc283621669)

[Differences between EBS and RBS 43](#_Toc283621670)

[Enabling an RBS Provider in SharePoint 2010 44](#_Toc283621671)

[Enabling the Provider on Multiple Content Databases 46](#_Toc283621672)

[Configuring RBS Security 47](#_Toc283621673)

[Migrating Content To and From the BLOB Store 47](#_Toc283621674)

[Migrate BLOB Data Back Inline in the Content Database 48](#_Toc283621675)

[Disable RBS in the Content Database 48](#_Toc283621676)

[Uninstalling RBS (and the FILESTREAM Provider for SharePoint) 48](#_Toc283621677)

[Migrate BLOB Data Back Inline in the Content Database 49](#_Toc283621678)

[Disable RBS in the Content Database 49](#_Toc283621679)

[Remove BLOB Store Database and Provider Configurations, Uninstall RBS 49](#_Toc283621680)

[Guidance: Implementing RBS in a Custom Application 50](#_Toc283621681)

[Introduction: The Client Application Explained 50](#_Toc283621682)

[Client Library Reference 50](#_Toc283621683)

[Using the RBS Client Library 51](#_Toc283621684)

[Environment Configuration 51](#_Toc283621685)

[BLOB Storage and Retrieval 53](#_Toc283621686)

[Deleting a BLOB 54](#_Toc283621687)

[Implementation Best Practices 55](#_Toc283621688)

[Guidance: Building an RBS Provider 55](#_Toc283621689)

[Implementation Guidance 55](#_Toc283621690)

[Max Buffer Size 55](#_Toc283621691)

[OptimizationSpecifiedIds 56](#_Toc283621692)

[Summary 56](#_Toc283621693)

# Introduction

As the volume of data generated by applications and services increase, the distribution of data types that are stored in databases is also changing. Take for instance a document management server system. Such a system may have thousands of versioned documents stored as blobs (binary large objects) directly in the database along with the document metadata. As the usage increases, the total size of binary document blobs will far outweigh the size of the document metadata and other structured data. This skews the ratio of structured vs. unstructured binary data far to the side of unstructured data. There are several advantages in moving the unstructured data, simply referred to as BLOB data hereafter, outside of the main database. This reserves the relational database storage for data which will be referenced in queries, and places the bulk, unstructured data on storage more appropriate for it.

[SQL Server 2008 R2 Remote BLOB Store (RBS)](http://go.microsoft.com/fwlink/?LinkID=188395&clcid=0x409) is an add-on feature pack for Microsoft SQL Server 2008 R2. RBS is designed to move the storage of binary large objects (BLOBs) from database servers to commodity storage solutions. With RBS, BLOB data is stored in storage solutions such as Content Addressable Stores (CAS), commodity hardware storage systems attached to the Databases and File systems or cloud storage solutions like Microsoft Windows Azure.

A client application, such as [Microsoft SharePoint 2010](http://sharepoint.microsoft.com/en-us/Pages/default.aspx), can take advantage of SQL Server 2008 R2 and Remote BLOB Storage to store BLOB data. SharePoint 2010 supports the RBS Feature Pack which facilitates the storage of the BLOB data outside of the SharePoint content database.

**NOTE**: SharePoint 2010 may be generically referenced in this whitepaper. Unless otherwise indicated this term applies to both Microsoft SharePoint Foundation 2010 and Microsoft SharePoint Server 2010.

## The Case for RBS

While RBS can provide tremendous benefits, it can also add overhead to the document management process. There are additional administrative considerations that must be accounted for when RBS is deployed. These considerations will be discussed later in this whitepaper. In this section, we’ll address common scenarios when the benefits of RBS outweigh the additional administrative considerations.

### Large Database of Mostly Binary Data

If a given SQL Server database would grow to 500GB without RBS enabled, then RBS would be a beneficial option. A 500GB database is considerably large. Having a very large database can have a negative impact on business continuity and maintenance operations.

* Backup and restore operations take considerably longer
* Index and statistics defragmentation takes considerably longer. This is a particular concern if the database must be taken offline during defragmentation.
* Regular DBCC consistency checks will take much longer. If database integrity is not regularly monitored, the risk of a corrupted database is considerably increased. Larger databases will have a higher risk of corruption due to physical storage errors simply because of the large quantity of storage they consume.

For these reasons, enabling RBS on an otherwise very large database can be very beneficial as each of the concerns addressed above are alleviated.

### Digital Asset Management Databases

SQL Server databases that are expected to contain fewer but larger files will also see a significant benefit. Storing video files, graphics files, and audio files (digital assets) in a SQL Server database would cause the database to inflate quickly.

Consider the scenario of a custom application that is designed to provide training content to users. The application will contains less than 1000 documents but they are all training videos that average 200MB in size. The database for this application would be nearly 200GB in size even though there are relatively few documents. By enabling RBS, the binary data for these videos can be kept out of the database while the structured metadata in the database remains responsive and easy to manage. This is a much more natural and integrated solution than allowing the video files to inflate the database or forcing the actual source videos to be stored in an unmanaged location outside of the control of the application.

### When Storage Tiers Need to be Implemented

One of the most significant benefits of RBS lies in extensibility. RBS doesn’t just have to be for getting BLOB data out of the database. It can serve other creative purposes such as facilitating highly efficient storage tiers.

Consider the scenario where a document management solution has been or will be deployed for an organization. A large percentage of the corporate user base will be adding and editing collaborative content on a daily basis. Over time, a very large number of customer centric documents are created, possibly declared as records for retention, and then archived.

In this scenario, an RBS provider that enables a tiered storage platform could provide tremendous cost savings by intelligently managing the storage location of BLOB data. New and frequently accessed content could be stored in high performance storage. Older documents that are accessed only occasionally could be automatically moved to lower cost and lower performance storage such as SATA arrays for example. Then over time, very old documents that are rarely accessed could be automatically moved again to extremely inexpensive cloud based storage. In all cases, end users are able to access content in real time but the responsiveness and cost of the storage is intelligently managed.

### When Storage Needs to be Optimized

When BLOB data is allowed to inflate a SQL database, file I/O and processing load is increased on the database server. If the average size of BLOB data is 80KB or higher, then implementing RBS reduces I/O and processing load which improves the performance of SQL Server.

Also, RBS providers have an additional advantage in that they can perform additional processing on the BLOB stream as it is being passed to the BLOB store. For example, some providers may compress and decompress the BLOB content on the way to and from the BLOB store. This results in a smaller overall storage footprint for the same BLOB data. This practice should be carefully managed such that significant additional storage and retrieval latency is not added to BLOB store operations.

# Installing, Configuring, and Administering RBS

In this section, the background concepts that support installation, configuration, and administration procedures will be discussed. It’s important to mention that this document will not provide step by step guidance for implementing RBS for SharePoint 2010 as that would be a duplication of effort. For step by step deployment procedures, please consult the [Manage Remote BLOB Storage (SharePoint Server 2010)](http://technet.microsoft.com/en-us/library/ee748638.aspx) TechNet topic. This document is, however, intended to provide supplemental and background information to the step by step process of RBS deployment.

The premise of the information in this section is the installation of the Local FILESTREAM RBS provider along with the rest of the RBS Feature Pack. TechNet provides additional procedures for deploying RBS without the FILESTREAM provider. If the FILESTREAM provider is not being installed, then it is assumed that additional installation guidance will be provided by creator of the RBS provider being implemented.

While the information in this section is directed towards the FILESTREAM RBS provider implementation for SharePoint 2010, most of the information is applicable even if the FILESTREAM RBS provider is not being implemented, such as when some other 3rd party RBS provider is being deployed. Similarly, most of this information is also applicable for client applications other than SharePoint 2010.

## Component Overview

The SQL RBS framework is entirely contained in the RBS Feature Pack. This feature pack contains elements that can be used to describe the core components of the RBS framework.

### RBS Feature Pack

The [RBS Feature Pack for SQL Server 2008 R2](http://www.microsoft.com/downloads/en/details.aspx?FamilyID=ceb4346f-657f-4d28-83f5-aae0c5c83d52&displaylang=en) installation provides three required components and one optional component that facilitate the implementation of the RBS framework. It is important to understand how the various components of the installation package affect the RBS deployment. For this reason, this section describes the four primary components that exist in the RBS Feature Pack installer.

#### Server

The Server component refers to the T-SQL script that is used to deploy the RBS database objects into the client database. The RBS database objects consist of ancillary tables, views, stored procedures, functions, schemas and security roles. When the Server component is selected for installation, this simply means that the T-SQL script will be installed to the RBS Feature Pack installation directory on the client server. It will not actually be executed on a client database unless the ***Execute Script*** subcomponent is also selected.

When the RBS objects are properly deployed to a content database, the RBS “server” is considered to be installed.

#### Client

The Client component refers to the RBS client library that is the key to the generic interface of the client application and an RBS provider. The client library provides an abstract class that must be implemented by the RBS provider. The abstract class in the client library ensures that the RBS provider will provide the required functional components necessary to interact with the BLOB store. When this component of the RBS Feature Pack is selected for installation, the install process will deploy the client library to the RBS Feature Pack install location.

Thanks to the provider model, the client application instantiates only the general client library to perform BLOB operations. It then invokes the RBS provider library to fulfill the request.

All BLOB objects that pass to and from the BLOB store are derived from the System.IO.Stream object. This allows for binary data to be transferred between the BLOB store and the client application in the most efficient way possible.

#### FILESTREAM Provider

Like any RBS provider, the FILESTREAM provider implements the BLOB Store abstract class in the Client library in order to provide BLOB operation functionality to the client application. The FILESTREAM provider utilizes SQL Server’s FILESTREAM technology to store BLOBs as files in the NTFS file system. For more details on the FILESTREAM technology, reference [the FILESTREAM Storage in SQL Server 2008 whitepaper](http://msdn.microsoft.com/en-us/library/cc949109.aspx).   
  
The FILESTREAM RBS provider has been extensively tested by the SQL Server development team to ensure that it meets the reliability and scalability needs of an enterprise class solution. Other software vendors have also developed RBS providers. There are positives and negatives regardless of the provider that is chosen. The most basic distinction between the FILESTREAM RBS provider and other third party providers is that deploying the FILESTREAM RBS provider, while fully functional, results in more administrative overhead when compared with third party providers. In contrast, third party providers typically have intuitive graphical user interfaces that ease the administrative burden. But perhaps a more relevant distinction for most organizations is that third party providers typically come at a significant cost. A licensing cost, annual maintenance cost, or both will typically be required before the provider can be deployed in production. In contrast, the FILESTREAM RBS provider from Microsoft is available at no cost as part of the SQL Server 2008 R2 feature pack.

During RBS Feature Pack installation, selecting the FILESTREAM provider component will cause the FILESTREAM provider to be installed and configured.

Selecting the ***Client*** subcomponent of the FILESTREAM provider will cause the FILESTREAM provider library to be deployed to the Global Assembly Cache (GAC) so that the client library is able to invoke the FILESTREAM provider as necessary. If a FILESTREAM provider has already been installed on the machine but RBS is being configured for a second client database, then it is not necessary to install the Client subcomponent a second time.

The FILESTREAM Provider ***Server*** subcomponent refers to the T-SQL scripts, specific to the FILESTREAM provider, that enable the provider to interact with the FILESTREAM file group for the BLOB store. Selecting this subcomponent instructs the installer to deploy these scripts to the RBS Feature Pack installation folder. But these scripts will not be executed unless the Server => ***Execute Scripts*** subcomponent is selected.

NOTE: If the ***Local*** FILESTREAM provider is being deployed, the FILESTREAM provider Server subcomponent T-SQL scripts will be executed in the context of the client database. If the ***Remote*** FILESTREAM provider is being deployed, the FILESTREAM provider Server subcomponent T-SQL scripts are executed in the context of an additional database, possibly on a different server. The only purpose of this second database is to facilitate BLOB store operations. An explanation of the LOCAL and REMOTE FILESTREAM providers are described in the section below titled “Differences between LOCAL and REMOTE FILESTREAM Providers”.

#### Maintainer

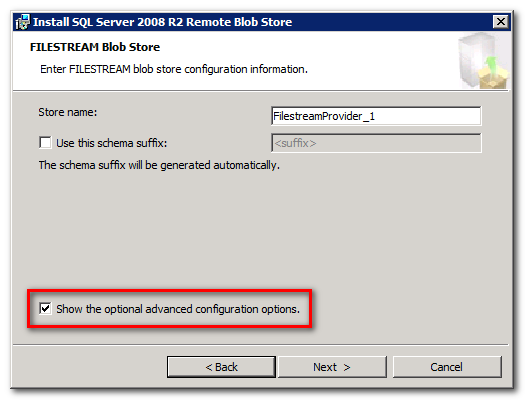
The Maintainer is a console application that facilitates lazy garbage collection processing and other maintenance tasks. It is a client application that invokes special methods in the client library to ensure that the BLOB store, the data in the RBS ancillary tables, and the BLOB references in the client database remain in a consistent state. Because Maintainer invokes the client library to perform all operations, it is provider agnostic. It doesn’t care what RBS provider is being invoked by the client library.

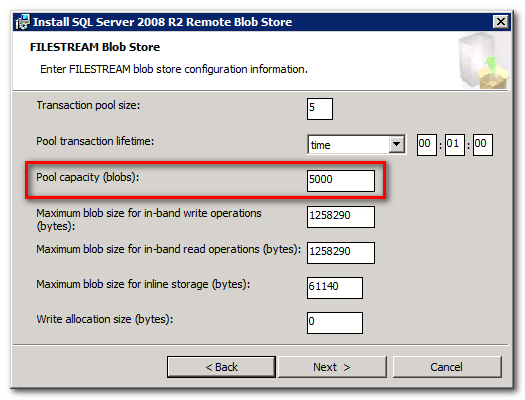
The Maintainer is intended to be executed on a regular basis via the Task Scheduler. A separate task should be scheduled for each client database that has been RBS enabled. The Maintainer can be executed in order to perform the following maintenance tasks:

Garbage collection performed by the maintainer needs to be performed in order to reclaim unused resources and maintain the RBS system. Garbage collection consists of Delete Propagation and Orphan Cleanup. Each of these tasks can be scheduled independently. They can be scheduled to run at a non-busy time so as to have lesser impact on the primary usage of the database.

* **Delete Propagation**. In order to support backup/restore SLA requirements, it would not be prudent to delete a BLOB from the BLOB store immediately when a document is “deleted” from the client application. This allows the BLOB to be available after a document has been deleted from a client database but then that database is restored to a time before the document was deleted. So delete propagation is the concept of finalizing the BLOB delete process in the BLOB store at some later point in time after the SLA required restore timeframe has passed. Deleted documents are tracked by the RBS server tables in the client database. The Maintainer application uses these tables to propagate the delete operation to the BLOB store after the appointed time has expired. Once the delete has been fully propagated by the RBS maintainer, the BLOB is no longer recoverable. Care should be taken to set the SLA required restore timeframe correctly.
* **Orphan Cleanup**. Orphans are created when a BLOB is created in the BLOB store but, due to a rare situation like an application or server crash, the document row is not committed to the client database. These orphans need to be garbage collected periodically. Orphan cleanup is done by enumerating all BLOBs present in the BLOB store which were created before the orphan cleanup time window, and from this list, removing those BLOBs that are referenced by the database. The BLOBs remaining in the list are deleted from the BLOB store. This process is executed on a subset of the BLOB store documents called a ***pool***.

It is normal for Orphan cleanup to take a lot of time. So it should be scheduled with a lower frequency than delete propagation. Orphan cleanup can affect farm performance; particularly if storage pools are allowed grow to contain very large number of BLOBs. This can be mitigated by limiting pool sizes to a number that can be processed in a relatively short amount of time. The pool size can be set during initial install or reconfigured after installation. When executing the RBS Feature Pack installer using the installer UI, an option will be presented on the FILESTREAM BLOB Store configuration dialog box that will allow advanced configuration options (see Figure 1-1). This will allow the configuration of the storage pool size during installation (see Figure 1-2).

*Figure 1-1*  


*Figure 1-2*  


But be careful not to set the pool size too small. The FILESTREAM provider creates a data table in the client database to manage each pool. Try to balance the number of documents in each pool and the number of pools with respect to the projected number of documents that will allowed in the client application database. Orphan cleanup is a metadata operation which means the full BLOB data is not parsed. The default value should be fine in most cases.

## Important RBS Concepts

There are several key concepts to the RBS framework. One of the most important concepts has already been mentioned. The RBS client library exposes a provider model which allows for the decoupling of the RBS provider from the client application. The RBS provider is not aware of the client application that causes it to store and retrieve documents. Similarly, the client application, is not aware of how or where the RBS provider stores documents. The RBS provider must implement certain methods and properties in the RBS API. The client application can then generically reference the RBS client assembly in order to store or retrieve BLOB content.

### Entity Relationships

Continuing on with important concepts, it is important to understand the various components of an RBS implementation. In this section the major components and their relationships will be described so that they can be referenced throughout this document.

#### BLOB Store

The BLOB store is the final storage location of all BLOB content. A BLOB store can be any storage location or device. Examples are NTFS file systems, SMB network shares, Content Addressable Stores (CAS) or some other commodity (cloud) storage solution. RBS providers are created to specifically support one or more BLOB Stores. It is the RBS provider that maintains the specific code that is necessary to interact with a given BLOB store.

It is possible to have multiple, populated BLOB Stores associated with a client database. For example, it is possible to initially register a FILESTREAM BLOB store in a client application database and then later register a BLOB store that uses a second provider in order to take advantage of a CAS device. While both BLOB stores can be active and serve BLOB content, only one BLOB store can be configured to receive new BLOB data at any given time.

### Differences between LOCAL and REMOTE FILESTREAM Providers

There are two possible implementations of the FILESTREAM provider. They are the ***Local*** FILESTREAM provider and the ***Remote*** FILESTREAM provider. The FILESTREAM provider implements a database FILESTREAM file group in order to essentially turn the SQL Server NTFS file system into a BLOB store.

When deploying the ***Local*** FILESTREAM provider the FILESTREAM file group is created directly in the database that is being RBS enabled. This means that the same SQL Server that is processing requests from the client application database is also acting as a BLOB store (see Figure 1-3).

*Figure 1-3*

When deploying the ***Remote*** FILESTREAM provider, the FILESTREAM file group can be created in a database that is NOT the content database being RBS enabled (see Figure 1-4). In fact, the FILESTREAM file group can even be created in a database that is hosted on an entirely DIFFERENT (or REMOTE) SQL Server (see Figure 1-5). This means that a second SQL Server can be deployed and dedicated to servicing RBS BLOB store requests, allowing the SQL Server that hosts application databases to be dedicated to application processing. In extreme scale environments this can result in a significant improvement to application scalability.

*Figure 1-4*  


Figure 1-5  


### High Availability Considerations

* **SQL Server Backup/Restore**. SQL Server backup and restore operations are supported for only the BLOB references in the database. BLOB data in the BLOB store does not get backed up. A separate solution, implemented at the scope of the BLOB store, must be implemented to ensure the availability of BLOB data.
* **Log Shipping**. Log shipping is directly supported with the FILESTREAM provider. Other RBS providers must specifically implement support for log shipping or the feature will not be available.
* **Clustering**. Clustering is supported by FILESTREAM provider because the FILESTREAM file group will be mounted locally for the cluster servers. The ancillary tables also support clustering so RBS providers other than FILESTREAM will work with a SQL cluster. However, it is important to note that a SQL cluster only guarantees high availability of the BLOB references in the database when the provider is a FILESTREAM provider. A separate solution, implemented at the scope of the BLOB store, must be implemented to ensure the availability of BLOB data.
* **Mirroring and Replication**. SQL mirroring and replication is supported for only the BLOB references in the database as long as BLOB data mirror processing can stay ahead of database (metadata) mirroring to ensure link level consistency. BLOB data in the BLOB store is not mirrored or replicated. Many BLOB store devices can provide mirroring and replication capabilities. These features must be supported by the RBS provider in order for mirroring and replication of both the SQL data and the BLOB store data to be successful. SQL mirroring can’t be used with SQL Server 2008 R2 if the LOCAL FILESTREAM provider is used. Mirroring can be used for the client/content database, but not on the remote BLOB store database if REMOTE FILESTREAM provider OR a third party provider is used.

## Process Flow

Now that a basic understanding of the underlying RBS entities has been established, let’s take a look at the process flow for various RBS activities. For each of the following flow charts, assume that RBS has been enabled for the content database.

### BLOB Insert

Reference the flow chart in Figure 1-6 using the step descriptions below the flow chart.

*Figure 1-6*

1. An end user sends a document into the application.
2. The application calls to the client library in order to create an RBS session context using the connection string to the database where the RBS auxiliary resources are located. The application uses the context to call the client library and create a new BLOB object and invoke a BLOB stream write operation.
3. The client library invokes the provider library in order to write the BLOB data to the BLOB store.
4. The provider library writes the BLOB stream to the BLOB store. The BLOB store generates the store\_blob\_id which is used as all or part of a file name as the BLOB data is stored in the BLOB store.
5. The BLOB may return a StoreBlobId. This is typical of CAS device, cloud storage, or other similar BLOB store.
6. The **StoreBlobId** is returned to the client library and stored in the RBS auxiliary tables.
7. The client library registers the BLOB in the RBS auxiliary tables in the content database using a BLOB Registration stored procedure. The CollectionId, BLOBStoreId, StorePoolId, StoreBlobId, CreatedTime, and BlobSize are passed into a stored procedure.
8. The registration stored procedure returns the BlobNumber to the client library. The client library uses the BlobNumber, CollectionId, and additional flags to construct a byte array that represents the BlobId.
9. The **BlobId** is returned to application.
10. The application sends the document metadata, including the BlobId, to the database. The BlobId is stored in the registered RBS column and table.
11. The success or failure of the document insert is returned to the calling application method.
12. The success or failure of the upload operation to the application is returned to the end user.

### BLOB Retrieval

Reference the flow chart in Figure 1-7 using the step descriptions below the flow chart.

*Figure 1-7*  


1. An end user requests a document from the application.
2. The application calls the database for the document data.
3. The client database returns the document metadata, including values from the registered RBS column which contains the **BlobId**.
4. Because the registered RBS column value is not null, the application knows that it must invoke the RBS client library to retrieve the BLOB data for the document. The application uses the RBS Client Library to create an RBS Session. The RBS Session is used to instantiate a BLOB object using the BlobId.
5. During BLOB object instantiation, the RBS Client Library passes the BlobId information to an RBS stored procedure in the client database.
6. The RBS stored procedure returns the information necessary to construct a **BlobReference**. Specifically, the **StoreBlobId**, which is part of the BlobReference, is needed before the BLOB stream can be returned to the application.
7. The application calls for a readable stream through client library. The client library invokes the provider library to retrieve the readable stream. The provider library uses the BlobReference in order to construct a request to the BLOB store.
8. The provider library sends a request for the BLOB stream to the BLOB store using the StoreBlobId from the BlobReference.
9. The BLOB store returns a BLOB stream to the provider library.
10. The provider library forwards the readable BLOB stream to the client library.
11. The client library returns the readable BLOB stream to the calling application method.
12. The application streams the document data to the end user.

### BLOB Delete

Document BLOB data should not be deleted immediately when a document in an RBS enabled application is deleted. RBS must conform to SLA requirements that support a defined document restoration window. If BLOB data is removed immediately and a restore operation returns metadata for a deleted document to the content database, then the end user would not be able to retrieve the document because the BLOB data does not exist in the BLOB store.

For this reason, deleting a document in an RBS enabled application is the same regardless of whether or not RBS is enabled.

1. The end user initiates a delete operation.
2. The application physically removes the document record from all relevant tables in the database. The BlobId data in the registered RBS column is deleted with the other document data.

At this point, all application metadata regarding the document has been deleted from the database. However, a BlobReference still exists in the RBS ancillary tables. This BlobReference will be used when a scheduled task launches the Maintainer application and initiates **Delete Propagation** which will finally remove the BLOB from the BLOB store.

### BLOB Delete Propagation

When a document record is deleted from an RBS enabled application, RBS is not yet aware that the document has been deleted. RBS becomes aware of delete during the **Reference Scan** maintenance phase, executed by the Maintainer application. During this phase, an RBS stored procedure compares the BLOB references in the RBS auxiliary tables with the registered RBS column in the application tables. The deleted items are then stored in the RBS auxiliary tables for delete propagation by the Maintainer after the Garbage Collection Time Window has elapsed.

Reference the flow chart in Figure 1-8 using the step descriptions below the flow chart for an explanation of the Delete Propagation Phase processing by the Maintainer.

*Figure 1-8*

1. Maintainer calls a stored procedure to request a list of BlobReference values for BLOBs that need to be deleted from the BLOB store.
2. The stored procedure returns a list of BlobReference values that have been deleted for a period of time that has exceeded the garbage collection time window.
3. Maintainer calls the client library in order to initiate a BLOB delete operation.
4. The client library invokes the appropriate provider library to initiate the BLOB delete operation.
5. The provider library commands the BLOB store to delete the BLOB.
6. The BLOB store returns the success or failure of the delete operation to the provider library.
7. The provider library passes the result of the delete operation to the client library.
8. Upon successful deletion from the BLOB store, the client library calls an RBS stored procedure which removes the BlobReference from the appropriate RBS auxiliary tables.
9. The result of the BlobReference delete stored procedure is returned to the client library.
10. The client library returns the result of the delete operation to the Maintainer.
11. Steps 3 through 10 are repeated until the entire list of BLOBs has been deleted, the time limit specified by TimeLimit parameter has elapsed, or the Maintainer application is forcibly terminated.

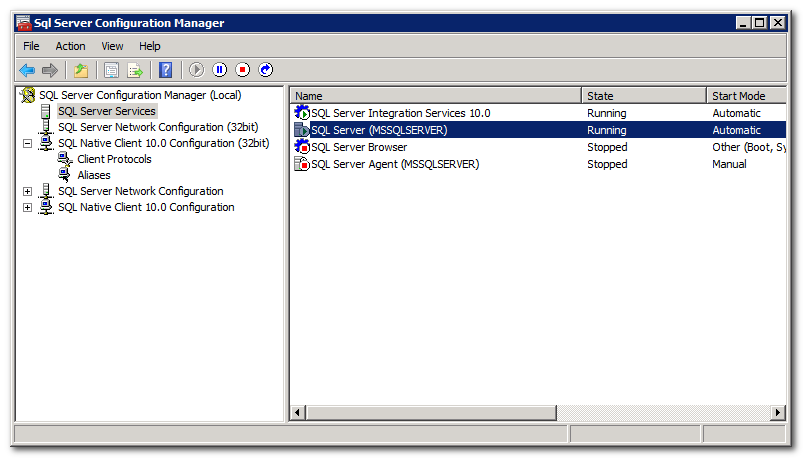
## Pre-requisites and Required Resources

Before the RBS Feature Pack is installed, there are a few steps that need to be taken to prepare the environment.

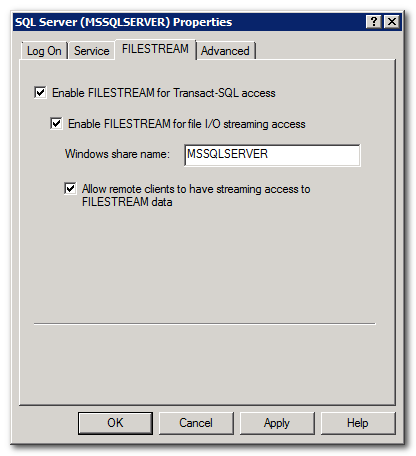
### Enabling FILESTREAM for file I/O Streaming Access

When deploying the FILESTREAM RBS provider, the “Enable FILESTREAM for file I/O streaming access” option must be activated during SQL Server installation on the server that will be used for the FILESTREAM BLOB store. It can be activated after installation by following these steps on the SQL Server that will host the content database (Local FILESTREAM provider) or the SQL Server that will host the BLOB store (Remote FILESTREAM Provider):

1. Click Start => All Programs => SQL Server 2008 R2 => Configuration Tools => **SQL Server Configuration Manager.**
2. Under SQL Server Configuration Manager in the left pane, select **SQL Server Services.**
3. Locate the instance of the SQL Server that needs to be enabled. (see Figure 1-9)

*Figure 1-9*

1. Right-click the SQL Server instance and select **Properties.**
2. In the SQL Server Properties dialog box, select the **FILESTREAM** tab.
3. Ensure that **Enable FILESTREAM for Transact-SQL access** is checked.
4. Ensure that **Enable FILESTREAM for file I/O stream access** is checked.
5. Ensure that **Allow remote clients to have streaming access to FILESTREAM data** is checked. (see Figure 1-10)

*Figure 1-10*  


Any changes to FILESTREAM configuration will not be implemented until the SQL Server service is restarted.

### Setting the filestream\_access\_level Parameter

By default, the filestream\_access\_level parameter in SQL Server is set to 1 (Enables FILESTREAM for Transact-SQL access). In order for the RBS provider to function properly, this parameter must be set to 2 (Enables FILESTREAM for Transact-SQL and Win32 streaming access).

1. Open SQL Management Studio and click New Query to launch the Query Editor
2. Execute the following commands using the Query Editor:

EXEC sp\_configure filestream\_access\_level, 2

RECONFIGURE

1. To verify that the configuration has been enabled, execute the following command using the Query Editor:

EXEC sp\_configure filestream\_access\_level

The results should look like this:  


## Storage Architecture Considerations

The FILESTREAM provider requires that the content database FILESTREAM file group be stored on a volume that is mounted locally to the SQL Server. It is possible to use NAS technology with iSCSI to mount a remote storage location as a local volume. However, this configuration is only supported if the network storage architecture responds to a ping within 1ms and must return the first byte of data within 20ms.

Regardless of the RBS provider that is implemented, be sure to test storage and retrieval latency as this can have a dramatic impact to the usability of the RBS enabled application. This can be of particular concern with cloud based strategies and certain extreme scale CAS storage systems.

## Implementation Guidance

There are a few possible pitfalls that can cause problems in an RBS implementation. Understand these considerations when deploying RBS:

* **Maintainer Scheduling**. When executing the RBS Feature Pack installer, do not use the installer UI to schedule the Maintainer application. This installation task can fail which can cause the entire RBS Feature Pack installation to roll back. It is better to manually schedule the Maintainer execution after the RBS Feature Pack has been installed.
* **Client Logging Configuration**. When executing the RBS Feature Pack installer using the installer UI, do not change the logging options unless directed to do so by a support technician. Verbose logging can have a negative impact on the performance of an RBS enabled application.

## Installing the RBS Feature Pack

This subsection will provide the background concepts that support the RBS Feature Pack installation. The information below is for any RBS implementation and are not intended to replace SharePoint specific procedures in SharePoint deployment usage of RBS.

### Database Preparation

Before the RBS Feature Pack installer can be executed, the database must first be prepared. First, you must ensure that a database master key is present as this is required for RBS installation. The master key is used for RBS credential encryption and the module signing / schema separation of the RBS internal tables and external views / stored procedures. More information on the master key requirement can be found [in this blog post](http://blogs.msdn.com/b/sqlrbs/archive/2010/08/05/rbs-security-model.aspx) from the SQL RBS Team.

Second, in the case of the FILESTREAM provider, the FILESTREAM file group must be created. [This TechNet topic](http://technet.microsoft.com/en-us/library/ee748631.aspx) addresses the process for properly creating a FILESTREAM file group.

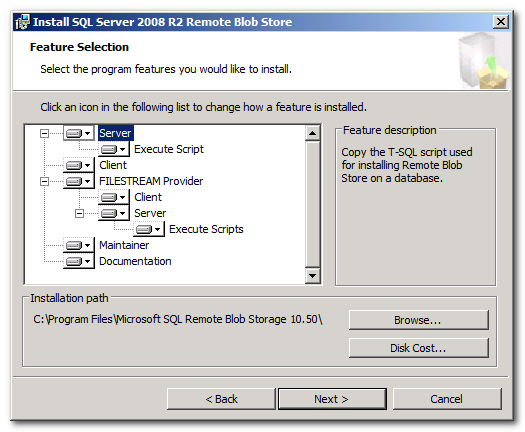
### Executing the RBS Feature Pack Installer

This section describes the individual features and configuration options available in the RBS Feature Pack Installer. The information below intends to use the graphical interface of the RBS Feature Pack installation wizard to illustrate the features and configuration options that are available during RBS Feature Pack installation.

When running the installer, the user will first be presented with the Welcome screen followed by standard License Agreement and Registration screens. The screen definitions below describe relevant information for each of the remaining configuration screens.

For each of the screens, the configuration information will be described and the related command line options will be defined.

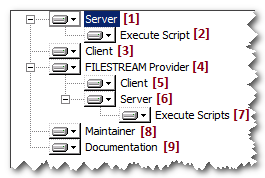
#### Feature Selection (see Figure 1-11)

*Figure 1-11*

The features listed in this screen have already been described in detail in the Component Overview section above. All selected feature components listed on this screen will be deployed and/or executed during the installation process.

The configuration options on this screen coincide with the ADDLOCAL command line parameter in the following way:

|  |
| --- |
| **[1]** ServerScript  **[2]** EnableRbs  **[3]** Client  **[4]** Filestream  **[5]** FilestreamClient  **[6]** FilestreamServer  **[7]** FilestreamRunScript  **[8]** Maintainer  **[9]** Docs |

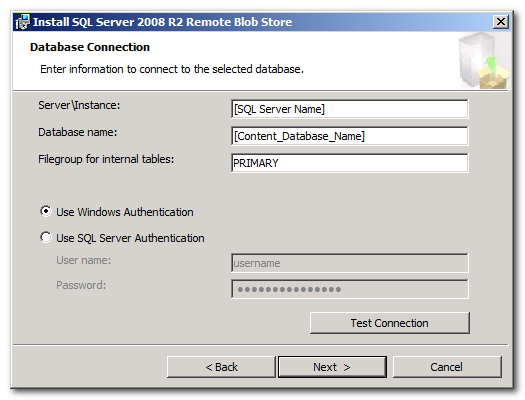


When executing the installer package by command line and using the **ADDLOCAL** command line parameter, one or more of these features can be installed. In order to install more than one feature, separate multiple feature names using a comma. An example of the ADDLOCAL command line parameter is provided:

ADDLOCAL="Client,Docs,Maintainer,ServerScript,FilestreamClient"

If the ADDLOCAL command line parameter is omitted, all features are installed by default.

#### Client Database Connection (see Figure 1-12)

*Figure 1-12*  


The database connection screen provides configuration options that identify the client database for which RBS will be enabled. These options are:

* **Server\Instance:** This is the name of the SQL Server Instance that services the client database. This field coincides with the **DBINSTANCE** command line parameter value. For example:

DBINSTANCE="[SQL Server or Instance Name]"

* **Database name**: This is the name of the client database for which RBS will be enabled. This field coincides with the DBNAME command line parameter. For example:

DBNAME="[Database\_Name]"

* **File group** **for internal tables**: This is the name of the file group in the client database where the RBS auxiliary tables will be stored. This field coincides with the **FILEGROUP** command line parameter. For example:

FILEGROUP="PRIMARY"

* **Windows / SQL Server Authentication:** This flag determines whether or not integrated security will be used when connecting to the client database. This field coincides with the **INTEGRATEDSECURITY** command line parameter. When using the command line, a value of [1] should be provided when integrated security should be used. A value of [0] should be provided when SQL Server Authentication should be used. For example:

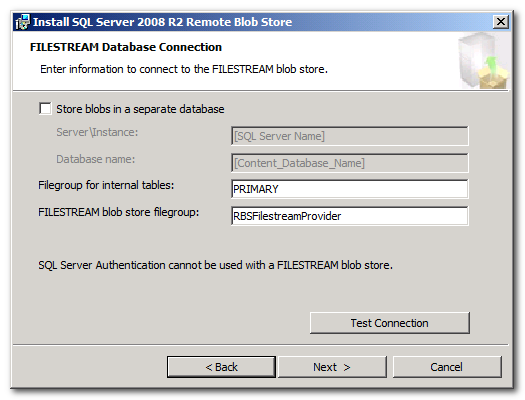
INTEGRATEDSECURITY=1

* **SQL Server Authentication => User Name and Password**: This is the user name and password of the SQL Server security account that should be used to connect to the client database. These values are only valid when SQL Server Authentication is being used.   
    
  **NOTE:** SQL Server Authentication is disabled and should not be used when the FILESTREAM provider is being installed. The FILESTREAM provider does not support SQL Server Authentication.   
    
  These fields coincide with the DBUSERNAME and DBPASSWORD command line parameter. When using the command line and SQL Server Authentication is being used, these parameters are required. For example:

INTEGRATEDSECURITY=0 DBUSERNAME=[UserName] DBPASSWORD=[PASSWORD]

The install user can use the “Test Connection” button to ensure that the installer is able to properly connect to the configured database using the provided credentials.

#### FILESTREAM Database Connection (FILESTREAM Provider Only) (see Figure 1-13)

*Figure 1-13*  


The FILESTREAM database connection screen allows for the configuration of the connection to the database that contains the FILESTREAM file group that will be used as the BLOB store. This configuration only applies when the FILESTREAM provider is being used. The configuration options are:

* **Store blobs in a separate database:** This flag determines if the file groups for the internal FILESTREAM provider tables and the FILESTREAM BLOB store will exist in the same database as the client database or if they will exist in a different database. When the FILESTREAM provider file groups are designated for the same for the same database, this is considered to be a ***Local*** FILESTREAM provider deployment.   
    
  When the FILESTREAM provider file groups are designated to be stored in a database that is not the client database, this is considered to be a ***Remote*** FILESTREAM provider deployment.   
    
  This flag coincides with the **REMOTEFSSTORE** command line parameter. Possible values are [0] which represents a ***Local*** FILESTREAM provider deployment or [1] which represents a ***Remote*** FILESTREAM provider deployment. For example:

REMOTEFSSTORE=0

* **Remote Store Server\Instance and Database** **name**: When the FILESTREAM provider file groups are designated to be stored in a database that is not the client database; these fields identify the SQL Server\Instance and database name where the internal FILESTREAM provider tables and the FIELSTREAM file group blob store should be deployed. These fields coincide with the **FSDBINSTANCE** and **FSDBNAME** command line parameters. For example:

FSDBINSTANCE=”[Server Name\Instance]” FSDBNAME=”[Remote DB Name]”

* **File group for internal tables**: This is the name of the file group where the FILESTREAM provider specific tables will be deployed. This field coincides with the **FILESTREAMDATAFILEGROUP** command line parameter. For example:

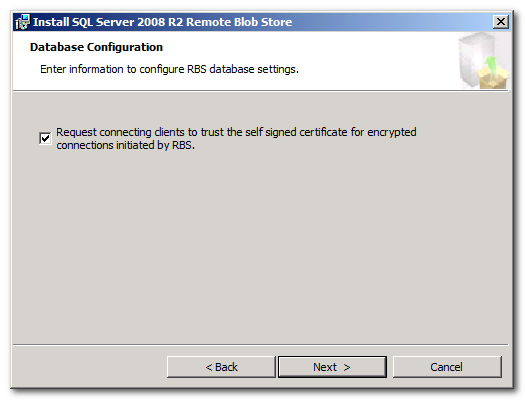
FILESTREAMDATAFILEGROUP=”PRIMARY”

* **FILESTREAM BLOB store file group:** This is the name of the FILESTREAM file group that will be used by the FILESTREAM provider as the BLOB store. The destination FILESTREAM file group must be created before the installer is executed. This field coincides with the **FILESTREAMFILEGROUP** command line parameter. For example:

FILESTREAMFILEGROUP=RBSFilestreamProvider

The install user can use the “Test Connection” button to ensure that the installer is able to properly connect to the configured database using the provided credentials.

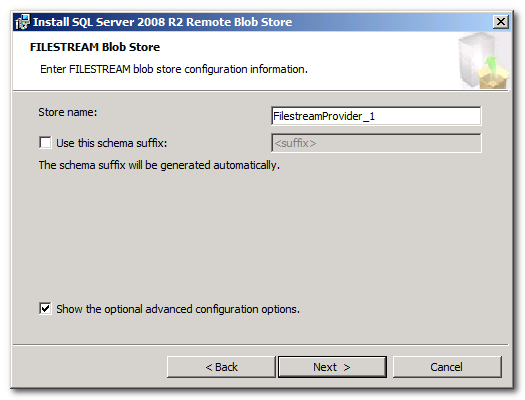
#### FILESTREAM Database Connection Trust (see Figure 1-14)

*Figure* 1-14  


This flag indicates whether or not connecting clients will be requested to trust the self-signed certificates for encrypted connections initiated by RBS. This means that an SSL channel can be set up without requiring the server to have a certificate chain that terminates in a root Certificate Authority (CA). This flag relates to the TrustServerCertificate keyword for SQL Server connection strings and should generally be enabled. This flag coincides with the **TRUSTSERVERCERTIFICATE** [connection string parameter in ADO.NET](http://msdn.microsoft.com/en-us/library/ms254500.aspx). Valid values are [1] to enable the request or [0] to not enable the request. For example:

TRUSTSERVERCERTIFICATE=1

#### FILESTREAM BLOB Store Configuration (see Figure 1-15)

*Figure 1-15*  


The FILESTREAM BLOB Store configuration screen allows for the configuration of the BLOB store name and schema suffix in order to allow for the installation of more than one provider instance into the same database without conflicting schemas in the database. It also provides a flag that determines if the installation UI will present the advanced configuration options on the subsequent screen in the wizard.

* **Store Name**: This is the name of the BLOB store that is used to distinguish different BLOB stores both internally and externally. This field coincides with the **FILESTREAMSTORENAME** command line parameter. For example:

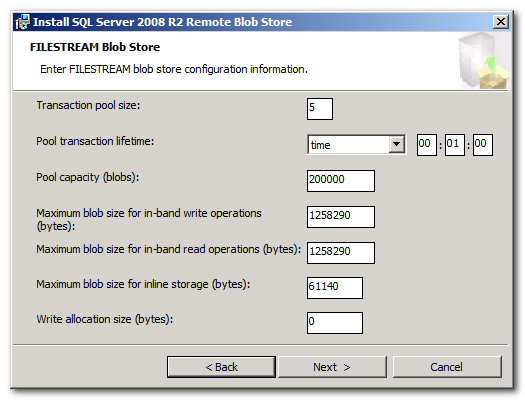
FILESTREAMSTORENAME=FilestreamProvider\_1

* **Schema Suffix:** When provided, this suffix will be appended to the name of the internal FILESTREAM provider schema in order to ensure that the instance of the FILESTREAM provider is unique in the database. This allows more than one provider instance to be deployed in the same database. This field coincides with the **SCHEMASUFFIX** command line parameter. For example:

SCHEMASUFFIX=samplesuffix1

* **Show Advanced Configuration Options:** This flag is for installation wizard UI purposes only. It determines whether or not the advanced configuration options screen (described below) will be displayed.

#### FILESTREAM BLOB Store Configuration (Advanced) (see Figure 1-16)

*Figure 1-16*

The FILESTREAM BLOB Store Advanced Configuration screen allows the installer to manipulate a more advanced set of parameters. In general, the default values represent the best value to ensure the best possible RBS performance. However, there may be times when it makes sense to adjust these values.

* **Transaction pool size**: This number represents the size of the internal “read” transaction pool. A read transaction is necessary for BLOB retrieval. Read transactions are opened on demand and then pooled for the transaction lifetime. The default value of 5 should be used unless instructed otherwise by Microsoft support. This field coincides with the FILESTREAMTRANPOOLSIZE command line parameter. For example:

FILESTREAMTRANPOOLSIZE=5

* **Pool transaction lifetime:** This is the length of time that “read” transactions (described above) are allowed to live before they are drained of any open handles and closed. The default value of 1 minute should be used unless instructed otherwise by Microsoft support. This field coincides with the FILESTREAMTRANPOOLTRANLIFETIME command line parameter. The value for this parameter is expressed as a text value that represents a duration expression. For example:

FILESTREAMTRANPOOLTRANLIFETIME=”time 00:01:00”

* **Pool Capacity:** This is number of BLOBs that constitute a BLOB store pool that is “full”. Once the pool is full, RBS creates a new pool for the storage of additional BLOBs. The size of the pool is important because it defines the maximum number of documents that can be processed by the Maintainer application during a garbage collection session.   
    
  Smaller pools decrease maintenance time and result in fewer documents being stored in a pool folder in the file system. However, the FILESTREAM provider creates a table for every pool in the database. So it is important to seek balance when determining the size of the pool. The 200,000 default value is a good start. It might be worth decreasing this value if the documents being stored are generally very large but fewer in numbers such as large graphics or video files that might be stored in a Digital Asset Management application.  
    
  This field coincides with the FILESTREAMMAXPOOLSIZE command line parameter. For example:

FILESTREAMMAXPOOLSIZE=200000

* **Maximum BLOB size for in-band write operations:** This value will determine, based on the number of bytes in the file being stored, whether the FILESTREAM provider will use SQL FILESTREAM (in-band) using TDS protocol or Win32 streaming APIs (out-of-band) to write the BLOB to the BLOB store. The default value of 1258290 should generally remain as the default. If write performance is less than optimal, testing file uploads using a higher or lower value may yield improved performance. This field coincides with the FILESTREAMMAXSIZEINBANDWRITE command line parameter. For example:

FILESTREAMMAXSIZEINBANDWRITE=1258290

* **Maximum BLOB size for in-band read operations:** This value will determine, based on the number of bytes in the file being retrieved, whether the FILESTREAM provider will use SQL FILESTREAM (in-band) using TDS protocol or Win32 streaming APIs (out-of-band) to read the BLOB data from the BLOB store. The default value of 1258290 should generally remain as the default. If read performance is less than optimal, testing file retrievals using a higher or lower value may yield improved performance. This field coincides with the FILESTREAMMAXSIZEINBANDREAD command line parameter. For example:

FILESTREAMMAXSIZEINBANDREAD=1258290

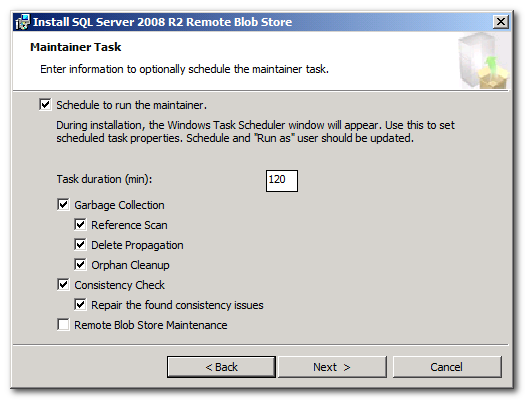
* **Maximum BLOB size for inline storage:** This value will determine, based on the number of bytes in the file being stored, whether the BLOB will be stored “inline” as a varbinary(max) value in the mdf file of the FILESTREAM provider database or as a FILESTREAM value in the file system. The SQL FILESTREAM feature operates more efficiently with larger files. Storing a large number of extremely small files can degrade performance. This value ensures that a minimum file size is present before the BLOB is written to the SQL FILESTREAM column. It is possible to ensure that ALL BLOBs are written to SQL FILESTREAM column by setting this field to a value of zero (0). This field coincides with the FILESTREAMMAXSIZEINLINEBLOB command line parameter. For example:

FILESTREAMMAXSIZEINLINEBLOB=61140

* **Write allocation size:** This value determines the initial size of the file that is created in the NTFS file system when a new BLOB file is being created. The default value of zero (0) should not be changed unless instructed to do so by Microsoft Support. This field coincides with the FILESTREAMWRITEALLOCATIONSIZE command line parameter. For example:

FILESTREAMWRITEALLOCATIONSIZE=0

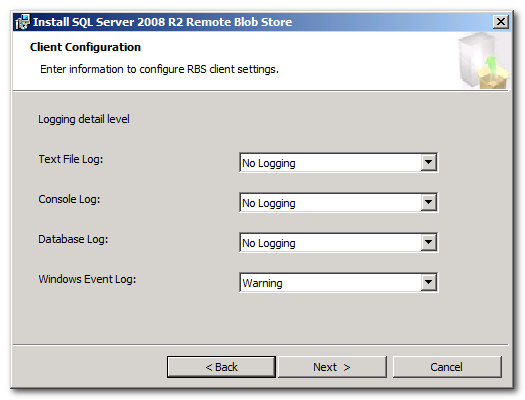
#### Maintainer Task Configuration (see Figure 1-17)

*Figure* 1-17  


The Maintainer task configuration screen facilitates the scheduling of the Maintainer in order to ensure that garbage collection is performed. Selecting the “Schedule to run the maintainer” will cause the installer UI to launch a scheduled task window. The installer will pre-configure some aspects of the task including task duration and the command line parameters that are present based on the options selected in this screen.

Detailed information regarding the configuration options of the Maintainer application will be defined in detail in a later section.

#### Client (Logging) Configuration (see Figure 1-18)

Figure 1-18  


The Client Configuration screen allows the installer to configure the logging options for the RBS client library. Each of the log configuration values can be expressed in a coinciding command line parameter numeric value ranging from 1 to 7. They are:

1. No Logging
2. Critical Error
3. Error
4. Warning
5. Information
6. Developer
7. Verbose

* **Text File Log:** This value will determine the log level of the client library text file log. This field coincides with the TEXTFILEVALUE command line parameter. For example:

TEXTFILEVALUE=1

* **Console Log:** This value will determine the log level when console commands invoke the client library. This field value affects the verbosity of command results when using the command console or PowerShell. This field coincides with the CONSOLEVALUE command line parameter. For example:

CONSOLEVALUE=1

* **Database Log:** This value will determine the log level for log entries stored in the RBS auxiliary table. This field coincides with the DBTABLEVALUE command line parameter. For example:

DBTABLEVALUE=1

* **Windows Event Log:** This value will determine the log level for log entries stored in the Windows Event Log. This field coincides with the EVENTLOGVALUE command line parameter. For example:

EVENTLOGVALUE=4

### Installing on Multiple Servers

Once the RBS Feature Pack installer has been used to deploy all RBS components to the application database and the initial application server, only a subset of the RBS Feature Pack components need to be installed on additional servers in a multiple server farm. For all subsequent servers, the following RBS Feature Pack installer command should be executed:

msiexec /qn /lvx\* rbs\_install\_log.txt /i RBS\_X64.msi DBNAME="ContentDbName" DBINSTANCE="DBInstanceName" ADDLOCAL="Client,Docs,Maintainer,ServerScript,FilestreamClient,FilestreamServer"

This command breaks down as follows:

* **msiexec:** This is the executable name for the MS Installer utility.
* **/qn**: Indicates that the installer should run **silently** with **no User Interface** (UI).
* **/lvx\***: Indicates that the **verbose** output of the install process should be **logged** to a file with **extra debugging information**. The **\*** indicates that **all information** other than verbose and extra debugging information should be logged.
* **Rbs\_install\_log.txt**: This is the path and file name of the **output log file**.
* **/i**: Indicates that MSIEXEC should install a certain product package.
* **Rbs\_x64.msi**: Is the path and file name of the product package to be installed.
* **DBNAME**: The name of the client database.
* **DBINSTANCE**: The SQL Server\Instance name where the client database is contained.
* **ADDLOCAL**: Represents the RBS features that will be installed. When installing to a server that hosts the client application, the Client, Docs, Maintainer, ServerScript, FilestreamClient, FilestreamServer features should be installed:
  + **Client**: Installs the client library (Microsoft.Data.SqlRemoteBLOBs.dll) and deploys it to the Global Assembly Cache (GAC)
  + **Docs**: Installs the RBS documentation. This component is optional.
  + **Maintainer**: Installs the Maintainer application (Microsoft.Data.SqlRemoteBLOBs.Maintainer.exe).
  + **ServerScript**: Installs the server script. This script can be executed manually at a later time to create the RBS auxiliary resources to other content databases.
  + **FilestreamClient**: Installs the FilestreamClient library (Microsoft.Data.BLOBStores.FilestreamBLOBStore.dll) and deploys it to the GAC.
  + **FilestreamServer**: Installs the necessary scripts that construct the FILESTREAM provider tables and resources in the content database. These scripts can be executed manually at a later time to install the FILESTREAM provider server resources to other content databases.

Essentially, this command does not execute any server scripts for creating a new BLOB store. It simply deploys the necessary libraries and configuration to allow a second server that hosts the client application to communicate with a client database that has already been RBS enabled.

## Configuring RBS

The RBS Feature Pack exposes little in the way of an administrative interface for configuring RBS components. However, there are quite a few knobs and buttons that can be tweaked, allowing for further control over RBS functionality.

### BLOB Store Core Configuration Settings

BLOB Store core configuration settings are typically configuration values that facilitate BLOB store READ operations. These values are stored as XML in the core\_configuration column of the mssql\_resources.rbs\_internal\_blob\_stores table in the RBS auxiliary tables of the client database.

For the FILESTREAM provider, several of the advanced configuration (see Figure 1-16) values as well as the schema suffix value (see Figure 1-15) are stored in this field:

* schema\_suffix
* max\_size\_inband\_read
* tran\_pool\_size
* tran\_pool\_tran\_lifetime

### BLOB Store Extended Configuration Settings

Extended configuration settings are typically configuration values that facilitate BLOB store WRITE operations. These values are stores as XML in the extended\_configuration column of the mssql\_resources.rbs\_internal\_blob\_stores table in the RBS auxiliary tables of the client database.

For the FILESTREAM provider, the BLOB store connection information (See Figure 1-13) values as well as several of the advanced configuration (see Figure 1-16) values are stored in this field:

* filegroup\_name
* data\_filegroup\_name
* max\_pool\_size
* max\_size\_inband\_write
* max\_size\_inline\_blob

### RBS Configuration Settings

In addition to the BLOB store configuration, there are several more general RBS configuration settings. The current values for RBS general settings can be seen by opening the mssqlrbs.rbs\_config view. To update these settings, use the mssqlrbs.rbs\_sp\_set\_config\_value stored procedure. These configuration items are:

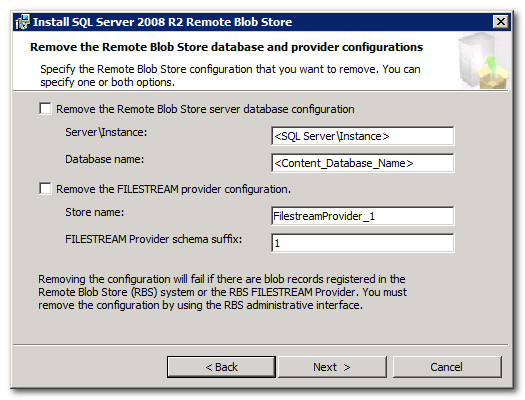
|  |  |
| --- | --- |
| **Configuration Item** | **Description** |
| default\_blob\_store\_name | Specifies the name of BLOB store that will be used if no default for collection is specified and no store is specified at runtime, must be a name of existing blob store. No default value is defined. |
| min\_client\_library\_version\_required | No application using a version of RBS lower than the value specified by this key will be allowed to work with this database. The default value is 10.0.0.0. |
| set\_xact\_abort | Determines whether xact\_abort should be set for RBS stored procedures. The default value is false. |
| set\_trust\_server\_certificate | Determines whether RBS should trust the server certificate for creating an encrypted connection to download BLOB store credentials. The default value is false. |
| maintain\_blob\_id\_compatibility | Determines whether the BlobId for new BLOBs must be formatted for SQL Server 2008. SQL Server 2008 used an old format for BlobId which has been deprecated since SQL Server 2008 R2. Set this value to true only if you need to interoperate with the first version of RBS (SQL Server 2008. The default value is false. |
| disable\_pool\_slicing | Determines whether optimization of the orphan cleanup phase of garbage collection is disabled. This optimization can provide better garbage collection task granularity, but can also cause compatibility issues for the SQL Server 2008 version of the RBS Maintainer. The default value is false. |
| gcr\_num\_blobs\_per\_work\_unit | Specifies the number of BLOBs in a single work unit for the reference scan phase of garbage collection. This setting can be used to control task granularity. The default value is 100000. |
| gcr\_num\_blobs\_per\_iteration | Specifies the number of BLOBs that can be checked in a single transaction for the reference scan garbage collection phase. This value can be used to control the tradeoff between the overhead of creating and committing transactions and number of locks held at any given time. The default value is 1000. |
| gco\_num\_blobs\_per\_iteration | Specifies the number of BLOBs that can be checked in a single transaction for the orphan cleanup garbage collection phase. This value can be used to control the tradeoff between the overhead of creating and committing transactions and number of locks held. The default value is 1000. |
| gcd\_num\_blobs\_per\_iteration | Specifies the number of BLOBs that can be checked in a single transaction for the delete propagation garbage collection phase. This value can be used to control the tradeoff between the overhead of creating and committing transactions and number of locks held. The default value is 1000. |
| gco\_enum\_num\_blobs\_per\_iteration | Specifies the number of BLOBs to enumerate in a single call to the BLOB store for orphan cleanup. The default value is 1000. |
| max\_consistency\_issues\_found | This applies to the consistency check operation performed by maintainer and is the number of consistency issues of a single type that can be found before further checks will be cancelled. The default value is 1000. |
| max\_consistency\_issues\_returned | Specifies the maximum number of consistency issues that will be reported by the maintainer. The default value is 100. |
| delete\_scan\_period | Specifies the minimum amount of time that must pass between two complete passes of reference scan garbage collection. Note that the reference scan phase can run multiple times within this period, making incremental progress during each run... The default value is 30 days. |
| orphan\_scan\_period | Specifies the minimum amount of time that must pass between two complete passes of reference scan garbage collection. Note that the reference scan phase can run multiple times within this period, making incremental progress during each run... The default value is 30 days. |
| garbage\_collection\_time\_window | Specifies the minimum time that must pass between identifying a blob as having no references in the database and deleting the blob from the store. This guarantees the availability of BLOBs for the specified time in case a backup is restored. The default value is 30 days. |
| garbage\_collection\_slice\_duration | Specifies the length of a time in a single orphan cleanup phase garbage collection work unit. A value of 0 means that a single garbage collection slice should cover the entire duration of the cleanup, without dividing it into smaller pieces. This setting is ignored if disable\_pool\_slicing is set to true. The default value is 7 days. |
| configuration\_check\_period | When the RBS Client Library encounters some errors in blob operations, it checks the database RBS configuration to determine whether some configuration settings have been changed recently. It will force a configuration refresh on the client if necessary. However, in some cases of transient failures these checks may result in a high burden for the database. This option may place a limit of frequency of these checks to avoid it. The default value is 00:00:00, meaning that the check will be performed every time blob operation errors are encountered. |

## Uninstalling RBS (and the FILESTREAM Provider)

There are certain circumstances when it is necessary to remove RBS and/or the FILESTREAM provider from a client database. This section covers the steps necessary for uninstalling RBS and/or the FILESTREAM provider.

### Remove BLOB Store Database and Provider Configurations, Uninstall RBS

Launch the RBS\_x64.msi installer. Select options for removing the RBS server database configuration and the FILESTREAM provider configuration. (see Figure 1-19)

*Figure 1-19*  


Executing this uninstaller with both options checked will cause the RBS Feature Pack installer application to execute a couple of stored procedures:

* **mssqlrbs\_filestream.rbs\_fs\_sp\_uninstall**: This stored procedure removes all FILESTREAM BLOB store database resources from the content database.
* **mssqlrbs.rbs\_sp\_uninstall\_rbs**: This stored procedure removes all RBS auxiliary resources from the content database.

**NOTE**: It will not be possible to execute the rbs\_sp\_uninstall\_rbs and rbs\_fs\_sp\_uninstall stored procedures if there are existing BLOBs registered in the BLOB store.

Continue with the uninstallation wizard to completely remove all RBS components from the local server. If the FILESTREAM and RBS database configuration components have already been removed during the uninstallation process of another server, it is not necessary to select the removal options during subsequent uninstallations. (see Figure 1-19)

After all RBS and FILESTREAM components have been removed, the final step is to manually remove the FILESTREAM file group that was manually created during installation. Open a query window in SQL Management Studio and execute the following command to completely remove the FILESTREAM file group:

ALTER DATABASE <Database\_Name> REMOVE FILEGROUP RBSFilestreamProvider

## Maintenance and Administration

RBS is not a “set it and forget it” solution. RBS must be monitored to ensure that it is functioning properly and maintenance jobs must be executed on a regular basis to ensure that the application database, the RBS database tables, and the BLOB store are synchronized at all times. As previously mentioned in this document, garbage collection needs to be performed periodically to reclaim unused resources and maintain RBS components.

### The Maintainer application

The RBS Maintainer is a console application that uses the client library for the purposes of garbage collection and other maintenance tasks. It is a component of the RBS Feature Pack that is both client application and RBS provider agnostic. It was created to ensure that a generic but fully tested maintenance utility is available for all RBS implementations.

Because Maintainer is a console application, it can be scheduled using Windows Task Scheduler or SQL Agent. It can be run on either the database server machine or on another server in the client application server farm. Maintainer operates in the context of a specific database connection.

TechNet has recently provided comprehensive information for scheduling and executing the Maintainer application in [this TechNet topic](http://technet.microsoft.com/en-us/library/ff943565.aspx). This document is not intended to replace the TechNet guidance however there are three RBS configuration settings that have a significant effect on Maintainer processing.

#### Garbage Collection Time Window

Consider this scenario:

* RBS is enabled and all document binary data is stored in the BLOB store.
* A user deletes a document from the client application on Monday which (in this scenario) deletes the corresponding binary data from the BLOB store.
* On Tuesday, the user determines that the document must be restored.
* SQL administrators work to restore the database to Monday morning, just before the document was deleted.

In this scenario, even after the database is restored, the document can’t be retrieved because even though the database contains a reference to the BLOB, the binary data has been irreversibly deleted from the BLOB store. This violates link-level consistency.

In order to prevent this scenario, RBS provides a time window parameter called garbage\_collection\_time\_window which ensures that even after a document record has been deleted from the client application database, the BLOB data will not be removed from the BLOB store until the restore SLA window has passed. When RBS is deployed to a client application database, the administrator should set the garbage\_collection\_time\_window value based on the restore SLA requirement for the organization. The default garbage collection time window is 30 days. Use the following example to set the default garbage collection time window to 14 days:

exec mssqlrbs.rbs\_sp\_set\_config\_value 'garbage\_collection\_time\_window', 'days 14'

#### Delete Scan Period

In order to prevent Reference Scan and Delete Propagation processing from impacting client database performance, RBS includes a delete\_scan\_period configuration parameter. By default, this parameter is set to 30 days. If Maintainer is used to run RS and DP garbage collection phases inside of 30 days from the last completed run, these phases will be ignored. However, if it is necessary to execute this garbage collection more frequently than the default 30 days, the delete\_scan\_period parameter may be modified:

exec mssqlrbs.rbs\_sp\_set\_config\_value delete\_scan\_period', 'days 7'

#### Orphan Scan Period

Similar to delete scan period, the orphan\_scan\_period parameter prevents Orphan Cleanup garbage collection phase from being executed too frequently by Maintainer. Once again, the default value is 30 days. If it is necessary to run this phase more frequently, the orphan\_scan\_period parameter may be modified:

exec mssqlrbs.rbs\_sp\_set\_config\_value orphan\_scan\_period', 'days 7'

### Monitoring the BLOB Store

It is important to monitor a couple of key components of the RBS system. Allowing these factors to grow wildly out of control can degrade client application performance and potentially lead to production down time.

#### BLOB Store Pools

BLOB Store pool size is something that needs to be carefully calculated, tested, and monitored. Some RBS providers, such as the FILESTREAM provider, create a tracking table for each pool. By setting pool sizes to a very small number, the number of tables created in the database can number in the thousands very quickly.

However, the size of the pool also reflects the size of the work unit that Maintainer uses to perform garbage collection phases. So a very high maximum pool size can result in the maintainer taking a very long time to complete a work unit.

It is important to monitor the number of pools being created in the client database and also Maintainer application performance, then adjust pool size up or down if necessary.

#### Disk Space Considerations

RBS storage monitoring is similar to SQL database storage monitoring such that if the BLOB store runs out of space, it will no longer be possible to add documents to the client database. It is important to keep an eye on the rate that storage is being consumed. If storage is beginning to run out, it is possible to add a new BLOB store to the client database. The new BLOB store would be located on an additional storage volume or CAS location.

### Performance Counters

After the RBS Feature Pack is installed, several counters are made available in the Remote BLOB Storage counter category. The following performance metrics can be tracked using Performance Monitor or other monitoring tool:

|  |  |
| --- | --- |
| Active context pools  Active contexts  BLOBs created avg. blob store time  BLOBs created avg. client library time  BLOBs created avg. SQL round trips  BLOBs created avg. time  BLOBs created/sec  BLOBs read avg. blob store time  BLOBs read avg. client library time  BLOBs read avg. SQL round trips  BLOBs read avg. time  BLOBs read/sec  Contexts created/sec | Context invalidated (configuration)/sec  Contexts invalidated (user)/sec  Contexts trimmed/sec  Contexts used/sec  Nonactive context pools  Nonactive contexts  Nonpooled contexts  Pools created avg. blob store time  Pools created avg. client library time  Pools created avg. SQL round trips  Pools created avg. time  Pools created/sec |

### Database Resources

Each of the RBS Auxiliary tables, views, and stored procedures in the client (content) are a member of either the Application Interface or the Administrative Interface. The Administrative Interface consists of one table and multiple views that can be accessed (read only) and multiple stored procedures that may be executed by administrators or custom code. The Application Interface consists of tables, views, and stored procedures that are used by the client library and Maintainer application for internal RBS operations. They should not be modified or directly accessed by administrators or custom code.

#### Administration View

Only the rbs\_log table is exposed for read only access by administrators and custom code. The following views in the mssqlrbs schema are exposed for read only access by administrators and custom code:

|  |  |
| --- | --- |
| rbs\_blob\_details  rbs\_blob\_references  rbs\_blob\_stores  rbs\_collections  rbs\_columns  rbs\_config | rbs\_consistency\_issues  rbs\_counters  rbs\_dictionary  rbs\_history  rbs\_pools |

The following functions and stored procedures in the mssqlrbs schema are exposed for execution by administrators and custom code:

|  |  |
| --- | --- |
| rbs\_fn\_get\_rbs\_state  rbs\_sp\_enable\_rbs  rbs\_sp\_uninstall\_rbs  rbs\_sp\_set\_config\_value  rbs\_sp\_delete\_config\_value  rbs\_sp\_register\_column  rbs\_sp\_unregister\_column  rbs\_sp\_add\_blob\_store  rbs\_sp\_add\_pool  rbs\_sp\_modify\_collection | rbs\_sp\_modify\_blob\_store  rbs\_sp\_delete\_blob\_store  rbs\_sp\_set\_blob\_store\_credential  rbs\_sp\_get\_all\_blob\_store\_credentials  rbs\_sp\_delete\_blob\_store\_credential  rbs\_sp\_reset\_counter  rbs\_sp\_mark\_pool\_as\_full  rbs\_sp\_register\_blob  rbs\_sp\_cleanup\_unmaintained\_collection \*  \* Being added in SQL 2008 R2 PCU1 |

Most of the stored procedures above will be used by utility and configuration applications during RBS deployment. However the rbs\_sp\_set\_config\_value stored procedure and the rbs\_config view are particularly useful. Administrators can use these to fine tune the configuration of RBS.

#### Application View

The following tables from the mssqlrbs\_resources schema are INTERNAL and should not be accessed or modified by administrators or end users:

|  |  |
| --- | --- |
| rbs\_internal\_blob\_store\_credentials  rbs\_internal\_blob\_stores  rbs\_internal\_blobs  rbs\_internal\_collections  rbs\_internal\_columns  rbs\_internal\_config  rbs\_internal\_consistency\_issues  rbs\_internal\_counters | rbs\_internal\_deleted\_blobs  rbs\_internal\_dictionary  rbs\_internal\_history  rbs\_internal\_indices  rbs\_internal\_pools  rbs\_work\_blob\_store\_blobs  rbs\_work\_units |

The following views from the mssqlrbs\_resources schema are INTERNAL and should not be accessed or modified by administrators or end users:

|  |  |
| --- | --- |
| rbs\_all\_counters  rbs\_all\_history | rbs\_work\_all\_blobs\_used\_view  rbs\_work\_blobs\_used\_view |

Essentially all remaining stored procedures that are not mentioned in the Administration Interface are called by the client library, provider libraries, or the Maintainer application. They should not be modified or executed by administrators or custom code.

## Backup and Restore

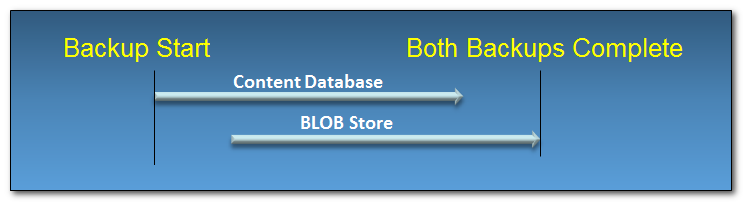
It is essential that backup and restore be carefully managed when RBS is enabled on a client database. With RBS enabled, the BLOB data is not automatically backed up when the content database is backed up. However some backup and restore solutions may account for this fact automatically. For example, in a SharePoint 2010 RBS implementation, SharePoint console backup commands do include BLOB data as long as the RBS provider supports it. However, there is an upper limit of 85GB for SharePoint 2010 site collection backups. Of course, this is a SharePoint specific example but the general principle applies to other RBS applications as well. The backup and restore procedures for any given RBS client application should ensure that link level consistency is not violated and Service Level Agreements (SLAs) are honored. Consider this backup scenario:

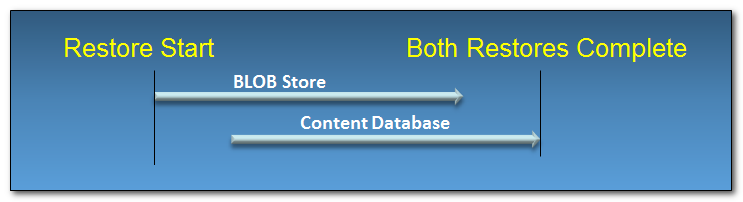
* The BLOB store backup process is started
* During the BLOB store backup, a user inserts a new document which is added to the BLOB store but not in time to be included in the backup.
* The related content database is backed up, including the record for the new document.
* A restore must be performed due to a catastrophic failure of the storage device that contains the BLOB store. The BLOB store is restored
* In order to ensure consistency, the client application database is restored to the backup that corresponds with the BLOB store backup.

In this scenario, link level consistency is broken because the restored application database will have a reference to the document that the user submitted during the BLOB store backup. Since the BLOB store will not have that document, the user is unable to open the document. In this scenario, the order of the backup and restore operations is not correct and inconsistent content retrieval could be experienced by end users.

### Prescriptive Guidance

Backup and restore operations should always be performed in this order:

**Backup**:  


**Restore**:  


**NOTE:** If LOCAL RBS FILESTREAM Provider is being used, content database backup operations will also backup the BLOBs at the same time which simplifies the backup and restore process.

### Synchronizing RBS Configuration with Backup Schedule

**IMPORTANT NOTE:** To insure that link level consistency is maintained, it is important to ensure that the Maintainer application is **NOT** running during backup operations.

Once the backup and restore SLA has been determined, the *Garbage Collection Time Window*, *Delete Scan Period*, and *Orphan Scan Period* should be appropriately set using the procedures defined above. Then the Maintainer application should be scheduled to execute in harmony with the garbage collection parameters and with respect to backup and restore processing windows.

# SharePoint 2010 and RBS

Now that Remote Blob Storage has been covered in detail, it is helpful to understand how RBS can facilitate the binary externalization requirements of a specific application. The most prolific example of an application that supports RBS is SharePoint 2010. The binary externalization option in SharePoint 2007 consisted only of External Binary Storage (EBS).

### Differences between EBS and RBS

External Binary Storage (EBS) was introduced as a first generation binary externalization solution for SharePoint 2007 (as of SP1). It is a basic API that at least provided an option for externalization. However there are several drawbacks to EBS. Below is a table that contains a matching of EBS issues or concerns with RBS functional improvements.

|  |  |
| --- | --- |
| **EBS Concern** | **RBS Improvement** |
| Every time a BLOB is updated, the EBS provider does not overwrite the existing BLOB. Instead a new BLOB is created and all references to the old BLOB are destroyed which results in an orphaned blob. This paradigm can cause significant concerns with respect to maintenance operations. The orphaned BLOB count can grow rapidly requiring frequent garbage collection to prevent significantly inflated storage requirements. | When a BLOB is updated, a new copy of the BLOB is stored in the BLOB store but old copies are tracked in the RBS auxiliary tables in order to support backup and restore operations. After a configurable amount of time has passed, the Maintainer and RBS client library take advantage of efficient SQL calls to identify old copies of the document that may be deleted. |
|  |  |
| EBS providers do not support transaction consistency or cancellation support. If a Write Once Read Many (WORM) storage device elects to “VETO” a delete operation, the delete operation is not halted. The SharePoint row would be deleted and the BLOB would be orphaned. | RBS supports transactional consistency which guarantees that if a link/pointer to a BLOB exists in the database, the BLOB will be guaranteed to exist in the BLOB store in a fully written/committed state. If the BLOB store fails to write the BLOB and throws an exception, the entire upload operation will fail and the record will not be written to the client database. |
|  |  |
| There is no support for maintenance or migration operations in the EBS API. Provider developers must build tools that perform orphan cleanup and manage the externalization of any BLOB data that exists in the content database before EBS was enabled. | The RBS Feature Pack includes the Maintainer application in an effort to provide efficient and consistent lazy garbage collection capabilities that independent of any RBS provider development. SharePoint provides Management Shell support for migrating BLOB data between the BLOB store and the content database. |
|  |  |
| EBS must be enabled at the scope of the farm (SPFarm). | RBS is enabled at the scope of the content database. So by creating site collection in different content databases, the scope can be as granular as the site collection. |
|  |  |
| Support for EBS is not guaranteed past SharePoint 2010 | RBS will be supported in future versions of SharePoint. |

It is important to understand that most of these concerns can be addressed by a properly architected and developed EBS provider. The point here is that RBS was designed from the ground up to support client applications such as SharePoint.

### Enabling an RBS Provider in SharePoint 2010

TechNet provides step by step procedures for [implementing RBS with FILESTREAM provider for SharePoint 2010](http://technet.microsoft.com/en-us/library/ee748631.aspx) and [implementing RBS without FILESTREAM provider for SharePoint 2010](http://technet.microsoft.com/en-us/library/ff629463.aspx). These procedures are more specific than those mentioned above for a generic RBS implementation. TechNet procedures for implementing RBS in SharePoint 2010 should always be referenced first. The information in this document is intended to be complimentary.

Once the RBS feature pack is installed and deployed to a given content database, PowerShell can be used to enable an RBS provider on that database. TechNet describes PowerShell commands that should be executed to enable an RBS provider [in this topic](http://technet.microsoft.com/en-us/library/ee748631.aspx). These PowerShell commands are part of the SharePoint 2010 command shell. They are unique to the SharePoint implementation of RBS. It is beyond the scope of this document to duplicate that information. However, it is important to understand what each of the commands accomplishes.

#### Acquire a Reference to the SPContentDatabase Object

An RBS provider is enabled at the scope of the content database. So in order to manipulate RBS settings, it is necessary to acquire a reference to the SPContentDatabase object. The procedure lists these two commands:

1. $cdb = Get-SPContentDatabase –WebApplication <http://SiteName>

2. $rbss = $cdb.RemoteBLOBStorageSettings

The first statement instantiates a reference to the SPWebApplication using the root url. The second statement stores the SPContentDatabase object reference from the SPWebApplication in the $rbss variable.

There are other ways to acquire a reference to the SPContentDatabase object. What if the content database contains only one site collection that is not located at the web application root? The commands below accommodate this situation by acquiring a reference to the content database using the url of a Site instead:

$cdb = Get-SPContentDatabase –Site <http://Root/Sites/SiteName>

$rbss = $cdb.RemoteBLOBStorageSettings

Or:

$site = Get-SPSite <http://Root/Sites/SiteName>

$rbss = $site.ContentDatabase.RemoteBLOBStorageSettings

#### Ensure RBS is “Installed”

Next, the procedure instructs the user to execute:

$rbss.Installed()

This command initiates a connection to the database and then checks to see if the RBS Auxiliary tables and resources have been deployed to the content database. If a value of TRUE is not returned from this call, then the RBS Feature Pack Installation (server component) did not install successfully and an RBS Provider can’t be enabled.

#### Enabling RBS for the Content Database

Next, the procedure instructs the user to execute:

$rbss.Enable()

This command causes SharePoint to open a SqlRemoteBLOBContext and create a new ***collection*** for this content database. This operation returns a new RbsCollectionId which SharePoint stores in the DatabaseInformation table in the content database. The RbsCollectionId is later used during BLOB storage and retrieval operations.

#### Setting the Active Provider BLOB Store

Next, the procedure instructs the user to set the active RBS provider name:

$rbss.SetActiveProviderName($rbss.GetProviderNames()[0])

This command uses the first provider name in the GetProviderNames() method enumeration and sets it as the active provider BLOB store name for the content database. The GetProviderNames() method retrieves a list of BLOB store names from the RBS auxiliary tables in the content database.   
  
Because a content database may have more than one BLOB store, the first BLOB store in the enumeration may not be the desired active BLOB store. In this case, use the GetProviderNames() method of the SPRemoteBLOBStorageSetting object ($rbss variable) to retrieve a list of all BLOB stores associated with the content database. It is then possible to set the active provider BLOB store name manually:

$rbss.SetActiveProviderName(“FilestreamProvider\_1”)

#### Verifying the Active Provider

Finally, the procedure instructs the user to output the value of the SPRemoteBLOBStorageSetting object:

$rbss

Enabled ActiveProviderName MinimumBLOBStorageS UpgradePersistedPr  
 ize operties

------- ------------------ ------------------- ---------------

True FilestreamProvid... 0 {}

This command retrieves the status of the RBS settings for the content database. If properly installed, Enabled should be True and the proper provider name should be listed under ActiveProviderName.

#### Disabling the Active Provider

It is possible to temporarily “turn off” RBS and return to normal content database storage operations without affecting the items that are already stored in the BLOB store. Execute the following command to temporarily disengage RBS processing:

$rbss.SetActiveProviderName(“”)

**WARNING!** Do not use the “Disable()” method of the RemoteBlobStorageSettings object as this will remove RBS collection information from the database. This method should ONLY be used if RBS is being completely uninstalled.

### Enabling the Provider on Multiple Content Databases

Once the RBS Feature Pack installer has been used to deploy all RBS components to the content database it is possible to use the RBS Feature Pack installer to execute an enable operation on other content databases. The complete procedure for executing this installation can be found in [this TechNet topic](http://technet.microsoft.com/en-us/library/ee748641.aspx). For all subsequent content databases, the following RBS Feature Pack installer command should be executed:

msiexec /qn /i RBS\_X64.msi REMOTEBLOBENABLE=1 FILESTREAMPROVIDERENABLE=1 DBNAME=<ContentDbName> FILESTREAMSTORENAME=FilestreamProvider\_1 ADDLOCAL=EnableRBS,FilestreamRunScript DBINSTANCE=<DBInstanceName>>

This command breaks down as follows:

* **msiexec:** This is the executable name for the MS Installer utility.
* **/qn**: Indicates that the installer should run **silently** with **no User Interface** (UI).
* **/i**: Indicates that MSIEXEC should install a certain product package.
* **Rbs\_x64.msi**: Is the path and file name of the product package to be installed.
* **DBINSTANCE**: The SQL Server\Instance name where the content database is contained.
* **REMOTEBLOBENABLE**: This flag indicates to the installer that another content database will be RBS enabled.
* **FILESTREAMPROVIDERENABLE**: This flag indicates to the installer that the FILESTREAM provider will be enabled on another content database.
* **DBNAME**: The name of the content database.
* **FILESTREAMSTORENAME**: This indicates what the name of the new FILESTREAM BLOB store should be set to during creation.
* **ADDLOCAL**: Represents the RBS features that will be installed or in this case, executed. When installing to an additional content database, EnableRBS and FilestreamRunScript features should be “installed”. These features cause the installer to execute the appropriate scripts to enable RBS and the FILESTREAM provider on the designated database.

### Configuring RBS Security

The SQL RBS Team has posted a comprehensive blog entry on the topic of the [RBS Security Model](http://blogs.msdn.com/b/sqlrbs/archive/2010/08/05/rbs-security-model.aspx). It is not in the scope of this document to extend that information. However, there are specific security concerns that must be addressed when deploying RBS for a client application. Specifically, there are two topics which must be discussed, particularly in reference to RBS deployment for SharePoint 2010.

First, in order to avoid code access security issues, the RBS client library is deployed to the GAC by the RBS Feature Pack installer. Any RBS provider libraries that will be used should also be deployed to the GAC. Not only are these libraries used by SharePoint, but they are also used by the Maintainer console application.

Second, it is important to understand how BLOB store permission requirements can affect SharePoint. Depending on the RBS provider being used, the RBS provider library may be invoked by SharePoint to access content directly in the BLOB store. In this case, the provider library is invoked in the security context of the SharePoint application pool of the web application that hosts the related content database. So this SharePoint application pool must have full access to the BLOB store. This is particularly important for providers that use NTFS file shares or CAS devices for the BLOB store.

## Migrating Content To and From the BLOB Store

Executing a migration of BLOB content to or from the BLOB store is a relatively simple task. The procedure to do so is described in [this TechNet topic](http://technet.microsoft.com/en-us/library/ff628254.aspx). It is important to understand that migration processing can impact the performance of the content database and the BLOB store. It is important to take into consideration the amount of content that will be migrated and whether or not migrating the BLOB data will impact end users. The Migrate() method in the SPRemoteBLOBStorageSettings processes the migration based on certain criteria.

Scenario 1: BLOB Data is in the Client (Content) Database

In this scenario, BLOB data currently exists inline in the content database. An RBS provider is configured and enabled. When Migrate() is called, all BLOB content will be migrated to the BLOB store.

Scenario 2: BLOB Data is in One or More BLOB Stores

In this scenario, BLOB data exists in one or more BLOB stores. The active provider name is set to “” (empty string) such that there is no active provider. When Migrate() is called, all BLOB content will be migrated back inline in the client database.

Scenario 3: BLOB Data Exists in Multiple BLOB Stores

In this scenario, BLOB data exists in one or more BLOB stores and there is more than one BLOB store configured for the client database. The active provider name is set to one of the BLOB stores such that there is an active provider. When Migrate() is called, all BLOB content will be migrated from the original BLOB store to the new active BLOB store.

Scenario 4: BLOB Data Has Been Externalized Using an EBS Provider

In this scenario, BLOB data exists in an EBS BLOB Store. The active provider name is set to one of the BLOB stores such that there is an active provider. When Migrate() is called, all BLOB data for document records in the RBS client database will be migrated from the EBS BLOB store to the active RBS BLOB store.

### Migrate BLOB Data Back Inline in the Content Database

Before uninstalling, it is necessary to migrate all BLOB content back inline into the client database. To perform this procedure, follow the steps listed in [this TechNet topic](http://technet.microsoft.com/en-us/library/ff628254.aspx).

### Disable RBS in the Content Database

Once all of the BLOB data is back inline in the client database, execute the following PowerShell commands below in the SharePoint 2010 Management Shell.

**WARNING!** Disabling RBS destroys RBS collection information in the content database. It should ONLY be used when RBS and/or the FILESTREAM provider need to be UNINSTALLED from the content database and SharePoint. Further, the Disable() method can only be called after all content has been migrated back inline in the content database.

$site = Get-SPSite <http://Root/Sites/SiteName>

$rbss = $site.ContentDatabase.RemoteBLOBStorageSettings

$rbss.Disable()

## Uninstalling RBS (and the FILESTREAM Provider for SharePoint)

There are certain circumstances when it is necessary to remove RBS and/or the FILESTREAM provider from a content database and SharePoint. One example would be when it is necessary to move a client (content) database to another SharePoint farm.

However, if the goal is to ensure that all future BLOBs are stored inline in the content database, then it is not necessary or even recommended to uninstall RBS or the FILESTREAM provider. Follow the steps listed in [this TechNet topic](http://technet.microsoft.com/en-us/library/ff628259.aspx) to disable the active BLOB store.

This section covers the steps necessary for uninstalling RBS and/or the FILESTREAM provider.

### Migrate BLOB Data Back Inline in the Content Database

Before uninstalling, it is necessary to migrate all BLOB content back inline into the content database. To perform this procedure, follow the steps listed in [this TechNet topic](http://technet.microsoft.com/en-us/library/ff628254.aspx).

### Disable RBS in the Content Database

Once all of the BLOB data is back inline in the content database, execute the following PowerShell commands below in the SharePoint 2010 Management Shell.

**WARNING!** Disabling RBS destroys RBS collection information in the content database. It should ONLY be used when RBS and/or the FILESTREAM provider need to be UNINSTALLED from the content database and SharePoint. Further, the Disable() method can only be called after all content has been migrated back inline in the content database.

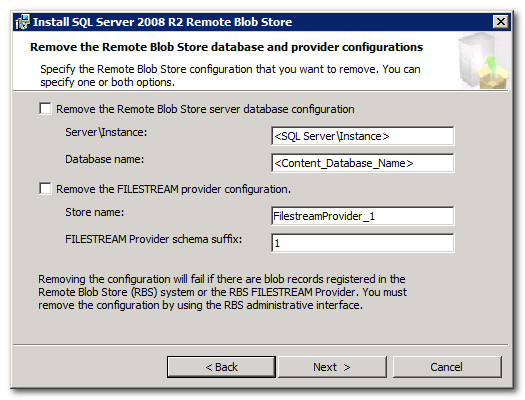
$site = Get-SPSite <http://Root/Sites/SiteName>

$rbss = $site.ContentDatabase.RemoteBLOBStorageSettings

$rbss.Disable()

### Remove BLOB Store Database and Provider Configurations, Uninstall RBS

Launch the RBS\_x64.msi installer. Select options for removing the RBS server database configuration and the FILESTREAM provider configuration. (see Figure 1-20)

*Figure 1-20*  


Executing this uninstaller with both options checked will cause the RBS Feature Pack installer application to execute a couple of stored procedures:

* **mssqlrbs\_filestream.rbs\_fs\_sp\_uninstall**: This stored procedure removes all FILESTREAM BLOB store database resources from the content database.
* **mssqlrbs.rbs\_sp\_uninstall\_rbs**: This stored procedure removes all RBS auxiliary resources from the content database.

**NOTE**: It will not be possible to execute the rbs\_sp\_uninstall\_rbs and rbs\_fs\_sp\_uninstall stored procedures if there are existing BLOBs registered in the BLOB store.

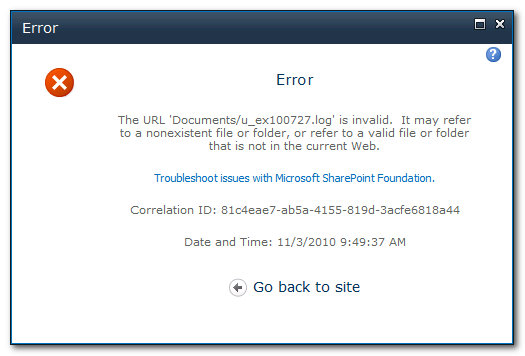
Continue with the uninstallation wizard to completely remove all RBS components from the local server. If the FILESTREAM and RBS database configuration components have already been removed during the uninstallation process of another server, it is not necessary to select the removal options during subsequent WFE uninstallations. (see Figure 1-20)

After all RBS and FILESTREAM components have been removed, the final step is to manually remove the FILESTREAM file group that was manually created during installation. Open a query window in SQL Management Studio and execute the following command to completely remove the FILESTREAM file group:

ALTER DATABASE <Database\_Name> REMOVE FILEGROUP RBSFilestreamProvider

## RBS Operational Issues

There are a couple issues that can present themselves during RBS operation. This list is likely not all-inclusive, but the issue(s) below may be encountered.

* With the FILESTREAM provider implemented, users experience a UI error similar to the one below when uploading a document to site in an RBS enabled content database:  
     
  There are two possible causes to this error. Check the SharePoint ULS logs and the Application log in the Event Viewer. The following RBS exception text may be discovered:

Exception: Microsoft.Data.SqlRemoteBlobs.BlobStoreException: There was a generic database error. For more information, see the included exception. ---> System.Data.SqlClient.SqlException: FILESTREAM feature doesn't have file system access enabled.

This exception suggests that one of two possible issues is at fault:

* + - The ***Enable FILESTREAM for file I/O Streaming Access*** feature may not been properly configured. Follow the ***Enable FILESTREAM for file I/O Streaming Access*** procedure in the Pre-Requisites and Required Resources section above.
    - The ***filestream\_access\_level*** parameter has not been properly configured in SQL Server. Follow the ***Setting the filestream\_access\_level Parameter*** procedure in the Pre-Requisites and Required Resources section above.

# Guidance: Implementing RBS in a Custom Application

Implementing RBS in a client application is a relatively easy proposition due to the fact that the RBS provider framework abstracts all of the code that is necessary to create and delete BLOBs in a particular BLOB store. It really doesn’t matter where the binary data ends up because the client side API calls are the same no matter which RBS provider has been implemented.

## Introduction: The Client Application Explained

An RBS client application is any application that uses the RBS client library for the purposes of storing, retrieving, or maintaining BLOB data in a BLOB store. The client application is abstracted from the actual provider that is used to execute BLOB operations.

SharePoint is a prime example of a client application. The SharePoint 2010 code base has the ability to interact with the client library in order to take advantage of the benefits of RBS.

The Maintainer application is another example application. It uses the client library to facilitate garbage collection operations that ensure that the client database, the RBS auxiliary tables, and the BLOB store all remain consistent.

## Client Library Reference

Support for creating or retrieving existing BLOBs is provided by classes in the **Microsoft.Data.SqlRemoteBlobs** namespace. The complete reference for the classes in this namespace can be found in the [Microsoft.Data.SqlRemoteBlobs Namespace documentation](http://msdn.microsoft.com/en-us/library/microsoft.data.sqlremoteblobs.aspx) on MSDN. This namespace defines the following classes for creating and accessing BLOB data:

|  |  |
| --- | --- |
| **Class** | **Description** |
| BlobStoreCredentials | Stores the credential information that is used to access a BLOB store. For SharePoint, the BLOB store uses the credentials of the application pool for the web application that hosts the content database with RBS enabled. |
| BlobStoreCredentialsList | Stores the list of credentials used to access a BLOB store. |
| BlobStoreException | The base class of exceptions thrown by BLOB store providers. Each BlobStoreException object must have a valid exception code set, which can be used by applications to handle the exception. |
| ConfigItem | Stores a (key, value) pair that is used in RBS to pass configuration and command options. |
| ConfigItemList | Represents a list of configuration items. This is used by applications and RBS client library to pass configuration and command options. |
| RemoteBlobClientException | The exception that is thrown for an error involving RBS client library operation. It usually indicates a runtime error and needs to be handled by the application. |
| RemoteBlobConfigurationException | The exception that is thrown for a configuration or installation issue. |
| RemoteBlobException | The base class for exceptions thrown by the RBS client library. |
| RemoteBlobStoreException | The exception that is associated with errors from the BLOB store provider. |
| RemoteBlobUsageException | The exception that is thrown for errors in the database configuration or application. |
| SqlRemoteBlob | Represents a BLOB created using the RBS feature. This is the primary BLOB object used for streaming BLOB data. |
| SqlRemoteBlobCollectionManager | Represents a logical collection that contains BLOBs created using RBS feature. |
| SqlRemoteBlobContext | Provides database context for the BLOB. A SqlRemoteBlobContext must be established before BLOB data can be submitted or retrieved. |

## Using the RBS Client Library

Implementing the RBS client library is not difficult. The sections below identify the basic code necessary to establish a reference to a SqlRemoteBlob object. This document aims to provide a basic understanding of how the classes in the client library are used to facilitate basic interaction with the BLOB store. A fully documented sample project can be found in [the Microsoft SQL Remote BLOB Storage (RBS) Samples CodePlex Site](http://sqlrbs.codeplex.com/).

### Environment Configuration

Of course, before code can be executed against an RBS provider, the provider and the RBS Feature Pack must be installed on the machine. The RBS client library uses information in the machine.config to instantiate the RBS provider. During installation, the information necessary to load the provider library is added to the machine.config, typically in this location:

C:\Windows\Microsoft.NET\**Framework64**\v2.0.50727\CONFIG\machine.config

However, when developing with Visual Studio 2010, even on a 64-bit operating system, it is possible that in the context of the custom client application, the client library may attempt to read the machine.config from this location:

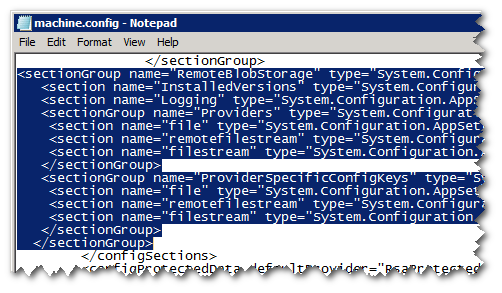
C:\Windows\Microsoft.NET\**Framework**\v2.0.50727\CONFIG\machine.config

Since the RBS provider information was not loaded to this machine.config, it may result in the following error during custom client execution:

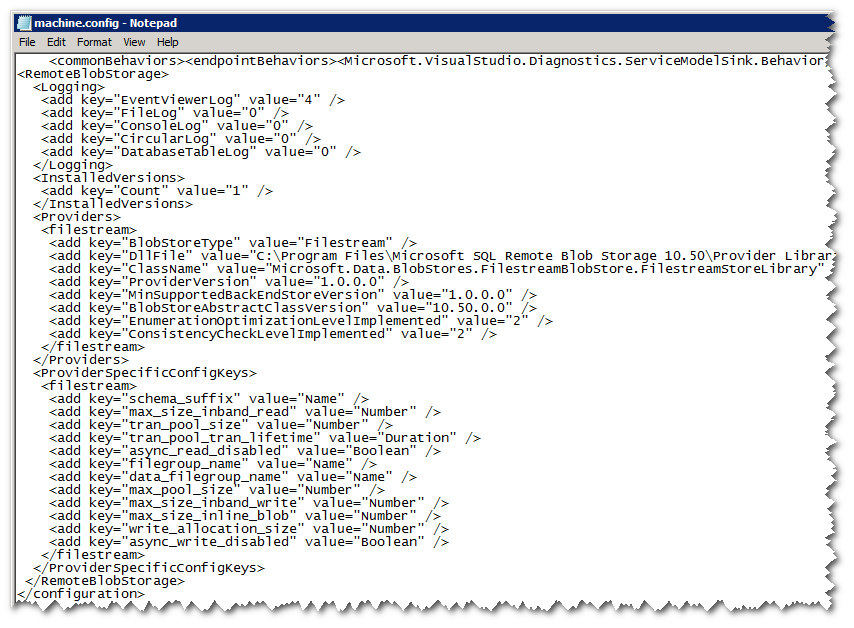
“**No provider of type <Filestream> found. Check the server configuration or install the provider on the client.”**

If this happens, copy the RBS provider information from the “Framework64” machine.config to the “Framework” machine.config. There are two sections that must be copied.

First, copy the RemoteStorage SectionGroup definition. It should be pasted into the destination machine.config file just before the </configsections> tag. *(see Figure 1-21)*

*Figure 1-21*  


Second, copy the entire contents of the <RemoteBlobStorage> configuration section and paste it into the destination machine.config just before the </configuration> tag. (see Figure 1-22)

*Figure 1-22*  


### BLOB Storage and Retrieval

All implementation models begin by establishing a connection to the database that contains the RBS auxiliary tables. The open connection is then used to create a SqlRemoteBlobContext object. Finally, using the SqlRemoteBlobContext object either the CreateNewBlob() method or the OpenBlob() method is typically called to create a SqlRemoteBlob object that can then be written to or read from using either the Simple, Push, or Pull models.

#### Simple Model (Write)

The most basic way to write binary data to a BLOB store is to write a byte array directly to SqlRemoteBlob object and then call the Commit() method to indicate that writing is complete. However, this method may not be the most efficient technique for large binary streams. Here is an example of a simple write operation:

byte[] blobData = Encoding.Unicode.GetBytes("Sample Content");

blob.Write(blobData, 0, blobData.Length);

blob.Commit();

#### Pull Model (Write)

The pull (write) model is the simplest of the more efficient models for storing blob data. In pull mode, the client application needs to have BLOB data in the form of a stream object that is derived from System.IO.Stream. This object must allow RBS to read data from it (CanRead is TRUE). The object must support querying the Length Property. Calling the Commit() method is not necessary because the end of stream marker indicates when the data transfer is complete.

blob.WriteFromStream(inStreamObject);

#### Push Model (Write)

The push model is useful when it is necessary to read from a source stream into a buffer and then write that buffer to the BLOB store. In push mode, it is necessary to call the Commit() method to indicate that writing has been completed:

int bufferLength = 256;

byte[] buffer = new byte[bufferLength];

int bytesRead = dataStream.Read(buffer, 0, bufferLength);

while (bytesRead > 0)

{

blob.Write(buffer, 0, bytesRead);

bytesRead = dataStream.Read(buffer, 0, bufferLength);

}

blob.Commit();

#### After the Write

After writing the stream to the SqlRemoteBlob object, the BlobId needs to be obtained and stored in the client application database. For example:

using (TransactionScope ts = new

TransactionScope(TransactionScopeOption.Required))

{

conn.EnlistTransaction(Transaction.Current);

string addBlobEntry = @"insert into ApplicationSampleTable

(SearchTags, BlobId) values

('Sample, RBS, Demo', @blobId)";

using (SqlCommand cmd = new SqlCommand(addBlobEntry, conn))

{

cmd.Parameters.Add(new SqlParameter("@blobId",

blob.GetBlobId()));

cmd.ExecuteNonQuery();

}

ts.Complete();

}

**WARNING!** Be careful with the “EnlistTransaction()” method highlighted above. Invoking this call can spin up a distributed transaction.

#### Pull Model (Read)

The most basic way to read binary data from a BLOB store is to execute the Read() method in order to read the SqlRemoteBlob data directly into a byte array. This method may not be the most efficient technique for large binary streams. Here is an example of a pull read operation:

byte[] readdata = new byte[blob.Length];

blob.Read(readdata, 0, readdata.Length);

#### Push Model (Read)

The push (read) model is the simplest and more efficient way to retrieve a BLOB stream. The client application prepares a stream object that is derived from System.IO.Stream. The object must allow RBS to write data to it (CanWrite is TRUE). For example:

MemoryStream dataStream = new MemoryStream()

blob.ReadToStream(dataStream);

### Deleting a BLOB

Deleting a BLOB is as simple as deleting the BlobId from the registered RBS column in the application database. Deleting the entire record will also accomplish the same effect. The actual deletion of the BLOB will take place later based on the Garbage Collection Time Window which should be based on the backup and restore Service Level Agreement (SLA) of the organization. This concept is known as ***Delete Propagation***.

### Implementation Best Practices

Most applications that have a need for RBS will benefit from stream based read and write operations. Using stream is typically more efficient, particularly for large operations, as well as being more flexible. Often times a stream will already be provided during upload and during retrieval a stream is often returned to the end user. So typically the Pull model will be best for write operations and the Push model will be best for read operations.

#### BlobReference vs BlobId

When using the OpenBlob() method of the SqlRemoteBlobContext object, the client application has the option of passing in either a BlobId or a BlobReference. A BlobReference contains all of the information necessary for retrieving the BLOB from the BLOB store. However, if a BlobId is passed in, an additional trip to the database is necessary to retrieve a BlobReference before the SqlRemoteBLOB object can be constructed. Whenever possible, the client application should provide a BlobReference as opposed to a BlobId.

The RBS auxiliary resources in the client database provide a function for converting a BlobId to a BlobReference. Instead of this query…

SELECT UserData, UserBlobName, BlobId

FROM BlobUserTable

WHERE (UserBlobName = 'abc.txt')

Use this query…

SELECT UserData, UserBlobName, rbs.fn\_GetRemoteBlobReference(BlobId)

FROM BlobUserTable

WHERE (UserBlobName = 'abc.txt')

# Guidance: Building an RBS Provider

An RBS provider library provides an implementation of abstract classes in the RBS client library in order to enable the storage and management of BLOB data in a specific type of BLOB store storage device. Practical guidance for building an RBS provider can be found in the [Remote BLOB Store Provider Library Implementation Specification](http://msdn.microsoft.com/en-us/library/cc905212(SQL.100).aspx).

## Implementation Guidance

There a couple of points that should be considered when developing an RBS provider.

### Max Buffer Size

Repeatedly allocating and releasing buffers of size more than 85000 bytes is bad for CLR performance due to the way garbage collection works. A server process can literally freeze for a few seconds while CLR GC runs for these large buffers. Consider avoiding the allocation of such large buffers. Rather, allocate many, smaller buffers (perhaps 80KB) and stitch them together to hold larger amounts of data. The RBS development team learned this while developing the FILESTREAM provider and it is recommended that other provider writers do the same.

### OptimizationSpecifiedIds

While this property is mentioned a few times in the Provider Implementation Specification, it was not included in the released version of the client library in the RBS Feature Pack. Any statements including a reference to OptimizationSpecifiedIds should be disregarded.

# Summary

Remote BLOB Storage is a powerful feature of SQL Server that can facilitate the separation of BLOB data from the client application database. By using the RBS client library, client applications can take advantage of RBS without having to worry about the code that is necessary to access a specific BLOB store. Responsible development departments and Independent Software Vendors (ISVs) can develop RBS provider libraries that can provide client applications with the ability to store BLOB data in a specific BLOB store. The Maintainer application, included in the RBS Feature Pack, is provider agnostic and provides standardized garbage collection capabilities. Together, these RBS components combine to provide a powerful feature set that can improve scalability, increase functionality, and reduce long term storage costs for any client application that needs to store a high volume of file content.

Did this paper help you? Please give us your feedback. Tell us on a scale of 1 (poor) to 5 (excellent), how would you rate this paper and why have you given it this rating? For example:

Are you rating it high due to having good examples, excellent screen shots, clear writing, or another reason?

Are you rating it low due to poor examples, fuzzy screen shots, or unclear writing?

This feedback will help us improve the quality of white papers we release.

[Send feedback](mailto:sqlfback@microsoft.com?subject=Feedback%20for%20SQL%20Server%202008%20R2%20Remote%20Blob%20Storage%20White%20Paper).