

SharePoint Server Caches Overview

**Advanced details on the SharePoint BLOB, Output, and Object Caches**

This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2010 Microsoft Corporation. All rights reserved.

Contents

[Introduction 4](#_Toc261351153)

[A deeper look at the caches 5](#_Toc261351154)

[The BLOB cache 5](#_Toc261351155)

[Logical architecture and location 7](#_Toc261351156)

[Cache population and file serving mechanics 7](#_Toc261351157)

[Data storage procedures and cache size on disk 8](#_Toc261351158)

[BLOB cache persistence across application pool recycles 8](#_Toc261351159)

[Cache invalidation mechanics 9](#_Toc261351160)

[Authentication and the BLOB cache 10](#_Toc261351161)

[The output cache 11](#_Toc261351162)

[Time to live vs. check for changes 12](#_Toc261351163)

[Output cache memory requirements 12](#_Toc261351164)

[Vary by parameters and other details about cache profiles 13](#_Toc261351165)

[The object cache 14](#_Toc261351166)

[Caching user-specific data 14](#_Toc261351167)

[The object cache multiplier 16](#_Toc261351168)

[Cache invalidation 16](#_Toc261351169)

[Configuring the caches 17](#_Toc261351170)

[The BLOB cache 17](#_Toc261351171)

[Considerations for the BLOB cache 17](#_Toc261351172)

[Configuring the BLOB cache 18](#_Toc261351173)

[Very large BLOB caches 21](#_Toc261351174)

[Output cache 22](#_Toc261351175)

[Considerations for the output cache 22](#_Toc261351176)

[Configuring the output cache 22](#_Toc261351177)

[Object cache 26](#_Toc261351178)

[Configuring the object cache 26](#_Toc261351179)

[More information 29](#_Toc261351180)

[Appendix 29](#_Toc261351181)

# Introduction

You can use Microsoft® SharePoint® Server 2010 to build many different types of business solutions ranging from collaboration portals and record archives to Internet-facing web sites. However you decide to use SharePoint Server 2010 or whatever you decide to build on top of it, you should understand how the SharePoint Server caches work. Knowing this can help you decide whether and how to use the caches to make your application scalable and fast. It’s important to really understand the benefits and limitations of each cache and how the caches must be configured to ensure you’re getting the most out of this versatile platform.

There are three caches that are described in this paper. Each of these caches has a unique function in helping SharePoint Server scale and grow to meet the demands of your business application. However, the caches are not a panacea; each of the caches has trade-offs, and not all of the caches will be appropriate for your scenario. As with any cache, the SharePoint Server caches require system resources to store and maintain content and indexes. Enabling the caches without correctly configuring them might not result in the expected performance improvements.

A SharePoint Server installation consists of an instance of Microsoft® SQL Server® and at least one front-end web server. When users request data (for example, a page or document) from SharePoint Server, the SharePoint application, running on the front-end web server, retrieves all the relevant data from SQL Server to handle the user request. Although this ensures that all users always see the most up-to-date data, it has the downside of requiring a sub-optimal amount of data traffic between SQL Server and the front-end web servers and therefore potentially sub-optimal performance and responsiveness to user requests, in addition to limited scalability of the hardware to achieve higher throughput levels. The SharePoint Server Caches run on the front-end web servers of your SharePoint Server farm. Each cache maintains copies of data locally on the front-end web server so that user requests can be serviced by using the cached data where appropriate, avoiding excessive traffic between SQL Server and the front-end web servers and CPU load on the front-end web server, thereby improving performance and scalability.





# A deeper look at the caches

The following section provides a deeper look into the function of the caches. Understanding how the caches work can help guide decisions when you are planning a deployment and can also help when you are troubleshooting performance problems.

## The BLOB cache

The BLOB cache reduces load on the farm’s instance of SQL Server by saving the contents of requested files (mainly page assets such as JavaScript, CSS and image files) that are stored in SharePoint Server on the disk of the front-end web server. When another request for a cached file is made, the BLOB cache returns the file from disk instead of having to fetch it from SQL Server again.

When you are developing a web site to be hosted on SharePoint Server, there are a few places where page assets can be stored. They can be stored on the file system of the front-end web server (typically in the “\_layouts directory”), or they can be stored in a SharePoint library or folder. Files that are kept in the \_layouts directory can be served quickly off disk, but if files must be updated, a farm administrator must edit them on each front-end web server. Storing files in a SharePoint library offers the advantage of centralized storage and management, where users other than farm administrators can add, update, and approve content. However, there will be higher latency when these files are served because they must be retrieved from SQL Server. The BLOB cache addresses this aspect of files stored in SharePoint Server by caching the content on the front-end web servers. By storing files in SharePoint Server but caching them by using the BLOB cache, page assets can be served quickly and centrally managed at the same time.

These benefits come with a tradeoff. To add a new file, the BLOB cache pays a first hit penalty of around five times more requests to SQL Server (SQL Server round trips) than with the BLOB cached disabled. These extra round trips are to fetch permission information and other metadata so the cache can serve the file securely. Also, to avoid serving stale content, the BLOB cache will toss out files if there is any chance that the cached copy might be out of date. After a file has been removed from the cache, the first hit penalty applies again to re-cache the file.

* **BLOB caching is most useful in the following scenarios:**
  + **Frequently accessed page resources such as JavaScript, CSS and page images**
  + **Rich media, especially larger files, such as videos**
* **BLOB caching is less useful in the following scenarios:**
  + **Files that are infrequently accessed**
  + **Files that are modified frequently**

In addition to reducing SQL Server round trips, the BLOB cache also helps reduce the time it takes to reload web pages by adding cache control headers to the HTTP responses for the files it serves. These headers tell the user’s browser to save these files in the browser’s cache. When the browser needs one of the cached files, it can use its cache instead of making requests to SharePoint Server. This results in fewer HTTP requests and a significantly reduced page load time.

The BLOB cache is particularly helpful when serving rich media files from SharePoint Server, especially larger files. SharePoint Server is highly optimized to serve small files. It can serve files smaller than *FileReadChunkSize[[1]](#footnote-2)* in a single SQL Server round trip and files up to the *LargeFileChunkSize[[2]](#footnote-3)* are served directly from SQL Server without disk buffering, resulting in low latency. For files sizes larger than *LargeFileChunkSize*, SharePoint Server buffers files on the front-end web server’s disk before serving them. This is to conserve memory, but it also adds some latency to serving these files[[3]](#footnote-4). The BLOB cache can reduce latency in this situation. When a file is BLOB cached, the file can be served nearly as quickly as serving it from IIS directly.

Another advantage that the BLOB cache provides is HTTP range request support. This enables a browser (or other client application) to request pieces of a file instead of the entire file. For example, if a browser only needs the last 1 MB of a 10 MB file, it can make a range request and the cache will serve only the last 1 MB. Without the BLOB cache, SharePoint Server ignores the HTTP range request and serves all 10 megabytes. The BLOB cache will help increase throughput by reducing unnecessary network load.

Media players are client applications that greatly benefit from the BLOB cache’s range request support. Whether it’s a stand-alone media player such as Windows® Media® Player or a Silverlight® media player that is embedded in a web page, media players make range requests for files when a user seeks (jumps ahead) in the player. The BLOB cache returns the requested part of the file, and the user can jump ahead in media even if it hasn’t been fully downloaded on the client.

### Logical architecture and location

The BLOB cache runs on each front-end web server in the farm. More precisely, each cache runs per web application and each virtual server has its own BLOB cache. A virtual server is synonymous with an IIS Web Site. In SharePoint Server, each web application is usually associated with a single virtual server. Only one BLOB cache instance can run on a given virtual server at a time. This means that the BLOB cache can’t be used with web gardens[[4]](#footnote-5). If a web application has been extended to a second virtual server – for example, to extend Intranet content to an Extranet ([How to: Expose a SharePoint Application to the Extranet and Use Forms-Based Authentication](http://msdn.microsoft.com/en-us/library/ee413985.aspx)) – requests handled by the second virtual server will be handled by a second BLOB cache. Having the BLOB cache partitioned at the virtual server allows the cache to be tuned to the authentication model and usage patterns for each zone. For example, requests to the Intranet might not be BLOB cached, but requests to the Extranet will be. It can also be set up so that requests from both zones will be BLOB cached. Although both virtual servers are serving the same content, they each have their own BLOB cache which requires twice as many resources to BLOB cache the content for that single web application.

### Cache population and file serving mechanics

Files with certain extensions get into the BLOB cache when they are requested. The list of extensions to cache is configurable and can be tuned to fit the scenario. When the BLOB cache is serving files for the first time, there can be more latency for small files then there would if the same files were served by SharePoint Server. This is due to the first hit penalty. Large files, on the other hand, are served more quickly because of an optimization that the BLOB cache makes when serving them. The BLOB cache starts serving the file as soon as it reads the first bytes from SQL Server. The file is served back to the client while the rest is still being downloaded from SQL Server. That applies only to the first request because subsequent requests are served from the BLOB cache.

The BLOB cache handles multiple requests for the same file by sharing the cached copy with all the requests. It can even do this before the file has been completely read from SQL Server. For example, a 500 MB video of a keynote presentation is posted on SharePoint Server and then a link to this video is sent out via email. There can be many users accessing the file at the same time. SharePoint Server (without the BLOB cache) would fetch the video from SQL Server once for every user requesting it. That would have huge performance implications for anyone else using the site. With the BLOB cache, even if the video isn’t yet fully cached, the video would be retrieved only once (per front-end web server) from SQL Server to handle all the requests. This optimization makes the BLOB cache indispensible for serving large files from SharePoint Server.

### ****Data storage procedures and cache size on disk****

**The structure with which the BLOB cache stores files on disk is useful for clarifying some of the performance and functional characteristics of the BLOB cache; however, it should be used for illustrative purposes only. You should not directly edit any of the files or folders stored on disk by the cache.**

**The BLOB cache stores its files on disk in a folder structure that mirrors the structure of the SharePoint Server site. For example, a SharePoint Server file with a URL of *http://contoso/sites/publishing/documents/somefile.jpg* will be stored on disk with a path similar to *c:\BlobCache\14\11111111\AB25499AF39572\sites\publishing\documents\somefile-1238DEF8097AB.jpg*. This path contains random strings that prevent the cache from overwriting old versions of files when adding new ones to the cache because the old versions on disk might still be in use. Files in host header sites collections will have an extra unique string added to their paths. This prevents *http://contoso/images/logo.jpg* from colliding with *http://northwinds/images/logo.jpg*.** There’s a 260 character limit for file paths in the Windows® operating system. Because the BLOB cache adds unique strings to the paths of files in the cache, it is possible to have a URL to a file in SharePoint **Server** that is long enough to exceed this limit when the cached file is written to disk. Try to keep URLs reasonably short because files with excessively long URLs cannot be served from the BLOB cache. **Server-relative URLs should be shorter than 160 characters to be BLOB cached.**

In addition to disk space, the BLOB cache requires a small amount of RAM to maintain an index of the files on disk. Each entry in the index uses about 800 bytes of memory. Under most user scenarios, the size of the BLOB cache will be only a small portion of the entire SharePoint application’s memory footprint. However, if hundreds of thousands of files are to be stored in the BLOB cache, RAM consumption can be something to plan for.

**The BLOB cache index uses 800 bytes of RAM per entry.**

### BLOB cache persistence across application pool recycles

The BLOB cache is the only persistent cache, which means that it will survive an IIS application pool recycle or shutdown. It can do this because it writes its index to disk periodically and before a recycle. The serialized index is approximately one third the size of the index in memory. Like all IO operations, the size of the index determines the time it takes to serialize and deserialize. Very large BLOB caches – containing hundreds of thousands of items or more – can take a minute or more to write out their indexes. While this serialization process is going on, new items cannot be added to the cache. This means that requests for files that are not yet in the cache must wait until the cache is finished serializing before those requests are served. If the index is extremely large (for example, a million items), then this time could exceed the client’s request timeout and the request could be dropped. Also, during application pool recycle, if writing the index takes longer than the application pool’s Shutdown Time Limit, then the cache can be forcibly terminated by IIS. In this case the index file can be corrupted and the cache can fail to release the resources that are used to prevent instances of the BLOB cache from running concurrently[[5]](#footnote-6). **If a scenario requires storing more than a few hundred thousand items, be sure to follow the guidance around very large BLOB caches (described in the configuration section of this paper).**

### ****Cache invalidation mechanics****

**The BLOB cache invalidates cached items by polling SharePoint Server for changes. This poll interval defaults to five seconds, but it is configurable. When the BLOB cache detects that one of the items in the cache has been modified, the cache removes the item from its index of cached items. The actual file on disk is deleted later (on a configurable interval) after any HTTP connections that might be reading the file have disconnected. Invalidated files are not re-added to the cache until they have been requested again. Due to the way SharePoint Server records changes to site content, the BLOB cache sometimes has to be more aggressive when it removes items from the cache. The following table lists some common operations (this is not comprehensive) that are performed in SharePoint Server and their effect on cached items.**

|  |  |
| --- | --- |
| ****Operation**** | ****Effect on BLOB Cache**** |
| ****Editing a file that is BLOB cached**** | **The cached file is removed from the cache.** |
| ****Deleting a file that is BLOB cached**** | **The cached file is removed from the cache.** |
| ****Deleting a file that is not BLOB cached**** | **All cached files in the list containing the file that was deleted will be removed from the cache** |
| ****Deleting a folder in a list**** | **All cached files in the list containing the folder that was deleted will be removed from the cache** |
| ****Renaming or Deleting a list**** | **All cached files that were in the list will be removed from the cache** |
| ****Making any changes to a list****   * ****Permissions**** * ****Title, or Description**** * ****Adding or Removing Content Types**** | **All cached files that were in the list will be removed from the cache** |
| ****Renaming or Deleting a web**** | **All cached entries in the web are removed from the cache** |
| ****Adding or removing permissions to a web, changing inheritance, adding, updating, or deleting roles**** | **All cached entries in the web are removed from the cache** |
| ****Deleting a site collection**** | **All cached entries contained in the site collection are removed from the cache** |
| ****Modifying User Policy on the web application**** | **Entire cache is abandoned, a new cache folder is started** |

**The cache also has a configurable size limit to keep the BLOB cache from using too much disk space. When the total size of all files in the BLOB cache exceeds the configured size limit, the BLOB cache removes the least popular files until the cache is below 70% of its configured maximum size. This process is called compaction. Compaction can be expensive because of the first hit penalty to re-cache the removed files. An occasional compaction allows unpopular files to be removed to make room for more popular contact, but what out for a situation where compactions are occurring frequently. If compaction occurs frequently**[[6]](#footnote-7)**, that indicates the cache doesn’t have enough disk space allocated to hold all of the popular content at the same time. Allocating more disk space to the BLOB cache is the best solution to frequent compaction. Ideally, the cache would have enough disk space allocated to it to store all popular content and avoid compaction.**

**Another way cache files can be invalidated is during a cache flush. When the cache is flushed, a new cache folder is created and the old cache folder is abandoned. This is to let any open requests for files in the old cache be completed. The old cache folder and its contents are deleted later (at a configurable interval). The cache can be flushed for several reasons: the cache index fails to deserialize correctly on startup, a web application user policy has changed, the content DB couldn’t be read, or the cache was manually flushed. The cache can be flushed manually by calling Microsoft.SharePoint.Publishing.PublishingCache.FlushBlobCache() from PowerShell or from a .NET application.**

### ****Authentication and the BLOB cache****

**The BLOB cache is highly optimized to serve files anonymously. When an anonymously visible cached file is requested, the BLOB cache can serve the file before the request is authenticated and without making any SQL Server round trips. There are two ways that an application can take advantage of this optimization.**

* **Enable anonymous access to the site.**
* **Store frequently requested files, such as page assets, in a SharePoint Server library that has the *AllowEveryoneViewItems* property set to true.**

**In SharePoint Server, creating a Publishing Portal site collection will create two lists with this *AllowEveryoneViewItems* already set. They are the “Images” and “Site Collection Images” lists. In either case the BLOB cache can serve the files without having to make a SQL Server round trip to check user ACLs. If there is content in the site that doesn’t have to be secured it would be beneficial to enable anonymous access to those files to take advantage of the BLOB cache’s optimizations for anonymous authentication.**

## The output cache

SharePoint **Server** uses the ASP.NET output cache, which is an in-memory cache that stores rendered ASPX pages per ASP.NET application. This cache can help increase page throughput and reduce CPU load on the front-end web server and the instance of SQL Server because pages won’t have to be re-rendered on each request. If a site is configured for anonymous access, the front-end web servers won’t even have to request data from SQL Server to do a permission check for cached requests. A highly simplified flow diagram of how the output cache affects request handling is shown below.



In SharePoint Server only Publishing[[7]](#footnote-8) pages can be cached in the output cache. When a Publishing page is rendered, the Publishing feature assembles the page dynamically. To do this, Publishing must first fetch the page layout and then fetch the contents of the page. Both of these operations incur SQL Server round trips that affect throughput and latency. To increase throughput and scalability, Publishing pages can use the output cache so that pages aren’t rendered on each request.

The cache is configured per site collection.[[8]](#footnote-9) Here it can be enabled and cache profiles – which control cache behavior – can be configured. The cache profile is a collection of settings and parameters that are used to determine if and how a page will be cached. For example, a profile might require that pages are not cached when the requestor is someone that can edit the page so that the user can see the latest version. Another profile might specify that different copies of the pages should be generated depending on what browser is making the request. Along with those settings, the cache profile also specifies how pages in the cache are invalidated. When a page is invalidated, the next request for that page is re-rendered. This new rendered copy replaces the invalidated copy in the cache.

**Publishing pages are output cached when the following two conditions are met:**

* **The cache has to be enabled on the site collection.**
* **The cache profile that applies to the pages must specify they should be cached.**

### Time to live vs. check for changes

There are two options for cache invalidation. If a profile specifies a “time to live” (TTL) then a page is served until this length of time has passed. If “Check for Changes” is specified in a profile, all pages using that profile are invalidated if there is *any* site change or until TTL expires (whichever comes first). This is because it’s not possible to know which site changes could alter the rendering of every page. For example, someone changes the URL to the site logo that is shown on the top of each page. If the cache relied on TTL alone, then pages could appear inconsistent after a site change until each page in the cache has expired.

When you select an invalidation method, weigh the cost of rendering pages against the need for the most current content. Checking for site changes might seem like the best option because it provides the benefits of the cache while showing the latest data, but unless the site rarely changes, pages will be re-rendered after any site change. A site that is serving pages from the output cache should expect a ninefold improvement in page throughput compared to rendering each page. In our testing, the limit to page throughput with the cache enabled is due to available network bandwidth, versus bottlenecks on the front-end web server and the SQL Server CPU when the output cache isn’t used*.* Further, when checking for site changes, each change causes the cache to flush the entries for that profile. There is a big performance penalty to flush and re-fill the cache. Therefore using **TTL based cache invalidation is one way to ensure good performance on sites that experience frequent changes.**

### Output cache memory requirements

The main consideration to make before you use the output cache is the memory requirement on the front-end web servers. ***Each rendered copy of a page can use as much as 2x+32 KB of memory where x is the size of the rendered page in KB.*** For example, a page with a rendered size[[9]](#footnote-10) of 42 KB would require 2\*42+32 or about 116 KB in the output cache. This memory cost must be weighed against the cost to render the page on each request and should be considered when selecting “vary by” criteria which can result in multiple copies of a page being cached.

***Each rendered copy of a page may use as much as  
2x+32 KB of memory where x is the size of the rendered page in KB.***

### Vary by parameters and other details about cache profiles

The vary by custom feature of the output cache allows multiple versions of a page, rendered for different targets, to be output cached. An example might be rendering one copy for users that can see draft items on a page and another copy for users that cannot see draft items on a page. Another common case (one that is configured by default) is to vary by browser. In this case, cached copy is rendered for each type of browser that accesses the site. This is done because the server can render a page differently depending on the browser that makes the request.

The vary by mechanism is easiest to think of it as a format string. In the output cache, the cache profile creates this format string depending cache profile settings. Each setting such as Vary by Custom, Vary by HTTP Header, or Vary by User Rights adds a parameter to the string. The output of this format string is used as the key to retrieve a cached page from the cache. Consider the following format string, generated from a hypothetical cache profile that is set to vary by browser and user rights:

Browser;UserType;URL

When a contributor user visits homepage.aspx in Internet Explorer, the following key is output from the format string:

IE8;Contributor;homepage.aspx

Likewise, when a reader user visits the same page in Firefox, the following key is output:

FF;Reader;homepage.aspx

Notice that the same page (homepage.aspx) now has two entries in the cache and that the number of entries increases as the number of different types of users and unique browsers increases. **The number of vary by parameters will directly affect the amount of memory required by the output cache.**

Finally, SharePoint **Server** can also specify different cache profiles based on the page layout. This can be very useful to give certain types of pages a shorter TTL. For example, suppose there is a news portal that has a headlines page that is updated every 10 minutes and article pages that are updated only once a day. It wouldn’t make sense to have a cached page TTL of one day because the headlines would be wildly out of date. On the other hand, it’s expensive to re-render all of the article pages that change only once per day. One solution would be to have different cache profiles for the different types of cache layouts. A Headline Page page layout could be given a TTL of 10 minutes so that every 10 minutes the old cached page would expire and a new page, with fresh content would be rendered to replace it. The other pages, Article Page pages, could have a TTL of one day because the content of those pages will not change within that time. **Using different cache profiles for difference page layouts allows more control over the output cache.**

## The object cache

The last SharePoint Server cache is the Object Cache. This cache increases page throughput and reduces load on the instance of SQL Server by storing metadata about SharePoint **Server** objects like SPWeb or SPList. When a page is being rendered that requires some metadata about one of these objects, the metadata is retrieved from the cache instead of SQL Server. This is especially beneficial when a page needs data from multiple SPWebs (when rendering Navigation for example) in which case separate queries must be made for each web. **Some features that use the object cache include Publishing, the Content By Query Web Part, Navigation, the Search query box, and Metadata Navigation.** These features were written so that when they need data from SharePoint **Server**, instead of using the SharePoint API directly, they use the object cache API. Because all the interactions between these features and SharePoint **Server** go through the cache, the cache is always “on” and cannot be disabled.

One use of the object cache is storing data that is returned for Content By Query Web Part. Another is to store information about the site navigation so that the navigation controls can render quickly without load on SQL Server. These queries can be expensive , especially if navigation has been configured to include Publishing pages from multiple SPWebs. Developers can also write their own code that uses the cache to take advantage of the performance benefits that it offers. The PortalSiteMapProvider is a public API that can be used to execute a query, and the results that are returned will be cached. For example, an image rotator could be written that would query a list to get the URLs of 10 images. These results could be stored in the cache so that the control doesn’t make SQL Server round trips each time it renders.

### Caching user-specific data

In SharePoint **Server**, querying for items is tied to the user who is making the query. For example, if a user has rights to see draft items in a list and makes a query against that list, the query results will include the latest published version or draft version of the files. If another user who does *not* have rights to see draft items makes that same query, then the results for that user will not include any draft items. Instead, draft items without a published version will be omitted in the results and draft items with a published version will instead have the published version in the results. See the following diagram.



As mentioned previously, various Publishing features make queries for which the results are cached in the object cache. Consider the Navigation feature which shows the sub-sites and pages in a given site on the left of the site UI. If the user who is visiting the site has rights to see draft items, then draft pages are shown in the list of links. Likewise, if a user without rights to see draft items visits the site then Navigation does not show draft items. To correctly cache the results, there must be a way to cache based on the user making the query, but it would be pointless to cache results for every user who makes queries. The cache hit rate would be very low and the memory requirements would be very high. It would be better if the cache could somehow cache results based on the kind[[10]](#footnote-11) of user making the request. This is where the Portal Super Accounts come in.

When a control asks the object cache to make a query to get data, the cache will not make the query as the user making the request but instead will make the query twice: once as a user called the PortalSuperUserAccount (referred to as Super User) and once as PortalSuperReaderAccount (referred to as Super Reader). The results for the Super User query will include draft items and the results for the Super Reader query will include only published items. The cache detects duplicates and single instances the results set. The cache will then check the ACLs for the user that initiated the request and will return the appropriate results to that user. Because of the Portal Super Account users, the cache only has to store results for two users which will increase hit count and decrease the cache’s memory footprint.

Out of the box, the Super User account is the web application’s System Account and the Super User account is NT Authority\Local Service. Farm administrators are advised to change the default accounts for the following reasons. Some items get checked out to System Account (In Place Records is a feature that does this). When a query is made that includes these items, the checked-out version[[11]](#footnote-12) of the items is returned instead of the latest published version. That is usually not what is expected to be returned and the cache has to make a second query to fetch the versions of the file that is not checked out. This extra SQL Server round trip results in a performance hit for every query that includes these checked out items. The same would be true for any user account if that account were used for the Super User account and the user had items checked out. The user configured to be the Super User or Super Reader should not be an account that is used to log into the site or run programs that use the SharePoint API to make changes to a site. This is a precaution that will ensure the Portal accounts aren’t inadvertently used to check items out. Also, don’t use the application pool account or any account that has “Runs as System” checked in the user policy on the web application because these accounts will have the same undesirable behavior as the default account. Finally, the cache needs to have results from users who can and cannot see drafts so the Super User and Super Reader accounts must not be the set to the same account. **The cache will not perform correctly if the Super User and Super Reader accounts are set to the same account.**

### The object cache multiplier

The multiplier is a site-collection-level object cache setting that changes the number of rows that the cache will request from SharePoint **Server**. This is to attempt to fetch enough items to fulfill the original request after security trimming. Suppose that there is a list with 40 items in it. The first 10 items in this list are draft items, the second 10 items are only viewable to readers in group X and the rest are published and visible to everyone. A user makes a request to a page that has Content By Query Web Part which makes a query to the cache for 10 items. If the cache were to request only 10 items, then it would get the first 10 items as the Super User and then it would get the second 10 items as the Super Reader. Suppose that the user that made the request does not have edit permissions and this user isn’t a member of group X. In that case the number of items that the cache would return would be 0 even though there are 20 items that the user could see. By increasing the multiplier to 3, the cache will fetch the first 30 items as Super User and will fetch the last 30 items as the Super Reader. Now the cache will be able to serve files 21 to 30 to the user. Increasing the multiplier can increase the number of items that are returned, but it also requires more RAM to store the results.



### Cache invalidation

Object cache invalidation is configurable and behaves similarly to the output cache. The cache can be configured to check for site changes on every request or to check for updates periodically. Checking for site changes on each request carries a small performance penalty because each request must access SQL Server to determine if there have been changes. In a site that has frequent changes, this reduces the effectiveness of the cache. Using a periodic check results in better overall performance, but stale content can be served between updates. Keep this tradeoff in mind when you configure the object cache.

Unlike the output cache, the object cache is not completely invalidated by any site change. Instead, the cache tracks dependencies and will attempt to keep cached data unless it’s expected to be invalid. For example, if some information about a list were to change, only the cached object for the list and the cached contents of the list would be removed from the cache. However, if a whole site (SPWeb) were to be deleted, then a larger portion of the cache would be removed. The cache keeps a list of dependencies and dependents that each cached object has. This list is enumerated when an object is invalidated and all objects that depend on the invalidated object are also invalidated. For example, the navigation objects in the cache can be invalidated if a site has been deleted because the navigation controls will have to be updated to reflect this change.

# Configuring the caches

It’s important to know how to properly configure the caches and what considerations you should make about a deployment to determine which caches are right for the scenario. The following sections describe the process of enabling and configuring the SharePoint **Server** caches.

## The BLOB cache

### Considerations for the BLOB cache

When you are deciding to use the binary large object (BLOB) cache, consider the following:

1. **How often will the content change?** If the content to be cached has a high read to write ratio, the BLOB cache will provide the most benefit. A good candidate for caching is a site logo, which is unlikely to be modified and is accessed on almost every page request. An inappropriate candidate is a Word document in a collaboration site that is frequently updated.
2. **How large is the content?** The BLOB cache provides different key benefits for files of different sizes. For larger files, the BLOB cache saves on bandwidth between the back end and front end and improves the time to first byte (TTFB) for those files. For smaller files, the cache control headers that are set by the BLOB cache tell the client applications to cache the files locally, which can dramatically reduce the overall number of requests made to SharePoint **Server**.
3. **Is the content anonymously accessible?** The BLOB cache can serve anonymous requests without any SQL Server round trips, while authenticated requests require two round trips to SQL Server for permissions checks. Enabling anonymous access can result in dramatic throughput improvements.
4. **Do the clients that consume the files to be cached use range requests?** The BLOB cache can respond to HTTP range requests correctly. Some applications that make partial content requests are media players (for example, when they are seeking in videos), PDF viewers. Unless the BLOB cache is enabled, SharePoint **Server** ignores partial content requests and responds to these requests with the entire file.
5. **How many items are going to be cached?** The BLOB cache’s memory footprint becomes large, around 400 MB, when it is caching a half million files. Most scenarios, don't ever need to cache this many files, but if a scenario calls for more than a few hundred thousand files to be cached, be sure to read the section on [very large BLOB caches](#_Very_Large_large).

### Configuring the BLOB cache

#### Enabling the BLOB cache

The BLOB cache is enabled per Internet Information Services (IIS) web site (virtual server). If the web application has been extended to more than one zone, there will be as many virtual servers hosting the application. The BLOB cache must be enabled and configured independently for each zone on each front-end web server. At a high level, the following are the steps to enable the BLOB cache.

These steps must be performed on each front-end web server

1. *Edit the web.config file for the IIS web site*
2. *Find the XML node named “BlobCache”*
3. *Find the “enabled” attribute in this node and set its value to “true”*

For more detailed instructions, see [Configure disk-based cache settings](http://office.microsoft.com/en-us/sharepointserver/HA101762841033.aspx).

#### Configuring the application pool

When you enable the BLOB cache, increase the startup and shutdown time limits for the application pool. The recommended setting is at least 300 seconds. This will allow the BLOB cache enough time to initialize or serialize its index on startup or shutdown. Setting this value to 300 seconds does not cause it to take 300 seconds to start or stop, but it does prevent IIS from terminating the application pool until 300 seconds have elapsed. Increasing this limit, especially the shutdown limit, prevents the application from being terminated – possibly when the cache is serializing its index. If the index isn’t completely serialized it is considered corrupted and the cache is flushed on the next startup.

The BLOB cache does not support multiple processes per application pool, known as a web garden. When a web garden is configured, only the first process is correctly cached in a BLOB cache; the other processes are not cached.

For more information, see [Process Model Settings for an Application Pool](http://www.iis.net/ConfigReference/system.applicationHost/applicationPools/add/processModel).

#### Tuning the BLOB cache

The first step is to plan ahead. On the front-end web server, pick a disk that has enough free space to hold all (or at least the most popular) content in the site. This disk should have sufficient IOPs to handle the expected throughput while avoiding contention with other processes using the disk like search indexes or logging. Also, account for the memory that the cache index uses. Every cached file uses about 800 bytes of RAM in the cache index. This can be a significant factor in planning for hardware if the expected number of items is large.

The BLOB cache works for most scenarios out of the box without any tuning, but there are a few settings, such as where the cache should store its files, that you should consider for every deployment. The following tables list all the XML attributes that can be configured in the BlobCache node in web.config. The attributes have been grouped into two buckets: attributes that should be modified to suit scenario requirements and attributes that should not be modified. The second set is included for completeness. Not all of these attributes appear in the web.config BlobCache node by default.

**Attributes that can be tuned**

|  |  |  |
| --- | --- | --- |
| Attribute | Default | Notes |
| Enabled | False | Boolean value. Set to True to enable the BLOB cache. |
| Location | C:\BlobCache\14 | Path to the folder where the BLOB cache files are stored on disk. This Path cannot be on a network drive and cannot be a root of a drive. We recommend that this location be on a separate disk from the operation system and application files to avoid disk IO contention. |
| Path | See below | A regular expression that defines what URLs are handled by the BLOB cache. If an incoming file URL matches this regular expression, then the URL is served from the cache. The default path causes common page assets and media files to be cached. |
| maxSize | 10 | The maximum size in gigabytes that the BLOB cache is allowed to use to store cached files. This can’t be smaller than 1 GB. We recommend that you allocate enough disk space to the BLOB cache to store all popular content. If it’s possible to allocate enough disk space to store all content that can be cached, that’s preferred. |
| max-age | 86400 | The length of time in seconds that files served from the BLOB cache should be stored by a browser cache. The default is one day. Browsers do not make requests for locally (browser) cached files, which reduces the number of HTTP requests that the server must handle. |
| browserFileHandling | Nosniff | Write out an HTTP header to tell browsers to accept files without checking to ensure the contents match the MIME type. Possible values: <*nosniff*|*strict*>. For more information, see this blog: [IE8 Security Part VI](http://blogs.msdn.com/ie/archive/2008/09/02/ie8-security-part-vi-beta-2-update.aspx) |

Default Path Attribute Value: \.(gif|jpg|jpeg|jpe|jfif|bmp|dib|tif|tiff|ico|png|wdp|hdp|css|js|asf|avi|flv|m4v|mov|mp3|mp4|mpeg|mpg|rm|rmvb|wma|wmv)$

**Attributes that should not be modified (except where noted below for very large BLOB caches)**

|  |  |  |
| --- | --- | --- |
| Attribute | Default | Notes |
| changeCheckInterval | 5 | Length of time in seconds the BLOB cache waits before checking for site changes that invalidate cached files. |
| policyCheckInterval | 60 | Deprecated and has no effect on the BLOB cache. |
| writeIndexInterval | 60 | Length of time in seconds that the BLOB cache waits before serializing its index to disk. |
| deleteOldCacheFolders | 600 | Length of time in seconds that the BLOB cache waits before cleaning old cache folders from disk. Old cache folders are left on disk after the cache is flushed. |
| deleteCheckedInFiles | 600 | Length of time in seconds that the BLOB cache waits before cleaning up invalidated cache files from disk. |
| largeFileSize | 1048576 | Size in bytes that specifies how large a large file is. Large files are handled differently by the BLOB cache. They are not fetched from SQL Server on the request thread. Instead a second thread is created to download the file. This allows incomplete large files to be shared by multiple requesting threads. |
| readChunkSize | 65536 | Size in bytes of the buffer that is used to fetch large files. This affects the number of SQL Server round trips to fetch a large file and also affects latency. |
| throttleLimit | 10 | The maximum number of threads that the BLOB cache is allowed to create when fetching large files. If this number is too large, SQL Server load can be affected. |

#### Maintaining the BLOB cache

To ensure that the cache stays efficiently populated there are a few best practices:

1. Keep all content to be cached in a specific list (or specific small number of lists) and keep the site containing those list(s) stable. Making changes to the site collection, site, or list that contains cached content can cause the cache to invalidate all of its entries. There are a few other relatively common operations that can cause the cache to be flushed. These operations should be avoided:
   1. Modifying anything about any container of the cached content could invalidate all of the cached files within that container. For example, if a list’s description is changed, the cached files in that list will be invalidated.
   2. Making security changes on containers. Adding a user or group, changing ACLs, or breaking or restoring ACL inheritance will all cause the container to be flushed.
   3. Deleting a file that is not in the cache from a list that contains cached items will cause every item in that list to be flushed.
2. To keep the cache healthy, let it maintain its own files.
   1. Never manually delete files from the cache folder on disk.
   2. Don’t open or modify the files in the cache folder.
   3. Add the cache folder to antivirus software exclude lists. Antivirus software can delete file from the cache or, more likely, lock files during scans.
3. **Windows has a maximum allowable length of 260 characters for file paths. Because of this limit, SharePoint Server-relative URLs should not exceed 160 characters** (or fewer, depending on the length of the path to the folder where BLOB cache files will be stored on disk) **or the path for the cached file might exceed this limit.**

#### Flushing the BLOB cache

Occasionally, the BLOB cache can get into a bad state. For instance a file on the disk might have been deleted and now won’t be cached. If the cache seems to be in a bad state there are a few options that you can try to get the cache back into a good state.

1. Run IISReset on the front-end web server that seems to be having a problem. This causes the cache to serialize its state and then reinitialize.
2. Flush the cache by using the object model. Using PowerShell or a program written with the SharePoint API, call **Microsoft.SharePoint.Publishing.PublishingCache.FlushBlobCache(). Using this method causes all the front-end web servers to flush their own BLOB caches for the web application passed to the method.**
3. Disable the cache, delete the cache folder, and re-enable the cache.

### Very large BLOB caches

This section talks about specific recommendations for deployments hundreds of thousands of items in the BLOB cache. When you plan for these extremely large BLOB caches, you should be careful to avoid operations that would result in the cache being flushed because the cost of repopulation is high. Also, because the size of the BLOB cache index also becomes fairly large, you should increase the interval at which the BLOB cache index is written out to cut down on IO. Consider the following recommendations when you plan very large BLOB caches.

* Increase the WriteIndexInterval in the BlobCache node in web.config so that the cache index isn’t serialized as often. By default, this is set to 1 minute. We recommend that you increase this value to one hour.
* Increase the startup and shutdown time limits on the web application to accommodate the extra time it takes to initialize or serialize the cache index. In the standard guidance, we recommend that these time limits be increased to 300 seconds. For very large BLOB caches, set time limits to 600 seconds. A web front end that is under a lot of load can take a long time to write out a very large cache index (sometimes as large as 300 MB) and giving SharePoint **Server** and the BLOB cache enough time to shut down is critical so the BLOB cache isn’t stopped while its index is only partially written. A partially written cache index is considered corrupted and the cache is flushed.
* Take extra care to avoid flushing or compacting the BLOB cache. If having the cache populated is critical to the deployment’s scale and performance goals, it’s best to lock down the sites where the content will be stored to ensure that the cache isn’t inadvertently flushed.
  + Create libraries that contain only cached content. Lock down these libraries so that only a few users can add or remove files from this list. Make sure that these users understand the consequences to the cache of deleting uncached files or folders from this list.
  + If possible, have enough disk space to hold all of your content that can be cached on each front-end web server.

## Output cache

### Considerations for the output cache

When you are deciding to use the output cache, consider the following:

1. **What kind of content will be cached?** The output cache only works with Publishing pages – pages that are based on page layouts. If the scenario doesn’t use the SharePoint Server Publishing features, for example, a collaboration site or a document store, the output cache will not provide any benefit.
2. **What kind of read to write ratio is expected for the site?** As a general rule, a site with a low read to write ratio is not a good candidate for the output cache. The frequent changes will make it hard to populate the cache with fresh data.
3. **How important is having the most current content on the site?** The benefit that the cache provides is proportional to the number of times a cached page is served. Ideally, a page could be rendered once per application recycle. If the scenario requires that pages be re-rendered frequently, site throughput will degrade accordingly.
4. **Will the site contain dynamic content, especially per-user content?** To properly support personalized cached content – for example, changing the contents of a Content By Query Web Part based on a user’s team – requires a variant cached page to be rendered and cached. As the number of variants increases, memory use increases and the cache hit ratio decreases.
5. **Is memory utilization a concern in the deployment?** The memory requirement of the cache is directly related to the size of the content and the number of variants of the page. For example, if the cache profile is configured to create a variant page for each of group in a deployment, the increase in memory use is equal to the number of groups multiplied by the number of pages that are cached.

### Configuring the output cache

For information about describing the output cache, see [Output Caching and Cache Profiles](http://msdn.microsoft.com/en-us/library/aa661294.aspx). The output cache has not changed much since Microsoft Office SharePoint Server 2007 with the exception of cache profile overrides which are described later in this paper. For a discussion of the output cache, based on IIS, see [Configure IIS 7 Output Caching](http://learn.iis.net/page.aspx/710/configure-iis-7-output-caching/).

The following section provides high level details about output cache configuration.

#### Enabling the cache

To enable the output cache, the Publishing Infrastructure site collection feature and the Publishing site feature must be enabled. After the features are enabled, the high level steps to enable the cache are as follows:

1. As a site collection administrator, browse to the site collection settings page.
2. Click the “Site collection output cache” link.
3. Select the **Enable output cache** check box
4. Optionally: To help verify that the cache is enabled, selecting the last check box enables debug information that describes when a page was cached or if it wasn’t cached, why not.
5. Click **OK.**

#### Configuring the cache profiles

Cache profiles specify the different variants of a page that can be stored in an output cache. For example a cache profile can specify that a page have different rendered copies depending on the browser that is used to access the page. The following list briefly describes the different fields that can be configured on a cache profile and their effect on what the output cache stores.

|  |  |
| --- | --- |
| Field Name | Notes |
| Perform ACL Check | Check to verify that a user has rights to view a page before serving the page. This verification adds latency to the request but ensures that authenticated content isn’t visible to users who wouldn’t have access otherwise. In an anonymous only scenario, this setting should not be used. |
| Enabled | Enables output caching for pages that use this profile. |
| Duration | The length of time, in seconds, that a cached copy of the page is served from the cache. A longer time means that the cached copy’s hit ratio is higher at the expense of content possibly being stale for longer. Another consideration is that a longer time means that cache entries with low hit ratios are taking memory for longer. |
| Check for Changes | When this attribute is set, the cache checks for site updates on each request. If there have been changes, then a new cached copy of the requested page is rendered. Setting this attribute can reduce performance on a site that is frequently modified, but it ensures that the latest content is always available. Regardless of this setting, cached pages still expire when the duration has been exceeded. |
| Vary by Customer Parameter | This attribute allows arbitrary parameters to be added to the cache profile. These parameters have no effect on caching without adding a custom vary by handler attribute that replaces the parameter with actual data which then creates a new entry in the output cache. For example, a custom handler could be built to replace the parameter ***GeoIP*** with a country name. Then the output cache would store a different cached copy for each country.  ***Browser*** is a built in custom vary by parameter. Adding this to the custom vary parameters causes the output cache to store a copy of the page based on the user-agent that the browser sends.  For more information about vary by custom parameters, see: [HttpCachePolicy.SetVaryByCustom Method](http://msdn.microsoft.com/en-us/library/system.web.httpcachepolicy.setvarybycustom.aspx)  Also, for more information about developing a custom varyByCustom handler, see [Extend Caching by Using the VaryByCustom Event Handler](http://msdn.microsoft.com/en-us/library/ms550239.aspx) |
| Vary by HTTP Header | This attribute is a list of HTTP header names. The output cache creates a new cached copy for each value that is sent in the header. An example is ***Accept-Language*** which causes the cache to cache entries for each combination of languages that browsers send when they make page requests.  For more information about vary by HTTP header, see [HttpCachePolicy.VaryByHeaders Property](http://msdn.microsoft.com/en-us/library/system.web.httpcachepolicy.varybyheaders.aspx) |
| Vary by Query String Parameters | This attribute is a list of query parameter names that cause the cache to create cached copies based the values of the query parameters. See the detailed explanation following this table for examples of and considerations for using vary by query.  For more information about vary by query string parameter, see [HttpCachePolicy.VaryByParams Property](http://msdn.microsoft.com/en-us/library/system.web.httpcachepolicy.varybyparams.aspx) |
| Vary by User Rights | Create a different cached copy for each combination of user rights. For example, different copies are made for users that have and do not have edit rights on the site. |
| Cacheability | This attribute modifies the HTTP cache control header that is written on the page, which changes how downstream proxy and client caches store the file. For more information, see: [Response.CacheControl](http://msdn.microsoft.com/en-us/library/ms524721(VS.90).aspx) |
| Safe for Authenticated Use | This attribute has no effect on the contents of the cache. It is a flag that, when checked, adds this cache profile to the list of authenticated profiles that are shown when cache profiles are selected. |
| Allow writers to view cached content | When this attribute is set, users who have permission to edit pages are served cached pages. There are a few caveats here. Because the rendered copy of the page can include items or content that is not checked in or approved, enabling this could cause draft content to get cached. This draft content would then be served until the cached copy expires. When using this setting, configure the cache to perform an ACL check to prevent viewers from seeing draft content. |

Carefully consider the settings that are made in a cache profile – especially on a high volume site – because they directly affect the size of the output cache (see the first section for more information about why). When you determine how much memory your output cache will require use the following formula:

***Cached Copy size in KB = 2x+32 KB where x is the size of the rendered page in KB.***

After determining a rough estimate, try to determine how many cached copies of each page will be created as a result of the cache profile settings. For example, when varying by browser and two browsers are expected to be used to access the site, there will be twice the number of cache entries in the output cache.

All settings have the potential to affect the memory use of the output cache by increasing the number of cached copied that are in the cache, but **one setting in particular, Vary by Query String Parameter, can cause low hit rates and increase memory use if not selected carefully**. A good query string parameter to vary on will have a limited number of values. Consider a page with a Content By Query Web Part on it that checks a query string parameter called vehicle – that could have the following values: car, truck, van – and renders something special depending on that value. Under typical circumstances, that parameter result in only three cached copies of that page and each page will probably have a high hit ratio. A bad query string parameter to vary on is unbounded. Think about a page with a Content By Query Web Part that uses a date range from the query string to present a list of items to a user. Date ranges can have an infinite number of permutations and so could also create an infinite number of cached copies, and each cached copy will have a very low hit ratio because it’s unlikely that two user will select the same date range.

#### Different profiles for different page layouts

Instead of having profiles that affect the entire site collection, cache profiles can also be defined for page layouts. This is useful when certain content is updated less frequently than other content. For example, a site home page might have dynamic content that lists the latest articles on the site. The actual content articles do not change. If these two types of pages are based on two different page layouts, then the home page could be given a duration of five minutes and the content articles could be given a duration of two hours. In this case the home page content would be only five minutes out of date, but at the same time, the front-end web server is not wasting resources to re-render the static content pages every five minutes.

#### Cache profile overrides

The cache profile overrides are required when the output cache with host header site collections is used. The overrides are specified in the web.config for the IIS virtual server that is hosting the web application and apply globally to all site collections in that web application. When the overrides are used, certain parts of the cache profiles that are defined at the site collection level are overwritten with these settings. The cache profile fields that are available for overrides follow; their functions are the same as those described previously in the table:

* varyByHeader: a semicolon-separated list of HTTP headers to vary by
* varyByParam: a semicolon-separated list of URL query parameter names
* varyByCustom: a semicolon-separated list of Custom vary by parameters
* varyByRights: Boolean set to true to vary by user rights
* cacheForEditRights: Boolean set to true to allow editors to see output cached content

To enable and configure the cache overrides, modify the web.config file and search for an XML node called *OutputCacheProfiles*. Set the *useCacheProfileOverrides* attribute to true and save the changes.

When you set the overrides, carefully weigh the caching needs of all the site collections in the application against the memory use of the cache. For example, some site owners might want to use a Content By Query Web Part that displays different content depending on the value of a query parameter. If varyByParam is not on, then those users will get a cached copy of a page that is rendered with the first query parameter that is specified. Subsequent page requests receive the same cached copy regardless of the query parameter that is specified.

## Object cache

The object cache cannot be disabled, and there is very little that can be configured. None of the out of box settings should have to be changed, but there are two user accounts that should be set up to ensure the best performance. The following article was written for Office Server 2007 but is still relevant and provides some additional information on this cache: [Object Caching](http://msdn.microsoft.com/en-us/library/aa622758.aspx).

### Configuring the object cache

The first thing that you should always do when you are using the object cache is to set up the Portal Super Accounts. The rest of the information in this section relates to performance and capacity considerations of the object cache.

#### Configuring the Portal Super Accounts

Below are the steps to configure the portal accounts manually. A script that performs these steps automatically can be found in the [appendix](#_Appendix).

1. Browse to the application management page and Manage Web Applications.
2. Select the web application to edit, and click the **User Policy** button on the ribbon
3. Add a user for All Zones.
4. In the add user form, add the first username and make sure to select the user name with the check button on the user control. (This ensures that the username can be resolved by the authentication providers on the application.)
5. Assign Full Control to this user.
6. Click **OK**.
7. Perform steps 3-5 again using a different user account and assign this user Full Read.

* Do not select Runs as System.
* Do not use these accounts to log in to the site.
* Do not use the same account that the application pool is running as.
* The two accounts must be set to different accounts.

Next, these two accounts must be added to the web application properties collection. The easiest way is to use PowerShell.

On the user policy form for the web application, next to the user names that were added, there will be the encoded username that corresponds to the authentication providers that have been configured for the web application. For example: *contoso\user1* for Windows Classic authentication and *i:0#.w|contoso\user1* for Windows Claims authentication. Copy the encoded names and perform the following steps.

Open the SharePoint Management Console (PowerShell with SharePoint SnapIn) and run the following:

1. $wa = Get-SPWebApplication –Identity “SharePoint – 80”
   * This will store a reference to the web application is variable $wa
   * The identity that is passed in here should be the name of the web application that is being configured.
2. $wa.Properties[“portalsuperuseraccount”] = “i:0#.w`|contoso\user1”
   * The string that is being assigned should be the encoded username for the account that was given Full Control in the previous steps. The pipe character has been escaped with a back tick.
3. $wa.Properties[“portalsuperreaderaccount”] = “i:0#.w`|contoso\user2”
   * This should be the user that was given Full Read.
4. $wa.Update()

Run IISReset on each front-end web server in the farm to complete this configuration.

#### Planning for the object cache

For most deployments with a small number of site collections (less than 50), the object cache should have a negligible memory footprint. When you plan to have more site collections it might be necessary to consider the memory cost of the cache. When calculating the size of the object cache on a front-end web server, assume that each site collection requires 500 KB of memory.

**Allocate 500 KB of RAM per site collection for object caching.**

#### Tuning the object cache

As stated previously, the object cache will probably not require tuning for most scenarios. The following is a list of the different tuning options and their effect on the system. These options are on the object cache settings page that can be accessed under the site collection administration group on the site settings page.

* Max Size: when the object cache becomes larger than this value, least popular entries (based on hit count) will be trimmed from the cache. This prevents the cache from growing unbounded. This value is for the site collection and all of the sites in it. For more information about this setting, see the maxSize override later in this paper.
* Flushing: The object cache can be flushed by selecting the flush cache check box and clicking **OK**. To flush the caches on all the front-end web servers, then select both check boxes and click **OK**. We do not recommend flushing the whole farm on high throughput sites because it will cause a performance to degrade while the cache is repopulating.
* Cache Invalidation: The cache can either check for changes on each request, or it can check for changes periodically. As with the other caches, checking for changes results in lower performance because more SQL Server round trips are made for each page load. Using periodic invalidation should produce a higher cache hit rate, so the cache will provide more performance benefits.
* The multiplier function is used to help ensure that enough results are available after security trimming to fulfill the original query. For example a Content By Query Web Part might issue a query to the cache for 10 items from list A. The cache takes that query and makes a query for 30 items from list A to SharePoint **Server**. This way, after security trimming, which in this example removes 15 items, the Content By Query Web Part will still have 10 items returned.
  + The multiplier increases the number of results that are requested and stored in the cache.
  + A higher multiplier can be used if controls that make queries through the object cache are not getting as many rows as expected.
  + A higher multiplier causes a query to return more data and is more expensive to execute. Because there is more data, more RAM is used to store the results.
  + The default multiplier, 3, should be sufficient for most scenarios, but this value should be modified to meet the requirements of the site. Memory use shouldn’t be a concern unless you are using very large multipliers (greater than 50).
  + If a site requires a very large multiplier, it might be a permission or configuration problem that should be addressed.

#### Writing custom query controls

If a scenario requires a custom control that aggregates or queries site content, that control can use the object cache API to attain higher performance while lowering its effect on the system. To utilize the cache in a custom control, use the PortalSiteMapProvider class that is specified in the Microsoft.SharePoint.Publishing namespace. For more information about this class and how to use it, see the following articles.

* [PortalSiteMapProvider Class (Microsoft.SharePoint.Publishing.Navigation)](http://msdn.microsoft.com/en-us/library/ms582344(v=office.12).aspx)
* [Increased performance for MOSS apps using the PortalSiteMapProvider](http://blogs.msdn.com/ecm/archive/2007/05/23/increased-performance-for-moss-apps-using-the-portalsitemapprovider.aspx)

#### Object cache maxSize override

In the web config file for the IIS virtual server that is hosting SharePoint **Server**, there is a configuration setting that was added in SharePoint Server 2010 to set an application level limit on the output cache size. By default, this is set to 100 MB per web application.

The sum of all the object caches (which are maintained per site collection) is limited to the value specified here.

The following are some additional notes about this setting:

* The application-wide limit can only be set to an integer between 1 and 2048.
* The web.config setting supersedes the value that is shown in the UI. This means that if a site administrator specifies the object cache size to be 200 MB, the cache is still trimmed at 100 MB (or whatever the web.config setting is). However, the site UI can specify a lower value, say 50 MB, than the web.config setting. In that case, the cache for that site collection is trimmed at 50 MB. Site collection administrators are not informed about the web.config limit.
* If the value is not set in web.config, the application wide limit is set to 25 MB.
* A general rule for setting the application-wide limit is to leave it unchanged unless there are more than 200 site collections in the application. In that case, multiply the number of expected site collections by .5 MB and set the application limit to that. For example with 400 sites, the application should be set to 200 MB.

# More information

For more information about the SharePoint Server caches see the following:

* [Caching in Office SharePoint Server 2007](http://technet.microsoft.com/en-us/library/cc298466.aspx)
* [Custom Caching Overview](http://msdn.microsoft.com/en-us/library/aa589700(v=office.12).aspx)
* [How to Optimize a SharePoint Server 2007 Web Content Management Site for Performance](http://msdn.microsoft.com/en-us/library/bb727371.aspx)
* [Output Caching and Cache Profiles](http://msdn.microsoft.com/en-us/library/aa661294.aspx)
* [Disk-based Caching for Binary Large Objects](http://msdn.microsoft.com/en-us/library/aa604896.aspx)
* [Object Caching](http://msdn.microsoft.com/en-us/library/aa622758.aspx)
* [Performance and capacity test results and recommendations (SharePoint Server 2010)](http://technet.microsoft.com/en-us/library/ff608068(office.14).aspx)

# Appendix

The following is a sample Windows PowerShell™ script that can be used to configure the Portal Super User/Reader Accounts.

**ConfigurePortalAccountsInteractive.ps1**

function Prompt($s)

{

Write-Host

Write-Host $s

$r = Read-host

return $r

}

function ConfigureAccounts($application, $readerAccountString, $userAccountString)

{

$superReaderPropertyString = "portalsuperreaderaccount"

$superUserPropertyString = "portalsuperuseraccount"

$fullReadRoleName = "Full Read"

$fullControlRoleName = "Full Control"

Write-Host

# Super Reader

Write-Host "Adding policy user for Super Reader $readerAccountString"

$superReaderPolicy = $application.Policies.Add($readerAccountString, "Object Cache Super Reader")

Write-Host "Getting Policy Role: $fullReadRoleName..."

$role = $application.PolicyRoles | where {$\_.Name -like $fullReadRoleName}

Write-Host ("Assigning Policy Role: " + $role.Name + "...")

$superReaderPolicy.PolicyRoleBindings.Add($role)

Write-Host "Setting $superReaderPropertyString property..."

$application.Properties[$superReaderPropertyString] = [System.String]$readerAccountString

Write-Host

# Super User

Write-Host "Adding policy user for Super User $userAccountString"

$superUserPolicy = $application.Policies.Add($userAccountString, "Object Cache Super User")

Write-Host "Getting Policy Role: $fullControlRoleName..."

$role = $application.PolicyRoles | where {$\_.Name -like $fullControlRoleName}

Write-Host ("Assigning Policy Role: " + $role.Name + "...")

$superUserPolicy.PolicyRoleBindings.Add($role)

Write-Host "Setting $superUserPropertyString property..."

$application.Properties[$superUserPropertyString] = [System.String]$userAccountString

Write-Host

Write-Host "Updating Web Application..."

$application.Update() | Out-Null

}

$windowsAuthName = "Windows Authentication"

$formsAuthName = "Forms Authentication"

$genevaAuthName = "Geneva"

if(((Get-PSSnapin | foreach {$\_.Name}) -contains "Microsoft.SharePoint.PowerShell") -eq $false)

{

Add-PSSnapIn Microsoft.SharePoint.PowerShell

}

$webAppCollection = @(Get-SPWebApplication)

$i = 1

foreach($webApp in $webAppCollection)

{

Write-Host "[$i]:" $webApp.Name

$i = $i + 1

}

$appIndex = [System.Int32](Prompt "Enter the number of the application to configure")

if($appIndex -gt $webAppCollection.Length -or $appIndex -lt 1)

{

Write-Host "Invalid Index"

Exit

}

$app = $webAppCollection[$appIndex - 1]

Write-Host "Got web application:" $app.Name

$zone = $app.IISSettings.Item("Default")

$authProviderNames = $zone.ClaimsAuthenticationProviders | foreach {$\_.DisplayName}

$isMultiAuth = ($zone.ClaimsAuthenticationProviders.Count -gt 1)

$superReaderString = ""

$superUserString = ""

Write-Host

Write-Host "Information about the default zone:"

Write-Host "Using Claims:" $zone.UseClaimsAuthentication

Write-Host "Is Multi Auth:" $isMultiAuth

write-Host "Providers Enabled:" $authProviderNames

Write-Host

Write-Host "------------------------------------------------------"

Write-Host -ForegroundColor Red "WHEN SETTING ACCOUNTS REMEMBER THE FOLLOWING:"

Write-Host " 1. DO NOT USE THE APPLICATION POOL ACCOUNT"

Write-Host " 2. DO NOT USE THE SAME USER FOR BOTH SUPER READER AND SUPER USER"

Write-Host " 3. DO NOT USE AN ACCOUNT THAT WILL BE USED TO LOG IN TO A SITE"

Write-Host " 4. DO ADD NEW USERS IF YOU DON'T HAVE 2 USERS THAT MEET THE ABOVE CRITERIA"

Write-Host "------------------------------------------------------"

Write-Host

if($zone.UseClaimsAuthentication -eq $false)

{

Write-Host "Windows Auth is enabled, creating portal accounts with Windows users"

Write-Host "Windows usernames should have a domain e.g. redmond\testuser"

$superReaderString = Prompt "Enter a Windows user name to use for the super reader account"

$superUserString = Prompt "Enter a Windows user name to use for the super user account"

}

else

{

if($authProviderNames -contains $windowsAuthName)

{

Write-Host "Windows Auth is enabled, creating portal accounts with Windows users"

Write-Host "Windows usernames should have a domain e.g. redmond\testuser"

$superReaderString = Prompt "Enter a Windows user name to use for the super reader account"

$superUserString = Prompt "Enter a Windows user name to use for the super user account"

$superReaderPrincipal = New-SPClaimsPrincipal -Identity $superReaderString -IdentityType WindowsSamAccountName

$superReaderString = $superReaderPrincipal.ToEncodedString()

$superUserPrincipal = New-SPClaimsPrincipal -Identity $superUserString -IdentityType WindowsSamAccountName

$superUserString = $superUserPrincipal.ToEncodedString()

}

elseif($authProviderNames -contains $formsAuthName)

{

$formsProvider = $zone.FormsClaimsAuthenticationProvider

$formsPrefix = "Forms"

if($formsProvider -eq $null)

{

Write-Host "Error getting claims forms provider, defaulting to prefix `'Forms`'"

}

else

{

$formsPrefix = $formsProvider.MembershipProvider

}

Write-Host "Forms Auth is enabled, creating portal accounts with Forms users"

Write-Host "Forms usernames will be prefixed with this provider:" $formsPrefix

$superReaderString = Prompt "Enter a forms user name to use for the super reader account"

$superUserString = Prompt "Enter a forms user name to use for the super user account"

$superReaderPrincipal = New-SPClaimsPrincipal -Identity ($formsPrefix + ":" + $superReaderString) -IdentityType FormsUser

$superReaderString = $superReaderPrincipal.ToEncodedString()

$superUserPrincipal = New-SPClaimsPrincipal -Identity ($formsPrefix + ":" + $superUserString) -IdentityType FormsUser

$superUserString = $superUserPrincipal.ToEncodedString()

}

elseif($authProviderNames -contains $genevaAuthName)

{

$tokenIssuer = Get-SPTrustedIdentityTokenIssuer

Write-Host "Geneva Auth is enabled, creating portal accounts with Geneva users"

Write-Host "Geneva usernames should not have a domain or provider in the string e.g. testuser"

$superReaderString = Prompt "Enter a geneva user name to use for the super reader account"

$superUserString = Prompt "Enter a geneva user name to use for the super user account"

$superReaderPrincipal = New-SPClaimsPrincipal -Identity $superReaderString -TrustedIdentityTokenIssuer $tokenIssuer

$superReaderString = $superReaderPrincipal.ToEncodedString()

$superUserPrincipal = New-SPClaimsPrincipal -Identity $superUserString -TrustedIdentityTokenIssuer $tokenIssuer

$superUserString = $superUserPrincipal.ToEncodedString()

}

else

{

Write-Host "This application cannot be configured with this script"

Exit

}

}

if($superReaderString -eq $superUserString)

{

Write-Host

Write-Host "You must use different accounts for Super User and Super Reader! Exiting..."

Exit

}

ConfigureAccounts $app $superReaderString $superUserString

Write-Host

Write-Host "You need to IISReset on each WFE to complete the operation"

1. SPWebService.FileReadChunkSize, defaults to 100KB ([SPWebService.FileReadChunkSize Property](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.spwebservice.filereadchunksize(office.14).aspx)) [↑](#footnote-ref-2)
2. SPWebService.LargeFileChunkSize, defaults to 5MB ([SPWebService.LargeFileChunkSize Property (Microsoft.SharePoint.Administration)](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.administration.spwebservice.largefilechunksize.aspx)) [↑](#footnote-ref-3)
3. In deployments where the bandwidth between the WFE and the client is slower than the bandwidth out of SQL's disk (or buffer pool) and the bandwidth between SQL and the WFE, additional round trips to retrieve the file after 100k do not contribute to the latency of the request. Since we expect this to be the case in most deployments, that means that the latency of the request is governed by: {the size of the file}/{WFE-to-client bandwidth} + min(FileReadChunkSize, {size of the file})/{SQL-to-WFE bandwidth} + {fixed overhead for serving a request}. In the case when the file is larger than LargeFileChunkSize, you add the time it takes to make a temp file large enough to hold on the WFE file system plus the time it takes to read/write the FileReadChunkSize bytes of data to the temp file [↑](#footnote-ref-4)
4. A web garden is an application pool that has more than one process handling requests. See [Performance Application Pool Settings](http://msdn.microsoft.com/en-us/library/aa720391(VS.71).aspx) [↑](#footnote-ref-5)
5. If the cache index becomes corrupted, the cache will be flushed the next time the cache is deserialized. If the cache mutex is not released because the process is terminated, a reboot may be required to clear the mutex. [↑](#footnote-ref-6)
6. Compaction can be monitored using the “Total number of cache compactions” Performance Monitor counter under the SharePoint Disk-Based Cache group. [↑](#footnote-ref-7)
7. Publishing is a feature in SharePoint Server that allows the separation of content and its presentation. In this model a page layout is created that defines the place holders for content; pages are the containers for the content. The target scenario for this feature is an Internet facing web site that has multiple content contributors where all of the content shares a common page structure. [↑](#footnote-ref-8)
8. This can also be configured for an entire web application. More on that is in the configuration section. [↑](#footnote-ref-9)
9. The rendered size is the size of the HTML source that is sent to the browser without any compression. [↑](#footnote-ref-10)
10. Kind of user in this case refers to whether or not the user can see draft items. [↑](#footnote-ref-11)
11. SharePoint always returns the checked-out version for items that a user has checked out. [↑](#footnote-ref-12)