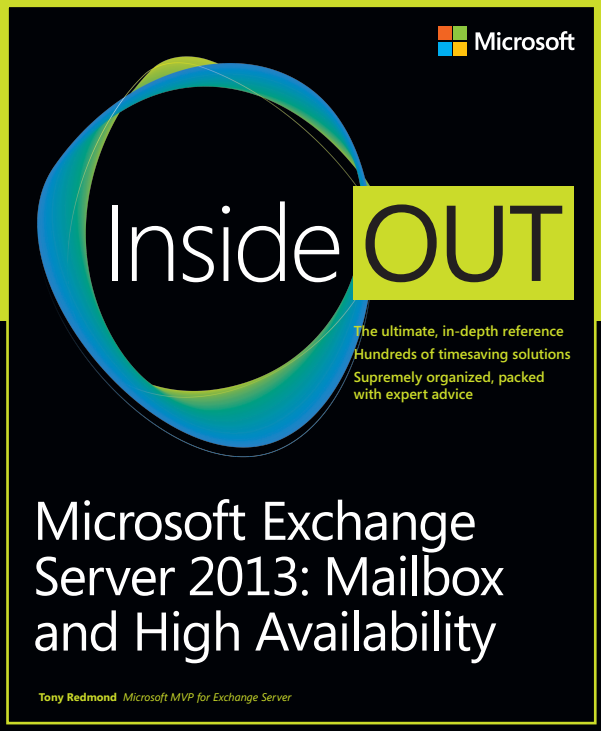
EXCERPT



Chapters 8-9

Managing the Store & DAG

Tony Redmond

Microsoft

Managing the Store & DAG: EXCERPT from Microsoft® Exchange Server 2013 Inside Out

Tony Redmond

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2013948705
ISBN: 978-0-7356-8080-7

Printed and bound in the United States of America.

First Printing

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at mspinput@microsoft.com. Please tell us what you think of this book at http://www.microsoft.com/learning/booksurvey.

Acquisitions Editor: Anne Hamilton
Developmental Editor: Karen Szall
Project Editor: Karen Szall
Editorial Production: nSight, Inc.
Technical Reviewer: Paul Robichaux; Technical Review services provided by Content Master, a member of CM Group, Ltd.
Copyeditor: Kerin Forsyth
Indexer: Lucie Haskins
Cover: Twist Creative • Seattle

Contents at a Glance

Introduction

When Paul Robichaux and I first sat down to discuss how we might cooperate in writing Microsoft Exchange Server 2013 Inside Out, we were conscious that the sheer complexity and breadth of the product meant that many months of fact-gathering, writing, checking, and editing would be necessary to produce a book that described Exchange 2013 in sufficient depth and detail to warrant the "Inside Out" title. In fact, we knew that two books were necessary to avoid ending up with one 1,600-page mega-volume, so we divided the task to align with the Mailbox and Client Access Server roles, which is the plan that we've used to create the books.

At the same time, we knew that there were particular parts of Exchange 2013 that justified their own book. I've often felt that Unified Messaging was a very misunderstood part of the product. Paul has worked in this area for many years and has taught the subject to students aspiring to become Microsoft Certified Masters for Exchange. He's the best possible person to write about Unified Messaging. I have had a particular interest in compliance, and it's an area in which Microsoft has invested massively over Exchange 2010 and Exchange 2013, so it was easy (relatively) to create a mini-book on this topic. Then we were faced with High Availability, an area that is so important to so many companies who depend on their email being available all the time. The advent of the Database Availability Group (DAG) was a tremendously important advance for the product in Exchange 2010, and it's even better in Exchange 2013. Making High Availability the focus of the third mini-book made a lot of sense.

Each mini-book is a chapter from the larger "Inside Out" title, but each stands on its own merits and can be read in isolation. However, if you really want to get acquainted with Exchange 2013, you might just want to check out the two-volume set. We think you'll like it.

Errata & book support

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site:

http://aka.ms/ExIOv1/errata

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software is not offered through the addresses above.

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://www.microsoft.com/learning/booksurvey

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

Stay in touch

Let's keep the conversation going! We're on Twitter: *http://twitter.com/MicrosoftPress*.

Paul is available on Twitter *@PaulRobichaux*. His blog is available at *http://paulrobichaux.wordpress.com/*. Tony is available on Twitter *@12Knocksinna*, while his blog is at *http://thoughtsofanidlemind.wordpress.com/*.

CHAPTER 8

The Exchange 2013 Store

Maintaining and evolving a database isn't an easy engineering challenge, especially when the database has to deal with transactions that vary widely, from a simple 10 KB message sent to a single recipient to a message sent to a large distribution (with each address individually expanded in the header) accompanied by a 10 MB Microsoft PowerPoint attachment. Today's competitive environment and the demands of the service create the need to support very large mailboxes and to answer the continuing demands of the corporate sector for improvements in areas such as high availability. Microsoft might well have invested more engineering in the Store even if Google hadn't come along, but there is no doubt that competition can have a stunning effect on innovation.

The overall goals Microsoft engineering took on as it rewrote the Store into managed C# code (the reason it's called the Managed Store) in Exchange Server 2013 were the following:

- Continue to reduce I/O to take advantage of denser, cheaper disks and increase the flexibility available to storage architects when laying out storage designs for Exchange.
- Make 25 GB mailboxes work as well in Exchange 2013 as 1 GB mailboxes work in Exchange 2007.
- Improve the operational effectiveness of Database Availability Groups (DAGs) in areas such as site transition, use of lagged database copies, and automation.

Whenever you think about the Exchange Store, three major components come to mind:

1. The Extensible Storage Engine (ESE), the database engine that organizes the contents of mailbox databases at page level. Strictly speaking, ESE is not part of the Store. Instead, it is the underlying database engine on which the Store is built. However, ESE and the Store go together like twins, or maybe the odd couple.

2. The Information Store, which lays down a logical schema that defines the contents of mailboxes and how those contents are organized.

3. The Exchange database (EDB) and transaction logs. The EDB is the database file on disk. The transaction logs record any change made to a page within a database (adds, modifies, deletes) and provide the ability for Exchange to recover from a database failure by replaying transactions recorded in the logs. The transaction logs are also used to replicate data between servers and keep database copies synchronized so that copies can be quickly switched in to provide continuous service to users if the primary (active) database fails.

This chapter reviews the components of the Exchange 2013 Store and the major changes made in this version.

Long live JET!

Exchange 2013 continues to use ESE, which has been highly optimized over the years to meet ongoing product requirements for Exchange. Some are surprised that Microsoft persists with ESE as the platform for Exchange when it seems to have another perfectly good enterprise-class, high-performance database engine in SQL Server. On the surface, it's an attractive notion to unify all database development activities into a common platform that Microsoft and third-party developers can support. Money would be saved on engineering, testing, and support activities, and everyone could use a single target platform through a unified set of application programming interfaces (APIs). Microsoft has been down the path to investigate the effort required to move Exchange to SQL on a number of occasions. However, apart from the sheer engineering effort, a number of technical and nontechnical challenges would have to be overcome before Exchange could use SQL.

The demands on the two database engines are very different: ESE is optimized to process messaging transactions that vary from a one-line message sent between two people to a message sent to a very large distribution complete with a multi-gigabyte attachment (your favorite communication from the marketing department). SQL is optimized to handle structured transactions that do not tend to be highly variable. Could performance and scalability be maintained in a single database that attempted to serve both demands?

ESE enables users to create their own indexes (views) on the fly with a simple click of a column heading in Microsoft Outlook. The result is a colossal number of temporary views that differ from database to database that ESE manages in an elegant manner. SQL doesn't tend to be so accommodating. Again, SQL could provide the functionality, but would it take away from database performance?

SQL is designed to be a platform for applications. Exchange has had many attempts to become a platform for different types of applications, such as workflow routing, but is now simply a messaging server. Microsoft SharePoint (which uses SQL) has supplanted Exchange as the cornerstone of the Microsoft collaboration applications strategy. If Exchange used SQL, would Microsoft still be able to sell SharePoint as successfully as it does today?

Both SQL and Exchange have developed third-party ecosystems around their platforms that add enormous value for anyone who wants to deploy these products. Given that most partners who support Exchange today use Exchange Web Services (EWS) rather than ESE to access the Store, it shouldn't be a problem to move them to SQL because EWS could hide the changeover. In any case, who really cares what database engine powers an application as long as it does the job and delivers robust and scalable performance in a reliable manner? Do people ask what database Google uses for Gmail? The choice of the underlying database technology really shouldn't be an issue as long as email continues to flow.

Maximum database size

Microsoft has recommended a maximum size for an Exchange mailbox database of 2 TB. The current database page size of 32 KB makes the maximum theoretical size of a database at the NTFS file system limit (16 TB–64 KB). To be safe, the absolute limit for an Exchange 2010 or Exchange 2013 database is 16,000 GB, just below the NTFS file system theoretical limit. Few customers are likely to operate databases that approach anywhere near this value in the next decade, especially when they consider just how long it would take to restore such a monster database if that need were to arise.

Allowing for a 2 TB maximum database size seems startling on the surface, but it can be justified by three major technical and operational factors:

- Exchange 2013 does not support streaming backups, and you are forced to use Volume Shadow Copy Services (VSS) disk-based backups instead. VSS backups are faster and more capable of dealing with huge volumes of data. Taking a snapshot is a lot faster, and if you need to create a copy on tape to satisfy off-site archival procedures, you can do this after the VSS backup is available. Note that changes in the way Exchange supports VSS writers means that third-party backup software vendors need to produce new versions of their products to support Exchange 2013. Alternatively, on smaller systems, you can use Windows Server Backup as described in *http://technet.microsoft.com/en-us/library/dd876854.aspx*.

Tip

Note that you do not need to use VSS to restore a database. You can certainly recover a database from a snapshot or cloned copy, but it is much faster to use continuous replication and replace a failed active copy with a passive copy in a DAG.

- Exchange 2013 supports a replication model that allows up to 16 copies of a database. If you create three or more copies of a database, there's a fair argument to make that backup operations become less important than they were in the past because you can recover faster and more simply from a passive copy than by reaching for the most current backup. This fact does not eliminate the need for backups because legal and other requirements might mandate backup copies for off-site storage. See Chapter 9, "The Database Availability Group," for a discussion about the Database Availability Group (DAG) and how Exchange 2013 handles multicopy database replication.
- Background maintenance performed by the Managed Folder Assistant and other components means that Exchange Mailbox servers are quite capable of keeping very large databases in good health.

A large database really doesn't take longer to mount and make available to users; the primary consideration in terms of mount time is the number of transaction logs that might need to be replayed to bring the database up to date during the mount operation. Larger databases might have more transaction logs to replay, but the net impact probably won't be noticed.

INSIDE OUT The assumptions behind capacity planning

Considerations such as availability, recovery, and business requirements must be factored in to server planning. Tools available from Microsoft such as the Exchange 2013 Server Role requirements calculator (available for download from TechNet) help determine the basic technical design for Exchange 2013, but be prepared to hand-tailor the resulting output as you incorporate other needs that cannot be easily input into a spreadsheet before you arrive at the final design. All sizing tools make assumptions such as the default client type, user profile, and concurrency ratio, and the recommendations they make will be most satisfactory when your situation closely matches these assumptions. In other circumstances, your users might be busier than the assumptions, they might not use Outlook and Outlook Web App, or they might not connect as often as you'd expect, and therefore the results generated by a sizing tool will need to be adjusted before they make sense for you.

Sizing mailboxes

Calculating database size by multiplying the number of users by the mailbox quotas is still a valid way to approach the problem of what size database you can support, and it's certainly one that works in small to medium deployments. However, email systems have become more complex in the past few years, and other factors must be taken into account. Microsoft has posted an excellent discussion of the factors that drive Exchange 2013 hardware planning on *http://blogs.technet.com/b/exchange/archive/2013/05/06/ask-the-perf-guy-sizing-exchange-2013-deployments.aspx*. The concepts explained there can be used with some or all of the following questions to help frame the discussion.

- Are any legal or regulatory conditions in place that force retention of mailbox data for a specified period?
- What is the company's backup regime? Will you use DAGs to achieve higher availability and protection with Exchange 2013?
- How are users coping with current mailbox quotas? Are most within limits, or do you see a constant stream of requests for additional space? Are you happy to have users spend time juggling mailbox contents to keep under their quota?
- What deleted-items retention policy do you operate? Is it sufficient or should it be extended? How often do you have to restore from backup to recover user data?
- Do you force automatic deletion of items in mailboxes? If not, should you be using a retention policy?
- Do you use an archiving product with Exchange? Will Exchange archive mailboxes replace or complement this product? Will archive mailboxes be collocated in the same database as the primary mailboxes or in another database, or even grouped on a separate, dedicated archive server?
- Do you need to journal messages? How often do you have to perform legal discovery searches or otherwise react to complaints when email forms essential evidence that has to be found and retained?
- Has the messaging profile of your users changed in the past few years? Are they sending and receiving more email? What is the average size of the messages? Are they using calendars and other mail-enabled applications more?

These questions address a mixture of technical and business requirements and lead to an understanding of how users generate and use mailbox data. There's no simple formula to turn the responses into an answer for the mailbox quota you should provide and how many databases of a particular size will be required. Instead, you're more likely to realize

that you have to deliver differentiated services to various categories of users. Executives and other high-impact staff will have different quotas, retention periods, and archiving and legal requirements than factory workers, and their mailboxes will probably be in databases located on the most highly available servers. These servers might use different hardware configurations, be operated by dedicated staff, and have to support access through a variety of devices (mobile, Outlook, and Outlook Web App) rather than the more limited variety that might be allowed to less-important users.

INSIDE OUT **Larger mailbox quotas required!**

One of the changes made in Exchange 2013 is in the way the Store calculates the space used for mailboxes that is then charged against a user's mailbox quota. All databases incur some level of overhead to manage internal structures. In the past, Exchange did not charge this overhead against mailbox quotas, but Exchange 2013 does. The net effect is that database sizes stay the same because the same level of overhead exists, but mailbox quotas might have to be increased because the overhead has been charged. In effect, the reported size of a user mailbox will appear to grow by some 30 percent after it is moved to Exchange 2013, despite nothing additional being held in the mailbox. At least, not in the eyes of the user. A proactive step is therefore to review the quotas assigned to user mailboxes before they are moved to Exchange 2013 to ensure that sufficient quota remains to enable Exchange 2013 to charge the additional overhead to the mailbox. Some organizations have simply increased all quotas by 50 percent to ensure that they won't encounter difficulties during migration, with the intention of reviewing quotas when everyone is on Exchange 2013. This seems to be a very practical and pragmatic approach.

Dealing with I/O

Even though the type and volume of email had changed dramatically since Microsoft first laid out the Exchange database schema in 1996, Exchange Server 2000, Exchange 2003, and Exchange 2007 use the same schema that emphasizes efficient storage over utility. Cheaper storage makes it less attractive to focus on storage efficiency. To achieve a reduction in I/O, the Store has steadily moved from forcing disks to do many small, random I/Os to fetch data to using larger, sequential I/Os. The physical performance difference between random and sequential I/O almost guarantees better performance and lower I/O activity for any application if the code is written to move away from random I/O. The current database schema emphasizes contiguity over storage, essentially by keeping mailbox content together. Because more data is contiguous, the Store can read data out in large sequential chunks rather than in many random and smaller chunks.

Exchange has not supported single-instance storage since Exchange 2010. You might assume that databases will grow larger because mailboxes hold their own copies of messages. In fact, this isn't always true because of the way databases compress message content. However, a side effect of the elimination of single-instance storage is that more transaction logs are generated to capture the insertion of individual copies of items in the table for each mailbox. For example, if you send a message with a 1-MB attachment to 10 mailboxes in the same database, the Store has to create at least 10 transaction logs.

Views (or secondary indexes) are a very valuable user feature. Outlook allows users to sort items within folders by using a wide variety of properties (author, date received, subject, and so on). Every time a user opts for a different sort order, the Store creates a new view. Thereafter, as new items arrive in the Store, a lot of work occurs to update the views. Eventually, if a view is not used, it expires, and the Store removes it.

Large mailboxes tend to have many views, and those views are usually for the default folders (Inbox, Sent Items, and Deleted Items). Many people don't like the work involved in filing email in different folders and are happy to let messages build up in the default folders, and they only ever take action when prompted to by quota exhaustion. Search technology facilitates this behavior because no penalty exists if you never file a message; search will always find an item faster than you can, even with a solid filing system. The result of declining discipline in filing is folders that hold many thousands of items, in turn causing the Store to generate additional I/O to access large folders. This prompted Microsoft to recommend that the largest folder should hold fewer than 5,000 items in Exchange 2007. The improvements made since enable Exchange 2013 to tolerate up to one million items in a single folder (assuming that the view for the folder is a standard view that's optimized within the schema and does not have to be rebuilt). However, this is very much a notional limit because even the most modern clients will run into performance difficulties when they attempt to deal with such a large folder. It's still best to coach users to keep fewer than 10,000 items in a folder whenever possible—or to deploy retention policies to help users keep their mailboxes under some degree of automated control.

How Exchange 2013 supports large mailboxes

Exchange 2013 is designed for a world in which mailboxes of 25 GB and above are often encountered in deployments. Here are some of the ways the Exchange 2013 database schema and layout facilitate large mailboxes.

- **Use an optimized Store schema designed to deal with large mailboxes more efficiently.**
- **Force pages within the Store to be laid out contiguously rather than having pages scattered around the database.**

- **Use large, 32-KB pages that can store complete messages and ensure that the Store allocates pages in large, contiguous chunks rather than randomly within the database.**
- **Use large I/Os to fetch sequential chunks of data. The logic here is very simple. Today's disks are capable of providing data at 300 or more sequential I/O operations per second (IOPS), whereas their ability to handle random I/O remains constrained at around 50 IOPS. Given that disks can provide sequential data six times faster than random data, it makes perfect sense for Microsoft to focus its engineering investment on moving activity within the Store from random to sequential. Apart from the performance benefit, this step enables a wider range of storage designs to support Exchange.**
- **Use a large, 100-MB cache size for mailbox databases in a DAG. This step enables the Store to keep more dirty pages in memory. A difference between Exchange 2010 and Exchange 2013 is that passive database copies move from a 5-MB to a 100-MB cache, which reduces the amount of I/O consumed to keep a database copy updated. This cache size (basically memory assigned to hold data that have been recorded in transaction logs but not yet committed to the database) is referred to as the checkpoint depth. Increasing the cache improves the chance that Exchange can update a page in memory rather than having to generate an I/O to retrieve it from disk. Increasing the cache size introduces some additional complexity that the Store takes care of, including the potential for longer shutdown times because the cache needs to be flushed to disk and longer recovery times from crashes because more transactions have to be replayed from transaction logs. However, the overall benefit in I/O improvement is worth taking these risks.**
- **Use compression for some object types within the Store.**

The work to enhance the Store over the last two releases has had a huge impact on I/O performance. Inside a reasonably small database, it's probably acceptable to assign pages to store new items in whatever pages are free within the database. For a 20-KB message, this meant that the item might be placed in several pages scattered in different locations within the database. From a logical perspective, the database can find the five pages and provide their locations to the Store whenever a client requests the item, but this activity generates a lot of very small I/Os. A 32-KB page size makes it more likely that a single item will fit on one page and ensures that pages required to hold new items are assigned contiguously in large chunks.

When clients need to access an item, the Store uses a technique called gap coalescing to read in a range of pages in a single large I/O rather than hunting around the database for individual pages. The Store also writes pages in contiguous chunks to smooth I/O. Consider when four pages in a contiguous set of six pages are updated in the cache and need to

be written to disk. Exchange 2007 writes each of the four dirty pages in individual I/Os. Exchange 2013 writes the six (dirty and clean) pages in one large I/O, and ESE discards the unchanged pages as it updates the database. Using gap coalescing reduces the overall I/O demand Exchange exerts and increases overall data throughput.

The change that updates views on demand rather than after every change is also important in terms of I/O reduction. Microsoft refers to this as a change from a nickel-and-dime approach, in which the Store is in a state of constant turmoil as views are updated after the arrival of new items, to a pay-to-play approach, in which the Store updates a view only when a client wants to use it. Consider when a user creates a view through Outlook Web App (Outlook clients that work in online mode also create views in this manner; when Outlook works in cached Exchange mode, it creates views on the client) to look at items in her inbox sorted by author (sender) rather than the default sort order (received date/time). Such a view is usually created so a user can look for items sent by a specific individual, when a view sorted by author is much more useful than the default view sorted by the time an item is received. However, assume that the user then switches back to the default view and doesn't look at a view by author for another week. Even though the client does not intend to access the view, the Store keeps updating it just in case. This might be good in that a view is immediately available, but it incurs a horrible overhead for the Store to keep track of all the views and update views as new items arrive. The resulting updates generate I/O activity because the changed views have to be written back into the database. Eliminating these updates and using an on-demand model is a very intelligent change, especially because the multicore servers used today are very capable of handling the small extra demand for the processor to compute a view when required.

In its efforts to improve I/O performance, Microsoft also paid attention to the concept of database write smoothing. Every disk has its performance limitations that restrict the amount of data it can handle at any time. If you attempt to write too much data to disk at one time, the disk will be swamped, and a disk contention condition will occur. The application cannot write its data as quickly as it wants to, and everything halts until the disk can accept more data and relieve the contention condition. Write smoothing means that the application is intelligent enough to detect when disks are approaching the limits of their performance and can throttle back its demand to allow the disks to continue to process data at a consistent rate.

Exchange uses the checkpoint depth as its measure to determine when it should throttle database writes. This approach is advantageous because it works for all disks. When the checkpoint depth is between 1.0 and 1.24 of the checkpoint target, the Store limits the outstanding write operations for each logical unit number (LUN) to one until the condition improves. If the checkpoint depth goes past 1.25 of the target, further throttling is imposed by increasing the maximum number of outstanding write operations per LUN until a balance is attained and the depth begins to reduce. Pending writes remain in cache until they can be written to disk.

Microsoft claims that it has driven a reduction of over 90 percent in I/O operations since Exchange 2003. The actual gain must be measured in your own specific hardware environment and might be better or worse, depending on server and storage configurations, user workloads, and types of clients. However, it's fair to say that the focus on reducing I/O has worked and has greatly improved the overall performance characteristics of the product.

INSIDE OUT When are large mailboxes justified?

Many older Exchange deployments started with mailbox quotas of 50 MB to 100 MB and have gradually increased over time to a range of 250 MB to 10 GB, with some select mailboxes being allocated a higher quota. Usually, these mailboxes are owned by executives or other users who have a business need for higher email volumes or for holding data for longer periods. The other users resort to PSTs to hold messages they want to keep so that they stay within their allocated quota.

Although the practical limit for an Exchange 2013 mailbox is around 100 GB, in reality, this usually translates into an average corporate mailbox size of 5 GB to 10 GB. A mailbox quota of 5 GB is sufficient to store approximately 100,000 items with a 50 KB average size. For most people, it will take five years to accumulate that much mail, and it is possible that the company's legal team would prefer users not to store quite so much mail!

Before you rush out to enable 25-GB mailbox quotas, it's wise to reflect on the business requirements you might meet with large mailboxes. These reasons might be valid for your company:

- **Reduction in unproductive time spent by users who struggle to stay within lower quotas. How much does it cost if a message containing some time-critical information cannot be delivered to a mailbox because its quota is exhausted?**
- **Elimination of PSTs through the deployment of archive mailboxes.**
- **Elimination of expensive deleted-item recovery from backups through the deployment of an extended deleted-items retention period (more than 60 days).**
- **Ability to conduct online searches through all email available to the company rather than the subset that is available in online mailboxes.**
- **Protection of all email rather than just the items in the Exchange database.**
- **Provision of all mailbox content to multiple clients. Outlook Web App and mobile devices cannot access PSTs, and mobile devices cannot access archive mailboxes.**

Of course, keeping vast quantities of email has its downside. Search facilities are excellent today, but reviewing a large set of messages retrieved by a search to find a particular item is never a thrilling experience. From the system perspective, increasing user mailbox quotas means larger databases, bigger backups, and more data to manage, including the very real concern some companies have that retaining so much data makes the company more vulnerable to discovery actions. Like everything else in life, a middle path should be walked between keeping everything and keeping nothing. When you have a set of reasons to justify the deployment of larger quotas, you can begin to figure out how to deploy the servers and storage necessary to support large mailboxes.

Maintaining contiguity

Enormous care is taken to allocate contiguous space within Store databases. However, this is not the end of the story because pages have a lifetime that begins with their creation and extends through modifications and deletes. Given the need to support multi-terabyte databases, the old way of maintaining page contiguity through scheduled background maintenance could not work.

As the Store processes transactions, it checks to ensure that maximum contiguity is maintained and that it is using available space efficiently. If necessary, a background thread is created to address any problems—perhaps it will shuffle some pages to make them more contiguous or to free up space in the database. All this work is throttled automatically so that background operations never interfere with the ability of the Store to service user requests, just like Windows Desktop Search only indexes items on a PC's hard disk when the PC is inactive.

The result of these operations is that Exchange 2013 databases start out and remain more contiguous than their predecessors. Of course, some extra CPU resources are consumed to perform run-time processing, but this should not be a problem in most situations because the normal bottleneck for Exchange has been I/O rather than CPU for the past decade.

As you read the discussion about contiguity and about the 32-KB page size Exchange now uses, the thought might run through your mind that large pages arrayed in bigger chunks might increase the overall on-disk size of a database. This conclusion is correct, but it is offset by the introduction of data compression within the database.

Microsoft believes that the potential extra growth in database size is fully mitigated by the way message headers and HTML and text body parts are compressed. Of course, the exact ratio of compression varies, depending on the mix of content within a database (Rich Text Format [RTF], text, HTML, different types of attachments). For example, RTF messages are

already compressed and therefore are not compressed again when they are written into a database. Outlook 2010 and Outlook 2013 generate HTML-format messages by default so databases that support these clients get more value from compression than databases that support earlier clients such as Outlook 2003, in which the default message format is likely to be RTF. Exchange achieves additional efficiency by first clustering records and tags into contiguous chunks within a page before attempting to compress the data.

INSIDE OUT **Data compression doesn't include attachments**

Although great value is attained by compressing message bodies, Exchange doesn't attempt to compress attachments. Tests by the Exchange development team demonstrate that most attachment types that circulate with messages such as newer versions of Word documents and Microsoft Excel workbooks don't compress well because they are already stored in a compressed format. The same is true of other attachment types you might not want to see circulated by email, including JPEG, PNG, MP3, and WMA. Some older formats such as Word 2003 do compress well, but because these versions are being rapidly replaced, their attachments will form a decreasing percentage of the overall attachment volume. The decision was therefore made to avoid incurring the CPU overhead of compression to achieve what might be a marginal decrease in overall storage requirement, especially at a time when storage costs are reducing rapidly. The same logic was used when the decision was made not to attempt to compress RTF message bodies.

The database schema

The schema sets out how data are physically and logically organized within a database. Apart from some minor tweaking, Microsoft used the same schema from Exchange 4.0 through Exchange 2007, but the schema used since Exchange 2010 made some major changes, the biggest of which is the way the contents of mailboxes and folders are organized. The previous schema laid out a mailbox database as follows:

- A mailbox table containing a pointer to every mailbox in the database.
- A folders table containing a pointer to every folder in every mailbox in the database.
- A message table containing an entry for every message in every folder in the database.
- An attachments table containing an entry for every attachment in the database.

- A message/folder table maintained for every folder that lists all the messages in a specific folder. For example, the message/folder table for your inbox contains pointers to every message in your inbox.

In essence, you have four big tables shared by every mailbox in the database and a set of message/folder tables for the different folders. This schema, shown in Figure 8-1, obviously works successfully, but ESE generates more I/O than necessary to navigate through the tables in response to user requests, especially as some of the tables become very large in databases that support thousands of mailboxes. For example, a database supporting 4,000 mailboxes, each of which has 150 folders containing 20,000 items, has a folders table with 600,000 rows and a messages table with 80,000,000 rows. The message/folder tables aren't so much of a problem because most users have noted the advice given over the years to restrict the number of items within a folder to less than 5,000.

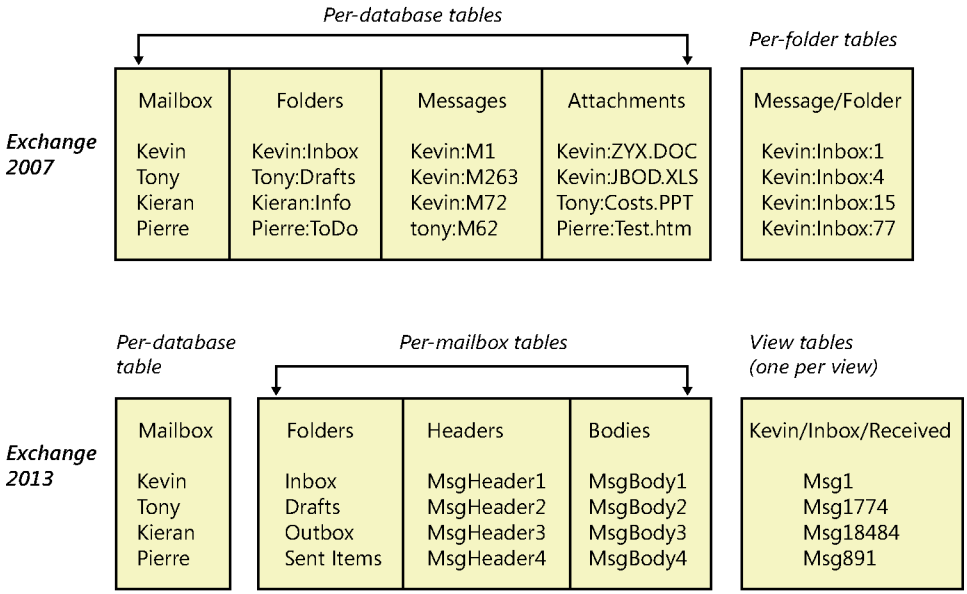


Figure 8-1 Difference between Exchange 2007 and current Exchange database schemas

As illustrated in Figure 8-1, the layout imposed by the current Exchange schema is as follows:

- A mailbox table containing a pointer to every mailbox in the database (as before).
- A folders table for each mailbox in the database. Although there are many more folders tables, the size of each is much smaller. Taking the example cited earlier, there are now 4,000 folders tables (one for each mailbox), each of which holds 150 rows.

- A message header table for each mailbox in the database. The message header table holds all the Messaging Application Programming Interface (MAPI) properties (subject, address list, priority, and so on) for the messages in a mailbox. Again, there are 4,000 message header tables, but the number of items in the largest mailbox sets the maximum size of a table, around 20,000 rows in this example.

- A message body table for each mailbox in the database. The message body table holds the message content (the body text), including attachments. Again, there are 4,000 message body tables for this example database, and the largest mailbox determines the number of items in the maximum size of the table.

- In addition, ESE maintains view tables on an as-required basis. A view is an ordering of a folder (for example, the Inbox ordered by date received). Previous versions of Exchange use secondary indexes to populate views. Exchange now holds views as tables. Although there can be many views (more than the number of folders because users can create multiple views per folder), the maximum size of a view table is determined by the number of items in the largest folder.

Two major types of tables are found within an Exchange mailbox database. Global tables hold information about all mailboxes in the database, whereas mailbox tables hold data specific to individual mailboxes. Global tables include the catalog; a registry of all tables that exist in the database; another table (Globals) that tracks information such as the database version; and the Mailbox table, containing basic information about the mailboxes in the database such as the owner, locale, and last logon time. Another global table tracks information about message delivery, and another is used by mailbox assistants to track events. The Mailbox table is a good example of one that is optimized for sequential I/O. Of the mailbox-specific tables, those that store folder information, messages, attachments, and the physical indexes are also optimized for sequential access.

In summary, the current database storage schema focuses on smaller tables that store data on a per-mailbox basis rather than on much larger per-database tables. This approach allows ESE to be more efficient as it navigates through a database and generates fewer I/O transactions for the storage system to process. It also enables Exchange to support up to 100,000 items per folder and still deliver reasonable responsiveness to clients that work online. The performance of cached mode clients depends on other factors such as the speed of the local hard disk and the version of the client. (Outlook 2013 or later is best because of the intelligent caching behavior incorporated into this client.)

Exchange 2013 I/O improvements

The overhaul of the database schema contributed enormously to the dramatic improvements in I/O reduction seen in the last two product releases. Exchange 2013 features a rewritten Store due to a transition to managed code. As part of the transition, the Exchange

developers were able to remove a large amount of redundant code that lingered from a time when the code might have been required in previous versions. At the same time, they were able to take advantage of the lessons learned from Exchange 2010 deployments, both by customers and at Microsoft, including Exchange Online running within Office 365.

The following changes contribute to better I/O performance in Exchange 2013:

- Within Exchange 2010 Database Availability Groups, passive database copies have a 5 MB checkpoint depth. This meant that the Store cached a relatively low amount of data for passive copies with the idea that this would enable faster activations of a passive database to become active. The low checkpoint depth and some aggressive pre-reading of data to maintain cached data at the 5 MB level meant that the I/O requirements for a passive database copy were the same as for an active database copy, despite the fact that all mailbox activity was focused on the active copy. Exchange 2013 increases the checkpoint depth for passive copies twenty-fold to 100 MB, which reduces the amount of processing required to maintain the depth. The result is that a passive copy now requires only 50 percent of the I/O of an active copy. Keeping more cached data for a passive copy might prevent fast failovers if the cache has to be flushed before activation (as is the case in Exchange 2010), so changes were made at the ESE level to persist the cache during activation and thus preserve the ability to move a database copy quickly from passive to active. In fact, with the changes, database transitions are reckoned to be somewhat faster in Exchange 2013 with the expected failover at around 10 seconds as opposed to 20 seconds for Exchange 2010.

- The Store performs ongoing background maintenance to keep databases healthy (see the "Background maintenance" section later in this chapter). In its early days, Exchange databases needed a lot of tender loving care because a mixture of bugs (server and client), storage glitches, and other issues caused frequent problems. Today, the more robust and resilient Store (including features such as single-page patching) means that a less aggressive approach can be taken to background maintenance, a feeling that was confirmed through instrumentation of databases running in the Exchange Online service. Exchange 2013 throttles back maintenance from 5 MB/second to 1 MB/second. Maintenance is still done, just a little more slowly than before. The upside is that less I/O is consumed for maintenance.

- Further database and schema tuning contribute to reduced I/Os too. For example, collection of MAPI properties that are frequently used in view operations are held in binary large objects (BLOBs) so that they can be fetched together in one chunk. Read operations have been examined and converted to sequential I/O whenever possible to eliminate random I/O further and make Exchange deployments feasible on even low-performing storage subsystems.

Chasing I/O improvements requires constant effort to track changes in the code (Exchange and Windows), disk technology (across all classes of storage), and to accommodate customer demands. The Exchange developers have done extremely well over the past decade to move from when storage was the Achilles heel of Exchange deployments, especially at scale when the necessary storage was expensive and often difficult to manage, to today's situation, when Exchange 2013 will run on just about any storage you care to deploy. It should be interesting to see what they'll do in the future.

Workers, controller, and memory

As part of the rewrite to managed code, Information Store processing is now separated into the work done by a Store controller process and a set of worker processes, one for each database (active or passive copies) mounted on a server. Each worker process deals with the transactions flowing into its database and does the work required to keep the database healthy, such as background maintenance. The major advantage gained by moving work into a set of worker processes is that any problem that occurs is automatically isolated to a single process and therefore can affect only the database to which the process is connected. Of course, if the controller process dies for some reason or is terminated by an administrator (for instance, to stop the Information Store service), all the worker processes also terminate. Figure 8-2 shows that four *Microsoft.Exchange.Store.Worker* processes are running on a server. You can therefore conclude that the server supports four mounted databases (which could be active or passive copies). You can also see the Store controller process, *Microsoft.Exchange.Store.Service*. You can discover which worker process controls a database by running the Get-MailboxDatabase cmdlet. The value of the WorkerProcessId property returned by Get-MailboxDatabase is the identifier, or PID, of the worker process.

The change from a single monolithic Store process to a coordinated set of worker processes also affects the way Exchange 2013 uses memory on Mailbox servers. The old Store process uses a mechanism called dynamic buffer allocation to seize as much memory as it can on a server and then throttles back its usage depending on the requirements of other applications and Windows. This is why Store.exe appears to occupy huge amounts of memory on previous Exchange servers.

As the Store becomes active, worker processes are initialized to manage each of the databases on a server. Some databases will be active and some passive. Exchange 2013 begins the process of memory allocation by building the ESE cache, using 25 percent of available system memory. This memory holds database structures in memory instead of having to go to disk to fetch pages because of client transactions. The cache is then divided by the number of mounted databases on the server to establish a maximum target memory for each database (Max Cache Target). If the Set-MailboxServer command is used to configure a *MaximumActiveDatabases* property to control the maximum number of databases that can be activated on the server, this value determines the dividing factor for the ESE cache.

Worker processes use this factor to determine how much of the cache they request ESE to allocate for their workload.

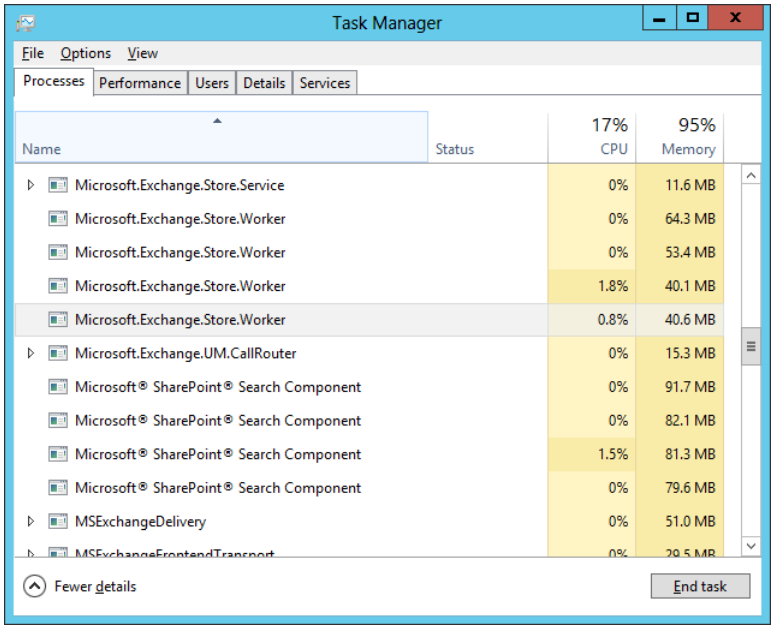


Figure 8-2 Store processes on an Exchange 2013 Mailbox server

The cache size is determined when the worker process starts and is not dynamically adjusted following the addition or reduction of mounted databases on the server, and setting the *MaximumActiveDatabases* property will not immediately affect the determined size for the ESE cache. If a need exists to calculate a new ESE cache value, such as adding or removing a database, you have to restart the Information Store service to force the worker process to recalculate the cache size.

A worker process can grow to the Max Cache Target limit that is in force when its database is mounted. The worker processes dealing with active databases are allocated the full amount, whereas those associated with passive databases get 20 percent of the maximum target. Of course, if a database copy is activated, the worker process increases the cache used to reflect the fact that the process is now handling active client connections.

For example, assume that you have a Mailbox server equipped with 32 GB of RAM. Following the rule, a quarter of the memory (8 GB) is computed as the ESE Max Cache Target. If four active databases are mounted on the server, their share of the ESE cache will be allowed to grow to a maximum of 2 GB. Such a server is likely to have many more databases mounted on it. If you were to add four passive copies, you would now have

eight mounted databases, so the maximum amount of memory each database can be allocated is now 1 GB. The worker processes for those copies will each start with 200 MB (20 percent of 1 GB) to enable them to process replicated transactions from the active copies. Even after increasing the number of local database copies to eight, you are still using only 4.8 GB of the 8 GB that Exchange assigns to the ESE cache to run four active and four passive copies. With this amount of free cache, the server is easily able to accommodate a scenario in which all eight of the mounted databases are activated. As you'll see in just a little while, you can make better use of the available memory by adjusting the value of the *MaximumActiveDatabases property for a server.*

Given that most Mailbox servers are now being configured with amounts of memory that would have been deemed scandalously over the top just a few years ago, a server equipped with 96 GB of memory is no longer unusual. On such a server, Exchange will reserve 24 GB for the ESE cache. If the server mounted 50 active databases, each would be allowed to have (24 GB/50) or approximately 490 MB of ESE cache.

A more likely scenario is that the mix of databases on a large server would be divided 50-50 active–passive. When 25 active copies and 25 passive copies are supported on a server, it's unlikely that more than a few of the passive copies will need to be activated at any one time. You can therefore use your knowledge about the operational environment to increase the amount of cache available to each active database copy by setting the value of MaximumActiveDatabases to restrict the number of databases Exchange can activate on a server. ESE now uses a different calculation to determine the Max Cache Target. It is:

Divider = (MaximumActiveDatabases × *0.8) + (number of databases* × *0.2)*

Max Cache Target Per Worker = (Total ESE cache allocated from server memory/Divider)

For instance, if you set the MaximumActiveDatabases to 30 for the server in question, the calculation is now:

Divider = (30 × 0.8) + (50 × 0.2) = 34

Max Cache Target Per Worker = (24 GB/34) = 723 MB

Overall, if the server mounts 25 active databases and 25 passive databases, the cache usage will be (25 × 723 MB) + (25 × (723 MB/5)) = 21,690 MB, or 21.18 GB. With this workload, the Store uses approximately 22 percent of the available 96 GB of system memory.

The increase in cache size from 490 MB to 723 MB obviously allows the worker process for each database to hold many more pages in memory and should therefore aid performance because the cache is more efficient if it can hold more data in memory. The additional cache can be assigned because Exchange does not allocate memory from the total cache to

passive copies that do not need to be activated, which is usual in a DAG in which multiple database copies are distributed across multiple servers.

A computer and its operating system will always be constrained in terms of available resources. On memory-scarce servers, it might not be possible to assign the amount of memory that the Store deems desirable for the worker processes. When insufficient system memory is available to allocate 25 percent to the ESE cache, the server moves pages for the least accessed memory to disk, and the worker processes are likely to follow suit and begin to page from disk. When a significant portion of the cache a worker process uses is on disk, ESE logs event 906 (Figure 8-3) to advise that it is having to use disk to store its buffers rather than maintaining them in memory. This is not an issue if the condition is temporary and caused by a peak in demand for memory from other processes or if you run Exchange on a server that is known to be underpowered, as in the case of a small, virtualized test server. However, it is a problem that will affect overall system performance that should be dealt with if event 906 appears an ongoing basis.

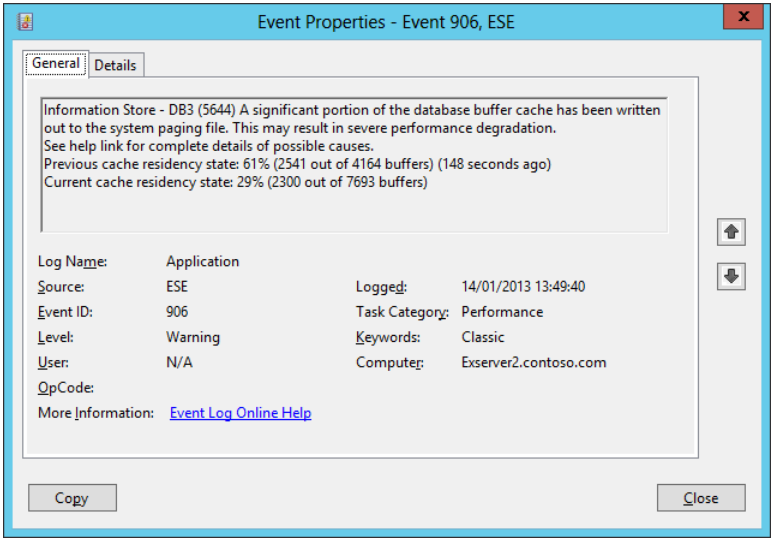


Figure 8-3 Event logged when memory is short on an Exchange 2013 server

On smaller Exchange servers that are relatively underequipped with memory yet expected to run Exchange and other applications, it is possible to constrain the total memory allocated to the ESE cache by using ADSIEdit to adjust the value of the *msExchESEParamCacheSizeMax* attribute for the Information Store object on the server so that the Store attempts to use less than 25 percent of available memory. This technique has been used to trim memory usage on Exchange servers since Exchange 2000 and is documented in *http://support.microsoft.com/?kbid=266768*. However, because Exchange 2013 uses 32-KB pages, you should use 32 KB as the divisor to calculate the number

of pages to assign to the cache and not 4 KB as specified in the article. For example, to constrain Exchange to use 1 GB for the ESE cache, the value is calculated as 1,024,000 KB/32 KB = 32,000. To complete the process, the same value should be set for the *msExchESEparamCacheSizeMin* attribute.

Over and above the ESE cache, the Store uses another set of memory to handle the general overhead to manage activity such as database transition (roughly 100 MB in the *Microsoft.Exchange.Store.Service* process) and to create and cache objects such as address book information, folder hierarchy, and the permissions used to execute business logic in the worker processes. The amount of memory used for this processing varies with the number of active and passive databases on a server and the number of client connections to each database. As a rule of thumb, the worker process for each active database will consume a private working set for its overhead of between 300 MB and 400 MB, whereas the worker process for a passive database will use about 80 MB. In total, this is more overhead than seen on an Exchange 2010 server, but the purpose of using memory is always to avoid going to disk for expensive I/O operations; this is another example of how Exchange continues to trade memory to reduce its I/O footprint.

Two other factors differ in the way Exchange 2013 Mailbox servers consume memory when compared to previous versions. First, the Search Foundation processes that index the content held in mailbox databases consume much more memory than does the equivalent MSSearch service on an Exchange 2010 or Exchange 2007 server. Four SharePoint Search component processes are active on Exchange 2013 Mailbox servers and are visible in Figure 8-2. Sometimes called *noderunner* processes after the executable name, the four processes perform the following functions:

- Content indexing
- Content processing
- Query handling
- Administration and orchestration

Overall, these processes can consume a large amount of memory on a Mailbox server, with between 10 percent and 12 percent of available memory deemed normal. The exact amount used depends on the number and content of the items stored in mailboxes, but a large Mailbox server hosting tens of millions of items across 20 mounted active databases might use a cumulative total of 10 GB of memory for content indexing.

The process that performs indexing (a separate index is maintained per mailbox database) usually consumes the most memory because it caches a substantial amount of information to be able to respond quickly to user queries. The actual amount of memory consumed depends on the number of items in mailboxes, the size (content) of the items, the

properties of the items, and whether the items have attachments that can be indexed. Broadly speaking, after a certain fixed cost in memory is absorbed to start up the search processes, the amount of memory the content indexing process consumes increases in line with the number of items.

INSIDE OUT **How many databases are supported by Exchange 2013?**

The Standard edition of Exchange 2013 supports up to five mounted databases, but the Enterprise edition supports up to 100 mounted databases. The important word here is "mounted," because any version of Exchange 2013 will allow you to create a maximum of 235 databases on a server. However, you will only be able to mount the number of databases permitted by the server license. Test or unlicensed servers have the same five-database restriction as the Standard edition of Exchange 2013.

Managed Availability and the Managed Store

All access to a mailbox is gained by the protocol stack running on a Mailbox server. If a protocol such as Outlook Web App is down, all active databases on a server cannot be accessed through that protocol. The introduction of the Managed Availability system to Exchange 2013 is of huge importance to the Store because it provides a framework to measure the health of components and then takes action to restore service for problematic components.

The Managed Availability worker process (*MSExchangeHWWorker*) is responsible for three major functions that are designed to detect and address protocol failure. Probes operate at regular intervals to assess the health of components by reference to performance counters and other system symptoms. Managed Availability uses health mailboxes in databases to measure the health of databases and the transport system. Monitors observe data from probes, which decide whether the components are operating normally. If a monitor observes an abnormal condition, a responder is activated in an attempt to bring the observed component back to health. Actions can be escalated if attempts to restore health are unsuccessful, up to and including a forced system restart.

Database management

Database management covers many topics. This section starts with a discussion about the basic files that constitute an Exchange mailbox database (see Figure 8-4) and then goes on to discuss some of the finer details about how transaction logs and the other files are used.

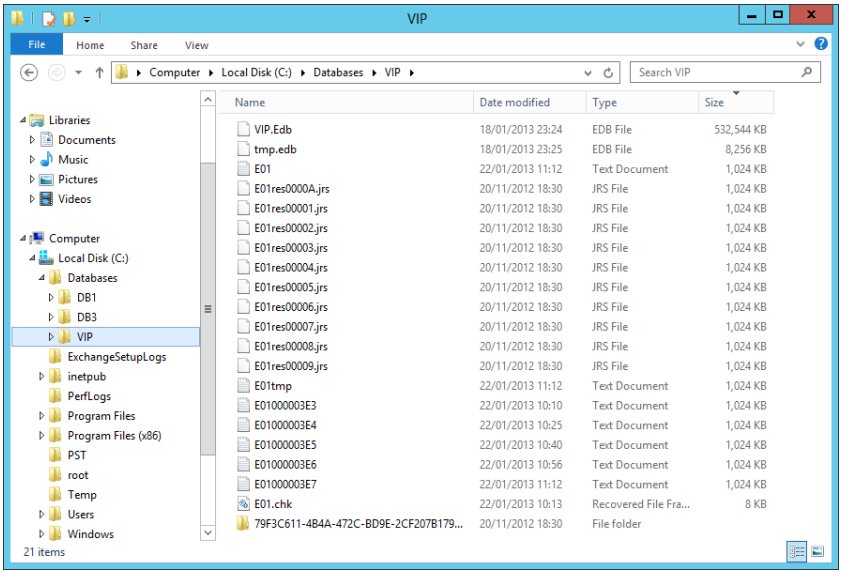


Figure 8-4 Files for a mailbox database

The most important files you see in the directory used to hold an Exchange database are:

- **Database file** For example, Vip.edb. All the tables and data required to provide content to clients are held in the database.
- **Current transaction log** For example, E01.log. This log file is the one into which the Store is currently writing active transactional data.
- **Previous transaction logs** The remainder of the transaction log set that holds previously captured transactions. The Store regularly removes these log files automatically (if circular logging is enabled) or following a successful full backup. The transaction log set forms the basis for replication of data between servers to keep database copies updated within a DAG. In Figure 8-4, the transaction log set goes from E0100003E3 to E0100003E7 (generations 995 to 999, indicating that the current log is generation 1000).
- **Checkpoint file** For example, E01.chk. The checkpoint file tracks the progress of the Store in writing logged transactions (those in its cache and captured in transaction logs) into the database.
- **Reserved logs** For example, E01res00001.jrs. Exchange uses these files to provide some additional space for the database to flush transactions into if the need arises.

Some data about active database transactions might be stored in Tmp.edb. This file is usually only a few megabytes in size and is deleted whenever the database is dismounted or when the Information Store service stops. The Store logs transactions for this database in Exxtmp.log (E01tmp.log in this example).

Ever since the introduction of the DAG, databases are not directly associated with servers (except that they are hosted by servers), and database management is performed at the organization level. Figure 8-5 illustrates a typical view of Exchange Administration Center (EAC) for a small organization positioned in the Servers section of the console. Because these databases don't have more than a single copy, you can conclude that they probably don't belong to a DAG.

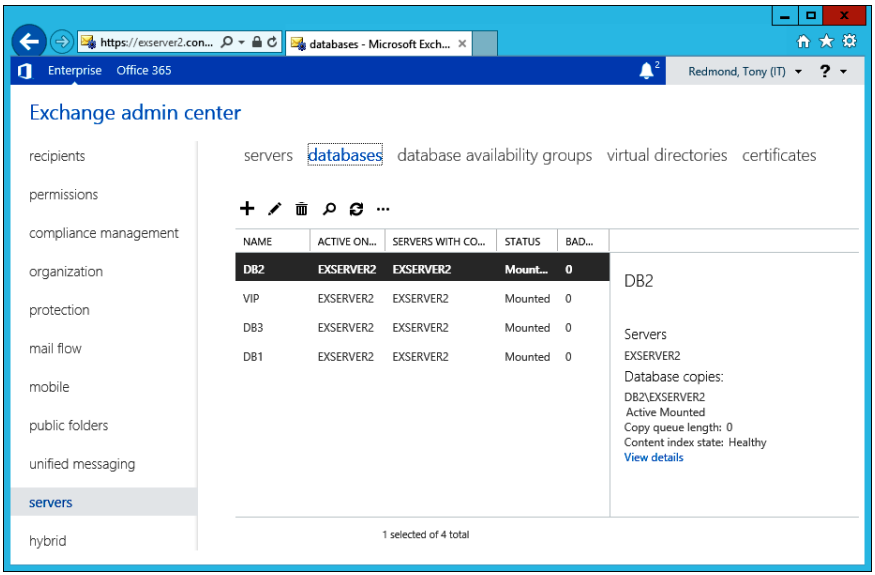


Figure 8-5 Managing databases through the Exchange Administration Center

Unlike EMC, EAC doesn't support filtering of databases to create a specific view. However, EAC does offer a search facility that enables you to look for specific databases. If you need to focus on a certain set of databases, you can do so by applying a naming convention to the databases so that those belonging to specific sites or other logical groupings can be easily identified. Exchange 2013 supports role-based access control (RBAC) scoping to restrict administrative access to specific databases. This is done on a write basis, meaning that EAC does not apply a filter to databases, and administrators are therefore able to view details of all available databases. The RBAC scope comes into effect when an administrator attempts to change a property of a database. If an administrator has the permission to work with the database, EAC will allow him to do so and effect the change. If blocked by an RBAC scope, he will see an error.

Creating new mailbox databases

New databases are created through the Servers node of EAC. It's always a good idea to create new mailbox databases on the server that will initially host them because this avoids the potential for any errors due to network connectivity. Navigate to the Servers section, choose Databases, and then click New (+). Figure 8-6 shows the first step in the creation process when you choose the name of the new database, the server on which Exchange will initially mount the new database, and the location of the database files.

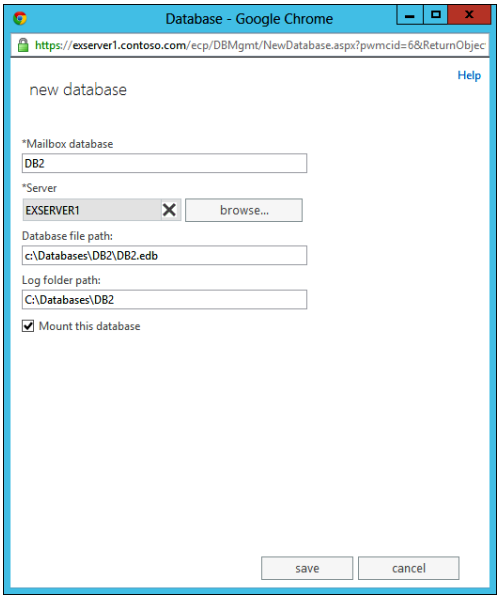


Figure 8-6 Choosing the name and server for a new mailbox database

Locations for both the new database file and the transaction logs must be on a fixed drive (one with a drive letter that does not change). When you create a new mailbox database with EAC, Exchange reads the value of the DataPath property of the server that will host the new database to discover the default database location. You can find this value with a command like this:

```
Get-ExchangeServer -Identity 'ExServer1' | Select Name, DataPath
```

Creating new databases and their transaction logs in a common location under the Exchange root directory is a convenient approach for test servers, but it is not recommended in production. The directories that store mailbox databases should be outside the C:\Program Files\Microsoft\Exchange Server structure because this should be used only for program binaries and other associated files. Best practice is to separate each mailbox database and its transaction logs by placing it in its own directory. Given the size of current

disks and in light of the improved I/O performance of the server, Exchange 2013 allows multiple databases to be accommodated on a single disk, but some still prefer to follow older practice and assign individual disks to databases on the basis that a failure of that disk will only affect a single database. If the directory you specify for a new database does not exist, Exchange will create it.

> **Note**
>
> **You cannot update the DataPath property for a server with EMS because this property is not exposed to the Set-ExchangeServer command. However, you can modify the default path using ADSIEdit by editing the MsExchDataPath attribute for the Exchange server object. Although Microsoft support won't recommend this approach, no side effects appear to be caused by making this change, and if you decide to do this, it is best to be consistent and make the same change on all Mailbox servers.**

In the example shown in Figure 8-6, the transaction logs are placed on the same drive as the database. Except on test systems, this used to be considered bad practice because a hardware failure that affected the database could also affect the transaction logs and potentially remove the ability to recover data into a restored database. Although transaction logs no longer provide the primary protection for database redundancy (unless you run standalone servers), some consideration still needs to be given to where transaction logs are placed and how they are managed.

Of course, if the database is part of a DAG and is protected through replication, things are a little easier because transaction logs are not the sole source for recoverable data, but even with DAG-protected databases, you need to take operational considerations into account before making the final decision of where to place transaction logs. For example, if you think you will move mailboxes on a regular basis, you might want to place transaction logs on a separate drive to ensure that there is sufficient space for the transaction logs that Exchange generates as mailboxes move between databases.

You must mount a new mailbox database after it is created before it can be used. The initial mount of a database prompts the Store to create the underlying files such as the database (.edb), transaction and reserved logs, catalog index, and so on. In Figure 8-6, the Mount This Database check box is selected, which is intended to instruct EAC to issue a *Mount-Database* command after the new database is successfully created. Click Save to give EAC the go-ahead to create the new database.

When the database is created, you'll see a warning message to tell you that the Information Store service should be restarted (Figure 8-7). This indicates that the new ESE cache allocation algorithm is not dynamic and will not factor the presence of the new database into its calculations until after the process is restarted. (See the "Workers, controller, and memory"

section earlier in this chapter.) You can safely leave the restart until you need to create active user mailboxes in the new database.

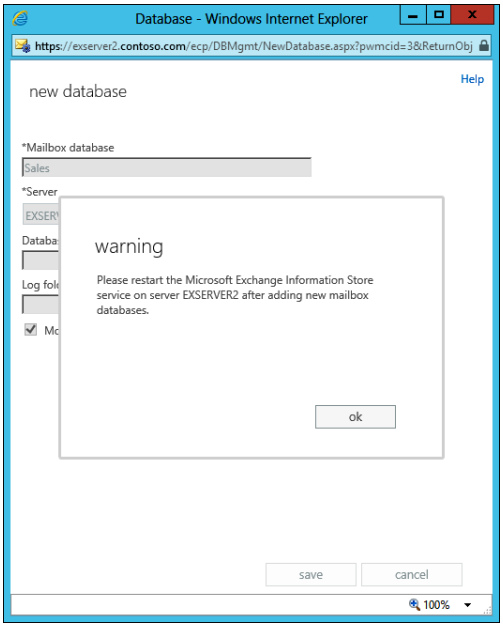


Figure 8-7 Creating a new mailbox database with EAC

An example of typical EMS code that creates a new mailbox database is:

```
New-MailboxDatabase -Server 'ExServer2' -Name 'Sales' -EdbFilePath 'c:\Databases\
Sales.edb' -LogFolderPath 'c:\Databases\Sales'
```

EMS signals the same warning about needing to restart the Information Store service when you run this command to create the new mailbox database. After the database is successfully created, you can mount it and make the database available to host new mailboxes. This step is not automatic, and the new database will remain dismounted until you explicitly mount it, even if the server is restarted to restart Exchange.

```
Mount-Database –Identity 'Sales'
```

Exchange doesn't assign an Offline Address Book (OAB) to a new database. This isn't an issue because Exchange will use the default OAB if no other OAB is assigned. However, it can be a problem if you create multiple OABs for use by different parts of the company and want to assign specific OABs to certain user communities. In such a scenario, it is common to create a mailbox database for each community and assign the correct OAB to the database. You can check the current OAB that is assigned to a database by selecting it and then viewing the Client Settings property tab (Figure 8-8). If the OAB property is left blank,

Exchange uses the default OAB. If this isn't what you want to happen, click Browse to see the full set of OABs known within the organization, select the correct OAB, and save the change.

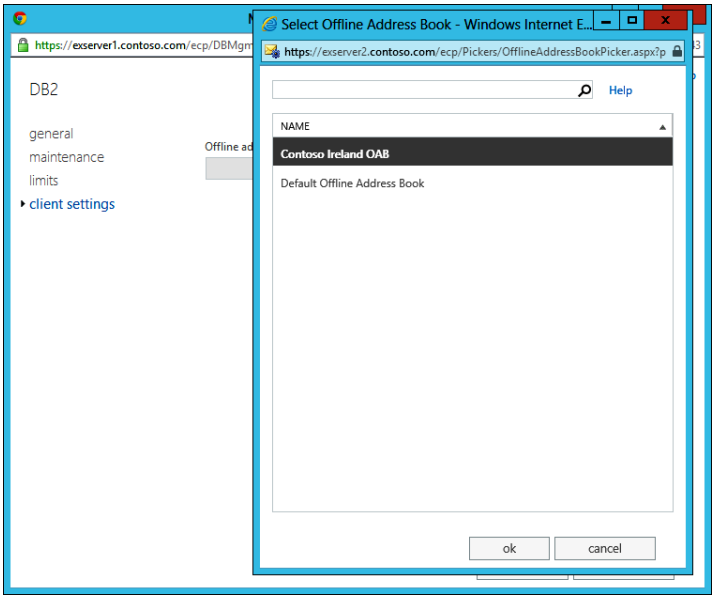


Figure 8-8 Assigning an Offline Address Book to a new mailbox database

In a coexistence situation, in which Exchange 2013 operates alongside older servers, it is important for OABs to be assigned to the databases mounted on the older servers. If this is not done, the danger exists for clients that connect to these databases then to attempt to connect to the Exchange 2013 OAB. You can check for mailbox databases that don't have assigned OABs with the following command. The databases with a blank value returned for the OfflineAddressBook property need to be updated.

```
Get-MailboxDatabase | Select Name, OfflineAddressBook
```

You can update any database that doesn't have an assigned OAB to use the default OAB with this command:

```
Get-MailboxDatabase | Where {$_.OfflineAddressBook -eq $Null} | Set-MailboxDatabase
-OfflineAddressBook '\Default Offline Address Book'
```

New mailbox databases automatically become available as targets for the automatic mailbox provisioning system, meaning that EAC or EMS can immediately create new mailboxes in these databases or that mailboxes can be moved to the database by the Mailbox Replication service (MRS). This might not be what you intended, especially if you create a new database with specific characteristics to support the needs of a particular user

community and want to reserve this database for that community. In this case, you need to block the new database from the automatic provisioning process. See Chapter 5, "Mailbox management," for details about how to exclude or suspend databases from automatic mailbox provisioning.

Updating mailbox databases after installation

When you install an Exchange 2013 Mailbox server, the Setup program creates a default mailbox database for that server and places the new database, its transaction logs, and its search catalog in a directory under the Exchange installation directory. A unique name is assigned to the database to ensure that it will not clash with any other mailbox database within the organization, so you end up with a database placed in a location such as:

C:\Program Files\Microsoft\Exchange Server\V15\Mailbox\Mailbox Database 2136564033\Mailbox Database 2136564033.edb

The location and name of the database is probably not optimal for a production server, so you should consider a number of tasks when the server is up and running properly:

1. Rename the database so that it complies with the organizational naming standard for databases. The name must still be unique, so you have to determine what it should be and then check that the name is unique.

2. Mailbox databases are created with circular logging disabled. You might want to enable circular logging, especially if the database will be used within a DAG. See the "The question of circular logging" section later in this chapter for more information about circular logging.

3. Move the database and transaction log files to a more suitable location away from the disk where the Exchange binaries and other system files are installed.

4. If required, assign an OAB to the database.

Renaming a database is easy. Select the database and view its properties, and then type the new name as shown in Figure 8-9. Alternatively, you can use the Set-MailboxDatabase cmdlet to do the job. For example:

```
Set-MailboxDatabase -Identity 'Mailbox Database 0518033349' -Name 'DB2'
```

You probably will want to move a database to a more suitable location. For example, because all copies of a database share the same path within a DAG, the location assigned to the default database created on a Mailbox server is seldom appropriate. You can either create additional databases and delete the default database or move the default database to a better location.

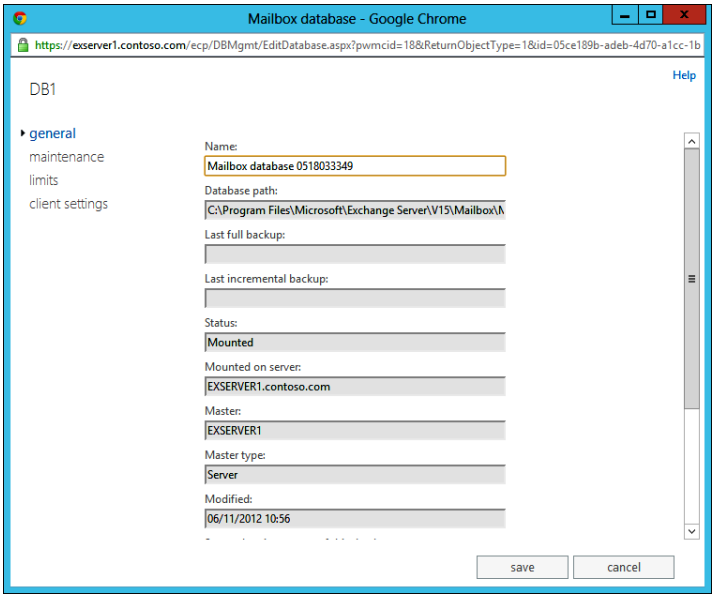


Figure 8-9 Renaming a database

Note

To prevent the potential of interruption caused by network glitches, attempt this operation only when you are connected to the server on which the database is currently mounted. In addition, it's best to settle on a path for a database as soon as possible after it is created and before it becomes part of a DAG because you only have smaller files and a single database copy to deal with. Do not attempt to move a database when a backup is active because the move will fail.

To move the database with EMS, use the Move-DatabasePath command, first dismounting the database if it is online and then mounting it again after the move completes. For example:

```
Dismount-Database –Identity DB4
Move-DatabasePath -Identity 'DB4' -EdbFilePath 'E:\Exchange\DB4\DB4.edb'
-LogFolderPath 'E:\Exchange\DB4'
Mount-Database –Identity DB4
```

The Store has to copy the physical database files from the original location to the new location. Obviously, if you move a database just after it is created, this operation will be very fast because relatively little data must be moved compared to an attempt to move a

500-GB database that has been in production for several months. The fact that a database has moved its location is noted as event 216 from source ESE in the Application Event Log.

If you look at the original location immediately after the database has been moved, you might see that the search metadata files are still there. This is because the Search Foundation is responsible for building the content indexes for the database, and it has to acknowledge the move. The search metadata will be moved to the new location in a short time, and all will be well.

Moving database locations inside a DAG is more complicated because multiple copies might exist for a database, and the copies must be kept in the same location on all servers. See Chapter 9 for more information on this topic.

Backups and permanent removal

You can stop the Store from permanently removing items from databases until a successful backup has been performed. You control this stage of deletion by setting a property of a database (Figure 8-10). The equivalent EMS command is:

```
Set-MailboxDatabase-Identity 'DB3' -RetainDeletedItemsUntilBackup $True
```

When this property is set, items remain in the recoverable items folder until a successful backup is performed, even if they exceed the retention period. You do not want items to expire from the dumpster, be removed by background maintenance, and then be unrecoverable from a backup because of a failure to make a backup or a failure during processing that rendered a backup invalid. It is best practice to set the property on all mailbox databases unless you have a database that is intended to hold only transient information.

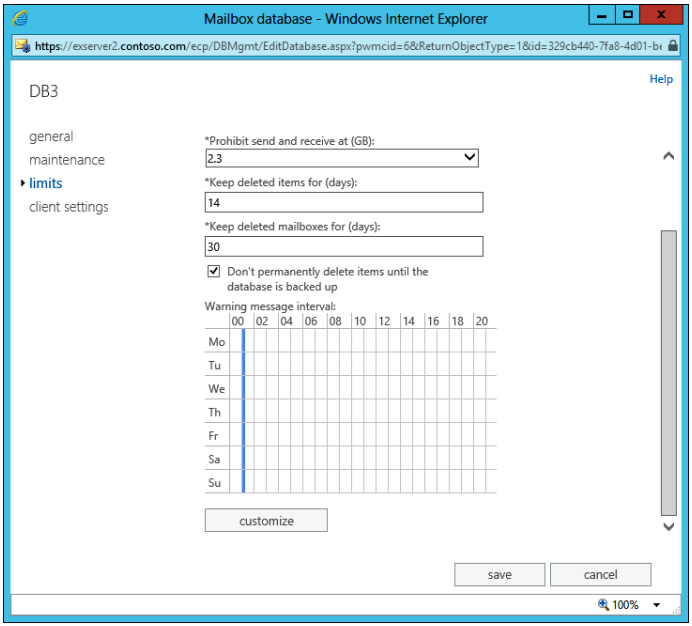


Figure 8-10 Don't delete until the database has been backed up

Removing a database

After you install a new Mailbox server, you might decide that the default mailbox database created on that server is not required. This often happens when you install a new server that will join a DAG and is intended to support copies of databases that are already active in the DAG. In this instance, no real value is to be gained by keeping the default mailbox database, and it is good practice to clean up things by removing the database immediately after the server is installed. Before attempting to remove a database, you should always check that no mailboxes are present, possibly as the result of a new mailbox being created and automatically placed in the database. Exchange won't allow you to remove a database if mailboxes exist in it, and EAC issues a warning if it detects any mailbox. (You can ignore the references to mailbox plans; these are only found in the cloud services.) In any case, it's better to know about potential issues beforehand rather than to find out in the middle of attempting to remove the database, and it's wise to check before attempting to delete a mailbox database that no arbitration, archive mailboxes, or public folder mailboxes exist in the database and that no mailbox move requests are in place that use the database as a source or target.

To check for these possibilities, you can run the Get-Mailbox and Get-MoveRequest cmdlets. For example, the first four of these commands check for mailboxes, arbitration mailboxes, public folder mailboxes, and archive mailboxes. (Note that this command checks for mailboxes on the server where the database is currently mounted to avoid a more general

search across all available mailboxes.) The last two commands look for mailbox move requests associated with the database:

```
Get-Mailbox –Database 'Mailbox Database 031899018'
Get-Mailbox –Database 'Mailbox Database 031899018' –Arbitration
Get-Mailbox –Database 'Mailbox Database 031899018' –PublicFolder
Get-Mailbox –Server ExServer2 | Where {$_.ArchiveDatabase –eq 'Mailbox Database
031899018'} | Select Name
Get-MoveRequest –SourceDatabase 'Mailbox Database 031899018'
Get-MoveRequest –TargetDatabase 'Mailbox Database 031899018'
```


Any mailboxes that are discovered must be moved to another mailbox database before you can remove the database. When all the mailboxes are moved, you can clear the move requests with:

```
Get-MoveRequest –SourceDatabase 'Mailbox Database 031899018' | Remove-MoveRequest
Get-MoveRequest –TargetDatabase 'Mailbox Database 031899018' | Remove-MoveRequest
```

Two calls to Get-MoveRequest are necessary to clear out any requests that use the database as a source or as a target. You can't combine these into a single Get-MoveRequest command. Move requests might linger for a while in a state of Removing after you run the Remove-MoveRequest command. When this happens, it is because the move requests are waiting for MRS to complete the requested clean-up operation. You will have to wait for MRS to do its work before you can delete the database. For more information about working with mailbox move requests, see Chapter 10, "Moving mailboxes."

When everything is ready to allow the deletion to proceed, you can select the mailbox database in EAC and click the wastebasket icon. EAC then prompts you to confirm the action and, if confirmation is given, removes the database.

Removing a database through EAC only deletes the entry for the database from Exchange configuration data in Active Directory. It does not remove the underlying database (.edb), transaction logs, or content index metadata. These files are retained just in case a mistake has happened and you want to bring the database back into Exchange. When you're absolutely sure that the database is no longer required, you can delete the files by using Windows Explorer or File Explorer.

If you prefer, you can run the Remove-MailboxDatabase command instead of going through EAC. Once again, this command only removes information from Active Directory, and you have to clean up the underlying files afterward.

```
Remove-MailboxDatabase –Identity 'Mailbox Database 031899018'
```

Just like creating a new mailbox database, the reduction in database numbers is not incorporated in the ESE cache allocation algorithm until the next time the Information Store service is restarted. This might mean that cache is not managed as effectively as it could be, so it is good to arrange for a restart as soon as convenient.

Transaction logs

Transaction logs are the basic mechanism to capture and record transactions that occur for an ESE database. ESE uses a dual-phase commit for transactions to meet the Atomicity, Consistency, Isolation, and Durability (ACID) test. A transaction is defined as a series of database page modifications that ESE considers a single logical unit. All the modifications must be permanently saved before a transaction is complete and held in the database. For example, the arrival of a new message in a user's inbox is represented by a number of page modifications. The message header might occupy one page, the contents could be held on other pages, and the message might be shared with a number of users, including the originator, so it appears in different folders within the database. ESE has to be able to perform all the updates to modify all pages affected by the transaction to save the transaction. If it does not, ESE discards the transaction.

All transactions that occur in a mailbox database are captured in transaction logs, including system transactions generated because of housekeeping or other background maintenance operations. Because of the asynchronous way the Store saves transactions to the log while batching transactions for efficient committal to the database, it is entirely possible for users to read and write messages in memory without the Store ever going to disk to fetch data.

> **Note**
>
> **Fast, efficient, and secure access to data is the major advantage delivered by the write-ahead logging model Exchange uses, and it is important for every administrator to understand how the model works.**

Log sets

ESE deals with transaction logs as if they formed one very large logical log, which the Store divides into a set of generations to be more convenient to manage. Each log represents a single generation within the log set (also called a *log stream*) and is assigned a sequential generation number for identification purposes. The header of a transaction log contains an identifier (or signature) that associates it with the correct database.

Obviously, a single message that has a number of large attachments can easily span many log files. ESE manages the split of data across the logs automatically and can retrieve data from several logs to form the single message and its attachment if the need arises to replay the transaction into a database. On a busy server, millions of transactions might flow through the logs daily, and it is common to see hundreds if not thousands of logs created each day.

Apart from the activity generated by users, transaction logs capture background and maintenance activity such as the generation of nondelivery reports (NDRs), mailbox moves, the import of mailbox data from other messaging systems, and so on. Any operation that causes data to flow in and out of a database is captured in a transaction log, as is any operation that changes data in place within the database. If you want to estimate the number of transaction logs a server will generate daily, you can use the rule of thumb of 25 MB of data (or 25 logs) per active user per eight-hour working day. The Exchange team's blog asserts that the number of logs per user per day varies from 7 to 42, depending on the average size of message and the number of messages users send and receive, so taking 25 logs is a reasonable middle value that has stood the test of time. Use of circular logging within a DAG reduces the number of transaction logs you have to manage. This topic is discussed in Chapter 9.

INSIDE OUT Transaction logs should be 1 MB

Transaction logs are 1 MB. (Earlier versions use 5 MB files.) The 1 MB size was selected from Exchange 2007 onward to facilitate easy replication between servers and to ensure that Exchange can quickly resolve any data divergence that might occur due to an interruption in replication. Choosing a small log size also makes it less likely that data will be lost if a log file cannot be copied or is corrupted due to a storage or other hardware failure. Any variation from the expected 1 MB file size is an indication that the file might be corrupt for some reason. If you see unexpected log sizes, you should stop the Information Store service and check the event log, database, and disks for errors.

The transaction log set for a database is assigned a two-character prefix followed by an 8-digit hex number representing the generation number of the log. Each log in the set is named by combining the log set prefix with the eight-digit hex number. For instance, the file E0000000F61.log represents generation 3937 of the log set associated with the database that uses the E00 prefix. More than 4 billion log files can be created for a database before the Store has to reuse file names. The prefix for the first database created on a server is E00, the prefix for the second is E01, the third E02, and so on, so the current transaction log for the first database is E00.log. The same log prefix is used for the transaction log set for all database copies within a DAG.

In the older form, public folders have their own databases with a quite separate set of transaction logs. These logs capture the same kind of transactions that occur for items in mailbox databases but are maintained as quite separate entities. Modern public folders are stored in mailboxes in regular mailbox databases, and their transactions are interleaved in

the transactions for all the other mailboxes in the same database. Email activity associated with site mailboxes is also captured in transaction logs, as is the synchronization of metadata between Exchange and SharePoint; the work done with documents held in the underlying SharePoint site is recorded by SharePoint.

Every time the Information Store service starts up, the Store automatically checks the databases as it mounts them to verify that the databases are consistent. A flag in the database header indicates whether the database is consistent or inconsistent, depending on whether the Store was able to shut down the database cleanly the last time it shut down. A clean shutdown flushes all data from the Store's cache and commits any outstanding transactions into the database to make the database consistent.

An inconsistent database is one that has some outstanding transactions that the Store has not yet committed. If the Store detects that a database is inconsistent, it attempts to read the outstanding transactions from the transaction logs to replay the transactions into the database and make the database consistent. This operation is referred to as a soft recovery. Exchange initiates a soft recovery for each database on a server when the Information Store service starts up and mounts the databases (Figure 8-11). Event 302 is logged when the soft recovery completes. You see the same set of events recorded if you dismount and then mount a database. Although event 300 proclaims that the database engine "is initiating recovery steps," in this instance, it really means that Exchange is validating the database just in case any transactions are outstanding. If the Information Store service had previously been shut down cleanly, any waiting transactions will have been flushed from cache to make the databases consistent, and no recovery is necessary, but it's good to check.

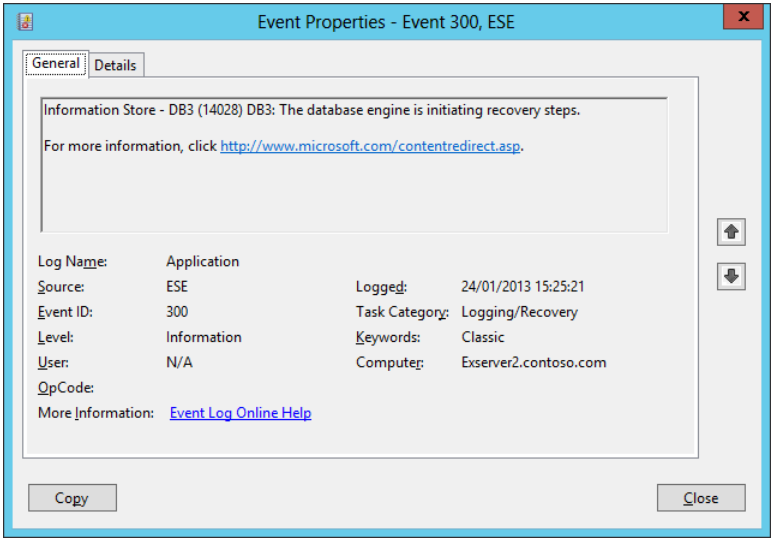


Figure 8-11 ESE logs event 300 as a mailbox database is mounted

When a database has been shut down abnormally, such as when a bug or hardware failure caused a worker process to terminate abruptly or the complete server to halt, Exchange goes through a real soft recovery process to replay all outstanding transactions from the logs to update the databases before clients are allowed to connect. Such recoveries occur much less frequently today than with earlier versions of Exchange. No database property is accessible through EAC or EMS to validate that a database is consistent, so if you really need to know this, you can run the ESEUTIL utility with the /MH parameter to check the database header. Databases that have been dismounted properly will report a state of Clean Shutdown.

Dismounting databases or stopping the Store makes them consistent, albeit inaccessible. Taking a full backup creates a somewhat similar situation in that the database after the backup is consistent. However, because the database is still online, the Store continues to commit transactions immediately after the backup finishes, so the database is only consistent for a very short time.

Transaction logs are tied to their databases in two ways. First, ESE writes a unique identifier (or signature) into each log as it creates the file. The log signature must match the signature of the corresponding database before ESE can use the contents of a log to recover transactions. Second, ESE records the path to the directory where the database is located in the transaction logs. You can find information about identifiers and locations by running the ESEUTIL utility with the /ML parameter to dump the header information from a transaction log as shown in the following sample. Note the signature reported for the transaction log and the identical value reported for the database.

```
Extensible Storage Engine Utilities for Microsoft(R) Exchange Server
Version 15.00
Initiating FILE DUMP mode...

    Base name: e01
    Log file: c:\databases\vip\e01000003fe.log
    lGeneration: 1022 (0x3FE)
        Checkpoint: NOT AVAILABLE
    creation time: 01/25/2013 10:54:51.743
    prev gen time: 01/25/2013 10:47:24.289
    Signature: Create time:11/20/2012 18:30:16.574 Rand:1021831792 Computer:
    Env SystemPath: c:\databases\VIP\
    Env LogFilePath: c:\databases\VIP\
    1 c:\Databases\VIP\VIP.Edb
        dbtime: 1540528 (0-1540528)
        objidLast: 5397
        Signature: Create time:11/20/2012 18:30:16.699 Rand:365102769 Computer:
```

After a client submits a message to the Store, an ESE session that is responsible for the transaction follows a well-defined order to apply the transaction to commit the new message to the Store. The same order is followed for other transactions such as deletes and moves. First, ESE obtains a timestamp using the internal time (called a *dbtime* held in an 8-byte value) maintained in the database header. To modify a page, ESE must calculate a new dbtime based on the current value in the header. When it has calculated the new dbtime, ESE writes the records that make up the transaction into the current transaction log. After this operation completes, the session can modify the appropriate pages in the database. Page modifications occur in an in-memory cache of dirty pages, so ESE might first have to fetch the necessary pages off the on-disk database.

When a client performs an operation that modifies a database page, ESE follows this sequence:

1. The database page is fetched from the ESE in-memory cache. If the page is not cached, ESE fetches it from disk.

2. A log record is generated to describe the page fetch and update the cache.

3. The database page is modified and ESE marks the page as dirty. The page is not written immediately to disk because it might be modified by subsequent transactions. The version store tracks dirty pages to make sure that they are accounted for in case the database session is terminated abnormally. The version store is an internal component that keeps an in-memory list of modifications made to the database that the Store uses for purposes such as transaction rollback or to resolve attempts to apply multiple modifications to the same page.

4. The database page is linked to the record in the cache to prevent ESE from flushing the page to disk before the transaction log record is written. ESE always commits pages to the database after it is sure that the pages have been successfully captured in a transaction log.

5. When the log buffer (1 MB) is full or a commit record is recorded for the transaction, ESE commits the changed page by recording it in the current transaction log. On idle servers, this operation might require a log roll (the creation of a new log generation). If a database is replicated, Exchange can use block replication to share information with servers hosting other copies of the database. Transaction logs continue to be copied. See Chapter 9 for more information about how block replication works.

6. Eventually, the dirty pages are flushed from memory and written into the database.

7. The checkpoint is advanced.

To ensure that data are always protected, it is a cardinal rule for ESE that database writes cannot occur before their transactions are first committed into a transaction log. If you look at the steps that make up a complete transaction, you see that the last step is to commit the transaction to disk. For example, the last step shown in Figure 8-12 is a commit command for the transaction identified as ESE session 8. Other prior steps in session 8 begin a transaction and insert some data to replace an existing page. The commit is a synchronous operation, so no other transaction can occur for that session until the write to disk is complete. Enabling battery-protected write-back caching on the disk that holds the transaction logs improves performance by allowing the write to complete in the controller's memory and thus release the synchronous wait. The controller is then responsible for writing the data to disk.

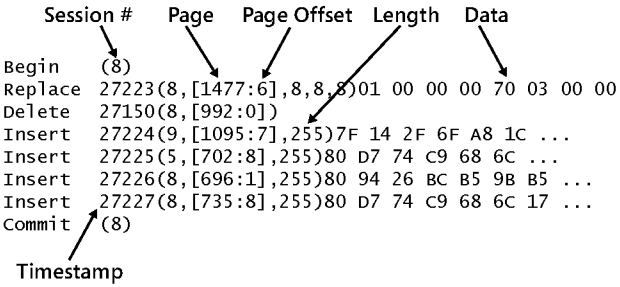


Figure 8-12 Data in a transaction log (from Exchange 2010 Inside Out)

Each log record represents an individual page modification. Transactions start with a Begin transaction record and proceed with the individual operations until the transaction is complete, a Commit transaction record is added, and the transaction is committed to the transaction log. The sequence of transaction records enables ESE to replay complete transactions into databases when it needs to recover data. If ESE reads in a transaction and cannot find a commit record, it considers the transaction incomplete and will not replay it into a database.

The delivery of a single new message causes ESE to modify many pages because many tables might need to be updated. ESE also updates the index for each table, and if the message contains a large attachment, its content will be broken up into several *long value* chunks (literally, large chunks of data that collectively form the item's content), all of which generate log records. All these transactions are captured in log files. If you monitor the log file directory for a database when a large message is delivered to multiple mailboxes in that database, you will observe a spurt of log generation activity. For example, a 1-MB message sent to 10 mailboxes generates at least 10 transaction logs.

However, if you delete an item that has a very large attachment, Exchange only needs to capture the page numbers of the pages that now contain deleted data, and the actual content does not appear in the logs. As you can see from Figure 8-12, the entries in a log

represent low-level physical modifications to the database, and records for different transactions are interspersed throughout a log file. Replaying a transaction is therefore quite a complex matter. Each transaction log contains a sequential list of operations the Store has performed on pages in memory. The log captures details of when a transaction begins, when it is committed, and whether the Store needs to roll it back for some reason.

Each record in the log is of a certain type. Record types include Begin (a transaction is starting), Replace (some data in a page is being updated), Delete (data is removed), Insert (data is added), and Commit. Transactions from multiple sessions are interleaved throughout a transaction log. This means that the Begin record type also identifies the session that performed a transaction. You can think of a session as a thread running within the Store process. The session forms the context within which ESE manages the transaction and all the associated database modifications. Each session could be tied back to a particular client, but the database has no knowledge of individual clients (MAPI or otherwise) because all it sees are the threads that operate on its contents.

Figure 8-12 illustrates how a set of transactions might appear in a log. In this example, the first transaction in session 8 (or thread 8) is replacing a record in the database. Every physical modification to a database is time stamped. ESE uses timestamps later if it has to replay transactions from a particular point in time. The page number and an offset within the page are also recorded. The length of the data to be replaced is then noted and followed with the actual binary data that is inserted into the page. The next transaction is a record delete. The set of insert transactions demonstrates that transactions from multiple sessions are intermingled within a log. Sessions write data into the log file as they process transactions. Any dump of a log file from even a moderately busy server will record transactions from scores of sessions.

When the Store replays transactions from logs to make a database consistent, it has to interpret the contents of all the transaction logs that have accumulated since the last good backup to find the transactions that it requires and then assemble the different records for those transactions from the logs and replay them into the database.

Dirty pages in the cache are flushed to disk every 30 seconds. Pages are not necessarily flushed in order because they might be modified several times during their life in the cache. Along with flushing, ESE calculates a checkpoint to mark the current recovery point so that it knows what transaction logs are required to update a database and make it current if it needs to replay logs in case of an outage. Each transaction log has a generation number that is incremented by 1 as each 1-MB transaction log file is closed off. The log generation number tracks the position and sequencing of transactions in a log file in the log stream. ESE uses the generation numbers to determine the exact log files it needs for any recovery operation. The generation number of the last committed log file is also written into the checkpoint so ESE can calculate the required log files. For example, assume that the current

log file generation is 0x36b0 (14,000), and the log file generation written into the checkpoint file is 0x3683 (13,955). In simple terms, using these values, you can say that:

- Any log files with a generation value less than 0x3683 that contain complete transactions (begin and commit records) have had their transactions written into the database.
- Any log files between the checkpoint value (0x3683) and the current generation (0x36b0) might have some of their transactions written to the database, but because you cannot be sure of what these transactions are, all the transaction logs between 0x3683 and 0x36b0 are required for replay purposes if you have an outage and need to recover the database. The last generation number required is called the waypoint, so you can say that the difference (in generation numbers) between the checkpoint and the waypoint represents the set of transaction logs required for recovery.

Transaction logs hold only 1 MB of data. On heavily loaded systems, when new transaction logs are being created all the time, it is possible for new generations of logs to exist past the waypoint. These logs contain transactions that have not been committed into the database. The transactions could be incomplete (no Commit record exists) or are in the process of being written when the failure occurs. For whatever reason, these logs are not required for recovery purposes. However, if a failure occurs and the database is updated, the transactions in these log files will be ignored and lost.

From this description, you'll realize that transaction logs are critical to the good health of any Exchange server. Although Exchange servers can use block replication to transfer data between them to synchronize database copies, this mechanism only works when networks are very reliable. Transaction logs provide the backstop for Exchange replication, and, without the logs, databases would not be as reliable and robust as they have proved to be in production.

Checkpoint file

The checkpoint file enables the Store to know which transaction logs are required if it needs to perform a recovery. The Store updates the checkpoint file every 30 seconds when it flushes dirty pages into the database. There is always a separate checkpoint file for a set of transaction logs (otherwise known as a log stream), and because Exchange maintains a set of transaction logs per database, it follows that there's a separate checkpoint file per database.

When a database is mounted, Exchange reads the checkpoint file to determine whether it should process any outstanding transaction logs to bring the database completely up to date. This is known as a soft recovery, and it happens every time a database is mounted to ensure that an inconsistent database (one that has outstanding

transactions) is restored to full health even if it has been shut down unexpectedly due to a computer failure or software bug. Unless tens of thousands of logs are necessary to update a database, a set of transaction logs usually doesn't take long to replay. The checkpoint file enables Exchange to quickly identify the point at which the last written transaction occurred and avoid the overhead of examining all available transaction logs and assessing whether the transactions they contain are in the database.

However, if the checkpoint file is lost for some reason, Exchange examines all available transaction logs to determine whether any transactions exist in the logs that have not been committed into the database. If some transaction logs are unavailable due to storage failure or another reason, you have to run the ESEUTIL utility with the /p switch to make the database consistent again and allow Exchange to mount it. In this situation, it is more than likely that users will experience some data loss because there is no way to recover the transactions in the lost logs.

Transaction log checksum

Every transaction log contains a checksum that Exchange validates to ensure that the log data is consistent and valid. Microsoft introduced the checksum to prevent logical corruption from occurring as the Store replays transactions into a database during a recovery process.

Exchange uses a type of sliding-window algorithm called Log Record Checksum (LRCK) to validate checksums for a selected group of records in a log to ensure log integrity. The Store reads and verifies these checksums during backup and recovery operations to ensure that invalid data are not used. If the Store detects invalid data through a checksum failure, it logs a 463 error in the system event log. If the Store fails to read the header of a transaction log and is unable to validate the checksum, it signals error 412 in the Application Event Log. Transaction log failure inevitably leads to data loss because the only way to recover from this error is to restore the last good backup. All the transactions since that backup might be lost.

Transaction log I/O

The Store always writes transactions in sequential order and appends the data to the end of the current transaction log. All of the I/O activity is generated by writes, so it is logical to assume that the disk where the logs are located must be capable of supporting a reasonably heavy I/O write load. In comparison, the disks where the databases are located experience read and write activity as users access items held in their mailboxes.

On large servers, the classic approach to managing the I/O activity generated by transaction logs is by placing the logs on a dedicated drive or LUN. This solves two problems. First, the size of the disk (today, usually 2 TB or greater) means that free space should always be available. If you accumulate 500 GB of logs (500,000 individual log files), it means that your server is under a tremendous load, that you're using a lagged database copy, or that the Store has not been able to truncate the logs for some reason (for example, no recent successful full backup). Second, in all but extreme circumstances, a dedicated drive is capable of handling the I/O load generated by transaction logs.

Arguing against the classic approach is the fact that the Exchange developers now favor placing transaction logs on the same volume as their database. This would have been heresy in the past because of I/O concerns and the perceived need to keep logs and database separated so that a disk failure would not affect both. The reason the developers have changed tack is simple. First, disks are getting larger and are capable of holding both database and logs without ever running out of space. In fact, individual disks are now so large that they can host multiple databases. Second, the advent of multiple database copies means that losing a disk is a lot less serious than it was in the past. Indeed, database auto-reseed now allows a new database copy to be created without administrator intervention. Third, the dramatic and continued reduction in I/O generated by Exchange means that single disks are capable of handling demand. In fact, the continuing improvement in the I/O profile of Exchange makes fast disks, including solid-state drives (SSDs), somewhat of an unnecessary luxury except in very specific circumstances in which their deployment is justified by requirements that are verified through testing. All of this goes to prove that there is no one good answer to the question of how to lay out databases and transaction logs across available storage. Instead, the flexibility of Exchange is seen in that multiple approaches work.

INSIDE OUT Applying some logic

Having a dedicated drive for log files is a luxury you might not be able to afford. But the logic applied to justify the drive—reserve enough space for log file growth and keep an eye on I/O activity—should be remembered when you decide where the logs should be stored on your system. For example, it's a bad idea to locate the logs on the same drive as other hot files such as a Windows page file.

As noted in the earlier discussion about creating a new mailbox database, another classic best practice is never to place the logs on the same drive as a database. The logic is that keeping the logs with their database might seem like a good idea, but like keeping all your eggs in one basket, if a problem afflicts the disk holding the database, the same problem will strike down the transaction logs, and you will lose data. This advice

still holds for databases that are not replicated, but it becomes less valuable the more database copies you have within a DAG. If you only have two database copies, database and log isolation across different disks is still a good idea because your tolerance for failure is limited to one copy. However, if you have three or more database copies, you can collocate databases and transaction logs because any failure will still result in the availability of at least two databases.

The question of circular logging

Circular logging means that the Store generates a limited number of transaction logs by reusing logs as their contents are committed into the database. This feature was originally introduced into Exchange as a mechanism to restrict disk usage on low-end servers because storage was an expensive commodity in 1996. Circular logging is also common on test servers and edge transport servers, where data are transient and don't usually need to be captured for a permanent record. From an operation perspective, the Store uses a relatively small set of between four and eight logs to capture transactions at low periods of demand. However, this figure depends on system load and how quickly the Store can commit transactions to the database—and whether the database is replicated. It is quite common to see thousands of transaction logs created for a database that is under heavy load (for example, when many mailboxes are being moved into the database), and these logs might persist for some time, especially if they are being replicated to other servers. Eventually, as system load decreases and replication occurs, the set of logs used for circular logging is truncated back to a minimal set and kept at this level until system load grows again.

In the past, circular logging was never enabled for production databases. The introduction of the DAG and the ability to have multiple copies for databases challenged the need to retain transaction logs until backups were complete. Circular logging within a DAG is explored in more detail in Chapter 9.

Note

By default, new mailbox databases do not use circular logging, so if you want to enable circular logging to save disk space, you have to update the database properties after it is created. The new logging behavior becomes effective the next time the database is mounted. Don't enable circular logging if you plan to replicate the database within a DAG because you'll have to disable it before you can transition the database to have copies.

To enable circular logging on a mailbox database, select the database with EAC, view its maintenance property page, and then select the Enable Circular Logging check box. The equivalent EMS command is:

```
Set-MailboxDatabase –Identity 'DB1' –CircularLoggingEnabled $True
```

After you enable circular logging for a nonreplicated database, you must dismount and mount the database to make the setting effective. These operations enable the Store to clear its cache of transactions for the database and close off the current transaction log stream in preparation for switching mode. Use the following commands to dismount and mount a database in EMS:

```
Dismount-Database –Identity DB1 | Mount-Database DB1
```

To dismount and mount a database from EAC, make sure that the correct database is selected, and then click the ellipses (...) button to reveal the option to change database mount status.

Circular logging will not prevent disk space exhaustion, and disks still need to be monitored carefully to ensure that sufficient storage is available to accommodate logs even if they have to be retained for an unexpected period.

When circular logging is disabled, the Store continuously creates new transaction logs as it fills the current log with data. The Store performs the following steps to create a new log and switch it to become the current log. First, the Store advances the checkpoint in the checkpoint file to indicate that it has committed the transactions in the oldest log into the database. Next, the Store creates a temporary file called <*database prefix*>*tmp.log*. For example, E00tmp.log is created for the first database created on a server. The Store switches this file into use to become the current log when it closes off the current log to ensure that transactions can be continually written into a log without pausing.

> **Note**
>
> **If the Store fails to create the temporary file, it usually means that no more disk space is available on the disk that holds the transaction logs. In this case, the Store proceeds to an orderly shutdown of the Information Store service and uses the two reserved log files to hold transaction data that it flushes from its cache during the shutdown.**

When the temporary log file is created, the Store initializes its log header with the generation number, database signature, and timestamp information. When the current log file is full, the Store stops writing transactional data and renames the current log file to incorporate the generation number in its file name (for example, E000007E71.log). The Store then renames the temporary log file to be E00.log (or whatever the prefix is for the database)

to make it the current transaction log and resumes writing transaction data into the current log file, flushing any transactions that have accumulated in the short time required to switch logs.

The Store continues with the process of creating the temporary log file and switching it to take the place of the current log file until the Information Store service shuts down. The number of logs created on a server varies according to message traffic and other activity, such as mailbox moves. Busy and large servers can generate tens of gigabytes of transaction logs daily, so this is obviously an important factor to take into account when you size storage for servers. It's also important to include a healthy margin to account for peaks in demand, including those that might occur due to software bugs such as the extreme growth in transaction logs caused by malfunctioning Apple iOS ActiveSync clients in 2012. The space occupied by transaction logs is released when the files are truncated following a successful full backup. The logic here is that you do not need the transaction logs anymore if you have a full backup. The database is consistent after the backup, and you have a backup copy if a problem arises. The backup copy also contains the transaction logs that are necessary for the Store to make the database consistent if you need to restore it.

Reserved logs

Reserved logs provide the Store with some emergency storage in case it is unable to create new transaction logs for any reason. The reserved logs are called databasePrefixres00001.jrs through databasePrefixres0000A.jrs. For example, E00res00001.jrs is the first reserved log file for the first database created on a server.

The only time I have seen these files used is when space is exhausted on the disk that holds the transaction logs. Running out of disk space was a reasonably common occurrence in the early days of Exchange because disks were small and expensive, but such an event is much less common today. Most administrators now monitor free disk space on all disks Exchange uses and take action to release space if less than 1 GB is available.

You should never get close to the point at which the reserved logs are called into play. First, you should ensure that all the disks that hold mailbox databases or transaction logs are sufficiently large to accommodate normal operation and any expected growth over the next year or so. The exact size of the drives you need to use differs from organization to organization and should be dictated by actual data for database growth over time. Second, you should monitor available space on all relevant disks on an ongoing basis. Third, if your monitoring fails or you haven't provided sufficient space to account for an expected spurt in growth, Exchange includes a self-protection mechanism (backpressure) that prevents the transport service from delivering new messages to the mailboxes in a database if the available space on a disk that holds a mailbox database or transaction log falls below 1 GB. (For instance, a database and its set of transaction logs might grow considerably if you move many mailboxes to the database.) This step prevents new messages from causing a

problem, but it might not stop other procedures (such as mailbox moves) from grabbing some more space. Exchange releases the block on new message deliveries when the available space on the disk grows past 1.5 GB. Exchange could release the additional space as a result of removing transaction logs after their contents are committed into the database or through administrator intervention.

INSIDE OUT **Less than 1 GB of free space requires action**

The fact that free space is reduced to less than 1 GB is a danger signal that warrants immediate and proactive attention from the administrator to take the necessary steps to free space on the disk so the situation cannot recur.

Background maintenance

All databases have internal structures that require some degree of maintenance to ensure logical consistency, and Exchange is no different. You can take a database offline to rebuild it with the ESEUTIL utility, but a database rebuild requires you to deprive users of access to their mailboxes. Microsoft designed Exchange to be highly available with as little downtime as possible, and online maintenance is necessary to keep the databases in good order while allowing users to continue working.

The maintenance Exchange performs can be grouped into the following categories:

- **Miscellaneous content maintenance** These tasks are performed during the maintenance schedule assigned as a property of mailbox databases, typically from 1 A.M. to 4 A.M. nightly. Given the speed of modern servers, these tasks usually complete quickly.
- **Checksum scans** A checksum is determined for every page in mailbox databases to detect possible physical corruption in an Online Database Scanning process. This process usually occurs on a 24/7 basis but can be assigned a specific schedule.
- **Defragmentation** Exchange attempts to reorder pages in mailbox databases so that data that belong together are stored sequentially to make I/O more efficient. This process runs on a 24/7 basis.
- **Compaction** Exchange attempts to free space within mailbox databases by rearranging data into as few pages as possible. This process runs on a 24/7 basis.

- **Page zeroing** Exchange writes zeros into deleted pages to obscure the data these pages once held. This process runs on a 24/7 basis.

Exchange 2013 uses two work cycle–based agents called StoreMaintenance and StoreDirectoryServiceMaintenance to perform Store maintenance operations in the background. These operations happen constantly and are not constrained by an assigned maintenance window.

Database checksums

To ensure that no physical page corruption exists in its databases, Exchange reads sequential 256-KB chunks of data at a rate of approximately 1 MB per second (per database) and then verifies the checksum for each of the eight pages in the chunk. This process is called Online Database or ESE Scanning, and its goal is to make sure that every page in every database (active and passive) is scanned on a server at least once weekly. If the scan takes longer than seven days to complete, ESE logs event 733 in the Application Event Log to notify that processing is taking longer than expected and then logs event 735 when the scan completes.

The default for Exchange 2013 is to run online scans on a 24/7 basis. On most systems, online scanning deals with pages at a rate of roughly 3.5 GB/hour (depending on server load, speed, and the storage system), so it is relatively easy for Exchange to go through every page in a database at least once every three or four days. These values are throttled back in Exchange 2013 from the previous rates used in Exchange 2010 (roughly 5 MB/sec) to improve the I/O profile of the Mailbox server.

If you look at the properties of a mailbox database (Figure 8-13), you see that the check box to enable background maintenance is selected. In fact, "background maintenance" is a misnomer because this check box only controls how Exchange performs checksum scans. The property can also be set through EMS by running the Set-MailboxDatabase command:

```
Set-MailboxDatabase –Identity 'DB3' –BackgroundDatabaseMaintenance $True
```

If you opt not to enable 24/7 scanning, Exchange performs checksum scans as the last stage in all the other background tasks that it performs to keep databases healthy, such as removing deleted mailboxes when their retention period expires. These tasks are executed during the nominated maintenance window, which you can also see in Figure 8-13.

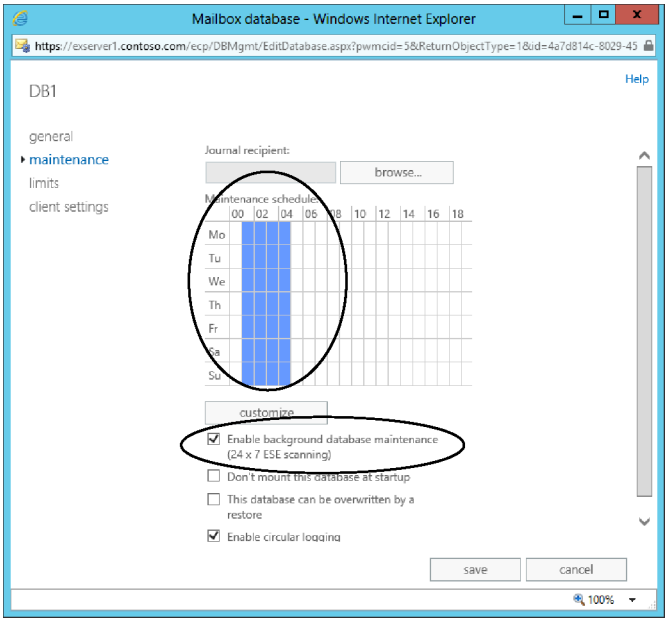


Figure 8-13 Controlling background database maintenance

You can also use the Get-MailboxDatabase command to retrieve the current maintenance schedule for a database. For example:

```
Get-MailboxDatabase –Identity 'DB1' | Format-Table Name, MaintenanceSchedule
–AutoSize
```

Given the format of the schedule, manipulating it by running the Set-MailboxDatabase command could be tricky. It's usually much better to make changes through the graphical interface of EAC to set the maintenance schedule on a database.

If you clear the Enable Background Database Maintenance check box in Figure 8-13, you constrain checksum scans for active databases to whatever maintenance period is assigned to the database (1 A.M. to 4 A.M. in the example shown). Given that other work needs to be done during this time, it's likely that checksum scans would only be performed in the last 150 minutes of the period. This is sufficient for small to medium databases but not recommended for any database that is larger than 500 GB due to the risk that scanning cannot process every page in the database in a reasonable time and so might not discover a corrupt page quickly enough for patching to be possible. Note that online scanning occurs continuously for passive database copies, even if you set up a customized schedule for a database.

All things considered, it is best to leave the default arrangement in place and let Exchange throttle operations to meet the goal of scanning every page in every database at least once weekly.

If a corrupt page is discovered during scanning, Exchange can patch the page from another copy of the database. However, this can only work when multiple copies are available, so it is not a feature available to databases located on servers outside a DAG. See Chapter 9 for more details of how page patching is performed.

INSIDE OUT **So many pages, so little time**

The reason for introducing 24/7 checking is simple. Confining the work to a set period at night is not a problem with small databases because you can be reasonably confident that Exchange can process all the pages in a database during the maintenance period. However, as databases become larger and larger, more pages must be processed. The Store processes pages sequentially from the start of the database to the end and, as the number of pages to be validated grows, it becomes more difficult for the Store to complete the task within the allocated maintenance window, hence the need for Exchange to accommodate a 24/7 processing cycle.

Database defragmentation

Online defragmentation (known as *OLD* or *B+ tree defragmentation*) reorganizes pages within the database so that they are in the most efficient structure. The current method of online defragmentation was first introduced in Exchange 2010 to replace an older and less efficient version and operates on a 24/7 basis. You cannot change a setting to influence how Exchange performs defragmentation because the Store is designed to monitor database operations with an eye to optimization. If the Store detects that requests coming in to read information are for records that are not held sequentially, it defragments that portion of the database to reorganize data into pages so that future requests can be dealt with sequentially. The intention is to minimize the number of random I/O requests the Store has to process and gradually improve the efficiency of the database. In addition, the Store monitors all database tables for free pages, and if it detects that a table has too many free pages, it moves those pages back to the root and makes the pages available for reuse.

You can track the work done to defragment databases with Performance Monitor by using the MsExchange Database/Defragmentation Tasks counters.

Database compaction

Because the nature of email is for information to accumulate steadily in mailboxes over time, it's reasonable to expect that an Exchange mailbox database will grow and grow, which is exactly what happens. An Exchange database will never shrink and return space to the Windows file system unless the database is taken offline and rebuilt using ESEUTIL.

Given that the tendency is for databases to swell over time, it's important for the internal structures of the database to be as efficient as possible in terms of how pages are used to store information. If information is stuffed into pages any old way, databases would grow even faster as disk space is continually requested from Windows to store new items.

Database compaction helps by moving information around within a database so that data are held within the smallest number of pages possible. At the same time, compaction attempts to keep data ordered contiguously so that all the data required to satisfy requests can be retrieved using sequential reads. For example, all the pages needed to store information about item headers might be moved to sequential pages so that Exchange can fetch all the headers for items in a folder using one sequential I/O rather than having to issue multiple smaller I/Os to different points within the database.

Page zeroing

Exchange overwrites the contents of deleted database pages with zeros to ensure that information previously held in those pages cannot be easily retrieved. Many companies that are concerned about the possibility that a rogue administrator might compromise the information held in a mailbox database request this security measure. Given that a rogue administrator has easier methods available to her to access user mailboxes if she desires, it's hard to see her making a copy of a database and then rummaging around in its internals to uncover deep and dark secrets, but it is possible.

In any case, Exchange 2013 overwrites pages with zeros as pages are freed because of items being hard deleted from the database. This process happens continuously and cannot be controlled by an administrator. In addition, during online scanning, Exchange checks for deleted pages that have not yet been zeroed for some reason, such as the Store being overloaded when a page was freed or a bug interfering with normal processing. Online scanning also takes care of zeroing the pages used for deleted tables and indexes because these are not zeroed when they are freed.

Content maintenance tasks

The maintenance tasks discussed so far have focused on raw database structures such as pages. Some of the information held in the database also needs some tender loving care. Clients such as Outlook make extensive use of the Store's ability to generate indexes or views on a dynamic basis. For example, if you decide that you want to view your inbox by

author rather than by date, Outlook asks the Store for this view. If the Store has previously generated the view and has it cached, the response is very quick, but the Store can process a request for a brand-new view quickly, too. Over time, the Store accumulates large numbers of views; each folder in every mailbox can have several views.

Expiry dates control the accumulation of views in the database

It is not desirable to retain all the views because each view occupies space within the database. In addition, users might not require a view after its first use. The Store assigns an expiry time to each view and monitors these data in an internal table called the index-aging table. When background maintenance runs, the Store scans the index-aging table to discover views that are older than 40 days and removes any view that has expired. Of course, if a client accesses a view, the Store resets its expiry time.

Another task is to perform *tombstone maintenance*. The Store maintains a list of deleted messages for each folder and a list of tombstones to indicate that a message was deleted. After successful replication, the Store can clear the entries from the tombstone list. Background maintenance ensures that the lists are accurate.

During background maintenance, the Store examines every message that has the deleted flag set to determine whether its retention period has expired. (If enough time is available in the maintenance window, the Store processes the entire contents of the deleted items cache.) If this is true, the Store removes the message from the Store (a hard delete) and makes the pages that hold the message available for reuse. You can set retention periods on a per-mailbox, per-database, or per-public-folder basis and can control these settings through EAC or EMS. The Store also checks for deleted mailboxes and removes any that have expired.

Corrupt item detection and isolation

Bad items or messages that contain some corruption such as a malformed header have a nasty habit of causing extreme difficulty for an Exchange administrator. Exchange 2007 introduced the ability to detect and isolate corrupt messages in the transport system. Essentially, if the transport system thinks that a message is corrupt and likely to cause problems such as provoking a software crash, it considers the message poison and isolates it in a queue the administrator can examine to decide whether the message really is a problem. Exchange can identify and then isolate or quarantine problematic mailboxes to address when a mailbox that contains one or more corrupt items can cause the Information Store process to terminate abnormally when a client attempts to access a corrupt item.

As you know, the Information Store process is multithreaded. The threads are connected to mailboxes to do work, and the Store detects a problem mailbox when:

- The Store has more than five threads connected to a mailbox that have been frozen for 60 seconds or more.
- One of the threads connected to a mailbox crashes abnormally. The nature of software is that crashes do occur, so Exchange allows for up to three crashes within a two-hour period before it regards a mailbox to have exceeded the acceptable threshold for crashes.

When the Store identifies a problem mailbox, it writes details about the mailbox into the system registry. These details include the mailbox globally unique identifier (GUID) to identify the mailbox, the time a crash occurred, and the number of times the problem has occurred. This information is stored under the root for the mailbox database that contains the mailbox in two entries written for a problem mailbox. The first entry captures the last crash time, the second the count of crashes.

- HKLM\System\CurrentControlSet\Services\MSExchangeIS\Server Name\Private-{Database GUID}\QuarantinedMailboxes\{Mailbox GUID}\LastCrashTime
- HKLM\System\CurrentControlSet\Services\MSExchangeIS\Server Name\Private-{Database GUID}\QuarantinedMailboxes\{Mailbox GUID}\CrashCount

When a mailbox exceeds the crash threshold or the Store considers that threads are being blocked by corrupt content, the Store puts the mailbox into quarantine. This means that normal client interaction, including access by Exchange mailbox assistants, is blocked. If you check the Application Event Log, you'll find a 10018 event from MSExchangeIS logged for each problem mailbox. The text of the event tells you the distinguished name of the problem mailbox and looks like this:

```
The mailbox for user /o=TonyR/ou=Exchange Administrative Group (FYDIBOHF23SPDLT)/
cn=Recipients/cn=TRedmond has been quarantined. Access to this mailbox will be
restricted to administrative logons for the next 6 hours.
```

A quarantined mailbox cannot be logged on to with Outlook or Outlook Web App, even by an administrator who has Full Access permission for the mailbox. In addition, Exchange ceases delivering messages to it. (The messages remain on the transport queues.) Apart from opening it with MFCMAPI, the only operation you can perform against a quarantined mailbox is to move it to another database. By default, Exchange maintains a mailbox in quarantine for up to six hours (21,600 seconds). You can change the quarantine period and the threshold for crashes by entering two new registry keys. These are specific to a mailbox database and are read when the database is mounted. The first key is in seconds and sets the quarantine period; the second sets the threshold for thread crashes Exchange will tolerate before considering a mailbox to contain some corrupt items.

- HKLM\System\CurrentControlSet\Services\MSExchangeIS\Server Name\Private-{Database GUID}\QuarantinedMailboxes}\MailboxQuarantineDurationInSeconds
- HKLM\System\CurrentControlSet\Services\MSExchangeIS\Server Name\Private-{Database GUID}\QuarantinedMailboxes}\MailboxQuarantineCrashThreshold

INSIDE OUT A cmdlet for identifying a quarantined mailbox

An administrator will probably not be aware of a quarantined mailbox until a user complains that he cannot access his mailbox, in which case, a check against the registry might reveal that the mailbox is in quarantine. The Get-MailboxStatistics cmdlet can be used to check for quarantined mailboxes. This command lists all mailboxes currently in quarantine:

```
Get-MailboxStatistics | Where {$_.IsQuarantine –eq $True}
```

You can access a quarantined mailbox with a utility such as MFCMAPI and use it to remove any corrupt items you detect. However, determining that an item is corrupt requires a deep knowledge of MAPI and how it expects data to be organized and is probably far past the capabilities of the typical Exchange administrator. To open the problematic mailbox with MFCMAPI, you need to:

1. Create a MAPI profile that points to the mailbox.
2. Run MFCMAPI, select Logon from the Session menu, and select the MAPI profile you created to force MFCMAPI to connect to the mailbox.
3. Select Default Store and then choose Open Default Message Store from the MDB menu. This forces MFCMAPI to display a dialog box to enable you to select the flags you want to use for the session. In this case, you want to use administrative privilege to open the mailbox, so it's sufficient to input OPENSTORE_USE_ADMIN_PRIVILEGE or 0x0000001 as shown in Figure 8-14.

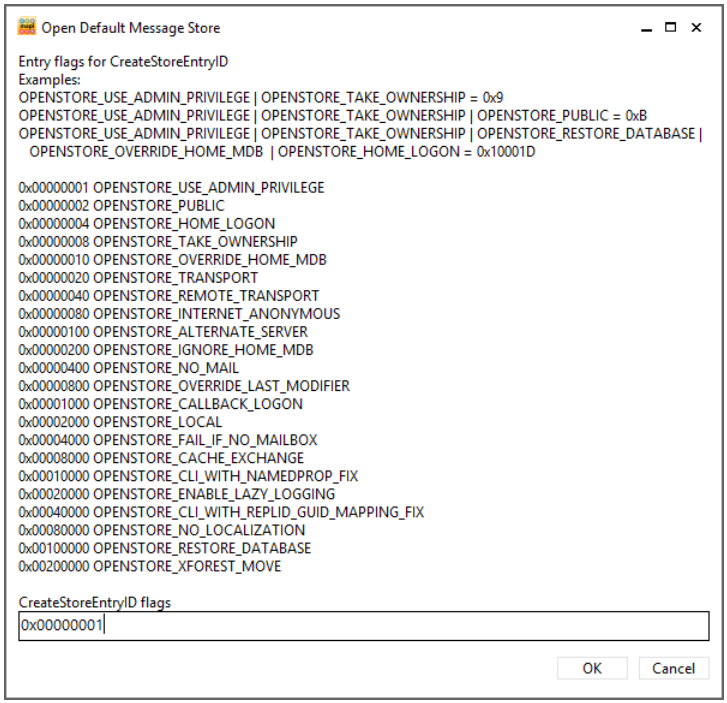


Figure 8-14 Specifying flags to use with an MFCMAPI session

4. Click OK to continue. MFCMAPI opens the folder tree so you can now navigate within the mailbox.

Even after gaining access to a mailbox with MFCMAPI, it will be difficult to find the items about which the Store is complaining. It's often a more pragmatic and therefore better solution to move quarantined mailboxes to a different database. Before you attempt to move a quarantined mailbox, you should run the Enable-QuarantineMailbox cmdlet to release it from quarantine. Because the condition that caused the mailbox to be quarantined probably still exists, you should not allow the user to access his mailbox after it is released. Instead, immediately move the mailbox either individually by running the New-MoveRequest cmdlet or in a group by including the mailbox in a migration batch (which could be a single-mailbox migration batch). To release the mailbox before the move, use:

```
Enable-QuarantineMailbox –Identity 'Tony Redmond'
```

Two good things can result from moving the mailbox. First, you can isolate the mailbox to a database where crashing will not affect service to other users. Second, the act of moving the mailbox can remove potential problems such as consolidating the named properties items in the mailbox use or dropping corrupt (bad) items as the Mailbox Replication service

(MRS) transfers items from the source to the target mailbox. See Chapter 10 for more information about how mailbox moves are performed.

The Store keeps the problem mailbox in quarantine until the quarantine duration elapses and then releases it by removing the registry keys for last crash time and crash count. The Store also removes the registry keys if they are more than two hours old when a database is mounted. The logic here is that dismounting and remounting the database might well have cleared any lurking issue that caused problems with Store threads. Of course, releasing a mailbox that contains corrupt data from quarantine could set off a new cycle of crashes, leading to further quarantine.

TROUBLESHOOTING

How to deal with a mailbox stuck in a quarantine cycle

If a mailbox goes into a cycle of constant quarantining, it's an indication that some deep problem exists that you might be able to resolve by moving the mailbox to another database. In this instance, you instruct Exchange to ignore any corrupt items it detects during the move operation so that the mailbox is as clean as possible after the move. If this doesn't work, and the mailbox continues to cause problems, the only route open to you is to export as much information as possible to a PST and then delete the mailbox. If the export operation fails because of corruption, you might be able to recover data through a database restore.

The Store periodically monitors mailboxes that have caused thread crashes. If no further crashes occur in a two-hour period, the Store concludes that whatever problem existed was transient and removes the mailbox from the list of quarantined mailboxes in the system registry. Obviously, an administrator can decide to remove a mailbox from the list, too, by manually deleting the entry in the system registry. This is not recommended unless you know that the underlying problem no longer exists or you want to restore client access to a mailbox to enable the user to recover as much data as she can from the mailbox before you delete and re-create the mailbox.

If you don't see the QuarantinedMailboxes entry under a database root in the registry, you know that the Store has never detected a problem mailbox for that database. Note that Exchange replicates mailbox quarantine data from the registry to other servers in a DAG through the failover cluster service to ensure that a failover retains knowledge of problem mailboxes.

Protection against high latency

Store databases can suffer from high latency for a number of reasons. A disk might be failing and responding to requests in seconds rather than milliseconds, or the server might be under extreme demand for CPU or suffering from memory shortage. Then it's possible that a mailbox could be causing an issue from a client-side bug or other problem.

Exchange includes a script, Troubleshoot-DatabaseLatency.ps1, which you can run if a server seems to be under strain to check whether it can determine the cause of excessive latency. For example, to check the latency of a database called DB1, the syntax of the script is:

```
Troubleshoot-DatabaseLatency.ps1 –MailboxDatabaseName "DB1" -LatencyThreshold 60
-TimeInServerThreshold 10 –Quarantine:$True –MonitoringContext:$True
```

The parameters that can be supplied to the script are as follows:

- *LatencyThreshold* is the threshold for latency and is 70 milliseconds by default. Normally, you won't need to amend this threshold unless you want to experiment to see what results are returned in your environment under different conditions or you are advised to do so by Microsoft PSS as part of a troubleshooting exercise.

- *TimeInServerThreshold* indicates the number of seconds of work per minute that can be performed on behalf of a single mailbox before the mailbox is considered hazardous to the health of the database. The number of seconds of work is measured by aggregating the time spent inside the Store (CPU, waiting for I/O and other operations) by all threads working on behalf of the mailbox over the period reported by the Get-StoreUsageStatistics command. The number of seconds of work per minute is calculated by dividing the aggregate number by the measurement period the Get-StoreUsageStatistics command uses (10 minutes). By default, the threshold is 60 seconds.

- *Quarantine* should be set to $True if you want mailboxes that are observed to be the cause of high server demand to be placed into quarantine.

- *MonitoringContext* determines whether the script is running on a server that is monitored by a system such as Microsoft System Center Operations Manager (SCOM). If this is the case, set the property to $True to force the script to write events to the Application Event Log.

If the problem is caused by a mailbox that occupies more than one CPU thread over a 10-minute period, Exchange puts the mailbox in quarantine and keeps it there for the default quarantine period unless it is moved to a different database. The same mechanism is used as for corrupt-item detection (see earlier) with the crash count set to three to force immediate quarantine. A mailbox is also put into quarantine if it peaks close to the 1 thread/10 minute threshold three times in a two-hour period.

Protection against excessive database or log growth

Many situations have been observed over the years when a Mailbox server seems to grow the size of a database excessively or generate a very large number of transaction logs. A database usually grows at a rate consistent with the growth of the mailboxes the database holds. The same is true of transaction logs, although you can see a spurt in transaction log generation with no increase in database size if a lot of white space is available in the database that can be filled by new transactions or if other activities, such as deleting a large number of mailboxes, cause a spike in transaction log growth.

For instance, if a database grows at the rate of 100 MB/month and suddenly expands by 1 GB overnight with the attendant increase in transaction logs, it might be due to legitimate activity or a software bug. Legitimate activity includes the movement of many mailboxes into a database or the import of data from PSTs. Known bugs in the past include the submission of messages by Outlook that go into a loop and continually generate new transactions. The problem therefore is to determine whether database growth is due to a problem or to administrative activity. The former might not go away and could need a patch to fix a bug; the latter will subside over time.

Microsoft provides a troubleshooter script called Troubleshoot-DatabaseSpace.ps1 that you can run against a Mailbox server that is currently experiencing excessive database or log growth. The script takes a reasonably hands-off approach to problems in that it doesn't rush to conclusions. Instead, it lets the potential problem go until available space on the disk that hosts the database or the transaction logs falls below a threshold at which it will be exhausted within a configurable time period at the current rate of growth. At this point, the script determines the mailboxes that are responsible for the growth and quarantines them to throttle back growth. If the current rate of growth will not exhaust available space within the period, Exchange logs an event that describes the potential for disk exhaustion due to database or log growth in the Application Event Log or, if SCOM is installed, a SCOM alert is raised.

The script is run as follows:

```
Troubleshoot-DatabaseSpace.ps1 –MailboxDatabaseName DB1
-PercentEdbFreeSpaceThreshold 29 -PercentLogFreeSpaceThreshold 20 –HourThreshold 6
-Quarantine:$True –MonitoringContext:$True
```

The parameters are the following:

- *PercentEdbFreeSpaceThreshold* sets the threshold for the percentage of free space available on the disk that holds the database. Exchange takes available white space in the database into account when it calculates how much free space exists on the disk.
- *PercentLogFreeSpaceThreshold* sets the threshold for the percentage of free space available on the disk that holds the database's transaction logs.

- *HourThreshold* sets the threshold for the number of hours of free space that must be available if the database continues to grow its size or transaction logs. In this example, the troubleshooter script responds (and quarantines mailboxes determined to be causing excessive growth) if disk space will be exhausted in six hours or less based on the current rate of consumption. If sufficient space exists to allow the database to grow for the specified period, the script logs an event and leaves it to the administrator to take the necessary action to restrict further growth.

- *Quarantine* determines whether the troubleshooter is allowed to quarantine mailboxes. If set to $False, the script will not attempt to quarantine problematic mailboxes.

Debugging swelling databases

It is possible for a database suddenly to begin to swell at an abnormal rate. For example, a database that normally grows at 50 MB daily that suddenly starts to expand by 2 GB daily without an obvious reason (such as a large-scale movement of mailboxes into the database) becomes a concern because user activity cannot explain the 40-times increase in growth. Another indication that heightens suspicions that all is not well is when a database reports a large proportion of white (unused) space. For example, if a database is 60 GB and the sum of the space occupied by mailboxes in the database is much lower than this amount, you should ask yourself why this might be. Again, moving many mailboxes out of a database creates white space as pages are freed up by the now-moved mailboxes, but a database that has more than 10 to 15 percent white space is suspect.

You can run the Get-MailboxDatabase cmdlet to check the size on disk and available white space for all databases on a server. Normally, to avoid any performance impact, a call to this cmdlet only reports the static information about a database that is held in Active Directory. In this case, you need to force EMS to make a call to the Store to retrieve internal database data, so pass the –Status parameter to retrieve the necessary data:

```
Get-MailboxDatabase –Server ExServer1 –Status | Format-Table Name, DatabaseSize,
AvailableNewMailboxSpace
```

Comparing the overall database size against the available white space is an easy way to determine whether any database has an unexpectedly high ratio of white space.

The white space within a database is reported as the available amount of space Exchange uses to hold new mailboxes. The same space is also used to expand existing mailboxes as users add new items. More white space will exist in databases immediately after mailboxes are moved to other databases, but this white space will be reused over time because the Store always prefers to use pages that exist inside a database rather than extending the database on disk.

The Exchange 2013 Store includes routines to take action, such as quarantining suspect mailboxes, when problems are detected. However, it's always possible for a new software bug to be exposed through interaction with a particular type of item stored in a mailbox or through the way a client attempts to interact with an item and, in turn, the bug might result in an unexpected growth in the database. One way to deal with quarantined mailboxes is to move them to another database to see whether MRS detects any bad items that are suppressed en route. Even if no bad items are found, the rebuilding of the mailbox in a different database might be sufficient to resolve the problem.

Experience shows that calendar items often are the cause of a swelling database, perhaps because many clients interact with user calendars, using different protocols. You can run the Calendar Checking Tool for Outlook (Calcheck), also downloadable from the Microsoft website, to validate that the calendar items in a suspect mailbox are properly formulated and intact.

Finally, if moving the mailbox or Calcheck cannot cure the problem, you can resort to the examination of transaction logs in an attempt to identify any patterns that show you where the problem lies (for example, the subject of a message). There is no officially supported method to examine Exchange transaction logs because these files are intended to be used for data replication and recovery purposes only. However, an interesting blog from a senior Microsoft support engineer at *http://blogs.msdn.com/b/scottos/archive/2007/07/12/rough-and-tough-guide-to-identifying-patterns-in-ese-transaction-log-files.aspx* explains how you can use a set of utilities to examine transaction logs. No guarantee is given that this approach will turn up any clues, and this is not an activity you should undertake without some planning and preparation. For instance, don't just start reviewing the live transaction logs used to replicate databases within a DAG. Always take copies of logs to work with, preferably from a time when some database swelling was observed, and use them rather than live data.

At the end of the day, none of these methods might turn up any results if you are dealing with a new and previously unknown bug. However, these methods have been successful in the past in identifying problematic client activity caused by corrupt data, so they are worth considering if you have to deal with a swelling database.

Online repair cmdlets

Those who have worked with Exchange for many years know of ISINTEG, the Information Store Integrity maintenance utility Microsoft provided to address problems that occurred in the logical structure of Exchange databases. (By comparison, ESEUTIL handles much lower-level problems that cause physical corruption.) ISINTEG is no more because its function in repairing logical corruptions is now performed through a set of online repair cmdlets.

Prior to Exchange 2010, the database schema featured many tables maintained at database level. A corruption that occurred in a table can therefore affect all the mailboxes in a database. The current schema is more focused on mailbox tables, so the chance of a logical corruption within a database that affects the entire database is remote. Although the likelihood of corruptions is greatly lowered by the new schema, it's also true that new mailbox-level corruptions are possible. However, because most data are now stored in mailbox tables, the simple act of moving a mailbox from one database to another is sufficient to clean out many problems that might exist in these tables. Bad items are dropped en route from the source to target database, and problems that might exist in views and item counts are fixed as the new mailbox is built in the target database.

The mailbox repair cmdlets enable administrators to create online repair requests for mailboxes that address the most common causes of corruption for views and item counts. These are the following:

- Search folder corruptions
- Incorrect aggregate counts on folders
- Incorrect contents returned by folder views
- Incorrect link to a parent folder

The repair cmdlets use roughly the same model as mailbox move, import, and export requests in that an administrator creates a repair request that the Store queues for processing, which then performs whatever repairs are required asynchronously with the database online, so there's no need for the user to log out of her mailbox while the Store examines and adjusts internal mailbox structures. The New-MailboxRepairRequest cmdlet creates a repair request for a mailbox. For example:

```
New-MailboxRepairRequest –Mailbox JSmith –CorruptionType FolderView
```

If you add the –DetectOnly parameter to the request, Exchange reports any corruption it finds but doesn't repair it. The other corruption types that can be fixed in a mailbox are SearchFolder, AggregateCounts, and ProvisionedFolder. These repairs fix problems with search folders, counts on folders, and provisioned fields. You can perform several repairs at one time by specifying a list of the fixes you want to make. For example:

```
New-MailboxRepairRequest –Mailbox JSmith –CorruptionType FolderView,
AggregrateCounts, ProvisionedFolder
```

By default, the Store only repairs the primary mailbox. If you want to include the mailbox's personal archive in the repair, you must specify the –Archive parameter:

```
New-MailboxRepairRequest –Mailbox JSmith –CorruptionType FolderView, AggregrateCounts
-Archive
```

You can also scan all the mailboxes in a database at one time to fix any corruption that is found in any mailbox. For example:

```
New-MailboxRepairRequest –Database DB1 –CorruptionType FolderView, SearchFolder,
AggregateCounts
```

When you submit a new repair request, Exchange responds with a task identifier and the name of the server that will handle the request. This is the Mailbox server that currently hosts the active copy of the database. The only evidence that Exchange has processed the repair request exists in the Event Log, which captures an event (Figure 8-15) when a mailbox repair request is completed. Events are logged on the server that processes the repair request.

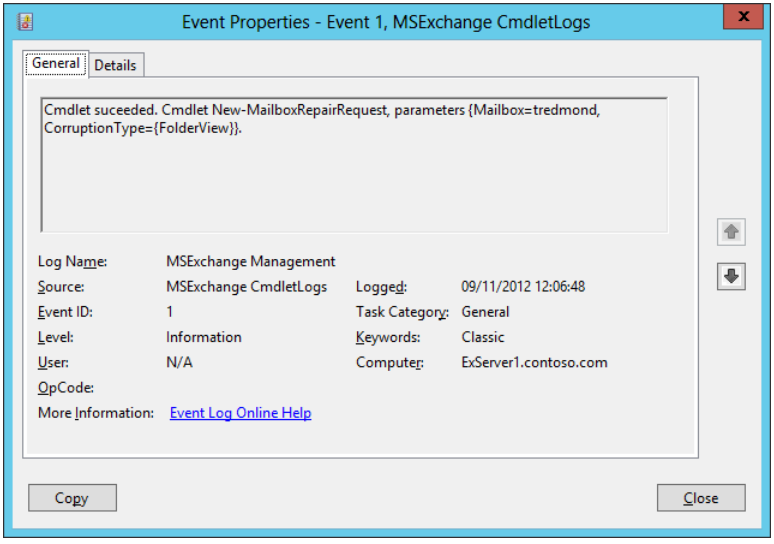


Figure 8-15 Viewing details of a mailbox repair request logged in the event log

INSIDE OUT Canceling a repair job

You can't cancel or review the current status of a repair job. This functionality might be added by Microsoft in a future release. For now, the only way to terminate a repair job is to dismount a database or move the database to another server (or if the database crashes due to a software bug). These actions clear out any pending or active repair jobs within the database.

To ensure that performance is not affected, you can only run a single repair against a complete database on a server at one time. However, you can run up to 100 individual mailbox repairs concurrently on a server (spread across multiple databases).

If the database has copies within a DAG, the results of any repairs made to fix problems found in the tables within the mailbox are replicated along with other transactions to the database copies and are logged as events in the Application Event Log on the server on which the repair is performed.

Rebuilding a database

As you know, Exchange performs ongoing online defragmentation to achieve maximum page contiguity and space usage within the database. The overall size of the database is not affected by online defragmentation because pages freed by processing are recycled and made available to the Store to hold new items. Offline defragmentation or a database rebuild is accomplished by taking the database offline and then running the ESEUTIL utility in /D mode to build a new copy of the database by reading all the pages out of the old database and writing them into the new. Depending on system performance and the size of the database, an offline rebuild can take a long time, and the only thing it accomplishes is the return of unused space to the file system. In the days of Exchange 5.5 and Exchange 2000, when background defragmentation was inefficient and ineffective, it was deemed best practice to perform an offline rebuild for databases on a regular basis to prevent them from swelling to obscene sizes.

Microsoft has improved the effectiveness of background defragmentation significantly, and you should never have to take a database offline to rebuild it except in extraordinary circumstances. For example, mailboxes allocated to students while they attend universities are more transient than mailboxes used in corporate email systems because there is typically a very large turnover of the student body annually. If you had to delete 10,000 student mailboxes from a database, performing a rebuild would shrink the database dramatically and make backups faster. However, given this scenario, it would be better to create a new database, move all the remaining mailboxes to the new database, and then delete the old database. Users aren't disturbed by this operation because the Mailbox Replication service (MRS) moves their mailboxes in the background.

A rebuild might also be required in previous versions of Exchange when the window assigned for background maintenance operation is insufficient for defragmentation to complete, so over time, the database bloats with unused pages. However, given that Exchange performs online defragmentation on an ongoing basis, this argument to justify an offline rebuild does not hold water now.

Unlike previous versions of Exchange, you won't see events logged to indicate when background maintenance has started and finished online defragmentation, and you will not see

any reports of the white space that's available in a database following a defragmentation pass. This is quite logical because the continuous nature of Exchange online defragmentation means that it starts when the Information Store mounts a database and finishes when the Store service shuts down. The loss of events to log the start and finish of defragmentation does not mean that you cannot see evidence that defragmentation is proceeding. Microsoft has added a new set of counters to the system Performance Monitor, so you can start the PerfMon utility and add the MSExchange Database/Instances/DB Maintenance IO Reads/sec counter to the set being monitored. The level of activity varies with system load, and you should see more activity when the system is lightly loaded and less when users are active.

As noted in the "Debugging swelling databases" section earlier in this chapter, you can still gain a sense of the amount of available white space in a database with the Get-MailboxDatabase cmdlet. You might decide to check the databases in use within the organization every six months or so to decide whether any are candidates to be taken offline for a rebuild. In reality, given the efficiency of Exchange databases today, it is unlikely that you will discover anything that requires intervention, so this is something that is done for peace of mind rather than an imperative operational requirement. The command shown below can take some time to complete because it requires EMS to check the status for each database on every server. The data is sorted by database size.

```
Get-MailboxDatabase –Status | Sort-Object DatabaseSize –Descending | Format-Table
Name, DatabaseSize, AvailableNewMailboxSpace
```

```
Name            DatabaseSize                    AvailableNewMailboxSpace
-------         ----------------                ------------------------
MBDatabase3     1.57 GB (1,686,175,744 bytes)   121.1 MB (126,943,232 bytes)
MBDatabase1     1.195 GB (1,283,522,560 bytes)  69.44 MB (72,810,496 bytes)
IT Department   936.1 MB (981,532,672 bytes)    6.063 MB (6,356,992 bytes)
```

The reported database size should be pretty close to the actual file size on disk. The figure for available new mailbox space gives an indication of the available white space in the database and, as you can see, the percentage of free space varies from database to database, depending on recent activity in the database. For example, a database that hosts a set of mailboxes that has remained constant over time will have a lower amount of free space than a database that has had some mailbox moved out to another database. This is obvious when you compare the data reported for the IT Department (0.64 percent free) and MBDatabase2 (85.3 percent free) databases. Over time, the percentages for the two databases will come much closer together as user activity consumes space. If more than 15 percent of space is available in a database, it might be due to incomplete maintenance or another reason such as the recent deletion of many mailboxes.

The available white space reported by Get-MailboxDatabase only tells you how much free space is available in the root tree of the database. It does not include any free space that is available within mailbox or index tables that might be recovered and eventually returned to the root tree by database defragmentation. To get a 100 percent accurate picture of how much white space actually exists within a database, you have to dismount the database, run ESEUTIL /MS, and then calculate the white space by multiplying the number of free pages by 32 KB (because Exchange 2013 uses a 32 KB page size). The article posted at *http://technet.microsoft.com/en-us/library/aa996139(v=EXCHG.65).aspx* explains how to run ESEUTIL /MS, but remember that this was written in 2006 when Exchange 2003 used a 4 KB page size. Most people aren't interested in taking databases offline simply to find out how much free white space exists and are happy to use the data Get-MailboxDatabase reports instead.

The available new mailbox space value is a fair approximation

The value reported for available new mailbox space is not an accurate indication of available white space because it includes only pages that are immediately available to be assigned to new mailboxes, which use space from the root tree within the database. It does not include free pages that exist in mailbox or index tables. However, it's a good approximation of how much white space exists in the database.

Moving away from internal structures, at the level of the database file, Exchange uses a 128-MB default file extension whenever it requests more disk space from Windows to grow an active or passive database.

Using ESEUTIL

ESEUTIL is a useful but dangerous utility. It's useful because it can fix some low-level problems in a database. (However, it cannot fix fundamental corruption caused by hardware failures.) It's dangerous because some administrators and other commentators consider it to be a good thing to run ESEUTIL against Exchange databases on a regular basis just in case. Such an attitude is firmly entrenched in the 1990s when, to be blunt, the Exchange database wasn't as robust and reliable as it is today—and has been since Exchange 2003.

In the early days, the predominant reason for running ESEUTIL was to recover disk space from the database. Exchange was like a swollen pig when it came to growing its database and wasn't very good at reusing database pages. A 10-GB database (big in those days) might only contain 7 GB of mailbox data; you had to rebuild the database with ESEUTIL to return the unused 3 GB storage to the system. Of course, disks were expensive and small, so

recovering 3 GB was a big thing, making the exercise usually worthwhile, even if you had to take the database offline over the weekend to perform the rebuild.

It's a different situation today. Disks are larger, they are radically less expensive, and Exchange is much better at recovering and reusing pages within its databases. It's true that software bugs occasionally cause an excessive growth in the size of a database, but the software bugs are being closed off one by one, and additional safeguards are being added to Exchange to prevent a database from suddenly swelling.

I don't cover the correct use of ESEUTIL in detail in this book. The utility has been around since the earliest days of Exchange and is well covered in other books, blogs, and TechNet. Instead, I refer to ESEUTIL when it is needed, such as validating a recovery database before it is mounted.

INSIDE OUT **Don't use ESEUTIL to rebuild a database**

You should not have to rebuild a database with ESEUTIL unless Microsoft Support advises you to do so. If anyone else tells you that running ESEUTIL to rebuild databases regularly is good for Exchange, you should give him the same look you'd give someone who told you to have a monthly colonic irrigation to improve your complexion.

Those considering using ESEUTIL to rebuild a database just to return some disk space to Windows need to remember that a database rebuild invalidates the current set of its transaction logs. Essentially, the rebuild transforms the internal structure of the database, so any of the transactions in the logs cannot be applied if the need occurs to recover transactions through log replay. This situation is not a problem on a standard server because you can make a full backup immediately after the rebuild to establish a new secure baseline for the database. It's different when a DAG is involved because the transaction logs provide the replication mechanism for the database copies; if you rebuild an active database, you essentially reset the database, and you will have to reseed all the database copies after the rebuild is complete. Not recommended!

Database usage statistics

The Get-StoreUsageStatistics cmdlet is designed to be a diagnostic aid for administrators who are concerned about Store performance for a particular mailbox, database, or server. The cmdlet retrieves data about Store activity for an individual mailbox or for all the mailboxes in a database or server to report the amount of server time executed for Store operations over the past 10 minutes. The cmdlet analyzes the mailboxes with the highest

TimeInServer value over the past 10 minutes, using performance data sampled every second.

The following example retrieves data for the activity in a specific database. In this instance, you see that the time of highest demand was during the "3" sample (three minutes ago) because activity for three mailboxes is noted during this time. The TimeInServer field provides a relative measurement of the demand that a mailbox exerts on the server. It is calculated from the total time spent processing synchronous and asynchronous requests sent to the Store for a mailbox, so it is a pretty good catchall metric that loosely encapsulates the performance indicated by the other metrics.

For example, if the activity for a mailbox is I/O intensive, the latencies for that mailbox tend to be higher, and this increases the TimeInServer figure. Heavy demand for the CPU also increases the TimeInServer figure because the Store has to spend more time processing requests on behalf of the mailbox. By looking at the following data, you can also see that the user who generated the most activity was the EMEA Help Desk in the fourth sample; the TimeInServer rating of 1163 was over twice as high as any other mailbox during the 10-minute duration of the sample.

```
Get-StoreUsageStatistics –Database 'VIP'
```

```
DigestCategory       SampleId       DisplayName                  TimeInServer
------------------   ---------      ------------                 -------------
TimeInServer          0             Mailbox – Akers,Kim          485
TimeInServer          1             Mailbox – Smith, John ...    32
TimeInServer          1             Mailbox - Online Archi...    0
TimeInServer          2             Mailbox – EMEA Help Desk.    1163
TimeInServer          3             Mailbox – Ruth, Andy         16
```

To access statistics for all the databases on a server, you use a command like this:

```
Get-StoreUsageStatistics –Server 'ExServer1'
```

Note that Exchange returns statistics only for mounted databases. Database copies are noted, but you won't see any statistics. This is quite logical because no mailboxes can connect to these database copies. An individual mailbox can appear multiple times in the output for a database or a server. To drill down on the activity for a mailbox, you can retrieve more detailed statistics for a specific mailbox by passing its name to the cmdlet.

```
Get-StoreUsageStatistics –Identity 'Akers, Kim'
```

```
DigestCategory    : TimeInServer
SampleId          : 1
SampleTime        : 5/23/2010 2:39:10 AM
DisplayName       : Mailbox – Akers, Kim
```

```
TimeInServer      : 424
TimeInCPU         : 79
ROPCount          : 222
PageRead          : 0
PagePreread       : 0
LogRecordCount    : 975
LogRecordBytes    : 467303
LdapReads         : 1
LdapSearches      : 0
ServerName        : ExServer1
DatabaseName      : VIP Data
```


Apart from the TimeInServer metric, the data includes:

- **TimeInCPU** The number of milliseconds of CPU time used for operations for this mailbox. Samples are taken on a one-minute basis; if the server has multiple processors, this number can exceed 60 seconds. The timer is very simple and is triggered whenever the CPU dedicates time to the thread handling operations for the mailbox. This information is useful when tracking down problematic clients that are imposing a high load on the server.
- **ROPCount** The number of remote (client) operations performed on the mailbox during the sample period.
- **PageRead** The number of noncached page reads required by the mailbox.
- **PagePreRead** The number of pre-read pages required by the mailbox.
- **LogRecordCount** The number of log records written out for this mailbox during the sample period. Together with TimeInServer, this is a good indication of the amount of activity the Store performs to handle mailbox operations.
- **LogRecordBytes** LogRecordCount converted to bytes.
- **LDAPReads** The number of asynchronous LDAP reads required by mailbox operations.
- **LDAPSearches** The number of LDAP searches performed on behalf of the mailbox.

The measurement data are cleared out after 10 minutes. This means that Get-StoreUsageStatistics returns blank results if you look for data 10 minutes after the last mailbox activity was recorded. If fewer than 25 users have been active during the past 10 minutes, only these users will be listed in the summary.

Mailbox assistants

Exchange includes a number of mailbox assistants to perform automated processing on different aspects of the data held in mailboxes. An assistant is just another name for a thread that executes within the Exchange System Attendant process on a Mailbox server. Exchange 2013 uses two types of assistants:

- **Event-based assistants** These respond to events as they occur. For example, the resource booking assistant monitors requests to book rooms and lets users know whether the room is unavailable.
- **Throttle-based assistants** These operate all the time to process data and are subject to throttling to ensure that their workload does not affect the responsiveness of a server. The Managed Folder Assistant is the best known of these assistants.

Among the better-known assistants are the following:

- **Calendar assistant** Checks mailboxes for incoming meeting requests and processes the requests according to mailbox settings
- **Resource booking assistant** Processes meeting requests for room and equipment mailboxes
- **Scheduling assistant** Scans attendee calendars for suitable meeting slots and suggests them to meeting organizers
- **Junk email options assistant** Copies the safelist data from user mailboxes to their Active Directory accounts so the anti-spam agents can use it
- **Managed folder assistant** Processes retention policies for managed folders and applies retention policy tags to items in mailboxes

These assistants are relatively well known because they exist in previous versions of Exchange.Exchange 2013 includes a new test cmdlet to enable administrators to check that the mailbox assistants are functioning properly on a server and to recover from any problem the test cmdlet detects. For example, this command runs the cmdlet to test that the mailbox assistants are functioning properly on server Exserver1 and then reports the results in list format:

```
Test-AssistantHealth –Server ExServer2 –ResolveProblems | Format-List
```

Exchange uses a work cycle model to define how its assistants perform work while remaining active in the background to accomplish goals over a set period. For example, you might define that the Managed Folder Assistant must process every mailbox on a server at least once every two days. Given the constraints determined in a work cycle, the assistant will

work out how much work it has to do over the period and create an internal schedule to do the work on a phased basis so that a smooth and predictable demand is generated on the server rather than the peaks created by the older model. The assistant monitors system conditions on an ongoing basis to make any required adjustments to perform well. If the server is under high demand for a period due to user activity, the assistant can back off its work and then speed up when server load drops.

The Get-MailboxServer and Set-MailboxServer cmdlets retrieve and set work cycle information for assistant processes. First, find out what the current work cycles are for the assistants that run on a multirole server:

```
Get-MailboxServer –Identity ExServer2 | Format-List *WorkCycle*
```

The majority of the assistant processes process the data for which they are responsible at least once daily because the work cycle period is set to one day. In effect, the goal for the assistant is to process all relevant data for all mailboxes on the server in a 24-hour period, assuming that system resources allow it and the assistants are not throttled back to allow more important processes to proceed. The checkpoint for an assistant tells it how often it should check for new objects that need to be processed, such as looking to see whether new mailboxes have been added to the databases on a server. In most cases, the checkpoint is equivalent to the work cycle period. It makes sense to have the same work cycle settings applied to all Mailbox servers in the organization. This is easily done by running the Get-MailboxServer command to fetch a collection of all Mailbox servers and then using Set-MailboxServer to set the properties.

And now for something completely different

Much of the technology discussed so far has existed in one form or another for at least several versions of Exchange. The Store is powered by a database engine, and its physical instantiation is a set of databases on disk. The interesting subject is to discuss just what can be done with those databases using the high-availability technology included in Exchange 2013, so that is where this book goes next.