Microsoft Exchange Server 2013 Mailbox and High Availability

Conquer Mailbox administration—from the inside out!
Focusing on the Mailbox server role, dive into Exchange Server 2013—and really put your enterprise messaging to work! This supremely organized reference packs hundreds of timesaving solutions, troubleshooting tips, and workarounds for managing mailboxes and high availability. Discover how the experts manage core operations and support tasks—and challenge yourself to new levels of mastery.

- Prepare for installation or upgrade
- Master role-based access control (RBAC) fundamentals
- Create, manage, move, and archive mailboxes
- Implement email address policies
- Configure and manage distribution groups
- Understand Store components and functionality
- Deliver high availability through database availability groups (DAGs)
- Manage compliance, retention, mailbox search, and data loss prevention
- Use the Exchange Management Shell and cmdlets
- Administer public folder architecture

About the Author
Tony Redmond is a Microsoft Most Valuable Professional (MVP) and one of the leading voices in the Exchange Server community. He has two decades of experience with enterprise mail, focusing on Exchange Server since version 4.0. As an industry consultant, he guides customers through Exchange Server deployment and management, and he’s written 10 books.

Foreword by Rajesh Jha
Corporate Vice President, Exchange Server Group, Microsoft Corporation

Inside OUT

Microsoft Exchange Server 2013 Mailbox and High Availability

For experienced Exchange Server administrators

Also look for
Microsoft Exchange Server 2013 Inside Out: Connectivity, Clients, and UM
9780735678378

Also look for
Microsoft Exchange Server 2013 Inside Out: Connectivity, Clients, and UM
9780735678378

Recommended

microsoft.com/mspress

U.S.A. $49.99
Canada $52.99
[Recommended]

Microsoft Press
Celebrating 30 years!

Microsoft Exchange Server 2013: Mailbox and High Availability

Tony Redmond Award-winning author + Microsoft MVP
## Contents at a Glance

Chapter 1  
**Introducing Microsoft Exchange Server 2013**  
1

Chapter 2  
**Installing Exchange 2013**  
43

Chapter 3  
**The Exchange Management Shell**  
83

Chapter 4  
**Role-based access control**  
131

Chapter 5  
**Mailbox management**  
169

Chapter 6  
**More about the Exchange Administration Center**  
267

Chapter 7  
**Addressing Exchange**  
333

Chapter 8  
**The Exchange 2013 Store**  
387

Chapter 9  
**The Database Availability Group**  
457

Chapter 10  
**Moving mailboxes**  
567

Chapter 11  
**Compliance management**  
641

Chapter 12  
**Public folders and site mailboxes**  
765
# Table of Contents

## Introduction

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
<td>xvii</td>
</tr>
<tr>
<td>Errata &amp; book support</td>
<td>xviii</td>
</tr>
<tr>
<td>We want to hear from you</td>
<td>xix</td>
</tr>
<tr>
<td>Stay in touch</td>
<td>xix</td>
</tr>
</tbody>
</table>

## Chapter 1

### Introducing Microsoft Exchange Server 2013

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understanding development priorities</td>
<td>1</td>
</tr>
<tr>
<td>The influence of The Service</td>
<td>2</td>
</tr>
<tr>
<td>Exchange Online and Exchange development</td>
<td>5</td>
</tr>
<tr>
<td>Transition into the cloud</td>
<td>7</td>
</tr>
<tr>
<td>Exchange 2013 architecture</td>
<td>9</td>
</tr>
<tr>
<td>The motivation to upgrade</td>
<td>10</td>
</tr>
<tr>
<td>Evolving from earlier versions of Exchange</td>
<td>11</td>
</tr>
<tr>
<td>Waiting for updates</td>
<td>12</td>
</tr>
<tr>
<td>Fundamental questions before you upgrade</td>
<td>13</td>
</tr>
<tr>
<td>Selecting the right Windows Server for Exchange 2013</td>
<td>15</td>
</tr>
<tr>
<td>Using virtualization</td>
<td>22</td>
</tr>
<tr>
<td>Preparing for Exchange 2013</td>
<td>24</td>
</tr>
<tr>
<td>A word about transport</td>
<td>26</td>
</tr>
<tr>
<td>The test plan</td>
<td>27</td>
</tr>
<tr>
<td>Testing for operational processes</td>
<td>28</td>
</tr>
<tr>
<td>Testing for programming and customizations</td>
<td>29</td>
</tr>
<tr>
<td>Updating earlier versions of Exchange</td>
<td>30</td>
</tr>
<tr>
<td>Deploying earlier versions of Exchange servers alongside Exchange 2013</td>
<td>31</td>
</tr>
<tr>
<td>Exchange 2013 editions</td>
<td>32</td>
</tr>
<tr>
<td>Active Directory</td>
<td>34</td>
</tr>
<tr>
<td>Using the strong link between Exchange and Active Directory</td>
<td>34</td>
</tr>
<tr>
<td>ADSIEdit</td>
<td>36</td>
</tr>
</tbody>
</table>

---

What do you think of this book? We want to hear from you!

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

[microsoft.com/learning/booksurvey](microsoft.com/learning/booksurvey)
## Table of Contents

**Useful utilities**

- MFCMAPI ......................................................... 38
- Exchange Web Services Editor ....................... 40

**Installing Exchange 2013** .................................. 41

### Chapter 2

**Installing Exchange 2013** .................................. 43

- Approaching the installation .................................. 43
- Types of Active Directory deployment that support Exchange ........................................ 45
- Preparing Active Directory for Exchange .................. 47
- Creating the Exchange 2013 organization ................ 49
- Deploying an Exchange 2013 server ......................
  - Installing UCMA ............................................... 53
  - Running Setup ................................................ 53
  - Setup logs ..................................................... 56
  - Uninstalling Exchange ...................................... 58
  - Repairing Exchange ......................................... 61
  - Recovering a failed server ............................... 61

### Chapter 3

**The Exchange Management Shell** ....................... 83

- How Exchange uses Windows PowerShell ................ 83
- Using remote Windows PowerShell ......................
  - Connecting to remote PowerShell .................... 88
  - Limiting user functionality ............................ 91

**EMS basics** .................................................. 92

- Command editing .............................................. 95
- Handling information EMS returns ...................... 98
- Selective output ............................................. 98
- Using common and user-defined variables ............ 101
- Using PowerShell ISE with Exchange .................... 103
- Identities .................................................... 105
- Piping ......................................................... 109
- OPATH filters ................................................ 110
- Server-side and client-side filters ...................... 110
- Transcripts ................................................... 114
- Bulk updates ................................................. 114
- Calling scripts ............................................... 117
Table of Contents

Chapter 4

Role-based access control ........................................... 131

RBAC basics ................................................................. 132
Roles ........................................................................ 136
Role groups .................................................................. 139
Scopes ........................................................................ 141
Working with RBAC ....................................................... 143
Creating a new role group ................................................ 145
Maintaining role group membership ................................... 149
Role group management .................................................. 151
Role assignment ............................................................ 152
Using role assignment policy to limit access ......................... 153
Creating roles for specific tasks ........................................ 155
Specific scopes for role groups ......................................... 157
Database scoping .......................................................... 158
Special roles ................................................................ 159
Unscoped roles ............................................................. 160
Which role groups do I belong to? ...................................... 162
Assignment policies ......................................................... 163
Implementing a split permissions model ................................ 166
Figuring out RBAC ......................................................... 167
On to management ......................................................... 168

Chapter 5

Mailbox management ..................................................... 169

Seeking perfection halts progress ....................................... 170
Starting EAC ................................................................. 173
How EAC accesses Exchange data ...................................... 176
Changing EAC columns ................................................... 177
Naming conventions ....................................................... 178
Managing Recipients ....................................................... 180
Recipient filtering .......................................................... 180
Exporting EAC information to CSV files ............................... 181
Some mysterious mailboxes ............................................. 182
The need for mailboxes ................................................... 183
Naming mailboxes .......................................................... 185
Creating new mailboxes ................................................... 187
Completing mailboxes ..................................................... 193
Bulk mailbox creation .................................................... 195
Creating new room and resource mailboxes ................................................. 195
Languages ........................................................................... 196
Default folders ........................................................................ 200
Manipulating mailbox settings ................................................................. 202
Automating mailbox settings .................................................................... 205
What’s in a mailbox? ........................................................................... 207
Ready-to-go custom attributes ................................................................... 207
Mailbox resources provisioning management agent ..................................... 209
Setting mailbox quotas ............................................................................ 213
Removing or disabling mailboxes ............................................................... 218
Reconnecting mailboxes ........................................................................... 220
Discovery mailboxes .................................................................................. 224
Creating additional discovery mailboxes .................................................... 224
Health mailboxes ...................................................................................... 226
Setting mailbox permissions ...................................................................... 229
Mailbox delegation ..................................................................................... 229
Managing Full Access permission ................................................................ 232
Mailbox auto-mapping through Autodiscover .............................................. 235
Opening another user’s mailbox with Outlook Web App .............................. 237
Sending messages on behalf of other users ................................................. 237
Outlook delegate access ............................................................................. 239
Shared mailboxes ...................................................................................... 240
Recalling messages ..................................................................................... 241
Moderated recipients ................................................................................... 243
Moderated groups ....................................................................................... 244
Processing moderation requests ................................................................... 247
Moderated mailboxes .................................................................................. 249
Mail-enabled contacts ................................................................................ 250
Mail users ................................................................................................ 252
Resource mailboxes ................................................................................... 254
Defining custom properties for resource mailboxes .................................... 255
Providing policy direction to the Resource Booking Attendant ...................... 256
Processing meeting requests according to policy ......................................... 261
Equipment mailboxes ................................................................................ 265
Enough about mailboxes ............................................................................. 266

Chapter 6  

More about the Exchange Administration Center ........................................... 267

Groups ................................................................................................. 267
Managing groups ....................................................................................... 269
Creating new groups .................................................................................. 270
Group owners ............................................................................................ 275
Group expansion ......................................................................................... 277
Protected groups ....................................................................................... 278
Viewing group members ............................................................................ 281
Security groups ........................................................................................ 284
Tracking group usage ................................................................................ 285
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allowing users to manage groups</td>
<td>286</td>
</tr>
<tr>
<td>Room lists</td>
<td>298</td>
</tr>
<tr>
<td>Dynamic distribution groups</td>
<td>300</td>
</tr>
<tr>
<td>OPATH queries</td>
<td>300</td>
</tr>
<tr>
<td>Creating new dynamic distribution groups</td>
<td>302</td>
</tr>
<tr>
<td>Validating query results</td>
<td>305</td>
</tr>
<tr>
<td>Creating dynamic groups with EMS</td>
<td>306</td>
</tr>
<tr>
<td>Using custom filters</td>
<td>308</td>
</tr>
<tr>
<td>Sharing</td>
<td>313</td>
</tr>
<tr>
<td>Certificate management</td>
<td>313</td>
</tr>
<tr>
<td>Mail flow and rules</td>
<td>318</td>
</tr>
<tr>
<td>Delivery reports</td>
<td>320</td>
</tr>
<tr>
<td>Administrator searches for delivery reports</td>
<td>322</td>
</tr>
<tr>
<td>Using EMS to search delivery reports</td>
<td>327</td>
</tr>
<tr>
<td>Running EAC without an Exchange mailbox</td>
<td>329</td>
</tr>
<tr>
<td>Setting diagnostics for Exchange servers</td>
<td>330</td>
</tr>
<tr>
<td>But what will you manage?</td>
<td>332</td>
</tr>
<tr>
<td>Chapter 7</td>
<td></td>
</tr>
<tr>
<td><strong>Addressing Exchange</strong></td>
<td>333</td>
</tr>
<tr>
<td>Email address policies</td>
<td>333</td>
</tr>
<tr>
<td>Email policy priority</td>
<td>334</td>
</tr>
<tr>
<td>Creating a new email address policy</td>
<td>337</td>
</tr>
<tr>
<td>Focusing on certain recipients by using filters</td>
<td>341</td>
</tr>
<tr>
<td>Creating email address policies with custom filters</td>
<td>343</td>
</tr>
<tr>
<td>Address lists</td>
<td>345</td>
</tr>
<tr>
<td>Address book policies</td>
<td>348</td>
</tr>
<tr>
<td>ABPs and groups</td>
<td>350</td>
</tr>
<tr>
<td>Creating and implementing an ABP</td>
<td>351</td>
</tr>
<tr>
<td>The Offline Address Book</td>
<td>357</td>
</tr>
<tr>
<td>The OAB and Outlook</td>
<td>359</td>
</tr>
<tr>
<td>How Exchange 2013 generates the Offline Address Book</td>
<td>362</td>
</tr>
<tr>
<td>Creating and using customized OABs</td>
<td>367</td>
</tr>
<tr>
<td>Hierarchical address book</td>
<td>371</td>
</tr>
<tr>
<td>MailTips and group metrics</td>
<td>373</td>
</tr>
<tr>
<td>Client interaction</td>
<td>376</td>
</tr>
<tr>
<td>Configuring MailTips</td>
<td>377</td>
</tr>
<tr>
<td>User experience</td>
<td>378</td>
</tr>
<tr>
<td>Custom MailTips</td>
<td>380</td>
</tr>
<tr>
<td>Multilingual custom MailTips</td>
<td>381</td>
</tr>
<tr>
<td>OAB support for MailTips</td>
<td>381</td>
</tr>
<tr>
<td>The display or Details Templates Editor</td>
<td>382</td>
</tr>
<tr>
<td>The next step</td>
<td>386</td>
</tr>
<tr>
<td>Chapter 8</td>
<td></td>
</tr>
<tr>
<td><strong>The Exchange 2013 Store</strong></td>
<td>387</td>
</tr>
<tr>
<td>Long live JET!</td>
<td>388</td>
</tr>
<tr>
<td>Maximum database size</td>
<td>389</td>
</tr>
<tr>
<td>Sizing mailboxes</td>
<td>391</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Dealing with I/O</td>
<td>392</td>
</tr>
<tr>
<td>Maintaining contiguity</td>
<td>397</td>
</tr>
<tr>
<td>The database schema</td>
<td>398</td>
</tr>
<tr>
<td>Exchange 2013 I/O improvements</td>
<td>400</td>
</tr>
<tr>
<td>Workers, controller, and memory</td>
<td>402</td>
</tr>
<tr>
<td>Managed Availability and the Managed Store</td>
<td>407</td>
</tr>
<tr>
<td>Database management</td>
<td>407</td>
</tr>
<tr>
<td>Creating new mailbox databases</td>
<td>410</td>
</tr>
<tr>
<td>Updating mailbox databases after installation</td>
<td>414</td>
</tr>
<tr>
<td>Backups and permanent removal</td>
<td>416</td>
</tr>
<tr>
<td>Removing a database</td>
<td>417</td>
</tr>
<tr>
<td>Transaction logs</td>
<td>419</td>
</tr>
<tr>
<td>Log sets</td>
<td>419</td>
</tr>
<tr>
<td>Transaction log checksum</td>
<td>427</td>
</tr>
<tr>
<td>Transaction log I/O</td>
<td>427</td>
</tr>
<tr>
<td>The question of circular logging</td>
<td>429</td>
</tr>
<tr>
<td>Reserved logs</td>
<td>431</td>
</tr>
<tr>
<td>Background maintenance</td>
<td>432</td>
</tr>
<tr>
<td>Database checksums</td>
<td>433</td>
</tr>
<tr>
<td>Database defragmentation</td>
<td>435</td>
</tr>
<tr>
<td>Database compaction</td>
<td>436</td>
</tr>
<tr>
<td>Page zeroing</td>
<td>436</td>
</tr>
<tr>
<td>Content maintenance tasks</td>
<td>436</td>
</tr>
<tr>
<td>Corrupt item detection and isolation</td>
<td>437</td>
</tr>
<tr>
<td>Protection against high latency</td>
<td>442</td>
</tr>
<tr>
<td>Protection against excessive database or log growth</td>
<td>443</td>
</tr>
<tr>
<td>Debugging swelling databases</td>
<td>444</td>
</tr>
<tr>
<td>Online repair cmdlets</td>
<td>445</td>
</tr>
<tr>
<td>Rebuilding a database</td>
<td>448</td>
</tr>
<tr>
<td>Using ESEUTIL</td>
<td>450</td>
</tr>
<tr>
<td>Database usage statistics</td>
<td>451</td>
</tr>
<tr>
<td>Mailbox assistants</td>
<td>454</td>
</tr>
<tr>
<td>And now for something completely different</td>
<td>455</td>
</tr>
<tr>
<td>The Database Availability Group</td>
<td>457</td>
</tr>
<tr>
<td>Changes in high availability in Exchange 2013</td>
<td>457</td>
</tr>
<tr>
<td>Database portability</td>
<td>459</td>
</tr>
<tr>
<td>Database Availability Group basics</td>
<td>461</td>
</tr>
<tr>
<td>The dependency on Windows clustering</td>
<td>463</td>
</tr>
<tr>
<td>Migrating an Exchange 2010 DAG</td>
<td>464</td>
</tr>
<tr>
<td>Active Manager</td>
<td>465</td>
</tr>
<tr>
<td>Automatic database transitions</td>
<td>467</td>
</tr>
<tr>
<td>Managed Availability and high availability</td>
<td>469</td>
</tr>
<tr>
<td>Best copy and server selection</td>
<td>472</td>
</tr>
<tr>
<td>ACLL: Attempt copy last logs</td>
<td>478</td>
</tr>
<tr>
<td>Transaction log replay: The foundation for DAG replication</td>
<td>479</td>
</tr>
<tr>
<td>Transaction log compression</td>
<td>483</td>
</tr>
</tbody>
</table>
Chapter 10  Moving mailboxes ................................................. 567

Moving mailboxes .......................................................... 567
The Mailbox Replication service .......................................... 567
Asynchronous moving ....................................................... 571
Mailbox Replication service processing ................................. 577
Creating and managing migration batches with EAC .............. 581
Using EMS with migration batches ...................................... 592
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reporting mailbox moves</td>
<td>595</td>
</tr>
<tr>
<td>Getting more information about a move</td>
<td>597</td>
</tr>
<tr>
<td>Accessing move report histories</td>
<td>599</td>
</tr>
<tr>
<td>Managing individual mailbox moves</td>
<td>605</td>
</tr>
<tr>
<td>Assigning move priority</td>
<td>607</td>
</tr>
<tr>
<td>Moves and mailbox provisioning</td>
<td>607</td>
</tr>
<tr>
<td>Handling move request errors</td>
<td>607</td>
</tr>
<tr>
<td>Checking and altering move request status</td>
<td>609</td>
</tr>
<tr>
<td>Clearing move requests</td>
<td>610</td>
</tr>
<tr>
<td>Migration endpoints</td>
<td>611</td>
</tr>
<tr>
<td>Enabling MRSProxy</td>
<td>612</td>
</tr>
<tr>
<td>Planning mailbox moves</td>
<td>614</td>
</tr>
<tr>
<td>Suspending mailbox moves</td>
<td>615</td>
</tr>
<tr>
<td>Ensuring high availability</td>
<td>619</td>
</tr>
<tr>
<td>MRS configuration file</td>
<td>621</td>
</tr>
<tr>
<td>Mailbox import and export</td>
<td>623</td>
</tr>
<tr>
<td>Gaining permission through RBAC to execute mailbox import and export</td>
<td>624</td>
</tr>
<tr>
<td>Planning the import of PST data</td>
<td>626</td>
</tr>
<tr>
<td>Importing and exporting mailbox data with EAC</td>
<td>629</td>
</tr>
<tr>
<td>Importing and exporting mailbox data with EMS</td>
<td>632</td>
</tr>
<tr>
<td>Alerts</td>
<td>638</td>
</tr>
<tr>
<td>Time to comply</td>
<td>640</td>
</tr>
</tbody>
</table>

Chapter 11: Compliance management ................................................. 641

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>The joy of legal discovery</td>
<td>642</td>
</tr>
<tr>
<td>Archive mailboxes</td>
<td>645</td>
</tr>
<tr>
<td>Enabling archives</td>
<td>647</td>
</tr>
<tr>
<td>The default archive and retention policy</td>
<td>652</td>
</tr>
<tr>
<td>Using an archive mailbox</td>
<td>655</td>
</tr>
<tr>
<td>Disabling an archive mailbox</td>
<td>656</td>
</tr>
<tr>
<td>Messaging records management</td>
<td>657</td>
</tr>
<tr>
<td>Types of retention tags</td>
<td>657</td>
</tr>
<tr>
<td>System tags</td>
<td>662</td>
</tr>
<tr>
<td>Designing a retention policy</td>
<td>663</td>
</tr>
<tr>
<td>Managed Folder Assistant and retention policies</td>
<td>665</td>
</tr>
<tr>
<td>Naming retention tags</td>
<td>666</td>
</tr>
<tr>
<td>Creating retention tags</td>
<td>668</td>
</tr>
<tr>
<td>Creating a retention policy</td>
<td>673</td>
</tr>
<tr>
<td>Applying a retention policy to mailboxes</td>
<td>677</td>
</tr>
<tr>
<td>Modifying a retention policy</td>
<td>681</td>
</tr>
<tr>
<td>Customizing retention policies for specific mailboxes</td>
<td>681</td>
</tr>
<tr>
<td>User interaction with retention policies</td>
<td>684</td>
</tr>
<tr>
<td>Setting a retention policy on a folder</td>
<td>685</td>
</tr>
<tr>
<td>Removing tags from policies</td>
<td>687</td>
</tr>
<tr>
<td>Removing a retention policy</td>
<td>688</td>
</tr>
<tr>
<td>Upgrading from managed folders</td>
<td>688</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>How the Managed Folder Assistant implements retention policies</td>
<td>689</td>
</tr>
<tr>
<td>Behind the scenes with the MFA</td>
<td>691</td>
</tr>
<tr>
<td>Retention date calculation</td>
<td>693</td>
</tr>
<tr>
<td>Preserving information</td>
<td>696</td>
</tr>
<tr>
<td>Putting a mailbox on retention hold</td>
<td>698</td>
</tr>
<tr>
<td>Putting a mailbox on litigation hold</td>
<td>699</td>
</tr>
<tr>
<td>Searching mailbox content</td>
<td>701</td>
</tr>
<tr>
<td>In-place holds</td>
<td>703</td>
</tr>
<tr>
<td>Creating a new search</td>
<td>708</td>
</tr>
<tr>
<td>Retrieving discovered content</td>
<td>716</td>
</tr>
<tr>
<td>Examining search results</td>
<td>720</td>
</tr>
<tr>
<td>Resource throttling for searches</td>
<td>724</td>
</tr>
<tr>
<td>How in-place holds work</td>
<td>726</td>
</tr>
<tr>
<td>Using groups with searches</td>
<td>728</td>
</tr>
<tr>
<td>Removing a search</td>
<td>730</td>
</tr>
<tr>
<td>Executing searches with EMS</td>
<td>731</td>
</tr>
<tr>
<td>What Exchange can search</td>
<td>733</td>
</tr>
<tr>
<td>Search syntaxes</td>
<td>736</td>
</tr>
<tr>
<td>The value of the Recoverable Items structure</td>
<td>737</td>
</tr>
<tr>
<td>The function of the Recoverable Items structure</td>
<td>737</td>
</tr>
<tr>
<td>Improvements in Exchange 2013</td>
<td>739</td>
</tr>
<tr>
<td>Single-item recovery</td>
<td>743</td>
</tr>
<tr>
<td>Knowing what’s in Recoverable Items</td>
<td>745</td>
</tr>
<tr>
<td>Managing quotas for Recoverable Items</td>
<td>746</td>
</tr>
<tr>
<td>Auditing administrator actions</td>
<td>748</td>
</tr>
<tr>
<td>The audit mailbox</td>
<td>749</td>
</tr>
<tr>
<td>How administrator auditing happens</td>
<td>750</td>
</tr>
<tr>
<td>Auditing mailbox access</td>
<td>754</td>
</tr>
<tr>
<td>Enabling mailboxes for auditing</td>
<td>757</td>
</tr>
<tr>
<td>Accessing mailbox audit data</td>
<td>759</td>
</tr>
<tr>
<td>Other compliance features</td>
<td>763</td>
</tr>
<tr>
<td>Public folders and site mailboxes</td>
<td>765</td>
</tr>
<tr>
<td>Public folders</td>
<td>765</td>
</tr>
<tr>
<td>New concepts</td>
<td>766</td>
</tr>
<tr>
<td>Creating public folder mailboxes</td>
<td>767</td>
</tr>
<tr>
<td>How many public folder mailboxes are needed?</td>
<td>770</td>
</tr>
<tr>
<td>Controlling the root</td>
<td>774</td>
</tr>
<tr>
<td>Creating public folders</td>
<td>776</td>
</tr>
<tr>
<td>Mail-enabling public folders</td>
<td>779</td>
</tr>
<tr>
<td>Moving public folders</td>
<td>783</td>
</tr>
<tr>
<td>Redirecting content</td>
<td>785</td>
</tr>
<tr>
<td>Removing a public folder</td>
<td>786</td>
</tr>
<tr>
<td>Mobile access</td>
<td>786</td>
</tr>
<tr>
<td>Organizational forms library</td>
<td>787</td>
</tr>
<tr>
<td>Migration to modern public folders</td>
<td>787</td>
</tr>
</tbody>
</table>
Foreword for Exchange 2013 Inside Out books

Those seeking an in-depth tour of Exchange Server 2013 couldn’t ask for better guides than Tony Redmond and Paul Robichaux. Tony and Paul have a relationship with the Exchange team that goes back two decades, to the days of Exchange 4.0. Few people have as much practical knowledge about Exchange, and even fewer have the teaching skills to match. You are in good hands.

Over the past few years, we have seen significant changes in the way people communicate; a growing number of devices, an explosion of information, increasingly complex compliance requirements, and a multigenerational workforce. This world of communication challenges has been accompanied by a shift toward cloud services. As we designed Exchange 2013, the Exchange team worked hard to build a product and service that address these challenges. As you read these books, you’ll get an up-close look at the outcome of our efforts.

*Microsoft Exchange Server 2013 Inside Out: Mailbox and High Availability* covers foundational topics such as the Exchange Store, role-based access control (RBAC), our simplified approach to high availability, and the new public folder architecture. It also covers our investments in eDiscovery and in-place hold. As you read, you’ll see how Exchange 2013 helps you achieve world-class reliability and provides a way to comply with internal and regulatory compliance requirements without the need for third-party products.

*Microsoft Exchange Server 2013 Inside Out: Connectivity, Clients, and UM* explores the technologies that give users anywhere access to their email, calendar, and contacts across multiple devices. It also explains how to protect your email environment from spam, viruses, and other threats and describes how Exchange 2013 can connect with Office 365 so you can take advantage of the power of the cloud.

From our new building-block architecture to data loss prevention, there’s a lot to explore in the newest version of Exchange. I hope that as you deploy and use Exchange 2013, you’ll agree that this is an exciting and innovative release.

Enjoy!

Rajesh Jha
Corporate Vice President - Exchange
Microsoft Corporation
Introduction

This book is for experienced Exchange administrators who want to get inside the soul of Exchange Server 2013, the latest version of the Microsoft enterprise messaging server first released in October 2012 and updated on a frequent basis since. You might learn how to work with Exchange 2013 by reading this book, but I sincerely doubt that this will happen simply because I have written it with experience in mind.

The book does not cover every possible topic relating to Exchange 2013. In fact, it focuses primarily on the Mailbox server role. Let me explain why. After completing Microsoft Exchange Server 2010 Inside Out (Microsoft Press, 2010), it became very clear that attempting to cover all of a complex product such as Exchange in any depth in just one book was a fool’s errand. There are too many details to master, too much work to do, too much information that can only be skimmed over to keep to a reasonable page count. The result would probably be a book that weighs 2 kilos, spanning 1,400 pages that takes 2 years to write. All in all, an unacceptable situation in both commercial and practical terms.

Paul Robichaux and I ran a number of Exchange 2010 Maestro seminars in the 2010–2011 period. Despite the infamous cockroach sandwich affair, the events were good fun, and we enjoyed discussing the technology in some depth, even if we tended to ramble on at times. Brian Desmond, an Active Directory MVP who did an excellent job of lab master and stand-in speaker when required, helped us. Because we worked well together and because Paul has an excellent record of writing both books and articles, it seemed like a good idea to consider a joint approach for Microsoft Exchange Server 2013 Inside Out. We arrived at the basic idea quickly—we would split coverage into the two server roles. I’d write about the Mailbox role and Paul took on client access, including all the various clients Exchange supports, and unified messaging, which, strictly speaking, is part of an Exchange 2013 Mailbox server. However, Paul is an acknowledged expert in this space, and it would have made no sense to have me write about a subject of which Paul is the master.

Because Exchange 2013 is an evolution of Exchange 2010, we decided to use Microsoft Exchange Server 2010 Inside Out as the base for the new book. An evolution it might be, but an extensive level of change at the detail level exists in Exchange 2013. The upshot is that I’m not sure how much of that book remains in the current text—maybe 20 percent. One thing I am glad of is that we did not rush to press after Exchange 2013 first appeared. Given the amount of change that has occurred in updates from Microsoft since, a book that describes the release to manufacturing (RTM) version of Exchange 2013 would have been obsolete very soon after publication. We hope that these volumes will last longer.
I hope that you enjoy this book and that you’ll read it alongside Paul’s *Microsoft Exchange Server 2013 Inside Out: Clients, Connectivity, and UM*. The two books really do go together. Paul has scrutinized every word in this book and I have done the same for his. We therefore share the blame for any error you might find.

**Acknowledgments**

I owe enormous thanks to the many people who agreed to look over chapters or portions of the book. Each has deep expertise in specific areas and all contributed greatly to eradicating errors and increasing clarity. These folks include Sanjay Ramaswamy, Jürgen Hasslauer, David Espinoza, William Rall, Todd Luttinen, Tim McMichael, Vineetha Kalvakunta, Fred Monteiro da Cruz Filho, Kanika Ramji, Lokesh Bhoobalan, Astrid McClean, Alfons Staerk, Kern Hardman, Andrew Friedman, Abram Jackson, and Scott Schnoll. Even if they didn’t realize it, many of the Exchange MVPs played their part in improving the book by prompting me to look into topics that I had forgotten to cover. I should also acknowledge the huge contribution made by my editor, Karen Szall. We fought many times about page counts, content, and too many other topics to list here but always kept the project moving.

I apologize sincerely if I have omitted to mention anyone who has contributed to making the text of the book as accurate and as informative as possible.

**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/ExlOv1/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.
Windows PowerShell is an extensible automation engine consisting of a command-line shell and a scripting language. Exchange Server 2007 was the first major Microsoft application to support Windows PowerShell in a comprehensive manner. Although not every administrator welcomed the opportunity to learn a new scripting language, the overall impact was extremely positive. The role of Windows PowerShell continues to expand across Microsoft products, and it now extends into the newest Microsoft offerings, including the deployment and management of applications on the Azure cloud computing platform.

Windows PowerShell is built on top of the Microsoft .NET Framework and is implemented in the form of cmdlets, specialized .NET classes that contain the code to implement a particular operation such as the creation of a new mailbox or the enumeration of the processes that are currently active on a server. Applications implement Windows PowerShell support by providing sets of application-specific cmdlets that collectively represent the functionality required to support the application, or they can be used to access different data stores such as the file system or system registry. Cmdlets can be run separately or combined by piping the output generated by one cmdlet to become the input of the next. Cmdlets can also be combined into scripts (with a .ps1 file extension) to provide more comprehensive processing and logic or included in executables when the need exists to launch a standalone application. Many scripts are available on different Internet sites to assist with Exchange management.

How Exchange uses Windows PowerShell

From an Exchange perspective, Windows PowerShell provides a way to perform tasks quickly and simply in a variety of manners, from one-off interventions to process one or more Exchange objects to complex scripts to perform tasks such as mailbox provisioning. Most administrators cut their teeth on PowerShell by using the Exchange Management Shell (EMS) to do simple things, such as using Get-Mailbox to report on a mailbox’s
properties and Set-Mailbox or Set-CASMailbox to set a property, before moving on to the more esoteric commands to manipulate connectors or control the ability of devices to connect through ActiveSync and so on. The saying is that almost anything is possible with Windows PowerShell, and this is certainly true when you dedicate enough energy and time to mastering the language, not to mention the time necessary to scan the Internet for useful examples of scripts that can be adapted to meet your needs.

Prior to Exchange Server 2007, business logic was scattered in components throughout the product. The management console did things—even simple things like setting a property on a server—by using different code and logic than in the setup program, and the application programming interfaces (APIs) included in the product usually provided a third way to approach a problem. The result was a total lack of consistency, duplication of code, and a tremendous opportunity to create bugs in multiple places. In addition, administrators could not automate common tasks to meet the needs of their organization; essentially, if an Exchange engineer didn’t code something into the product, it couldn’t be done.

Figure 3-1 illustrates the central role Windows PowerShell now plays in the Exchange architecture and shows how it provides a central place to encapsulate business logic that underpins the Exchange setup program, the Exchange Administration Center (EAC), the mailbox options that users can update through Outlook Web App, and the Exchange Management Shell (EMS).

![Diagram of Exchange architecture](image)

**Figure 3-1** Windows PowerShell at the heart of Exchange
The way Exchange uses Windows PowerShell to implement business functionality is probably the most extensive of any Microsoft application. As explored throughout this book, the options presented by EAC to work with mailboxes, connectors, servers, and other objects invariably result in a call to one or more PowerShell cmdlets that actually do the work. The functionality presented to administrators, specialist users (those who perform a subset of administrative tasks such as maintaining user details), and normal users is all based on PowerShell.

The exact scope and range of the functionality presented to any individual user is determined by the permissions granted to him through role-based access control (RBAC). RBAC is designed to function across a range of environments, from a single-server organization to an organization composed of a mixture of on-premises and hosted servers. The need to accommodate such a wide range of environments is also why Microsoft has moved from local PowerShell (by which all commands are executed on a local server) to remote PowerShell (by which commands are redirected through Internet Information Services [IIS] for execution on a target server). The details of just how remote PowerShell and RBAC work together in EMS are covered shortly.

**Simplifying the implementation of new functionality**

The administrative interfaces in Exchange all lead to the same place and execute the same business logic. Apart from removing redundant and overlapping code, having a single place to implement business logic enables the Exchange engineers to concentrate on implementing new functionality rather than re-implementing features specifically for use by EAC, EMS, or the setup program. The approach enables Exchange to deliver a more consistent administrative environment and a comprehensive method to automate tasks to deal with mailboxes, databases, connectors, and all the other components that collectively make up an Exchange organization.

At the time of writing, Exchange 2013 RTM CU2 includes 965 cmdlets that are added to the standard set of Windows PowerShell cmdlets, including cmdlets to work with the system registry, file system, variables (including environmental variables), and so on that are available in an EMS session. Depending on the RBAC role groups of which your account is a member, the number of cmdlets available to you might vary.

Collectively, the set of EMS cmdlets manages the objects and the properties of the objects that form Exchange. Objects include mailboxes, servers, transport rules, connectors, and so on. You can determine the exact number of cmdlets Exchange owns by using the following command (this command doesn’t work with Exchange Online):

```
Get-ExCommand | Measure-Object | Select Count
```
INSIDE OUT  Finding the cmdlets available to you

As you learn in the discussion about RBAC in Chapter 4, “Role-based access control,” an EMS session allows you access only to the cmdlets and parameters that are defined in the roles included in the role groups of which your account is a member. Accounts that are highly permissioned, such as those belonging to the Organization Management role group, can use many more cmdlets than those that belong to a less-permissioned role group, such as Help Desk or Recipient Management. You can use this command to generate a full list of all the Exchange 2013 cmdlets your account can access:

```
Get-ExCommand > C:\Temp\ExCommands.txt
```

By comparison, Exchange 2007 includes 394 cmdlets, Exchange 2010, 584; and the RTM version of Exchange 2013, 958. The hundreds of new cmdlets included in Exchange 2013 and subsequently augmented through cumulative updates reflect the new functionality in the product such as the introduction of site mailboxes and data loss protection policies, along with the expansion of existing functionality such as the changes to compliance.

PowerShell use and syntax are fundamental skills for Exchange administrators to master. In fact, many Exchange administrators prefer EMS to EAC because of the additional flexibility that EMS provides. This chapter lays out the basics of Windows PowerShell and sets the stage for the examples of PowerShell found in other chapters. To begin, review how the Exchange management tools actually connect to PowerShell.

Using remote Windows PowerShell

Exchange 2010 began the necessary transformation from a model that assumed an administrator would always have some form of physical access to a server to the point at which remote management has become the norm. Remote PowerShell provides the fundamental building block for connectivity to remote systems. The combination of remote PowerShell and RBAC enables administrators to manage objects residing on a server in a remote data-center as easily as managing objects on a local server.

Note
You can think of Windows PowerShell as implemented in Exchange Server 2007 as “local PowerShell” because cmdlets are executed in a local process. The only element of remote access in Exchange 2007 is when you pass the –Server parameter to identify a server against which to execute a command. Even so, if data are needed from a remote server, such as fetching a set of mailbox objects, it is retrieved across the network and processed locally.
Exchange 2010 and Exchange 2013 support the remote execution of commands in a secure manner, using HTTPS and a Kerberos-based encryption mechanism that is easily manageable through firewalls (assuming that port 80 is open). Remote PowerShell is now used for all EMS sessions. Even if you are logged on to an Exchange server and want to use EMS to change a property of that server, EMS still creates a remote session on the local server to do the work. The same applies for EAC because Exchange creates a remote session when you log on to connect to a server in the local Active Directory site to retrieve information about the organization and then display it in the console. In effect, remote PowerShell has replaced local PowerShell for all server roles except edge servers. The sole exception is for commands used during setup, which continue to execute locally. Remote PowerShell separates business logic into code that runs on the client and code that runs on the Exchange server. The logic for replacing local PowerShell with the remote model is simple. Just as the change in Exchange 2007 forced all messages to flow through the transport system so that a common place existed to apply features such as transport rules, remote PowerShell forces all aspects of Exchange administration to flow through RBAC so that tight control can be achieved over the actions an administrator—or, indeed, a user—is allowed to perform.

An RBAC role group defines the set of administrative actions a user is allowed to perform inside Exchange and can be resolved into a set of PowerShell cmdlets the user is allowed to use within her PowerShell session. After it is created, a PowerShell session is populated with cmdlets by reference to the RBAC role groups of which the user is a member so that it will only ever include the cmdlets that have been assigned to an administrator through her membership in role groups. Thus, the fact that an administrator is not a member of a particular role group is reflected in that any PowerShell session she initiates will never be able to call the cmdlets associated with the management group. Consider the case of an administrator who is a member of the Organization Management role group, typically believed to be the all-powerful role for Exchange management. Even though membership in the Organization Management role group grants a user access to the vast majority of Exchange cmdlets, out of the box, it does not grant access to the set that controls movement of data into and out of mailboxes. To protect user data, the Mailbox Import-Export role has to be explicitly assigned to any user who wants to perform these tasks, even those who are already members of the Organization Management role group.

The need to support massively scalable, multitenant platforms such as Office 365 was a major influence on the Exchange move to remote PowerShell. Providing a secure and controllable mechanism to permit administrators to execute privileged commands to control the subset of objects they own inside an infrastructure that is controlled and managed by someone else is always a difficult task, especially when all the data have to pass across the Internet. When you use PowerShell to create a session with Exchange Online, the session is by default remote because you cannot connect to the servers running in Microsoft datacenters in any other way. After it accepts the credentials you provide in the connection request, PowerShell loads in the cmdlets you are allowed to use through membership in
RBAC role groups. The fact that Exchange Online restricts the number of cmdlets available to administrators when compared to on-premises deployments is immaterial. Exactly the same mechanism populates your PowerShell session with cmdlets whether you connect to an on-premises deployment or to a cloud-based service.

**CAUTION**

Until Microsoft removes the functionality, it is possible to use local PowerShell with the Exchange 2013 snap-in to perform management operations on a server. However, Microsoft is not testing local PowerShell with Exchange anymore, and it is possible that problems will appear in local PowerShell that will never be resolved. In addition, running local PowerShell means that you bypass the controls otherwise imposed by RBAC. Given the engineering and strategic focus on remote PowerShell, it makes sense for everyone to make the transition now and embrace this platform as the future of command-line Exchange management.

**Connecting to remote PowerShell**

To understand how remote PowerShell and RBAC work together, examine how an administrator might create a new mailbox on a remote server. In this example, the administrator works on a help desk and has been assigned a role that enables him to create new mailboxes and update the properties of existing mailboxes. Further, assume that the user’s account is enabled to use remote PowerShell. In many cases, people in specialist roles such as help desk personnel use EAC to perform tasks, but an experienced Exchange administrator might prefer to use a command-line interface because of its power and flexibility, especially when processing multiple objects, when compared to EAC.

Figure 3-2 lays out the various components remote PowerShell uses from the local PowerShell host on a workstation or server across the network to IIS and the PowerShell application running there. The other components are the PowerShell engine and the complete set of cmdlets available to Exchange 2013, the Exchange authorization library that handles the interpretation of roles in terms of the cmdlets that each RBAC role can use, and the Active Directory driver that reads data from Active Directory. For the purpose of this discussion, assume that the account has been assigned a role such as Recipient Management and is enabled for remote PowerShell. If you are unsure about the account’s status, you can enable it to use remote PowerShell as follows:

```powershell
Set-User -Identity AccountName -RemotePowerShellEnabled $True
```

All PowerShell sessions flow through IIS because even a local connection goes through localhost. All Exchange 2013 servers support IIS and the PowerShell virtual directory, or
vdir; all are members of the Exchange Trusted Subsystem security group and therefore can manipulate any object in the organization.

If you run EMS on a workstation or server on which the Exchange management components are installed, EMS creates a remote session automatically as part of its initialization process. If you run PowerShell on a workstation that doesn’t have the Exchange management components installed, you must specify the name of the server with which you want to work. This is done by using the New-PSSession cmdlet, passing the name of the server to which to connect in the `https://fqdn/PowerShell/` form. This cmdlet creates a secure, authenticated connection to IIS running on the target server and begins a session there by checking the role held by the account that initiates the connection.

![Remote PowerShell architecture](Figure 3-2 Remote PowerShell architecture)

When you run EMS on a server on which the Exchange 2013 management components are installed, the EMS initialization script creates an environment by executing the code in the RemoteExchange.ps1 script (in the Exchange binaries folder), which first attempts to create a remote session with the local host. If successful, it then identifies your account to Exchange, uses RBAC to determine the cmdlet set you are allowed to use, and so on.
The RemoteExchange.ps1 script is quite complex because it has to handle the initiation of sessions in different circumstances, so it’s easier to look at the code that connects a new session with Exchange Online to see how the basic requirements to create a remote PowerShell session are met. The following code defines the Connect-ExchangeOnline function, which is suitable for inclusion in your PowerShell profile (a document that initializes a session with common commands and settings that you might want to use during the session). In fact, this is how I connect to Exchange Online when I want to manage my tenant domain. Three commands are executed:

- Get-Credential gathers the name of the account with which to connect and its password.
- New-PSSession establishes the remote PowerShell session with Exchange Online.
- Import-PSSession imports the Exchange commands from the Exchange Online session into your current session.

```powershell
function Connect-ExchangeOnline
{
    $LiveCred = Get-Credential
    $global:Session365 = New-PSSession -ConfigurationName Microsoft.Exchange
-ConnectionUri https://ps.outlook.com/powershell/ -Credential $LiveCred
-Authentication Basic -AllowRedirection
    Import-PSSession $global:Session365
}
```

**How the initialization script finds a server**

When a user creates a remote PowerShell session on an Exchange server, the initialization script attempts to connect him to the same server. If the attempt to establish a connection with the local server fails, the initialization script then enumerates the full set of Exchange servers in the local site and attempts to make a connection to one of the servers chosen at random. If this attempt fails, the script moves on to the next server and continues until a successful connection is established or all available servers have been attempted and have failed. The initialization script works in the same way when executed on a workstation that has the Exchange management components installed on it, except that the initialization begins by randomly selecting one of the servers from the site.

IIS uses the RBAC mechanism to check the user’s role and associated permissions through the Exchange Authorization Library. The Exchange Authorization Library (or ADDriver) connects to Active Directory to use it as the definitive source of information about accounts and supplements these data with its knowledge about the Exchange-specific roles that
administrators have assigned to users. During a PowerShell session, ADDriver connects to a domain controller in the local site to fetch data from Active Directory and keeps this connection throughout the session (referred to as DC affinity). Many PowerShell cmdlets support the DomainController parameter to enable you to connect to a specific domain controller (specifying the fully qualified domain name [FQDN]) if the need arises.

Because this user works with mailboxes as defined by the Recipient Management role group, the set of cmdlets he can use includes commands with easily identified purposes such as New-Mailbox, Set-Mailbox, Get-Mailbox, and so on.

Tip
Permissions granted through RBAC are evaluated during session initialization. If you are assigned a new role, you have to create a new session with EMS or EAC before you can access the cmdlets made available through the newly assigned role.

Users are not normally aware that they are restricted in terms of available cmdlets unless they attempt to use one to which they do not have access. The point is that they shouldn’t care that they can’t use hundreds of cmdlets, many of which do obscure things such as setting properties on messaging connectors or performing one-off operations such as creating a new Database Availability Group (DAG) or mailbox database. Instead, RBAC makes sure that users can access only the cmdlets they need to perform their jobs.

Limiting user functionality

When a new PowerShell session is created, you see no evidence that your role has forced RBAC to restrict the cmdlet set or the parameters you can use with cmdlets because the initialization of a session progresses just as it would for a fully privileged user. However, after you start to execute cmdlets, you quickly realize that you can’t do as much as you’d like. For instance, if you log on with a restricted user account and attempt to use the Get-Mailbox cmdlet to fetch a list of mailboxes, all you’ll see is your own mailbox. This is logical because your role allows you to see details of your own mailbox but not others’. In the same way, if you then attempt to use the Set-Mailbox cmdlet to update a property that only administrators can access, you won’t be able to use even tab completion to reveal a restricted property. However, unless the default role assignment policy has been amended to block access to basic user options, you can use the Set-Mailbox cmdlet to update properties that are generally exposed for user update through Outlook Web App options, so (assuming JSmith is the alias for your mailbox) you’ll be able to do things such as this:

Set-Mailbox -Identity JSmith -MailTip 'Hello World'
or this:

```powershell
Set-Mailbox –Identity JSmith –Languages 'EN-US', 'EN-IE'
```

**INSIDE OUT**  You can do some things; you can’t do others

Somewhat strangely, you’ll also be able to execute `Get-MailboxStatistics` to report the number of items in your mailbox but not `Get-MailboxFolderStatistics` to report on the folders and the items that each contains. This is all controlled by RBAC, the roles that your account holds, and the scope for the roles in terms of the cmdlets and parameters defined in each role. From this discussion, you should now understand how critical RBAC is to remote PowerShell and, by extension, to every aspect of the Exchange 2013 management toolset.

**EMS basics**

Exchange 2013 RTM CU2 includes 965 cmdlets, but you’re not likely to use the vast majority of these simply because many are designed for one-time use. For example, after you configure a receive connector, you probably will not revisit the `Set-ReceiveConnector` cmdlet very often after the connector is working. However, you’ll use cmdlets such as `Get-Mailbox` daily. Some examples (in no particular order) of frequently used Exchange cmdlets are the following:

- **Get-ExchangeServer**  Return a list of Exchange servers in the organization.
- **Disable-Mailbox**  Disable a user’s mailbox.
- **Add-DistributionGroupMember**  Add a new member to a distribution group.
- **Set-Mailbox**  Set a property of a user’s mailbox.
- **Get-MailboxDatabase**  Retrieve properties of a mailbox database.
- **Get-MailboxStatistics**  Return statistics about user mailboxes such as the total item count, quota used, and so on.

Note the consistent syntax of verb (Get, Set, Move, Remove, or Disable) and noun (Mailbox, User, and so on). Along with commands that operate on objects, you find commands that help you work with data, such as `Where-Object`, `Sort-Object`, and `Group-Object`. `Where-Object`, `Sort-Object`, and `Group-Object` are commonly shortened by using their aliases of
Where, Sort, and Group. You can type **Help** followed by a cmdlet name at any time to get help on the syntax of the command.

**Tip**

When you start to write scripts, consider spelling out cmdlet names completely and avoiding the use of aliases. This is important because you can never know in what environment a script will be run and therefore cannot assume that an alias will be defined and available for use in your code.

The Exchange developers have provided very accessible help for the EMS cmdlets. Apart from using the Help cmdlet, there are other ways of seeking help. RBAC controls limit help content so that a user sees help only for the set of cmdlets available to the roles that user holds. You can do the following:

- Use the Get-Command cmdlet to list the cmdlets you can use with different objects. The set of cmdlets will be limited to whatever is permitted by the RBAC roles held by your account. For example, Get-Command *contact* lists all the cmdlets available to work with contacts (shown in the following example). You can also use the shortened alias of gcm for Get-Command.

<table>
<thead>
<tr>
<th>CommandType</th>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Disable-MailContact</td>
<td>...</td>
</tr>
<tr>
<td>Function</td>
<td>Enable-MailContact</td>
<td>...</td>
</tr>
<tr>
<td>Function</td>
<td>Get-Contact</td>
<td>...</td>
</tr>
<tr>
<td>Function</td>
<td>Get-MailContact</td>
<td>...</td>
</tr>
<tr>
<td>Function</td>
<td>New-MailContact</td>
<td>...</td>
</tr>
<tr>
<td>Function</td>
<td>Remove-MailContact</td>
<td>...</td>
</tr>
<tr>
<td>Function</td>
<td>Set-Contact</td>
<td>...</td>
</tr>
<tr>
<td>Function</td>
<td>Set-MailContact</td>
<td>...</td>
</tr>
</tbody>
</table>

- Use the /detailed switch to get more detailed help about a cmdlet. For example: Get-Help Get-CASMailbox –Detailed.

- Use the /full switch to have EMS return every bit of information it knows about a cmdlet. For example, Get-Help Get-DistributionGroup –Full.

- Use the /examples switch to see whatever examples of a cmdlet in use EMS help includes. For example, Get-Help Get-MailboxServer –Examples.
Use the /parameter switch to get information about a selected parameter for a cmdlet. For example, Get-Help Get-Mailbox –Parameter Server. This switch supports wildcards, so you can do something like Get-Help Set-Mailbox –Parameter *Quota*.

**INSIDE OUT  Getting to know the cmdlets**

You will probably begin by using the /full switch to retrieve all available help for a cmdlet to get to know what each cmdlet does. After you learn more about the cmdlet, you can move on to the default view as you become more accustomed to working with EMS. Remember that the Exchange help file contains information about all the EMS cmdlets. The advantage of using the help file (which is always present on a server) is that you can use the help file’s index to search for specific entries.

Most of the time, you will probably work with commands by invoking EMS interactively and then typing whatever individual commands or scripts are necessary to perform a task. The user interface of EMS is based on the Win32 console with the addition of features such as customizable tab completion for commands. After you become accustomed to working with EMS, things flow smoothly, and work is easy. It is then usually faster to start EMS and issue the necessary code to change a property on a mailbox or a server than to start EAC and navigate to the right place to make the change through the graphical user interface (GUI).

**Tip**

Working through EMS is especially valuable if you have to perform management operations across an extended network link when waiting for the GUI to display can be painful. If you have a programmatic mind, you can also call EMS cmdlets through C# code, which is how Microsoft invokes them in EAC and other places throughout Exchange, such as to set up servers and databases in the setup program. (The blog Glen Scales writes at [http://gsexdev.blogspot.com/](http://gsexdev.blogspot.com/) provides many good examples of how to call EMS cmdlets from code.) In the past, the different groups that contributed to Exchange had to build their own programming interfaces, whereas now everyone uses PowerShell.

You can see that EMS focuses on performing tasks rather than taking the more object-focused approach implemented in the GUI, something that reflects a desire to accommodate administrators who think about how to do things rather than how to work with objects. After all, it is human nature to think in terms of the task of moving a mailbox to a different server rather than thinking about how to manipulate the properties of a mailbox object to reflect its new location.
Cmdlets accept structured pipelined input from one another in a common manner to allow them to process data in a consistent manner, no matter which cmdlet provides the data. Programmers therefore do not have to reformat data for input to specific cmdlets, so the task of assembling different cmdlets into a script to do a job is much easier. Microsoft built PowerShell around the concept of objects, so objects are accepted as input, and the output is in the form of objects that you can then pipe to other cmdlets. Even if the output from a cmdlet looks like plaintext, what you see is one or more objects that you can manipulate in a much more powerful manner than you can ever work with text output. The implementation is elegant.

**Command editing**

It should be apparent that you could do a lot of typing to enter commands into PowerShell, make the inevitable mistakes, correct them, and try again. To make the task a little easier, PowerShell supports the same kind of command-line editing as the Win32 console (CMD) does. Some of the more important keys you can use are described in Table 3-1.

**TABLE 3-1 Command editing keystrokes for PowerShell**

<table>
<thead>
<tr>
<th>Keyboard command</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2</td>
<td>Creates a new command based on your last command. A pop-up screen appears in which to enter a character. PowerShell then creates a new command, using the last entered command up to the character you specify. For example, if the last command is Get-MailboxStatistics –Identity TRedmond, and you enter F2 followed by c, PowerShell inserts &quot;Get-MailboxStatistics&quot;. You can then complete the command as you like.</td>
</tr>
<tr>
<td>F4</td>
<td>Deletes characters in the current command up to a specified position. For example, if the cursor is located at the &quot;M&quot; of Get-MailboxStatistics, and you enter F4 followed by x, PowerShell deletes &quot;Mailbo&quot; and the result is &quot;Get-xStatistics&quot;. Although this example wouldn’t result in a useful command, F4 is useful when you need to edit many parameters in a complex command.</td>
</tr>
<tr>
<td>F7</td>
<td>Opens a list of the last 50 commands used in the current session to enable you to select a command for reuse.</td>
</tr>
<tr>
<td>F8</td>
<td>Moves backward through the command history.</td>
</tr>
<tr>
<td>Tab</td>
<td>Requests PowerShell to complete a command based on what you’ve typed.</td>
</tr>
<tr>
<td>Left/Right arrows</td>
<td>Moves the cursor left and right through the current command line.</td>
</tr>
<tr>
<td>Up/Down arrows</td>
<td>Moves up and down through the history of previous commands.</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes the character under the cursor.</td>
</tr>
<tr>
<td>Insert</td>
<td>Toggles between character insert and character overwrite mode.</td>
</tr>
<tr>
<td>Backspace</td>
<td>Deletes the character before the cursor.</td>
</tr>
</tbody>
</table>
Most of these keys are straightforward. The two most interesting keys are F7 and Tab. F7 opens a list of the last 50 commands you have run in the current session (Figure 3-3) so that you can both see what you’ve done in the immediate past and select one of the commands to re-execute. You can type a couple of characters into the F7 list, and EMS will look for the first matching command, or you can use the Up and Down arrows to navigate through the command history. At times, it’s more convenient to use Up and Down arrows because you can retrieve more commands and edit a command before executing it. (F7 selects the command and executes it immediately.)

![Figure 3-3 Using F7 to recall EMS commands](image)

### INSIDE OUT

**An easy way to type a command**

Tab completion is a wonderful feature that Windows PowerShell inherited from CMD. You can partially enter a command and then press Tab to have PowerShell fill in the rest of the cmdlet name followed by its parameters. For example, type:

```powershell
get-dist
```

This isn’t the name of a valid cmdlet, but it is the root of several cmdlets, so when you press Tab, PowerShell completes the first valid cmdlet that matches and inserts:

```powershell
get-distributiongroup
```

If you press Tab again, PowerShell moves to the next cmdlet that matches and inserts:

```powershell
get-distributiongroupmember
```

If you press Tab again, PowerShell returns to Get-DistributionGroup because there are only two valid matches. PowerShell also supports completion for parameters. If you insert a dash to indicate a parameter value after Get-DistributionGroup and press Tab,
PowerShell starts with the first parameter and continues through all valid parameters. If you press Tab too many times and pass the parameter you want to use, you can press Shift+Tab to go back through the parameter list. If you add some characters to help PowerShell identify the parameter, it attempts to complete using that value. For example:

PowerShell completes Get-DistributionGroup -Ma into the command
Get-DistributionGroup -ManagedBy.

Even better, tab completion is context-sensitive, so it understands the structure of the object you are navigating. For example, if you want to move through the system registry, tab completion understands the hive structure, so you can type a location in the registry and then use the Tab key to move through the available choices from that point. For example, type:

CD HKLM:\Software\Microsoft\Exchange

Now press Tab, and PowerShell leads you through all the registry locations Exchange uses.

Windows PowerShell supports both named and positional parameters. Identifiers are a good example of a positional parameter. For example, if you enter Get-Mailbox Tony, PowerShell assumes that Tony is the value for the –Identity parameter.

Finally, PowerShell completes variables and even the properties of variables (such as their length) in a way similar to how the Microsoft Visual Studio IntelliSense feature works. If you type the incomplete name of a variable and press Tab, PowerShell completes it from the list of known variables. For example, if you fill a variable with details of a mailbox as in the following:

$Mailbox = Get-Mailbox –Identity Redmond

and then type $Ma and press Tab, PowerShell completes it and returns $Mailbox. This is a useful feature if you forget the names of variables you’ve defined. To see how properties are completed, type:

$Mailbox.Di

Pressing Tab now will request PowerShell to go through the list of properties beginning with Di. For a mailbox, the list is DistinguishedName and DisplayName.
Handling information EMS returns

Any cmdlet such as Get-EventLog that retrieves some information about an object will output a default set of properties about the object (or references to an object). Sometimes those properties are not exactly the ones you want to examine, so you will inevitably use the Format-List and Format-Table cmdlets to expand the set of properties a command returns. For example, if you use the Get-Mailbox cmdlet to view the properties of a mailbox, the information returned isn’t interesting:

```powershell
Get-Mailbox -Identity TRedmond
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Alias</th>
<th>ServerName</th>
<th>ProhibitSendQuota</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tony Redmond</td>
<td>TRedmond</td>
<td>ExServer1</td>
<td>unlimited</td>
</tr>
</tbody>
</table>

However, if you pipe the output to Format-List, you see much more information—far too much to review comfortably on screen—so it’s better to pipe the output to a text file and compare it at your leisure.

The Get-Mailbox cmdlet does not return every property you can set on a user object because EMS differentiates between general Active Directory properties for a user object and those that are specific to Exchange. For example, Get-Mailbox does not list the Office property for a user because every user object in Active Directory has this property regardless of whether it is mail-enabled. Thus, if you want to retrieve or update the Office property, you have to use the Get-User and Set-User cmdlets, respectively. The same differentiation exists for groups and contacts when the Get-Group/Set-Group and Get-Contact/Set-Contact cmdlets are available.

Selective output

It is easy to list every property, but when you have limited screen space, you need to be more selective about the properties you want to output, and that’s why it’s often a good idea to use the Select-Object cmdlet to select the data you need before you pipe to Format-Table. In this case, you use the Select alias for Select-Object just because this cmdlet is used so often and it is nice to use shorthand.

```powershell
Get-Mailbox -Identity Pelton | Select Name, PrimarySmtpAddress, Database
```

<table>
<thead>
<tr>
<th>Name</th>
<th>PrimarySmtpAddress</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>David Pelton</td>
<td><a href="mailto:David.Pelton@contoso.com">David.Pelton@contoso.com</a></td>
<td>ExServer1\DB1</td>
</tr>
</tbody>
</table>
PowerShell output can obscure data because it contains too many spaces. For example:

```
Get-ExchangeServer
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Site</th>
<th>ServerRole</th>
<th>Edition</th>
<th>AdminDisplayVersion</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXSERVER1</td>
<td>contoso.com/Conf.... Mailbox, ClientAccess</td>
<td>Mailbox,... Enterprise</td>
<td>Version 15.0 (Bu...</td>
<td></td>
</tr>
<tr>
<td>EXSERVER2</td>
<td>contoso.com/Conf.... Mailbox, ClientAccess</td>
<td>Mailbox,... Enterprise</td>
<td>Version 15.0 (Bu...</td>
<td></td>
</tr>
</tbody>
</table>

To force PowerShell to remove spaces and display more useful data, pipe the output to the Format-Table cmdlet and use the –AutoSize parameter to fit the output columns into the available space:

```
Get-ExchangeServer | Format-Table -AutoSize
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Site</th>
<th>ServerRole</th>
<th>Edition</th>
<th>AdminDisplayVersion</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXSERVER1 contoso.com/Conf....</td>
<td>Mailbox, ClientAccess Enterprise</td>
<td>Version 1...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXSERVER2 contoso.com/Conf....</td>
<td>Mailbox, ClientAccess Enterprise</td>
<td>Version 1...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Another way of extracting and then working with data is to direct the output of a command into a variable, in which case you have a complete picture of the object’s properties in the variable. For example, this command loads all the available information about the ExServer2 server into the $Server variable:

```
$Server = Get-ExchangeServer –Identity 'ExServer2' -Status
```

You can extract additional information about the server to use by including the name of the property in which you’re interested. (Specifying the –Status parameter requests Get-ExchangeServer to provide some additional information about the current domain controller and global catalog the server is using.) You can also use a variable as an array and populate the array with a call to a command.

In this example, you populate a $Mailboxes array with a call to Get-Mailbox, using a filter to extract details of all the mailboxes stored in a particular database. This output is a good example of how cmdlets can generate individual objects or an array of objects with each object being individually accessible within the array.

```
$Mailboxes = Get-Mailbox –Database DB2
```
When it is populated, you can then navigate through the array as follows:

$Mailboxes[0]
$Mailboxes[1]
$Mailboxes[2] etc etc etc.

You can reference specific properties of the objects by using the "." operator.

$Mailbox[2].Name
$Mailbox[53].PrimarySmtpAddress

**INSIDE OUT** Finding what you want when there’s a lot of output

The output from a cmdlet such as Get-Mailbox can easily result in a lot of data that are hard to read to find the piece of information in which you are really interested. One technique that helps is to pipe the output to the Out-String cmdlet and then use the FindStr cmdlet to search the output for a particular term. For example, here’s how to use the two cmdlets to search the output from Get-Mailbox to find a particular term. In this instance, EMS lists any occurrence of the word “Tony” if it exists in the list of mailbox names Get-Mailbox returns:

Get-Mailbox | Out-String | FindStr "Tony"

By default, EMS truncates the output of multivalue properties after 16 values. For example:

Get-Mailbox -Identity 'Pelton, David' | Format-List Name, EmailAddresses

<table>
<thead>
<tr>
<th>Name</th>
<th>Pelton, David</th>
</tr>
</thead>
<tbody>
<tr>
<td>EmailAddresses</td>
<td>{smtp:<a href="mailto:dp12@contoso.com">dp12@contoso.com</a>, smtp:<a href="mailto:dp11@contoso.com">dp11@contoso.com</a>, smtp:<a href="mailto:dp10@contoso.com">dp10@contoso.com</a>, smtp:<a href="mailto:dp9@contoso.com">dp9@contoso.com</a>, smtp:<a href="mailto:dp8@contoso.com">dp8@contoso.com</a>, smtp:<a href="mailto:dp7@contoso.com">dp7@contoso.com</a>, smtp:<a href="mailto:dp6@contoso.com">dp6@contoso.com</a>, smtp:<a href="mailto:dp4@contoso.com">dp4@contoso.com</a>, smtp:<a href="mailto:dp5@contoso.com">dp5@contoso.com</a>, smtp:<a href="mailto:dp3@contoso.com">dp3@contoso.com</a>, smtp:<a href="mailto:dp2@contoso.com">dp2@contoso.com</a>, smtp:<a href="mailto:dp1@contoso.com">dp1@contoso.com</a>...}</td>
</tr>
</tbody>
</table>

Truncation can hide some valuable data. In the preceding example, many of the email addresses are defined for a mailbox, but the default Simple Mail Transfer Protocol (SMTP) address is not shown. If this limitation becomes a concern, you can force EMS to output more values for a property by amending a $FormatEnumerationLimit variable. This variable is defined in the EMS initialization script (\bin\Exchange.ps1), and the default value of 16 is usually more than sufficient for normal purposes. If you want to see more variables, you can set the variable to a different limit or set it to -1 to instruct EMS that it can enumerate as many values as are available for any property. For example:

$FormatEnumerationLimit = -1
Get-Mailbox -Identity 'Pelton, David' | Format-List Name, EmailAddresses
Using common and user-defined variables

PowerShell includes a number of variables you will use a lot. `$True` and `$False` are variables you can pass to shell commands and scripts to check for true and false conditions. Usually, `$True` is equivalent to setting a check box for an option in EMC, and `$False` is equivalent to clearing a check box. If you prefer numeric values, you can replace `$True` and `$False` with 1 (one) and 0 (zero), respectively. Other global variables you commonly meet as you work with PowerShell include `$Null` (no value), `$home`, which returns the user’s home folder, and `$pwd`, which returns the current working folder. Important Exchange variables include the following:

- **$ExBin**  Points to the directory in which Exchange binaries and other important files are kept. On an Exchange 2013 server, this variable normally resolves to disk: \Program Files\Microsoft\Exchange Server\V15\bin.

- **$ExScripts**  Points to the directory in which important Exchange .ps1 scripts are kept. On an Exchange 2013 server, this variable resolves to disk: \Program Files\Microsoft\Exchange Server\V15\Scripts.

- **$ExInstall**  Points to the root directory for Exchange. On an Exchange 2013 server, this variable resolves to disk: \Program Files\Microsoft\Exchange Server\V15.

You can use these variables to access files in these directories. For example, to see a list of scripts Exchange provides, type **Dir $ExScripts**.

Checking that a value is `$True` or `$False` is a common occurrence. For positive conditions, you can shorten the check by just passing the property against which to check, and PowerShell will assume that you want to check whether it is true. For example, assume that you want to find out which mailboxes are enabled to use Outlook Web App. You can use this command and, as you can see, there is no mention of `$True`, but it works:

```
Get-CASMailbox | Where-Object {$_._OWAEnabled} | Select Name
```

Note the use of `$_` in the last command. `$_` is a very important variable because it points to the current object in the pipeline. Scripting languages on other platforms such as UNIX and Linux also support pipelines, which compose complex commands by allowing the output of
one command to be passed as the input to another. The | operator indicates that a pipeline is in place. Data are passed as fully formed objects rather than as a text stream. This enables PowerShell to operate on the full structure of data that are pipelined, including the attributes and types that define the objects piped from one cmdlet to another.

For example, if you create a filter to look for people in a certain department because you want to update the name of the department, you might do this:

```
Get-User | Where-Object {$_.Department -eq 'Legal'} | Set-User -Department 'Law'
```

The Department property is prefixed with $_ to indicate that you want to check this property for every object the call to Get-User passes through the pipeline. You actually use $_ as the prefix because it includes the “.” operator to specify that you want to access a property. If you just passed $$_ the comparison would not work because PowerShell would compare “Legal” against the complete object.

User-defined variables can be integer, decimal, or string—you decide by passing a value to the variable you want to use. For example:

```
$Tony = 'Tony Redmond'
$Figure = 15.16
```

This creates a string variable, and the second variable holds a decimal value. Variables are case-insensitive and case-preserving. Using the preceding example, you can refer to $Tony as $TONEY or $tony or even $Tony, and PowerShell will refer to the same variable. Variables are local unless you declare them to be global by prefixing them with Global, as in:

```
$Global:Tony = 'Tony Redmond'
```

When a variable is global, you can reference it interactively and in scripts you can call from anywhere.

---

A word of caution about PowerShell and quotation marks

Be careful how you use quotation marks in PowerShell because although it might appear that double and single quotation marks are interchangeable, there is a subtle difference that might catch you out. Single quotation marks represent a literal string, one that PowerShell will use exactly as you provide it. Double quotation marks mean that PowerShell should examine the string and resolve any variable it finds inside through a process called variable expansion. Consider this example:

```
$n = Date
$n1 = 'Right now, it is $n'
```

Right now it is $n
Right now, it is Tue Jan 16 17:59:54 2013

Can you see the difference a little quotation mark makes? Best practice is to use single quotation marks whenever you are sure that you want a string variable to stay exactly as you have typed it and to use double quotation marks elsewhere. Be careful about using editors that insert smart quotation marks because PowerShell cannot deal with them; it is best to use a simple text editor whenever you create or edit a script. You cannot mix and match the different types of quotation marks to enclose a variable because PowerShell will refuse to accept the command. You will not do any great harm if you use double quotation marks instead of single quotation marks, but it is best to use single quotation marks as the default.

Tip
Do not include hyphens when you name variables because PowerShell interprets the hyphens as parameters. In other words, $ServerName is a good name for a variable, but $Server-Name is not.

Like any good scripting language, PowerShell supports conditional checking with IF and ELSEIF that you will mostly use in scripts. It’s easy to generate code that goes through a certain number of iterations with constructs such as 1..100 | ForEach-Object <command…>. You will see examples of these constructs as you see more sophisticated PowerShell code in later chapters.

Using PowerShell ISE with Exchange

If you don’t like the bare-bones nature of EMS, you might prefer to use ISE, the PowerShell Integrated Scripting Environment. ISE is installed on Windows 2008 R2 SP1 and Windows 2012 servers to provide a GUI for PowerShell that allows users to write, test, and debug scripts. PowerShell ISE is also installed by default on Windows 7 and Windows 8 workstations.

ISE supports multiline editing, tab completion, syntax coloring (or highlighting of different parts of commands), context-sensitive help, and keyboard shortcuts. Because of its debug features, ISE is a good way to write complex scripts for use with Exchange 2013. All the code included in this book can be worked on through ISE.
When you start ISE, it has no knowledge of Exchange or how to create the kind of remote session with an Exchange server in the way EMS does when it starts. Some work is therefore necessary to integrate ISE with Exchange. The easiest way to do this is to insert some code in the PowerShell profile so that ISE learns enough about Exchange when it initializes to access Exchange when you need it to.

The code you need to use with ISE is very similar to the code you met earlier when discussing the basics of creating a remote PowerShell session. Start ISE and type Notepad $Profile to edit your PowerShell profile, and then insert the following code (amending the reference to contoso.com to reflect your own environment):

```powershell
$PSISE.CurrentPowerShellTab.AddOnsMenu.SubMenus.Add("Connect to Exchange", {
    $user = Get-Credential
    $Server = Read-Host "Connect to what Exchange server "
    $connectpoint = $Server + ".contoso.com/PowerShell/"
    $ExSession = New-PSSession -ConfigurationName Microsoft.Exchange -ConnectionUri $connectpoint -Credential $user
    Import-PSSession $ExSession
    }, "Control+Alt+1"
})
```

The code defines a new menu choice called Connect to Exchange that appears on the ISE Add-ins menu. The option can also be invoked with the Control/Alt/1 key combination. In either case, when invoked, the code prompts for user credentials and the server to which to connect and then initiates a new remote PowerShell session with the selected Exchange server. After the connection is established, you can work as with EMS except that extra information and facilities are available to you, such as a context-sensitive list of cmdlets that appears when you start typing a cmdlet name (Figure 3-4).
Identities

You might have noticed the –Identity parameter in some of the cmdlets you have explored so far. In many cases, a call to an Exchange cmdlet results in a set of objects being returned (for example, all the mailboxes on a server). In these instances, you might need to identify a specific object within the chosen set with which to work. (Think of a pointer to an item in an array.) For example, if you issue the Get-ExchangeServer cmdlet, you retrieve a list of all the Exchange servers in the organization. If you want to work with one server, you have to tell EMS which server you want to select by passing its identity. For example, to work with just the server named ExServer1:

Get-ExchangeServer –Identity 'ExServer1'

Apart from its obvious use to identify the object with which you want to work, –Identity has a special meaning within PowerShell because it is a positional parameter. You can specify the parameter’s value without specifying the parameter’s name, so the example previously used is just as valid if you use:

Get-ExchangeServer 'ExServer1'
INSIDE OUT  Best practice to include the –Identity parameter

Although you might find it faster to omit the –Identity parameter when you’re working interactively with EMS, it is best practice always to include the –Identity parameter when you write code for reusable scripts because this ensures that there is no possibility that another administrator or programmer will mistake the value passed for the identity for anything else.

If you want, you can retrieve a list of objects and store them in a variable and retrieve the values as you wish. The variable holds the objects as an array. For example, to populate a variable with a set of mailboxes hosted by a server:

```powershell
$Mbx= Get-Mailbox -Server 'ExServer1'
```

To retrieve the different objects in the array, pass the number of the object with which you want to work, starting from zero. For example, to fetch the first mailbox in the array:

```powershell
$Mbx[0]
```

You can be more specific and ask for one of the object’s properties. For example, to get the identity of the first mailbox in the array:

```powershell
$Mbx[0].Identity
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsDeleted</td>
<td>False</td>
</tr>
<tr>
<td>Rdn</td>
<td>CN=Eoin P. Redmond</td>
</tr>
<tr>
<td>Parent</td>
<td>contoso.com/Exchange Mailboxes</td>
</tr>
<tr>
<td>Depth</td>
<td>3</td>
</tr>
<tr>
<td>DistinguishedName</td>
<td>CN=Eoin P. Redmond,OU=Exchange Mailboxes,DC=contoso,DC=com</td>
</tr>
<tr>
<td>IsRelativeDn</td>
<td>False</td>
</tr>
<tr>
<td>DomainId</td>
<td>contoso.com</td>
</tr>
<tr>
<td>ObjectGuid</td>
<td>0bcd15b3-c418-43be-b678-2658614f732b</td>
</tr>
<tr>
<td>Name</td>
<td>Eoin P. Redmond</td>
</tr>
</tbody>
</table>

You might be surprised by the amount of information returned here for the mailbox’s identity (it’s all defined in the schema), but it contains all the ways you can navigate to this object through its relative distinguished name (shown here as the rdn property), distinguished name, globally unique identifier (GUID), and name. Normally, you’ll just use the name of a mailbox to find it, but you can use the other methods, and Exchange will find the mailbox. There is no requirement to parse out a specific piece of the identity you want to
use or to trim values; PowerShell does it all for you. For example, you can use an identity to
discover the groups to which a user belongs. Here’s the code:

```powershell
$U = (Get-User -Identity TRedmond).Identity; Get-Group | Where-Object {$_.Members -eq $U}
```

The Get-User cmdlet loads the user’s identity into a variable, and then the Get-Group and
the Where-Object cmdlets scan all groups to discover any that include the user in their
membership. Scanning the membership list of groups to discover string matches is never
going to be as quick (and will get slower as the number of groups in the forest grows)
because a string compare will never get close to the backward pointers that consoles such
as Active Directory Users and Computers or EMC use to display group membership in terms
of speed of access, so don’t be surprised. Scanning for group membership in this way takes
some time to complete.

If you don’t like user-friendly forms such as email addresses or mailbox names, Exchange
also allows you to use GUIDs as identifiers. Because they are obscure and long, GUIDs are
difficult to type, but you can still use them. One slightly complicating factor is that you
must know which GUID to use where. You might want the GUID that points to a user’s mail-
box, the GUID pointing to her Active Directory account, or even the one pointing to her
archive mailbox. For example, this command displays all GUIDs registered for a mailbox:

```powershell
Get-Mailbox -Identity 'Tony Redmond' | Format-List *Guid*
```

```
ExchangeGuid        : c2c4a3b5-c1a6-5a17-971d-8549123a78d0
ArchiveGuid         : 00000000-0000-0000-0000-000000000000
DisabledArchiveGuid : 00000000-0000-0000-0000-000000000000
Guid                : 288617d1-4592-4211-bb20-26ab755458c8
```

The ExchangeGuid property points to the user’s mailbox. This is a tremendously important
property because the GUID pointing to a mailbox can be guaranteed to be unique across
an Exchange organization, which is why the Store uses this value to locate a user’s mailbox.
It’s also why Outlook users see the ExchangeGuid of their mailbox instead of the server
name when viewing the server name property shown when viewing the server settings of
an Exchange 2013 mailbox (Figure 3-5).
It is confusing, but if you run Get-MailboxStatistics to retrieve summary details of the contents of a mailbox, EMS returns a MailboxGuid property. This is the same value as the ExchangeGuid when reported by Get-Mailbox. Why Microsoft felt that two names were required for the same GUID is beyond me.

The Guid property identifies the user’s Active Directory account and thus provides the essential link between a mailbox and an account. In this case, the ArchiveGuid is shown as all zeros, so no archive mailbox is associated with this mailbox. The DisabledArchiveGuid value is also all zeros. This GUID is used only when a user has been assigned an archive mailbox that was subsequently disabled for some reason. Exchange maintains the GUID so the archive can be reconnected to the mailbox up to the point at which it is permanently removed from a database after the expiry of the deleted mailboxes’ retention period.

Now that you know what the GUIDs are, you could use them to reference a mailbox. For example:

```
$GUID = (Get-Mailbox -Identity 'Tony Redmond').Guid
Get-User | Where {$_ Guid -eq $GUID} | Format-Table Name
```

The great thing about identities is that you sometimes don’t need to use them. This situation occurs when you pipe information from one cmdlet for processing by another because the shell understands that it needs to operate on the current object that has been fetched
through the pipe. For example, this command pipes a list of mailbox identities passed in strings to the Set-Mailbox cmdlet:

"TRedmond", "JSmith", "J Doe" | Set-Mailbox -Office "Dublin"

Piping

You'll pipe output from one cmdlet to another frequently as you work with Exchange data. The important thing to remember is that PowerShell outputs fully formed objects that can be manipulated when fed as input to other cmdlets through the pipeline. This wouldn't be possible if PowerShell output text strings. For example, assume that you want to change the value of the Office property for a set of users who have moved to a new building. It would be tedious if you had to fetch the identity of each user individually, determine each identity, and then pass the value to make the change to each user’s properties. A simple pipe works because PowerShell knows that it can use the stream of data from one command to identify the objects it has to process with another. Here's how you might update the Office property for a complete set of users without any mention of an identity. You'll see that the two cmdlets that do the work are separated by the pipe character, "|". This is the character that tells PowerShell to pipe the output from the first cmdlet to become the input to the second.

Get-User -Filter {Office -eq 'Building A'} | Set-User -Office "Building B"

Too many objects

By default, EMS returns up to 1,000 objects in response to cmdlets. (The value in Exchange 2007 is 5,000.) Therefore, if you run a cmdlet such as Get-Mailbox, Exchange will return up to 1,000 mailboxes if they are available. If you work in a small Exchange organization that supports fewer than 1,000 mailboxes, you don’t need to worry too much about the number of objects you have to deal with because PowerShell will likely return relatively few objects, and things usually progress quickly. However, it’s a different situation in large organizations, in which you have to pay attention to the filters you specify to retrieve data or override the default limit for returned objects by specifying the ResultSize parameter for cmdlets. For example, to let EMS return as many mailboxes as it can find, you could use a command like this:

Get-Mailbox -ResultSize Unlimited

This command will work, but it will be very slow because EMS has to read every mailbox in the organization. Think about how long this might take to execute in an organization that supports more than 300,000 mailboxes. In these situations it’s always better to specify a filter to restrict the number of objects EMS looks for and returns.
OPATH filters

OPATH is the basic syntax used for PowerShell queries. It is similar in concept to but uses different syntax from Lightweight Directory Access Protocol (LDAP) queries. Dynamic distribution groups (see Chapter 6, “Groups and other objects”) also use OPATH queries to locate objects in Active Directory when the transport system builds addressee lists to deliver message addresses to these groups.

Some base guidelines about the syntax OPATH queries are as follows:

- OPATH requires a hyphen before –and, –or, and –not operators.
- Comparison operators include –eq (equal), –ne (not equal), –lt (less than), –gt (greater than), –like (like), –ilike, and –notlike. –Like and –notlike are wildcard string compares. –iLike and –inotlike are case-insensitive.
- Filters should be expressed within braces; for example, {Office –eq ‘London’}.

You’ll see many more examples of OPATH queries in the remainder of this book.

Server-side and client-side filters

Windows PowerShell supports server-side and client-side filters. There’s a big difference in performance between the two types of filters, especially when you have to process more than a hundred objects. Client-side filters are the default. Any code that uses the Where cmdlet executes a client-side filter. Client-side filters request data from a server and then perform the filtering on the client. This is an effective approach if you only have 10 or 15 objects to process, but it obviously doesn’t scale too well as the number of objects increases.

Server-side filters have better scalability because the request for data forces the server to return a filtered data set to the client. Because Exchange servers often have to deal with tens of thousands of objects, a number of the Exchange cmdlets support server-side filters. If a cmdlet supports the –Filter parameter, it supports server-side filters. Usually, these are cmdlets that deal with objects that output large numbers, such as mail-enabled recipients or message queues. All the precanned filters generated for dynamic distribution groups, address lists, and email address policies use server-side filters.

As an example of server-side and client-side filtering in action, two methods are available to find all the mailboxes with “James” in their name, as demonstrated in these commands:

Get-Mailbox –Filter {Name –like '*James*'} –ResultSize 5000
Get-Mailbox –ResultSize 5000 | Where {$_.Name –like '*James*'}
On the surface, these two pieces of code seem reasonably similar, but they are very different in reality. The first difference is that the first code example uses a server-side filter, and the second uses a client-side filter. The second difference is that the two filter types can generate very different results because of the way the filters operate. If you omit the –ResultSize parameter, the same query is generated: Find all the mailboxes with a name that contains “James.” (The ResultSize parameter in the first example limits the total number of objects returned to 5,000.) However, if you time both queries, the server-side filter invariably executes faster than the client-side filter, largely because fewer data are transferred between server and client. To understand why the filters generate different results, you have to appreciate how the filters work:

- The server-side filter returns the first 5,000 mailboxes it finds that include “James” in the mailbox name.

- The client-side filter fetches data for the first 5,000 mailboxes and then applies the filter to find the mailboxes that include “James” in the mailbox name. However, the filter applies only to the set the client fetched and might not find all the mailboxes you actually want to discover.

Even though you ask the server-side filter to do more work (working with any reasonably sized set of mailboxes, the server-side filter will have to process significantly more data to find the first 5,000 mailboxes that match), it still executes faster. For example, when I executed similar commands within a very large Exchange organization (170,000 mailboxes), the server-side filter completed processing in 43 seconds, whereas the client-side filter completed in 81 seconds. The rule here is that the effect of server-side filtering gets better as the number of objects increases.

**INSIDE OUT**  **PowerShell and memory limits**

Another aspect to consider is that PowerShell cannot fetch and cache data on disk temporarily the way a database might. This is not an issue if you want to process only a few objects, but it can lead to memory issues if you attempt to process tens of thousands of mailboxes at one time, especially if you use client-side filters and want to pipe the output to another command. In this case, you ask PowerShell to find all the objects that match the specified filter, store the data in memory, process the data, and pipe the matching objects to the second command. Experience shows that these operations can cause PowerShell to complain that it is running out of memory. This is likely to be one of the growing pains through which all software goes and, apart from using loops to process data, no good solution to the memory exhaustion problem is available today.
Sometimes people make the mistake of assuming that client-side filters are faster because server-side filters provide the data in one motion after the server processes all the data. You therefore wait for a while without seeing anything and then see all the filtered records at one time. By comparison, client-side filters fetch and filter data continuously, so you see output as the command finds each matching record. However, the important indicator of performance is how long each type of filter takes to complete, and server-side filters are always faster.

The commands you are most likely to use with server-side filters are as follows:

- **Get-User**  Retrieve basic Active Directory properties for any user account, including mail-enabled accounts.

- **Get-Mailbox**  Retrieve Exchange-specific properties for mailboxes.

- **Get-DistributionGroup**  Retrieve Exchange-specific properties for mail-enabled groups.

Each of the commands you can use to work with user accounts, groups, and mailboxes supports a different set of filterable properties. To discover which properties are available for filtering, you can use PowerShell to query the properties of a returned object. For example:

```
Get-Mailbox -Identity Redmond | Get-Member | Where-Object {$__.MemberType -eq 'Property'} | Sort-Object Name | Format-Table Name
```

This set of commands calls a command to return some information about an object. It then pipes the information returned by the first command to the Get-Member cmdlet, which extracts information about the properties. You sort the properties by name and output them in table format. Here's an excerpt from the output:

```
Name
----
AcceptMessagesOnlyFrom
AcceptMessagesOnlyFromDLMembers
AddressListMembership
Alias
AntispamBypassEnabled
CustomAttribute1
CustomAttribute10
...
WindowsEmailAddress
```

This method works for the Get-Mailbox, Get-CASMailbox, Get-User, Get-Recipient, Get-DistributionGroup, and Get-DynamicDistributionGroup cmdlets. You can use any of the values reported in a –Filter statement. For instance, the call you just made to Get-Mailbox
reports that the custom attributes are available, so to find all mailboxes that have a value in the CustomAttribute10 property, you can generate a command like this:

`Get-Mailbox -Filter {CustomAttribute10 -ne $Null}`

If you look at the filterable properties reported by the Get-DynamicDistributionGroup cmdlet, you can see that the ManagedBy property is available for this dynamic distribution group, whereas it is not for mailboxes. Hence, you can execute a filter like this:

`Get-DynamicDistributionGroup -Filter {ManagedBy -ne $Null}`

When you create a filter, it is best to be as specific as possible. You can state several conditions within a filter. An example of a server-side filter that returns all the mailboxes in the Dublin office where the user name contains “Tony” is shown next. The Get-User cmdlet also works with this filter, but Get-Mailbox executes a tad faster because the server does not have to process accounts that are not mail-enabled.

`Get-Mailbox -Filter {Office -eq 'Dublin' -and Name -like '*Tony*'}`

After you have mastered server-side filtering, you will use it all the time to work with sets of users. For example, assume that you want to give a new mailbox quota to members of a certain department but no one else.

`Get-User -Filter {Department -Eq 'Advanced Technology'} | Set-Mailbox -UseDatabaseQuotaDefaults:$False -IssueWarningQuota 5000MB -ProhibitSendQuota 5050MB -ProhibitSendReceiveQuota 5075MB`

**INSIDE OUT  WhatIf and Confirm**

Before you execute any command to perform a bulk update of objects, you can run the command with the /whatIf switch added to force EMS to show you which objects will be altered. After you are sure that the correct set of objects will be updated, you can run the command without /whatIf, and EMS will perform the changes. The /confirm switch is also useful in terms of stopping administrators before they do something they should not. If you include the Confirm parameter, EMS prompts the administrator with “Are you sure that you want to perform this action” and waits for a “Y” or “Yes” response (or “A” for “all” if multiple objects are involved) before continuing. Act in haste, repent in leisure.
Transcripts

If you encounter a problem executing some EMS commands and need to produce some debug information to give to your support team or Microsoft, you can do this by generating a transcript. A transcript captures details of all commands executed in a session and is useful in terms of capturing the steps necessary to solve a problem or documenting steps to expose an issue that you want to report to Microsoft. You can combine this by adding the –Verbose parameter to most commands to gather a lot of information about what you’ve tried to do and what happened when you tried it. Use the Start-Transcript cmdlet to force EMS to capture debug information. For example:

```powershell
Start-Transcript c:\Temp\Transcript.txt
```

All commands and output will be captured until you stop the transcript by using the Stop-Transcript cmdlet. At this point, you can examine the output with any text editor, and you’ll see something like the output shown in the following example.

```
**********************
Windows PowerShell Transcript Start
Start time: 20130313093116
Username : CONTOSO\Administrator
Machine : ExServer1 (Microsoft Windows NT 6.2.9200.0)
**********************
PS C:\temp> $env:path
C:\Windows\system32\WindowsPowerShell\v1.0\;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem\C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Windows\idmu\common;C:\Program Files\System Center Operations Manager 2007\;C:\Program Files\Microsoft\Exchange Server\V14\bin;c:\temp
```

Bulk updates

Those faced with the task of bulk updates (either to create a lot of new mailboxes or other objects or to modify many existing objects) before the advent of PowerShell support for Exchange had quite a lot of work ahead of them because Exchange offered no good way to perform the work. You could create comma-separated value (CSV) or other load files and use utilities such as CSVDE or LDIFDE to process data in the files against Active Directory, or you could write your own code to use CDOEXM or ADSI to update Active Directory. Either approach involved a lot of detailed work and made it quite easy to make a mistake. Using a console to make the necessary changes was boring and an invitation to make a mistake. The cause of Exchange’s problems with bulk changes was the lack of a programmable way to automate common management operations, a situation that changed with the arrival of EMS.
You can combine the Get-User and Set-Mailbox cmdlets effectively to solve many problems. Here is an example in which you need to update the send quota property on every mailbox for a set of users whose business group has decided to fund additional storage. You can identify these users by their department, which always starts with “Advanced Tech” but sometimes varies into spellings such as “Advanced Technology” and “Advanced Technology Group.” Conceptually, the problem is easy to solve:

1. Look for all users who have a department name beginning with “Advanced Tech.”

2. Update the send quota property for each user.

You could use the Find option in Active Directory Users and Computers to build a suitable filter to establish the set of users, but then you have to open each user’s mailbox that Active Directory Users and Computers locates to update his quota through the GUI, which could become boring after several accounts. You could also export a CSV-formatted list of users to a text file, manipulate the file to find the desired users, and then process that list through CSVDE to make the changes, but you have to search for all matching users across the complete directory first. That is a lot of work to do.

The process is easier in EMS. First, you use the Get-User cmdlet with a suitable filter to establish the collection of mailboxes you want to change. The following command returns all users who have a department name that begins with “Advanced Tech” and then updates the ProhibitSendQuota property to the desired amount (say, 20 GB). Because you have a collection of user objects established, you can use the Set-Mailbox cmdlet to perform the update. Note that some of these users might not be mail-enabled, but error handling is another day’s work.

```
Get-User | Where {$_..Department -like '*Advanced Tech*'} | Set-Mailbox -ProhibitSendQuota 20GB -UseDatabaseQuotaDefaults $False
```

Mergers, acquisitions, and internal reorganizations pose all sorts of problems for email administrators. EMS will not solve the big problems, but it can automate many of the mundane tasks that are necessary. For example, department names tend to change during these events. EMS makes it easy to find all users who belong to a specific department and update their properties to reflect the new organizational naming conventions. If only executing organizational change were as easy as this one-line command, which transfers everyone who works for the Old Designs department over to the Cutting Edge Design department, things would be much easier:

```
Get-User | Where {$_..Department -eq 'Old Designs'} | Set-User -Department 'Cutting Edge Design'
```

Note the use of $_..Department; this indicates a value fetched from the current pipeline object. In this case, it is the department property of the current user object that Get-User
fetched. To verify that you have updated all the users you wanted to (and maybe provide a report to human resources or management), you can use code like this:

```powershell
Get-User | Where {$_.Department -eq 'Cutting Edge Design'} | Select Name, Department | Sort-Object Name | Format-Table > c:\temp\Cutting-Edge.tmp
```

A variation on this theme is to output the data to a CSV file to make the data easier to work with in Microsoft Excel, Microsoft Access, or another tool that can read CSV data.

```powershell
Get-User | Where {$_.Department -eq 'Cutting Edge Design'} | Select Name, Department | Sort Name | Export-CSV c:\temp\Cutting-Edge.CSV
```

Things are even easier if you just need to change everyone’s company name after your company is acquired.

```powershell
Get-User | Set-User -Company 'New Company'
```

You can even do such things as alter only the users whose mailbox belongs to a particular database:

```powershell
Get-Mailbox -Database 'VIP Mailboxes' | Set-User -company 'Big Bucks' -Department 'Executives'
```

**Tip**

All the examples discussed so far depend on you being able to identify some property you can use as the basis for a filter. But what about when you do not have a common property value to check for? In this case, you can build a simple list of mailbox names (or any other format the –Identity parameter will accept, such as a Universal Principal Name [UPN]), use the Get-Content cmdlet to read the names one by one, and pipe these values to whatever other command you need to use. For example, here is how you can use that trick to enable ActiveSync access for a set of users. In this example, the Get-Content cmdlet reads lines containing the identities of the mailboxes you want to change from a text file and pipes them as input to the Set-CASMailbox cmdlet:

```powershell
Get-Content c:\temp\Users.txt | Set-CASMailbox -ActiveSyncEnabled $True
```

Another example of when EMS excels is when you want to apply a common setting across all servers in your organization. For example, assume that you want to apply a new deleted item retention limit of 150 days (perhaps mandated by the legal department) to all servers:

```powershell
Get-MailboxDatabase | Set-MailboxDatabase -DeletedItemRetention 150.00:00:00
```

These simple examples demonstrate the value of having a scripting language that supports automation of common management tasks.
Calling scripts

After you have written a script, you have to decide where to keep it. You could put the new script in the directory that stores the Exchange binaries, but this is a bad idea for many reasons, not least because your script could be overwritten by the installation of a future Exchange service pack, a roll-up update, or even a completely new version.

**INSIDE OUT**  A wise practice

It is wise to maintain a clear separation between the code for which you are responsible and the code Microsoft distributes with Exchange. Therefore, you should create a directory to hold all the scripts you use to work with Exchange. You can then call your scripts safely in the knowledge that they will be available.

The basic rule of calling a script is that if the script is in the working directory (the directory you are currently in), you prefix the name with “.".

```
C:>.\Get-All-Users.ps1
```

If you’re not in the right directory, you can move to where you want to be by using the cd command:

```
C:> cd c:\Scripts\
```

Alternatively, you can supply the full path to where the script is located:

```
C:>c:\Scripts\Get-All-Users.ps1
```

If there are spaces in the directory names, then you need to enclose the path in single or double quotation marks:

```
C: '\Program Files\Microsoft\Exchange Server\V15\Scripts\CollectOverMetrics.ps1'
```

Even better, you can amend the path PowerShell uses by looking for scripts and adding your directory to it. For example, running this command adds the C:\MyScripts directory to the path:

```
$env:path = $env:path + ";c:\MyScripts"
```

After a script is in a directory that’s included in the path, you can invoke it by just typing its name.
Execution policies

EMS is powerful, and just a few cmdlets can have a tremendous effect on many objects throughout Exchange. You might have thought about how to control the ability of users to execute EMS commands.

RBAC provides the first line of protection. As you recall, users are permitted access only to the set of cmdlets and parameters available to the roles each user holds. Even though trusted users are assigned the roles they need to do their work, you still don’t want them to execute scripts they download from the Internet or obtain elsewhere.

A second line of defense is therefore provided by Execution Policies, which define the conditions under which Windows PowerShell loads files for execution. There are four policies: Restricted, AllSigned, RemoteSigned, and Unrestricted. You configure the execution policy used for a server by using the Set-ExecutionPolicy cmdlet. The default is RemoteSigned, which you can verify by using the Get-ExecutionPolicy cmdlet. In this mode, EMS permits the execution of any script created locally and any script downloaded from the Internet, provided the script includes a digital signature. All the scripts that come with Exchange are signed for this purpose (see Table 3-2). The caveat is that any script you attempt to run can contain only Exchange cmdlets that are supported by the role the user holds who invokes the script. Table 3-2 lists the alternate modes together with the potential trade-off in security that you might have to make for each mode.

### Table 3-2 Windows PowerShell execution policies

<table>
<thead>
<tr>
<th>Execution Policy mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restricted</td>
<td>No scripts can be run, even if they are signed by a trusted publisher.</td>
</tr>
<tr>
<td>AllSigned</td>
<td>Scripts must be digitally signed by a trusted partner before EMS will run them.</td>
</tr>
<tr>
<td>RemoteSigned</td>
<td>EMS will run any script created locally. Scripts that originate outside the system (such as those downloaded from the Internet) cannot be run.</td>
</tr>
<tr>
<td>Unrestricted</td>
<td>EMS will run any script. This mode should be used for test environments only.</td>
</tr>
</tbody>
</table>

If you attempt to run an unsigned script that doesn’t comply with policy, Windows PowerShell signals that it cannot load the script. Scripts are signed with the Set-AuthenticodeSignature cmdlet, but you need to get a valid certificate first. The certificate can be one you generate yourself or one you buy from a commercial vendor such as VeriSign.

Obviously, running an Exchange server with an unrestricted execution policy is a bad idea. In fact, you should avoid any deviation from the default policy unless you have an excellent reason to change. For example, you might decide that you want to run scripts you find on the Internet. This might be acceptable if you run the scripts on a test system only, but it’s a much better idea to take the time to go through the code to understand exactly what it does before you think of deploying to a production system. Remember that if you edit a script to create a new version on your computer, that version of the script is now considered local and can be run without changing the execution policy. Opening a downloaded script and saving it can lead to unintended consequences, so be sure that you only save a script that you didn’t write when you absolutely intend to create a new version.

If you deem it necessary to change the policy, use the `Set-ExecutionPolicy` command to update the default execution policy on an Exchange 2013 server. For example:

```
Set-ExecutionPolicy –ExecutionPolicy Unrestricted
```

The change to the execution policy is effective immediately. Be sure to test any change you want to make before you enable the change in production because it might break scripts on which you or applications depend. Execution policy is a server-specific setting. However, its setting is recorded in the system registry, and it is possible to use Group Policy to apply the same setting to every server within the organization. To do this, configure Group Policy to set the value of `ExecutionPolicy` to the desired execution mode. The key is located under:

```
HKLM\Software\Microsoft\PowerShell\1\ShellIds\Microsoft\PowerShell
```

Note that because the setting for the execution policy is held in the system registry, Windows will deny any attempt to update the value unless your account has the privilege to change the system registry.

**Profiles**

When you start EMS, PowerShell runs a script called `Bin\RemoteExchange.ps1` to initialize EMS by loading the Exchange snap-in and defining a set of variables that EMS uses, such as the default scope for Active Directory queries. The script also prints some welcome information for EMS.

If you use EMS frequently, consider creating a profile EMS can load when it initializes a new session. If it finds a profile, PowerShell executes the commands in it before it runs
Exchange ps1 to create the EMS session. This order ensures that you can’t interfere with the creation of the EMS session.

I like profiles because they remind me of the convoluted logon command procedures I used to create for OpenVMS. Typical examples of commands included in profiles are the following:

- Define some aliases (shorthand for commands). For example, you could use Set-Alias gmbx Get-Mailbox to use gmbx any time you want to run the Get-Mailbox cmdlet.
- Add one or more directories containing scripts to the path, as discussed earlier.
- Position your session in a specific directory in which you prefer to work.

PowerShell defines a global variable called $Profile to hold the location of your profile. The exact location varies across different versions of Windows. The profile doesn’t exist by default, and you might have to create it before you can edit it to add some commands. First, see whether a profile is available for the account you use:

```
Test-Path $Profile
```

If the response is $True, you know that a profile exists. If not, you have to create it with:

```
New-Item –Path $Profile –Type File –Force
```

After you have a profile, you can edit it as follows:

```
Notepad $Profile
```

Here’s a simple profile that you could begin with:

```
$env:path = $env:path + ";c:\Scripts"
'You are now entering PowerShell: ' + $env:Username
$StartTime = (Get-Date)
Write-Host "Session starting at $StartTime"
Set-Location c:\temp
```

After you finish updating the profile, save the file and restart EMS to see whether your changes are effective. There are endless possibilities for inventive code to run within a profile.

**Active Directory for PowerShell**

Active Directory is a huge dependency for Exchange, and it makes a lot of sense to be able to manage Active Directory through PowerShell. This was not always possible, but on Windows Server 2008 R2 SP1 servers, all you need to do is load the Active Directory Module for PowerShell that’s installed under Administrative Tools. Assuming that the Active
Directory module is available on a server or client, you can load it into any PowerShell session by using the following command:

```
PS C:\> Import-Module ActiveDirectory
```

The Active Directory module is loaded automatically into EMS on Windows 2012 servers, so you can execute commands against Active Directory data immediately. To get a list of the Active Directory cmdlets, type:

```
PS C:\> Get-Help *-AD*
```

In terms of navigation through the directory structure, Active Directory is represented to PowerShell like files on a hard drive that is referenced as the AD: drive. If your system is joined to a domain, you can then navigate Active Directory. For example, here’s how to create a new organizational unit (OU) called Marketing after navigating to the desired location in Active Directory. You can see the same in Figure 3-6.

```
PS C:\> CD AD:
PS AD:\> CD "DC=contoso,DC=com"
PS AD:\DC=contoso, DC=com> MD "OU=Marketing"
```

![Figure 3-6 Creating a new OU in Active Directory](image)

To compare how much easier it is to access Active Directory data by using the new module, the command to retrieve a list of domain controllers is:

```
PS C:\> Get-ADDomainController | Format-Table Name, OperatingSystem
```

<table>
<thead>
<tr>
<th>Name</th>
<th>OperatingSystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTOSO-DC07</td>
<td>Windows Server 2012 Standard</td>
</tr>
</tbody>
</table>
Another useful example is when you want to scan for inactive Active Directory accounts so that you can clean up the directory. In this command, you scan for any account that has not been logged on to in the past 120 days and report the account name and the date the user last logged on.

`Search-ADAccount -UsersOnly -AccountInActive -TimeSpan 120 | Format-Table Name, LastLogonDate`

You could then disable these accounts by piping the discovered list to the `Disable-ADAccount` cmdlet. However, this is a dangerous thing to do in an Exchange environment because so many accounts are never logged on to because they are used for purposes such as room and discovery mailboxes.

Another one-liner that is extremely useful on test systems searches for all Active Directory accounts that have an email address and sets the accounts so that the passwords never expire. This gets rid of a lot of annoying prompts you might otherwise encounter because passwords expire!

`Get-ADUser -Filter {EmailAddress -Like "*@contoso.com"}  | Set-ADUser -PasswordNeverExpires $True`


### Setting the right scope for objects in a multi-domain forest

When you start EMS, Exchange sets the default scope for queries performed against Active Directory to the domain to which the server belongs. This is fine if you operate a single-domain forest, but it is definitely not if you have to manage objects in a multi-domain forest because it means that any query you perform will return only objects from the local domain. To control the scope for Active Directory objects, use the `Set-ADServerSettings` cmdlet. Set the `ViewEntireForest` parameter to be `$True` (to see the entire forest) or `$False` (to see just the objects owned by the default domain). The logical place to do this is in your personal PowerShell profile. For example:

`Set-ADServerSettings -ViewEntireForest $True`

You can also use this command to point to a particular domain controller to retrieve Active Directory data. For example:

`Set-ADServerSettings -PreferredServer 'DC1.contoso.com'`
If you do not want to set your scope to the entire forest, a partial workaround is to specify a global catalog server in the remote domain to use for the query. Another way of forcing EMS to operate on a forest-wide basis is to specify the `–IgnoreDefaultScope` parameter for cmdlets such as `Get-Mailbox`. This parameter tells EMS to ignore the default recipient scope setting for EAC (typically the domain into which a server is installed) and use the entire forest instead. For example, if you wanted to set up a batch of mailboxes to move from an Exchange 2007 server to Exchange 2013 that used accounts in multiple domains, you could use a command like this:

```powershell
```

The natural question at this point is whether changing the scope for Active Directory queries will affect how you work with EMS. The answer is yes because when you set a forest-wide scope, EMS fetches data from across the forest rather than from the local domain. Unless you use parameters to focus on particular groups of objects, such as specifying that you want to work with the mailboxes from one server, you will probably have to wait longer for a response. This is because you will ask EMS to process cmdlets that deal with servers, mailboxes, databases, or other objects across a complete forest rather than with just one domain, but in most cases, the wait is worthwhile because you see the complete picture and do not run the risk of missing something.

### Exploring useful EMS examples

A scan of the Internet results in many interesting EMS code snippets that can be usefully employed by an Exchange administrator. This section discusses some good examples. The idea is not to present complete solutions. Rather, I hope to inspire you to experiment with EMS to see just how much value you can get from a few lines of reasonably straightforward code. After all, if you can do a lot of work in a couple of lines that take just a few minutes to type in and get running, think of how much you can do if you really set your mind to exploiting EMS!

Before reviewing the examples of EMS in use, I have two specific pieces of advice for the aspiring EMS aficionado. Because this book is emphatically not designed to be a reference guide for EMS, if you think that you will become heavily involved with EMS, purchase a copy of *Microsoft Exchange 2013 PowerShell Cookbook*, Second edition (Packt Publishing, 2013). The book is packed full of guidance, tips, and programming examples that are extremely useful for both on-premises and Exchange Online administrators.

Second, many of the Exchange MVPs provide an extremely valuable service to the Exchange community by publishing what become de facto standards for how to write a script to solve certain problems. You should download these scripts and use them as a
starting point for understanding just how to approach writing industrial-strength EMS code. I would also bookmark their websites and keep up to date with their activities so that you can learn from their future work. At the risk of offending others, among my favorite sites are:

- **Pat Richard** ([http://www.ehloworld.com/](http://www.ehloworld.com/)) Features a great script (Send-NewUserWelcome.ps1) that shows how to build a welcome message to new Exchange users on a scheduled basis.

- **Andy Grogan** ([http://www.telnetport25.com](http://www.telnetport25.com)) Look at his script for automating the setup of an Exchange lab environment.

- **Mike Crowley** ([http://mikecrowley.wordpress.com/](http://mikecrowley.wordpress.com/)) Contains a nice script to report on the proxy addresses assigned to email users.


- **Paul Cunningham** ([http://exchangeserverpro.com/](http://exchangeserverpro.com/)) Offers the best mailbox report script around (Get-MailboxReport.ps1). Paul also maintains a nice server health monitoring script that generates and sends an HTML format message to administrators on a regular basis.

These scripts can be downloaded from these sites; the code is fully revealed and is easily adapted to meet any particular needs that exist in your environment. New sites that feature great tips appear all the time, and I’m sure you will accumulate your own list of go-to people you consult when you meet a problem. In the meantime, look at some examples to get started with EMS.

### Looking for large folders

The first example shows how to discover users who might be suffering from performance problems because they have very large folders in their mailboxes. The number of items that is considered bad has grown over time in line with the updates Microsoft has made to tune the database schema. With Exchange 2000 or Exchange 2003, the danger mark is around 5,000 items. The threshold increases to 20,000 with Exchange 2007 and leaps to 100,000 for Exchange 2010 onward. The client used is also important because Outlook 2010 and Outlook 2013 are better at dealing with large folders than Outlook 2007 is. Having more than 20,000 items in a folder is evidence of solid pack-rat behavior by anyone, and it marks a folder that probably will never be cleaned out simply because it takes too much effort to explore the contents and decide what should be kept and what should be deleted. Assume
that you want to flag potential issues to users who have more than 5,000 items in a folder. You can use code like this:

```
Get-Mailbox -Server ExServer2 | Get-MailboxFolderStatistics | Where {$_ ItemsInFolder -GT 5000} | Sort ItemsInFolder -Descending | Format-Table Identity, ItemsInFolder -AutoSize
```

<table>
<thead>
<tr>
<th>Identity</th>
<th>ItemsInFolder</th>
</tr>
</thead>
<tbody>
<tr>
<td>contoso.com/Exchange Users/Redmond, Eoin\I</td>
<td>5271</td>
</tr>
<tr>
<td>contoso.com/Exchange Users/Ruth, Andy\Inbox</td>
<td>5265</td>
</tr>
<tr>
<td>contoso.com/Exchange Users/Andrews, Ben\Inbox</td>
<td>5263</td>
</tr>
<tr>
<td>contoso.com/Exchange Users/Pelton, David\Inbox</td>
<td>5230</td>
</tr>
<tr>
<td>contoso.com/Exchange Users/Simpson, David\Inbox</td>
<td>5218</td>
</tr>
<tr>
<td>contoso.com/Exchange Users/Redmond, Tony\Sent Items</td>
<td>5215</td>
</tr>
</tbody>
</table>

Of course, it would be impolite to send a note to these users to remind them that good filing practices lead to clean mailboxes, but you can still think about it!

This code does the following:

- Calls Get-Mailbox to generate a list of all mailboxes located on databases hosted by a server. It is possible to process all mailboxes in an organization by changing the code to Get-Mailbox -ResultSize Unlimited, but such a command will take a long time to process in any organization with more than a couple of thousand mailboxes (though you could use a server-side filter when appropriate).

- Calls Get-MailboxFolderStatistics to extract a count of items in each folder.

- Filters any folder with more than 5,000 items.

- Sorts the filtered folders by descending order.

- Outputs the information.

If you run this command against an Exchange 2010 or Exchange 2013 server, even details of the folders in the dumpster (for example, Deletions) will be shown that are not reported by an Exchange 2007 server.

**Outputting a CSV file**

Many examples of outputting CSV files from Exchange data use the Export-CSV cmdlet. For instance, here’s a two-line script that looks for any mailbox that has an ActiveSync partnership created, which indicates that the user has connected a mobile device to the mailbox by using ActiveSync. An expression is included to force a call to the Get-ActiveSyncDevice
cmdlet to retrieve the count of devices associated with each user. This kind of information is useful when understanding how many people actually connect mobile devices to Exchange!

```powershell
$Mbx = Get-CASMailbox –Filter {HasActiveSyncDevicePartnership –eq $True} | Get-Mailbox
$Mbx | Select DisplayName, UserPrincipalName, @{Name="Devices";Expression= {(Get-ActiveSyncDevice –Mailbox $_.Identity).Count}} | Export-CSV "c:\temp\ActiveSync.csv" –NoTypeInformation

Export-CSV is great because it takes care of all the formatting issues required to create a valid CSV file that will be recognized by applications such as Excel. However, there are other ways to generate CSV data. This script creates a CSV file you can use to analyze mailbox usage. A check in the code limits processing to the mailboxes found in a specific database and ignores anything but user mailboxes. (Objects such as room or arbitration mailboxes are ignored.) This script could take quite a while to finish if there are more than a few hundred mailboxes in the selected database, so be sure to test it on perhaps a smaller group before you launch it to process larger collections.

```powershell
$Outputfile = "C:\temp\Mailboxes.csv"
Out-File -FilePath $Outputfile -InputObject "UserPrincipalName, Items, Mailbox Size" -Encoding UTF8
$mbx = Get-Mailbox -Database DB2
Foreach ($M in $Mbx)
{
    if ($M.RecipientTypeDetails -eq "UserMailbox")
    {
        # Fetch information about the mailbox
        $UserMailbox = Get-Mailboxstatistics -Identity $($_.Identity)
        $UserPrincipalName = $M.UserPrincipalName
        $ItemCount = $UserMailbox.ItemCount
        $ItemCount = $ItemCount.ToString()
        $MailboxSize = $UserMailbox.TotalItemSize.ToString()
        $ItemCount = $ItemCount + 1
        $ItemCount = $ItemCount.Replace("","39")(0,1)
        $ItemCount = $ItemCount.Replace("","39")/1024/1024
        $UserDetails = $UserPrincipalName + "," + $ItemCount + "," + $MailboxSize
        Out-File -FilePath $Outputfile -InputObject $UserDetails -Encoding UTF8 -append
    }
}
```

This script generates fairly basic data about mailboxes, and if you scan the Internet, you can find many other approaches to the problem of mailbox reporting, some of which are much better than others. With anything to do with mailboxes, the key is speed because code that is quite good at processing one or two mailboxes might not be smart when confronted with
a few thousand. It's also a good idea to consider what information needs to be output and make sure that the data reported is formatted in a way that is most useful to the reader.

Creating a report in HTML

PowerShell is flexible in terms of processing output. Generated reports can show management and others the kind of work that servers do. The typical reports EMS generates are plaintext. You can also generate HTML reports by piping objects through the ConvertTo-HTML cmdlet. (The Out-HTML cmdlet at http://poshcode.org/1612 is also useful for generating HTML content.) This example explores how to generate a useful report that shows mailboxes that have exceeded their storage quota. You could use a report like this to check proactively for users who are experiencing problems with their quota and perhaps allocate them some additional quota to enable them to resume working. The output is shown in Figure 3-8.

Get-Mailbox -Database VIP | Get-MailboxStatistics | Sort TotalItemSize -Descending | ConvertTo-HTML DisplayName, Database, ItemCount, TotalItemSize > C:\Temp\Mbxs.html

You can enhance the output further by formatting the HTML with a style sheet or adding other information such as the date and time of the report. I leave that as an exercise for the reader.

Figure 3-7 User mailbox CSV data
It's worth noting that when you run the Get-MailboxStatistics cmdlet, you force EMS to make a remote procedure call (RPC) to the Information Store to retrieve the latest data for the mailboxes (individual, database, or server). The information is completely up to date and reflects the exact state of the mailbox rather than cached data that could be a couple of hours old. The Store caches information about mailbox quotas and updates the cache every two hours to avoid the overhead of the I/O that it would otherwise need to generate to check quotas every time a user attempts to send a message or to check that a mailbox can accept a new message.

**TROUBLESHOOTING**

Users report that they've deleted messages but still exceed quota

Given the dynamic flow of messages in and out of mailboxes, it's likely that a small difference exists between the cached data and the actual state. This sometimes causes confusion when a user reports that she has exceeded quota and can't send mail even though she has deleted many messages, and she has to wait until the Store refreshes its cache to determine the new mailbox size and respect the fact that she has reduced the size under quota. If this becomes a problem and users complain that Exchange takes too long before it allows them to resume email activity, you can amend the system registry to force Exchange to refresh the cache more often with the caveat that more frequent refreshes impose an extra overhead on the server. See [http://technet.microsoft.com/en-us/library/aa996988(EXCHG.80).aspx](http://technet.microsoft.com/en-us/library/aa996988(EXCHG.80).aspx) for details.
Verbose PowerShell

Usually, EMS gets on with whatever you ask it to do and doesn't give any indication of the processing it performs in the background. You ask for a new mailbox to be created, and it's created, or some problem occurs that stops the command from executing. If the problem originates with an error introduced by the user, such as an error in syntax or attempting to do something that doesn’t make sense, such as creating a mailbox in a database that doesn’t exist, you can just fix the problem and try again.

Sometimes you need to know exactly what EMS does to help track down a problem, perhaps to provide information to Microsoft support to help them figure out what's going on in your Exchange deployment. You might just want to know what's happening when you execute a command. In either case, you can add the /verbose switch to a command to have PowerShell generate details of exactly what it does as it proceeds. Figure 3-9 shows some of the output when the New-MailboxDatabase cmdlet is used to create a new mailbox database. You can see how EMS validates the context within which it is executing, including checks to locate a global catalog server, validate RBAC authorization, and confirm that the mailbox database doesn’t already exist.

![Figure 3-9 Examining some verbose PowerShell output](image)

Controlling access to Exchange

EMS is a great way to get work done with Exchange as long as you don’t mind grappling with the command-line interface. If no control were exerted, you could do massive damage to an Exchange organization with EMS, such as selecting all the mailboxes in a database and removing them with a single line of code. Only the people who need to control the full scope of the organization should be able to take such drastic action. Traditionally, control is
given through permissions and privileges. Exchange takes a different approach and adopts
the RBAC model. All administrators need a solid grounding in RBAC and its implementation
in Exchange, and that’s the next subject of discussion.
Numbers and Symbols
$_ variable, 101–102
$_ . variable, 102
{} (braces), 110
& (ampersand), 385
* (asterisk), 138, 481
- (hyphen), 103, 110
. operator, 100
" (quotation marks), 102–103
_ (underscore), 26
| operator, 102, 109

A
ABPs (address book policies)
about, 348–350
creating and implementing, 351–357
groups and, 350
accepted domains, 340
–AcceptLargeDataLoss cmdlet parameter, 632
AcceptMessagesOnlyFrom property, 279–280
AcceptMessagesOnlyFromDLMembers property, 279–280
AcceptMessagesOnlyFromSendersOrMembers property, 279
access control, role-based. See RBAC (role-based access control)
access control lists (ACLS), 131–132, 143, 282
ACID test, 419
ACLL (attempt copy last logs) process, 473, 478–479
ACLS (access control lists), 131–132, 143, 282
activating mailbox database copies, 533–534
activation blocks, 542–544
activation preference for databases, 477
ActivationPreference property, 521
Active Directory
about, 34–37
address book policies and, 350
ADSIEdit utility, 36–37, 59–60
deployment support, 45–47
ensuring full visibility, 364
Federated Services role, 313
forest functional mode, 26
preparing, 47–48
removing tags from, 687
storing information, 35–36
user accounts, 49
Windows PowerShell and, 120–123
Active Directory Domain Services (AD DS), 47–48
Active Directory Rights Management Services (AD RMS), 734
Active Directory Users and Computers
audit entries and, 754
changing group properties, 285
creating computer objects, 497
displaying group membership, 268
Find option, 115
moving groups, 289
USGs and, 141
Active Manager
about, 461, 465–467
ACLLs, 478–479
automatic database transitions, 467–469
best copy and server selection, 472–477
DAC and, 552
failover clustering and, 464
high availability and, 465–466, 469–472
Managed Availability and, 469–472, 536
scanning for failed copies, 458
ActiveCopy property, 521
ActiveDirectoryConnectivityConfigDCRestart responder, 470
ActiveSync, 29, 125–126
–ActiveSyncMailboxPolicy cmdlet parameter, 193
AD DS (Active Directory Domain Services), 47–48
AD RMS (Active Directory Rights Management Services), 734
Add-AdPermission cmdlet, 232, 794
Add-DatabaseAvailabilityGroupServer cmdlet, 494, 496
Add-DistributionGroupMember cmdlet, 92, 193, 296, 298
Add-MailboxDatabaseCopy cmdlet, 525, 527, 545
Add-MailboxPermission cmdlet, 202, 233–234, 236–237, 260
Add-PublicFolderClientPermission cmdlet, 787
Add-RoleGroupMember cmdlet, 150
Add-WindowsFeature cmdlet, 51, 53
AddNewRequestsTentatively property, 258
AddOrganizerToSubject property, 258
address book policies (ABPs)
  about, 348–350
creating and implementing, 351–357
groups and, 350
address lists, 345–349
addressing messages
  about, 333
  address book policies, 348–357
  address lists, 345–348
display templates, 382–386
e-mail address policies, 333–345
hierarchical address books, 349, 371–373
MailTips and group metrics, 373–382
Offline Address Book, 357–371
ADDriver (Exchange Authorization Library), 90–91
Admin Audit Log agent, 750
AdminAuditLogs folder, 749
Administrator user account, 163
administrators
  auditing actions of, 748–754
delivery reports, 321–327
  page zeroing, 436
  quorum voting, 494
  running EAC without Exchange mailbox, 329–330
  security and, 319–320
  user accounts, 163
ADSIEdit utility
  about, 36–37
  arbitration mailboxes, 59
  auditing and, 754
e-mail address formats, 344
  removing server objects, 60
  viewing group properties, 374
Advanced Search Syntax (AQS), 736
AgeLimitForRetention property, 666, 670
AggregateCounts corruption type, 446
AggregatePFData.ps1 script, 788–789
alerts
  EAC, 172–173
  health mailboxes, 227
  mail flow and rules, 318–319
  moving mailboxes and, 638–639
  self-signed certificates, 78
–Alias cmdlet parameter, 192, 253, 307
Alias property, 185
aliases, 93
All Contacts container, 345
All Groups container, 345
All Rooms container, 345
All Users container, 345
AllBookInPolicy property, 262–264
AllRestrictInPolicy property, 262, 264
AllRequestOutofPolicy property, 262–263
AllSigned execution policy, 118
AlternateWitnessDirectory property, 505
AlternateWitnessServer property, 505
ambiguous name resolution (ANR), 365–366
ampersand (&), 385
ANR (ambiguous name resolution), 365–366
ANSI format, 623
APIPA (Automatic Private IP Addressing), 492
Application and Service Logs, 502
Application Event Log
diagnostic levels and, 332
eTooManyMountedDatabases error, 544
excessive growth, 443
usage examples, 364, 433
Application Impersonation role, 159
AQS (Advanced Search Syntax), 736
arbitration mailboxes
  about, 182
  alerts and, 638–639
  checking, 547
discovery mailboxes, 224
migration, 573
moving, 59–60
organization mailboxes, 362
 suppressing, 312
ArbitrationMailbox property, 249
architecture, Exchange Server, 10–11
–Archive cmdlet parameter, 193, 446, 648, 652
archive mailboxes
  about, 645–647, 655–656
  checking space usage, 652
  compliance management and, 644
DAG design, 557
default archive and retention policy, 652–655
development priorities, 4
disabling, 656–657
enabling, 647–650
limitations of, 647
managing properties, 651–652
server failure and, 649
troubleshooting, 656
upgrading properties, 652
–ArchiveDatabase cmdlet parameter, 648–649
ArchiveDomain property, 651
ArchiveRelease property, 651
ArchiveState property, 651
ArchiveStatus property, 651
arrow keys, 95
–AssociatedMessagesCopyOption cmdlet parameter, 633
asterisk (*), 138, 481
asynchronous moving, 571–572
AsyncOperationNotification system retention tag, 639
attachments, 398
attempt copy last logs (ACLL) process, 473, 478–479
attributes. See specific properties
audit mailbox, 749–750
AuditAdmin property, 758
AuditDelegate property, 758
auditing
administrator actions, 748–754
mailbox access, 239, 754–762
AuditLogAgeLimit property, 757–758
AuditOwner property, 758
authentication
certificates and, 314
cmdlets and, 89
Kerberos, 79
sending messages to groups, 280–281
auto-mapping mailboxes, 235–237
–AutoDatabaseMountDial cmdlet parameter, 478–479, 539–542
AutoDatabaseMountDial setting, 478–479, 539–542
–Autodiscover cmdlet parameter, 612
Autodiscover feature, 235–237
AutomateProcessing property, 257, 261
automatic database transitions, 467–469
Automatic Private IP Addressing (APIPA), 492
automatic reseeding of failed databases, 458
automating mailbox settings, 205–206
–AutoSize cmdlet parameter, 99
–AutoStart cmdlet parameter, 593
autosynchronization, 588
Azure File Server, 490
B
B+ tree defragmentation, 435
backend (Mailbox server role), 1, 314
background maintenance
about, 390, 432–433
content maintenance tasks, 432, 436–437
continuous, 482
database checksums, 432–435
database compaction, 432, 436
database defragmentation, 432, 435
page zeroing, 433, 436
Backspace key, 95
backups
database, 416–417
streaming, 29, 389
VSS, 458
bad-item limit, 584–587, 590, 608
–BadItemLimit cmdlet parameter, 637
BadItemLimit property, 593
BalanceDbsByActivationPreference mode, 535
BalanceDbsBySiteAndActivationPreference mode, 536
–BatchName cmdlet parameter, 575, 634
BCS (best copy selection), 472
BCSS (best copy and server selection), 472–477
Begin log record type, 425
best copy and server selection (BCSS), 472–477
best copy selection (BCS), 472
binary large objects (BLOBs), 401
Binpatch.oab file, 361
BLOBs (binary large objects), 401
block mode replication, 484–486
BookInPolicy property, 262–264
BPOS (Business Productivity Online Services), 6
braces, 110
bulk mailbox creation, 195
bulk updates, 114–116
business cases, building, 16
Business Productivity Online Services (BPOS), 6
–BypassDelegateChecking cmdlet parameter, 328
–ByPassNestedModerationEnabled cmdlet parameter, 245
–BypassSecurityGroupManagerCheck cmdlet parameter, 277
C
Calcheck (Calendar Checking Tool for Outlook), 445
Calendar Assistant, 259, 261, 454
Calendar Checking Tool for Outlook (Calcheck), 445
Calendar folder, 658, 695
Calendar Repair Assistant, 746
CalendarLoggingQuota property, 746
calendars, retention policies and, 677
CalendarVersionStoreDisabled property, 746
Calender Logging folder, 746
CALs (Client Access Licenses)
about, 33
applying retention policy to mailboxes, 678
archiving and, 644
mailbox management and, 184, 191
requirements, 27, 72–73
Can Opener screen, 55
canceling repair jobs, 447
capacity planning, 389–392
CAS (Client Access Server)
about, 1
address book policies, 349
architectural overview, 10–11
calendar interaction, 376–377
DAG and, 461
development problems, 7–8
MailTips and, 376–377
MRS and, 567
MRSPProxy and, 612–613
namespace planning, 79–81
public folders and, 767
testing programming and customizations, 30–31
transport service, 27
case sensitivity, date/time formats, 199
CCR (cluster continuous replication), 29
cd command, 117
Central Help Desk security group, 323
CER file, 317
certificates
managing, 313–317
protocols and, 314
self-signed, 77–78, 314, 317
signed scripts, 118
SSL, 30
X.509, 26, 313
ChangedGroups.txt file, 376
ChangePermission right, 260
Checkbox control type, 385
CheckDatabaseRedundancy script, 522
checkpoint files, 408, 426–427
checksums
  database, 432–435
  transaction logs, 427
CI (Content Indexing) catalog, 554
circular logging
  about, 429–431
  DAG and, 510–514, 554
  default state, 414
  transaction log truncation, 486
clearing move requests, 610–611
Client Access Licenses (CALs)
  about, 33
  applying retention policy to mailboxes, 678
  archiving and, 644
  mailbox management and, 184, 191
  requirements, 27, 72–73
Client Access Server (CAS)
  about, 1
  address book policies, 349
  architectural overview, 10–11
  client interaction, 376–377
  DAG and, 461
devlopment problems, 7–8
  MailTips and, 376–377
  MRS and, 567
  MRSPProxy and, 612–613
  namespace planning, 79–81
  public folders and, 767
  testing programming and customizations, 30–31
  transport service, 27
client-side filters, 110–113
client-side rules, 318
closed groups, 270
cloud service, 5, 9, 646
cluster continuous replication (CCR), 29
Cluster Name Object (CNO), 490, 496–499
customer technology, 463–464, 492–493
cmdlets. See also specific cmdlets
  about, 21, 83–86, 92–95
  Active Directory module, 120–122
  bulk updates, 114–116
  command editing, 95–97
  default view, 94
determining availability of, 86
  execution policies, 118–119
  filtering, 110–113
  /full switch, 94
  handling information returns, 98
  HTML reports, 127–128
  online repair, 445–448
  output from, 100, 125–127
  /parameter switch, 94
  piping, 109
  quarantined mailboxes, 439
  RBAC restrictions, 154
  remote Windows PowerShell and, 86–92
  repair, 445–448
  role assignment policies, 153–155, 163–166
  roles and, 139
  in scripts, 93, 117
  selective output, 98–101
  transcripts, 114
  variables and, 101–103
  /verbose switch, 129
  /whatIf switch, 113
CNO (Cluster Name Object), 490, 496–499
Codeplex website, 38
columns, EAC, 177–178
comma-separated-value (CSV) files
  analyzing, 68
  CSVDE utility, 114
e-mailing EAC information to, 181–182
  migration batches, 592
  modern public folders and, 789
  moving mailboxes, 582, 590
  outputting, 125–127, 281–282
command editing, PowerShell, 95–97
Commit log record type, 425
compacting database, 432, 436
comparison operators, 110
Complete-MigrationBatch cmdlet, 594
compliance management
  about, 641–642
  archive mailboxes, 645–657
  auditing administrator actions, 748–754
  auditing mailbox access, 754–762
devlopment priorities, 4
evolving needs of, 3
  legal discovery actions, 642–645
  messaging records management, 657–689
  preserving information, 696–737
  Recoverable Items structure, 737–747
testing operational processes, 29
  upgrading considerations, 18
Compliance Management role group, 74
compression
  attachments and, 398
LZ77, 506
LZX, 361
mailbox database, 394, 397–398
transaction log, 483–484
Computer Administrator account, 329–330
concurrent searches, 733
conditional checking in scripts, 103
conditional parameters, 307–313
--ConditionalCompany cmdlet parameter, 307, 309
--ConditionalCustomAttribute cmdlet parameter, 307, 309
--ConditionalDepartment cmdlet parameter, 309
--ConditionalStateOrProvince cmdlet parameter, 307, 309
--ConflictResolutionOption cmdlet parameter, 632
configuration
Active Directory information, 35
MailTips, 377–378
XML files, 51–52, 205–206
Configuration read scope, 142
Configuration write scope, 142
--ConfigurationOnly cmdlet parameter, 545–546
--Confirm cmdlet parameter, 113, 656
ConflictPercentageAllowed property, 260
Connect-ExchangeOnline function, 90
Connect-Mailbox cmdlet, 222–223
contacts, mail-enabled, 179, 250–251
Contacts folder, 658
content indexes, 701, 776
Content Indexing (CI) catalog, 554
content maintenance tasks, 432, 436–437
ContentIndexState property, 542
ContentMailbox property, 780
contiguous space, 392, 394–395, 397–398
continuous replication circular logging (CRCL), 511
controller process, 402–407
ConvertTo-HTML cmdlet, 127
cookie files, 376
copies, database. See database copies
Copy action, 756
CopyQueueLength property, 521
corrupt items
detection and isolation, 437–448
moving mailboxes, 569
corruption, logical, 526
country property, 305
CrashCount system registry setting, 438
CRCL (continuous replication circular logging), 511
Create action, 756
--Credentials cmdlet parameter, 612
crimson channel, 475–476, 502–504
Crowley, Mike, 124
CSV (comma-separated-value) files
analyzing, 68
CSVDE utility, 114
exporting EAC information to, 181–182
migration batches, 592
modern public folders and, 789
moving mailboxes, 582, 590
outputting, 125–127, 281–282
CSVDE utility, 114
Ctrl+Click key combination, 361
CU (cumulative updates), 14, 27, 64–67
cumulative updates (CU), 14, 27, 64–67
Cunningham, Paul, 124
custom filters, 308–313
--CustomAttribute cmdlet parameter, 352–355, 385
--CustomConfigWriteScope cmdlet parameter, 143
--CustomRecipientWriteScope, 143, 157
D
%d variable, 339
DAC (Datacenter Activation Coordination), 551–552
DACP bit, 552
DACP (Datacenter Activation Coordination Protocol), 552
DAG (Database Availability Group)
about, 19–20, 32–33, 311, 457, 461–463, 489–494
activating mailbox database copies, 533–534
activation blocks, 542–544
Active Manager, 465–479
adding database copies, 514–517
adding database copies with EMS, 525
architectural overview, 10–11
AutoDatabaseMountDial setting, 539–542
building, 494–496
building DAG, 494–496
changes in high availability, 457–459
changes in message submission, 489
circular logging, 510–514, 554
cleaning up, 510
crimson events, 475–476, 502–504
DAG networks, 506–509
data center activation coordination, 551–552
databases and, 362, 411, 459–464
day-to-day management and operations, 489–551
design considerations, 552–562
handling storage failures, 549–551
I/O performance improvements, 401
lagged database copies, 525–532
managing properties, 504–506
migrating, 464–465
monitoring database copies, 517–522
moving database locations within, 544–546
moving databases, 539–542
naming conventions, 179
passive database copies, 460
performing server switchovers, 536–539
pre-staging Cluster Name Object, 497–499
public folders and, 767–768
rebalancing database copies, 534–536
removing database copies, 546–548
removing servers, 548–549
reseeding database copies, 523–525
role of FSW, 499–501
server maintenance, 562–565
server names, 44
sizing mailboxes, 394
stressed servers, 565–566
task logging, 502
transaction log replays, 479–489
virtual servers, 24
Windows Failover Clustering, 463–464

**Database Availability Group (DAG)**

about, 19–20, 32–33, 311, 457, 461–463, 489–494
activating mailbox database copies, 533–534
activation blocks, 542–544
Active Manager, 465–479
adding database copies, 514–517
adding database copies with EMS, 525
architectural overview, 10–11
AutoDatabaseMountDial setting, 539–542
building, 494–496
building DAG, 494–496
changes in high availability, 457–459
changes in message submission, 489
circular logging, 510–514, 554
cleaning up, 510
crimson events, 475–476, 502–504
DAG networks, 506–509
data center activation coordination, 551–552
databases and, 362, 411, 459–464
day-to-day management and operations, 489–551
design considerations, 552–562
handling storage failures, 549–551
I/O performance improvements, 401
lagged database copies, 525–532
managing properties, 504–506
migrating, 464–465
monitoring database copies, 517–522
moving database locations within, 544–546
moving databases, 539–542
naming conventions, 179
passive database copies, 460
performing server switchovers, 536–539
pre-staging Cluster Name Object, 497–499
public folders and, 767–768
rebalancing database copies, 534–536
removing database copies, 546–548
removing servers, 548–549
reseeding database copies, 523–525
role of FSW, 499–501
server maintenance, 562–565
server names, 44
sizing mailboxes, 394
stressed servers, 565–566
task logging, 502
transaction log replays, 479–489
virtual servers, 24
Windows Failover Clustering, 463–464

**DAMs (deferred action messages)**, 318–319
dark zone, 60
Data Guaranteed API, 609
data loss prevention (DLP), 18, 642, 646
database copies
activating mailbox, 533–534
activation blocks, 542–544
adding with EMS, 525
automatic database transitions, 467–469
cleaning up, 510
DAGs and, 465, 514–517, 555
failover, 466
identifying specific, 525
lagged, 525–532
monitoring, 517–522
passive, 460, 544
rebalancing, 534–536
removing, 546–548
reseeding, 523–525
scanning for, 458
switchover, 466, 533–534
database management
about, 407–409
activation preference, 477
automatic database transitions, 467–469
automatic reseeding of failed databases, 458
backups, 416–417
circular logging and, 429–430
cleaning up before creating copies, 510
creating mailbox databases, 410–414
custom filter to address mailboxes in databases, 310–313
DAGs and, 362, 555–556
database checksums, 432–435
database compaction, 432, 436
database copies, 510, 514–536
database defragmentation, 432, 435
database redundancy, 411, 522
database replication, 390, 411, 459, 479–489
database schemas, 398–400
database usage statistics, 451–453
downloading message databases, 444–445
dismounting databases, 422, 447, 469, 545
estimating database size, 217–218
log placement, 428–429
mailbox resources provisioning management agent, 209–213
maximum database size, 389–392
maximum number of active databases, 402–403
mounted databases, 407, 426–427, 462–463, 546
moving databases, 536, 539–542, 544–546
naming conventions, 179
numbers supported, 407
OAB and, 371
protection against excessive database growth, 443–444
rebuilding databases, 448–451
removing databases, 416–418
renaming databases, 414–415
seeding databases, 489
updating after installation, 414–416
write smoothing, 395
database portability, 459–464
database scope, 158–159
database usage statistics, 451–453
DatabaseAvailabilityGroupIpAddresses property, 505
Datacenter Activation Coordination (DAC), 551–552
Datacenter Activation Coordination Protocol (DACP), 552
--DatacenterActivationMode cmdlet parameter, 552
DataMoveReplicationConstraint property, 620–621
DataMoveReplicationConstraint setting, 481, 609
Data.oab file, 361
DataPath property, 410–411
date formats, 199
debugging
   generating transcripts, 114
   swelling databases, 444–445
Default Archive and Retention Policy, 653–655
Default Archive Policy, 653
default folders, 200–201, 393
default group location, 288–289
Default MRM Policy, 652, 655
default policy tags (DPT), 658–659, 666
Default Role Assignment Policy, 290, 295
DefaultPublicFolderMailbox property, 771
defferred action messages (DAMs), 318–319
Deferred Actions folder, 318
defragmentation, database, 432, 435
Defragmentation Tasks counter, 435
Delegated Setup role group, 63, 74, 141
delegating
   groups, 275
   mailboxes, 229–232
   Outlook access, 239–240
   role assignments, 160
   roles, 140, 163, 625
--Delegating cmdlet parameter, 154
Delete key, 95
delete tags, 672
DeleteAndAllowRecovery action, 672
DeleteAttachments property, 257
DeleteComments property, 257
--DeleteContent cmdlet parameter, 703
Deleted Items folder, 393, 649, 658, 695
--DeleteExistingFiles cmdlet parameter, 524
DeleteNonCalendarItems property, 258
DeleteSubject property, 257
Deletions folders, 746
delivery reports
   about, 320–322
   administrator searches for, 322–327
   EMS searches for, 327–329
   troubleshooting, 323
designing
   DAGs, 552–562
   retention policies, 663–665
detail templates, 382–386
Details Templates Editor, 382–386
detecting corrupt items. See corrupt items
--DetectOnly cmdlet parameter, 446
development priorities, 2–5
DHCP (Dynamic Host Configuration Protocol), 490
diagnostics for Exchange Server, 330–332
Direction property, 598
dirty pages, 425
Disable-Mailbox cmdlet, 92, 219, 656
Disable-MailPublicFolder cmdlet, 786
disabling
   administrative auditing, 748
   archive mailboxes, 656–657
   mailboxes, 218–220, 223
   retention tags, 687
disaster recovery mechanisms, 529
DisconnectDate property, 220
discovery actions, 642–643, 646
discovery mailboxes, 76, 182, 188, 224–226, 721
Discovery Management role group
   about, 74
   controlling access to discovery mailboxes, 722
default discovery mailboxes and, 234
   linking to tasks, 135
   performing searches, 225
   RBAC support, 174
role assignment and, 152
discovery metadata mailboxes, 224
discovery search mailboxes, 224
DiscoveryHolds folder, 726–727, 746
DiscoveryMaxConcurrency property, 725
DiscoveryMaxKeywords property, 725
DiscoveryMaxKeywordsPerPage property, 725
DiscoveryMaxMailboxes property, 725
DiscoveryMaxMailboxResultsOnly property, 725
DiscoveryPreviewSearchResultsPageSize property, 725
DiscoverySearchMailbox account, 76
DiskFreeSpace property, 522
DiskFreeSpacePercent property, 522
DiskTotalSpace property, 522
Dismount-Database cmdlet, 415, 512, 545
dismounting databases, 422, 447, 469, 545
display templates, 382–386
--DisplayName cmdlet parameter, 192, 307, 348
DisplayName property, 185, 222
DistinguishedName property, 185
distribution groups
about, 267–269
address book policies and, 350
creating, 270–275
defining default location, 288–289
dynamic, 179, 246, 300–313
group expansion, 277–278
group naming policy, 287, 289–295
group owners, 271, 275–277, 350
managing, 269–270, 286–297
metrics for, 275
moderated, 244–246, 270, 283
protected, 278–281
security groups, 252, 284–285
tracking usage, 285–286
viewing group members, 268–269, 281–283
Distribution Groups role, 147
DistributionGroupNamingPolicy property, 292
DLL (dynamic link library), 344–345
DLP (data loss prevention), 18, 642, 646
DNS (Domain Name System), 327, 462, 508
document library, 796
domain controllers, 495
Domain Name System (DNS), 327, 462, 508
–DomainController cmdlet parameter, 91
domains
accepted, 340
Active Directory information, 35
preparing, 48
–DoNotIncludeArchive cmdlet parameter, 731
DPT (default policy tags), 658–659, 666
drizzle synchronization, 578
.dsc file extension, 376
DSN error code 5.7.1, 280
dynamic buffer allocation, 402
dynamic distribution groups
about, 270, 300
Active Directory and, 176
creating, 302–305
creating with EMS, 306–308
custom filters, 308–313
OPATH queries, 300–302
validating query results, 305–306
Dynamic Host Configuration Protocol (DHCP), 490
dynamic link library (DLL), 344–345
dynamic quorums, 494
dynamic throttling, 580

E
E2EComplete migration product, 575
EAC (Exchange Administration Center)
about, 24
accessing data, 176–177
accessing mailbox audit data, 759
address book policies, 352–353
applying retention policy to mailboxes, 677
archive mailboxes, 650–652
bulk mailbox creation, 195
certificate management, 313–317
changing columns, 177–178
Compliance Management section, 171, 668, 673, 759
creating DAGs, 490–491
delivery reports, 320–329
development priorities, 4
distribution groups, 267–313
exporting information to CSV files, 181–182
filtering capabilities, 177, 342
Groups section, 302, 331
Hybrid section, 172
importing and exporting mailbox data with, 629–632
in-place holds, 705–706
language support, 196–200
mail flow and rules, 318–320
Mail Flow section, 172, 325, 333, 335
Mailbox Delegation section, 229–230, 232–233
mailbox import and export, 629–632
management interfaces, 21–22, 170–173
managing migration batches, 581–592
managing recipients, 180–183
migration batches with, 574, 581–592
Mobile section, 172–173
moving databases and, 544
mysterious mailboxes, 182–183
naming conventions, 178–180
Offline Address Book and, 362
Organization section, 171, 313, 346, 352
permissions, 143–144, 233–234
Permissions section, 133–134, 143–144, 171, 295
Protection section, 171
public folders and, 766
Public Folders section, 172, 188, 767
RBAC support, 132
Recipients section, 146, 171, 177, 182, 241, 250, 273–274, 581
refresh option, 176
replication queue lengths, 481
reporting licenses, 70–71
running without mailboxes, 329–330
Servers section, 172, 315, 409–410, 494, 537
setting diagnostics for Exchange Server, 330–332
setting up mailboxes, 191–193
sharing policies, 313
starting, 173–175
Unified Messaging section, 172
viewing correct, 174
viewing migration batch properties, 595–596
Windows PowerShell and, 84
ECP (Exchange Control Panel), 21–22, 170, 269, 707
ecTooManyMountedDatabases error, 544
EDB (Exchange database), 388
edge servers, 48
eDiscovery searches, 224–225, 233, 702, 726
Edit control type, 385
editions, Exchange Server, 22–23, 32–33
ELC (Email Lifecycle Assistant), 726–727
e-mail address policies
about, 333–334
creating, 337–340
e-mail policy priority, 334–336, 340
filters and, 341–345
mask variables, 339
Email Lifecycle Assistant (ELC), 726–727
email notification, 595–596
email service. See addressing messages; mailbox management
–EmailAddress cmdlet parameter, 612
EMC (Exchange Management Console), 21–22, 650
EMS (Exchange Management Shell)
about, 24, 31, 83, 92–120
accessing mailbox audit data, 759
Active Directory for PowerShell, 120–123
adding database copies, 525
address book policies, 352–353
creating dynamic distribution groups, 306–308
delegated setup and, 63
e-mail address policies, 343–345
enabling archives, 648
executing searches, 731–732
handling information returns, 98
mailbox import and export, 632–637
MailTips management, 377–378
management interfaces, 21–22, 170
migration batches with, 574, 592–595
public folders and, 766
room lists, 270, 298
searches for delivery reports, 327–329
useful examples, 123–128
Windows PowerShell, 83–92, 129
Enable-CmdletExtensionAgent cmdlet, 206
Enable-Mailbox cmdlet, 193–194, 206, 649, 651
Enable-QuarantineMailbox cmdlet, 440
day-user roles, 164–165
domains, migration, 611–612
Enterprise Administrators group, 48
Enterprise edition, 33, 462
EOP (Exchange Online Protection), 16
–eq comparison operator, 110
–Equipment cmdlet parameter, 195
equipment mailboxes
about, 188, 254–255, 265
creating, 195
defining custom properties, 255–256
naming conventions, 178–179
ESE (Extensible Storage Engine)
about, 388–389
DAG and, 466
handling storage failures, 549
memory allocation and, 402–403
timestamps, 425
transaction logs, 419–425, 479
view tables, 400
ESEUTIL utility
about, 57, 445, 450–451
/D parameter, 448
database compaction, 436
/K switch, 482
/MH parameter, 422
/MS parameter, 450
/p switch, 427
/R switch, 482
rebuilding databases, 451
–EstimateOnly cmdlet parameter, 731
event-based assistants, 454
Event Viewer, 471
EWS (Exchange Web Services)
about, 40–41
ESE versus, 389
message tracking, 326
synchronizing information, 792
testing considerations, 28, 30
EWSEditor (Exchange Web Services Editor), 40–41
examining search results, 720–724
$ExBin variable, 101
Exchange Administration Center (EAC)
about, 24
accessing data, 176–177
accessing mailbox audit data, 759
address book policies, 352–353
applying retention policy to mailboxes, 677
certificate management, 313–317
changing columns, 177–178
categories and, 171, 668, 673, 759
creating DAGs, 490–491
delivery reports, 320–329
development priorities, 4
distribution groups, 267–313
e-mail address policies, 343–345
exporting information to CSV files, 181–182
filtering capabilities, 177, 342
Groups section, 302, 331
Hybrid section, 172
I/O performance improvements, 401
importing and exporting mailbox data with, 629–632
in-place holds, 705–706
language support, 196–200
mail flow and rules, 318–320
Mail Flow section, 171–172, 325, 333, 335
Mailbox Delegation section, 229–230, 232–233
mailbox import and export, 629–632
management interfaces, 21–22, 170–173
managing recipients, 180–183
migration batches with, 574, 581–592
Mobile section, 172–173
moving databases and, 544
mysterious mailboxes, 182–183
naming conventions, 178–180
Offline Address Book and, 362
Organization section, 171, 313, 346, 352
permissions, 143–144, 233–234
Permissions section, 133–134, 143–144, 171, 295
Protection section, 171
public folders and, 766
Public Folders section, 172, 188, 767
RBAC support, 132
Recipients section, 146, 171, 177, 182, 241, 250, 273–274, 581
refresh option, 176
replication queue lengths, 481
reporting licenses, 70–71
running without mailboxes, 329–330
Servers section, 172, 315, 409–410, 494, 537
setting diagnostics for Exchange Server, 330–332
setting up mailboxes, 191–193
sharing policies, 313
starting, 173–175
Unified Messaging section, 172
viewing correct, 174
viewing migration batch properties, 595–596
Windows PowerShell and, 84
Exchange Authorization Library (ADDriver), 90–91
Exchange Control Panel (ECP), 21–22, 170, 269, 707
Exchange database (EDB), 388
Exchange Install Domain Servers group, 75
Exchange LegacyInterOP role group, 74
Exchange Management Console (EMC), 21–22, 650
Exchange Management Shell (EMS)
about, 24, 31, 83, 92–120
accessing mailbox audit data, 759
Active Directory for PowerShell, 120–123
adding database copies, 525
address book policies, 352–353
creating dynamic distribution groups, 306–308
delegated setup and, 63
e-mail address policies, 343–345
enabling archives, 648
executing searches, 731–732
mailbox import and export, 632–637
MailTips management, 377–378
management interfaces, 21–22, 170
migration batches with, 574, 592–595
public folders and, 766
room lists, 270, 298
searches for delivery reports, 327–329
useful examples, 123–128
Windows PowerShell, 83–92, 129
Exchange-MBX.xml file, 52
Exchange Online-ApplicationAccount, 76
Exchange Online Protection (EOP), 16
Exchange Online (Service), 5–9
Exchange Organization Administrators security group, 48
Exchange Personal Information property set, 35
Exchange Server
about, 1–2
Active Directory support, 34–37, 45–48
addressing, 333–386
architectural overview, 10–11
creating organization, 49–50
Database Availability Group, 457–566
deploying, 50–63
development priorities, 2–5
database management, 407–418
database usage statistics, 451–453
I/O operation, 392–402
mailbox assistants, 454–455
managed availability, 226–227, 407
maximum database size, 389–392
rebuilding databases, 448–451
transaction logs, 419–432
utilities supported, 37–41
virtualization, 24–25
Windows PowerShell, 83–130
Exchange Servers role group, 74–75
Exchange Store. See also Information Store
about, 19–20
background maintenance, 432–437
corrupt item detection and isolation, 437–448
database management, 407–418
database usage statistics, 451–453
I/O operation, 392–402
mailbox assistants, 454–455
managed availability, 226–227, 407
maximum database size, 389–392
rebuilding databases, 448–451
transaction logs, 419–432
utilities supported, 37–41
virtualization, 24–25
worker, controller, and memory, 402–407
Exchange Trace Analyzer (ExTRA) utility, 57
Exchange Trusted Subsystem role group
about, 74, 77
FSW and, 500
remote PowerShell and, 89
split permissions model, 166–167
Exchange Web Services Editor (EWSEditor), 40–41
Exchange Web Services (EWS)
  about, 40–41
  ESE versus, 389
  message tracking, 326
  synchronizing information, 792
  testing considerations, 28, 30
Exchange Windows Permissions role group, 74, 166–167
ExchangeGuid property, 107
–ExchangeRemoteMove cmdlet parameter, 612
ExchangeSetup.log file, 56–57
–ExcludeDuplicateMessages cmdlet parameter, 713, 731
–ExcludeFolders cmdlet parameter, 633, 636
executing searches, 731–732
execution policies, PowerShell, 118–119
$ExInstall variable, 101
–ExpansionServer cmdlet parameter, 277
Expert diagnostic level, 331
expiry dates, 437, 684
export. See import and export (mailbox)
Export-CSV cmdlet, 125–127, 281–282
Export-PublicFolderStatistics.ps1 script, 789
Export-RetentionTags.ps1 script, 681
$ExScripts variable, 101
Extensible Storage Engine (ESE)
  about, 388–389
  DAG and, 466
  handling storage failures, 549
  memory allocation and, 402–403
  timestamps, 425
  transaction logs, 419–425, 479
  view tables, 400
ExternalURL property, 366
ExTRA (Exchange Trace Analyzer) utility, 57

F
F2 key, 95
F4 key, 95
F7 key, 95–96
F8 key, 95
facilities provisioning, 191
FAI (folder-associated item), 207
failover, database copies, 466
Failover Cluster Manager, 463–464, 498
$False variable, 101
Federated Services role, 313
FederatedEmail account, 76, 638
file share witness (FSW), 491, 499–501
–Filter cmdlet parameter, 109–110
filters
  address mailboxes in databases, 310–313
  client-side, 110–113
  custom, 308–313
  database, 158–159
  dynamic distribution groups, 308–313
  EAC, 177
email address policies, 341–345
OPATH, 110, 346
server-side, 110–113
firewall rules, 492
–FirstName cmdlet parameter, 192
FirstName property, 185
folder-associated item (FAI), 207
FolderBind action, 756
folders. See also public folders
default, 200–201, 393
larger, 124–125
managed, 688–689
naming conventions, 179
retention policy tags, 661
setting retention policy on, 685–687
–FolderScope cmdlet parameter, 746
–ForceGroupMetricsGeneration cmdlet parameter, 376
Forefront Protection for Exchange (FPE), 16
forest functional mode, 26
Format-List cmdlet, 98
Format-Table cmdlet, 98
$FormatEnumerationLimit variable, 100
Fortune magazine, 643
FPE (Forefront Protection for Exchange), 16
FQDN (fully qualified domain name), 90, 314
free disk space, 432, 522
front-end. See CAS (Client Access Server)
FSW (file share witness), 491, 499–501
Full Access permission, 226, 230, 232–235
FullScanMoveJobsPollingInterval property, 622
fully qualified domain name (FQDN), 90, 314

G
%g variable, 339
GAL (Global Address List)
  about, 35
  address book policies and, 348–349, 351–357
  address lists and, 346
  group naming policy and, 287, 292
  mail-enabled contacts, 250–251
  mail users, 252
  naming mailboxes, 178, 185–187
  Offline Address Book and, 360, 367–371
  public folders, 775
  resource mailboxes, 254–255
  tracking group usage, 286
gap coalescing technique, 394–395
Get-ActiveSyncDevice cmdlet, 125–126
Get-ADDomainController cmdlet, 121
Get-AddressList cmdlet, 347
Get-AdminAuditLogConfig cmdlet, 749
Get-ADUser cmdlet, 122
Get-CalendarProcessing cmdlet, 202, 258–259, 261
Get-CASMailbox cmdlet, 112
Get-ClusterNode cmdlet, 564
Get-Command cmdlet, 85, 93–94
Get-Contact cmdlet, 98
Get-Credential cmdlet, 90, 612
Get-DatabaseAvailabilityGroup cmdlet, 467, 493
Get-DatabaseAvailabilityGroupNetwork cmdlet, 507, 509
Get-DistributionGroup cmdlet, 112
Get-DistributionGroupMember cmdlet, 217, 305
Get-EmailAddressPolicy cmdlet, 334, 341–342
Get-EventLog cmdlet, 98
Get-EventLogLevel cmdlet, 331–332
Get-ExchangeDiagnosticInfo cmdlet, 576
Get-ExchangeServer cmdlet
  about, 92
  finding property values, 410
  identities, 105–106
  reporting issues, 71–72
  selective output, 99
  version numbers, 67–68
Get-ExCommand cmdlet, 86
Get-ExecutionPolicy cmdlet, 118
Get-FailedContentIndexDocuments cmdlet, 734
Get-Group cmdlet, 98, 107, 350
Get-HealthReport cmdlet, 471
Get-InboxRule cmdlet, 320
Get-Mailbox cmdlet
  about, 83
  address book policies, 354–356
  applying retention policies, 678
  arbitration mailboxes, 59, 547
  archive mailboxes, 650
  checking language, 196
  checking mailboxes, 312, 362–363
  checking quotas, 217
  EAC support, 174, 182–183
  equipment mailboxes, 195, 265
  filter examples, 110, 112–113
  health mailboxes, 228, 547
  larger folders and, 125
  limiting number of objects returned, 109–111
  multi-domain forests, 123
  OAB generation mailboxes, 367
  object versions, 69
  removing databases, 417–418
  returning properties, 98, 108
  room mailboxes, 195, 262
  selective output, 99–100
  server names and, 44
  shared mailboxes, 241
Get-MailboxAutoReplyConfiguration cmdlet, 202–203
Get-MailboxCalendarConfiguration cmdlet, 202–204, 261
Get-MailboxCalendarSettings cmdlet, 261
Get-MailboxDatabase cmdlet
  about, 92
  checking mounted databases, 517
  checking OABs, 413
  debugging swelling databases, 444
  moving database locations, 544
  reconnecting mailboxes, 220
  removing database copies, 547
  removing mailboxes, 510
  removing servers, 548
  retrieving maintenance schedule, 434
  updated status information, 481
  viewing activation preferences, 535
  white space, 449–450
  worker processes, 402
Get-MailboxDatabaseCopyStatus cmdlet
  content index, 482
  database copy information, 520–522, 536, 542
  moving database locations, 545–546
  replay queue length, 529
  replication status, 516, 518–519
  server location, 362
  wildcard characters, 481
Get-MailboxExportRequest cmdlet, 626, 637
Get-MailboxExportRequestStatistics cmdlet, 626, 637
Get-MailboxFolder cmdlet, 202, 204, 320
Get-MailboxFolderPermission cmdlet, 240
Get-MailboxFolderStatistics cmdlet, 92, 125, 746, 798
Get-MailboxImportRequest cmdlet, 625, 631, 634–635
Get-MailboxImportRequestStatistics cmdlet, 625, 631, 634–635
Get-MailboxJunkEMailConfiguration cmdlet, 202, 204–205
Get-MailboxMessageConfiguration cmdlet, 202, 204
Get-MailboxRegionalConfiguration cmdlet, 199, 202
Get-MailboxSearch cmdlet, 702, 707, 731
Get-MailboxServer cmdlet, 375, 455, 519
Get-MailboxSpellingConfiguration cmdlet, 202
Get-MailboxStatistics cmdlet
  about, 92, 108, 207
  accessing move report histories, 600
  auditing mailboxes, 184
  checking space usage, 652
  getting more information about a move, 597
  health mailboxes, 228
  quarantined mailboxes, 439
  reconnecting mailboxes, 222
  remote procedure calls and, 128
  removing database copies, 547
  source of mailbox move information, 599
Get-ManagementRole cmdlet, 136, 138, 142
Get-ManagementRoleAssignment cmdlet, 152–155, 162–164, 296
Get-ManagementRoleEntry cmdlet, 136, 138, 146, 156
Get-Member cmdlet, 112
Get-MessageTrackingLog cmdlet, 286
Get-MigrationUser cmdlet, 594–595
Get-MigrationUserStatistics cmdlet, 585, 595, 599
Get-MonitoringItemIdentity cmdlet, 471
Get-MoveRequest cmdlet
  about, 605
  checking status of move request, 609
  clearing move requests, 610
  removing databases, 417–418
  suspending mailbox moves, 617
  viewing move request properties, 598
Get-MoveRequestStatistics cmdlet
  about, 576, 605
  percentage of move completed, 176, 598
  retrieving updated information, 481
  source of mailbox move information, 599
  status detail, 579
  suspending mailbox moves, 616–618
  total mailbox size, 597
  troubleshooting with, 599
Get-Notification cmdlet, 638
Get-OfflineAddressBook cmdlet, 362, 364–366
Get-OrganizationConfig cmdlet, 292, 770
Get-PublicFolder cmdlet, 769, 780, 786
Get-PublicFolderMailboxDiagnostics cmdlet, 772
Get-PublicFolderMoveRequest cmdlet, 784
Get-PublicFolderStatistics cmdlet, 773
Get-Queue cmdlet, 564
Get-Recipient cmdlet
  archive mailboxes, 650
  checking mailboxes, 312
  discovering hidden recipients, 357
  filters and, 112, 343
  validating query results, 305–306
Get-RetentionPolicy cmdlet, 662, 674–675
Get-RetentionPolicyTag cmdlet, 666, 670, 672
Get-RoleGroup cmdlet, 139, 144, 151, 162
Get-RoleGroupMember cmdlet, 144
Get-SearchDocumentFormat cmdlet, 730
Get-SendConnector cmdlet, 70
Get-SiteMailboxDiagnostics cmdlet, 148
Get-StoreUsageStatistics cmdlet, 442, 451–453
Get-ThrottlingPolicy cmdlet, 724
Get-TransportService cmdlet, 324
Get-User cmdlet
  bulk updates and, 115–116
  filters and, 112–113
  handling information returns, 98
  variables and, 102, 107
Get-WebServicesVirtualDirectory cmdlet, 613
  --GetEffectiveUsers cmdlet parameter, 163
GivenName property, 385
Global Address List (GAL)
  about, 35
  address book policies and, 348–349, 351–357
  address lists and, 346
  group naming policy and, 287, 292
mail-enabled contacts, 250–251
mail users, 252
  naming mailboxes, 178, 185–187
  Offline Address Book and, 360, 367–371
  public folders, 775
  resource mailboxes, 254–255
  tracking group usage, 286
global tables, 400
globally unique identifier (GUID)
  active copy of user's mailbox, 44
  archive mailboxes, 651
  database portability and, 459
  default OAB operation, 363
  as mailbox identifiers, 107–109
  system registry and, 438
  target mailboxes, 601
GlobalWebDistributionEnabled property, 366
GoDaddy certificate vendor, 314
Goodman, Steve, 124
Grant Full Access permission, 230
Grant Send As permission, 229
Grant Send On Behalf Of permission, 229
  --GrantSendOnBehalfTo cmdlet parameter, 232
Grogan, Andy, 124
group metrics, 373–382
Group Metrics mailbox assistant, 375–376
Group-Object cmdlet, 92–93, 286
Groupbox control type, 385
groups. See also distribution groups; role groups
  ABPs and, 350
  authentication and, 280–281
  closed, 270
  delegating, 275
  group metrics, 373–382
  moderated, 244–246, 270, 283
  public, 270
  with searches, 728–730
  searches and, 728–730
  security, 252, 284–285
  tracking usage, 285–286
  user-maintained, 286–299
  USGs, 73–77, 139–141, 267
  version numbers, 268
  viewing members, 268–269, 281–283
GroupType property, 284
  --gt operator, 110
GUID (globally unique identifier)
  active copy of user's mailbox, 44
  archive mailboxes, 651
  database portability and, 459
  default OAB operation, 363
  as mailbox identifiers, 107–109
  system registry and, 438
  target mailboxes, 601
Guid property, 108
HABs (hierarchical address books), 349, 371–373
HardDelete action, 756
hardware
development influenced by, 5
freeing disk space, 432, 522
protection against high latency, 442
recycling, 23
transaction log I/O, 428
health mailboxes, 182–183, 226–229, 547
Health Manager Service process (MSExchangeHMHost.exe), 226
Health Manager Worker process (MSExchangeHMWorker.exe), 226
health sets, 471
Help Desk role group, 74, 135, 141, 145–146
Hidden From Address Lists property, 357
hidden recipients, 357–358
hidden system folder, 786
hierarchical address books (HABs), 349, 371–373
hierarchy, public folders, 766–767
high availability. See also DAG (Database Availability Group)
about, 457–459
Active Manager and, 469–472
handling move request errors, 609
lagged copies and, 527
mailbox moves and, 619–621
High diagnostic level, 331
high latency, 442, 508, 550, 553
HighAvailability channel, 503
HoldForMigration parameter, 789
$home variable, 101
HomeMDBBL property, 59
HomeMTA property, 44
Hotmail, 6
HR provisioning, 190
HTML-format messages, 397–398
HTML reports, 127–128
HTTP proxy logs, 367
HTTPS protocol, 8
Hygiene Management role group, 74, 152
Hyper-V, 24–25
hypervisors, 24–25
hyphen (-), 103, 110
I/O operations
about, 392–397
database schema, 398–400
high availability and, 457–458
import of PST data, 626
improvements to, 400–402
maintaining contiguity, 392, 394–395, 397–398
memory and, 463
sizing mailboxes, 394
transaction logs, 427–429
%i variable, 339
ICS (Incremental Change Synchronization), 578, 772
identities (cmdlets), 105–109
–Identity cmdlet parameter, 105–109, 116, 690
–IgnoreDefaultScope cmdlet parameter, 123
–IgnoreNamingPolicy cmdlet parameter, 290
–IgnoreNetworkParameter cmdlet parameter, 509
IIS (Internet Information Services), 53, 85, 88–90
–i-like operator, 110
import and export (mailbox)
about, 623–624
with EAC, 629–632
with EMS, 632–637
exporting mailbox data, 635–637
gaining permission to execute, 624–626
planning import of PST data, 626–629
running multiple concurrent, 633
Import-Module cmdlet, 52, 121
Import-PSSession cmdlet, 90
Import-RetentionTags.ps1 script, 681
in-place holds, 703–708, 726–728, 736
Inbox folder, 393, 658
– IncludedRecipients cmdlet parameter, 307, 347
– IncludeFolders cmdlet parameter, 633, 636
IncludeKeywordStatistics property, 713
– IncludeMoveHistory cmdlet parameter, 600
– IncludeMoveReport cmdlet parameter, 601
– IncludeReport cmdlet parameter, 599
– IncludeSystemTags cmdlet parameter, 662
IncludeUnsearchableItems property, 713
IncomingLogCopyingNetwork property, 522
Incremental Change Synchronization (ICS), 578, 772
incremental resynchronization, 487–488
incremental synchronization, 575, 615
incremental updates, 578
indexes
content, 776
secondary, 393, 395, 437, 578
Information Assistant process, 248
Information Store
about, 388
DAG and, 466
improvements in, 19–20
MailTips and, 374
MRS and, 568
processing overview, 402–407
remote procedure calls to, 128
startup process, 421
Information Store Integrity maintenance utility (ISINTEG), 445
initialization scripts, 90, 100
–Initials cmdlet parameter, 192
InPlaceHoldEnabled property, 707, 714
InPlaceHoldPeriod property, 707
Insert key, 95
Insert log record type, 425
Install-WindowsFeature cmdlet, 47, 51, 53
installing Exchange Server
about, 43–44
Active Directory support, 45–48
creating Exchange organization, 49–50
deployment overview, 50–63
domain controllers and, 495
namespace planning, 77–81
security groups and accounts, 73–77
updates and, 63–73
updating mailbox databases, 414–416
InstallWindowsComponent.ps1 script, 52
Integrated Scripting Environment (ISE), 103–105
Integrated Windows Authentication (IWA), 79–80
interfaces, management, 21–22
Internet Information Services (IIS), 53, 85, 88–90
IP addresses, 492–493, 508
IPM_SUBTREE, 775
ISA Server, 6
–IsArchive cmdlet parameter, 637
iSCSI networks, 509
ISE (Integrated Scripting Environment), 103–105
IsExcludedFromInitialProvisioning property, 210
IsExcludedFromProvisioning property, 210
IsExcludedFromProvisioningBySpaceMonitoring property,
210
IsExcludedFromServingHierarchy flag, 771
ISHierarchicalGroup property, 372
ISINTEG (Information Store Integrity maintenance utility),
445
IsMemberOfDL property, 385
isolating corrupt items. See corrupt items
IssueWarningQuota property, 214, 803
IsSuspendedFromProvisioning property, 210
IT provisioning, 190–191
IWA (Integrated Windows Authentication), 79–80

J

JET database engine, 511
journaling messages, 249
Junk Email Options Assistant, 454

K

Kerberos authentication, 79
keyboard commands, 95–97
keyword query language (KQL), 710–711, 736
KQL (keyword query language), 710–711, 736

L

Label control type, 385
lagged database copies
about, 525–530
Safety Net and, 531–532
self-maintained, 530–532
large-item limit, 584–585, 590
LastAvailableLogTime property, 521
LastCrashTime system registry setting, 438
LastLogGenerated property, 521
LastLogInfoStale property, 521–522
–LastName cmdlet parameter, 192
LastName property, 185
LastNumberOfRecords property, 365
LastStatusTransitionTime property, 521
LastTouched property, 364–365
latency, 442, 508, 550, 553
LCR (local continuous replication), 29
LDAP (Lightweight Directory Access Protocol), 110, 270,
301, 349
LDAPReads metric, 453
LDAPSearches metric, 453
LDIFDE utility, 114, 386
LegacyExchangeDN property, 34, 222
legal compliance. See compliance management
licenses, reporting, 70–73
life cycle of site mailboxes, 800–803
Lightweight Directory Access Protocol (LDAP), 110, 270,
301, 349
–like operator, 110
linked mailboxes, 188
linked value replication (LVR), 268–269
Listbox control type, 385
migration batches, 592
load balancing
moving mailboxes, 568–569
upgrade considerations, 16
local continuous replication (LCR), 29
Log Record Checksum (LRCK) algorithm, 427
log sets (log streams), 419–427
logical corruption, 526
logical unit number (LUN), 395, 428
–LogLevel cmdlet parameter, 731
–LogonType cmdlet parameter, 760
LogRecordBytes metric, 453
LogRecordCount metric, 453
logs and logging. See also transaction logs
Application and Service Logs, 502
Application Event Log, 332, 364, 433, 443, 544
circular, 414, 429–431, 486, 510–514, 554
diagnostic levels, 330–332, 608
fluctuations in number of, 480
HTTP proxy, 367
message-subject, 324
migration batches, 592
placement considerations, 428–429
protection against excessive log growth, 443–444
rebuilding databases, 448–449
reserved, 408, 431–432
setup, 56–57
task, 502
temporary log files, 430–431
lossy failure, 580–581
Lotus Notes, 337, 344
Low diagnostic level, 331
Lowest diagnostic level, 331
LRCK (Log Record Checksum) algorithm, 427
–lt operator, 110
LUN (logical unit number), 395, 428
LVR (linked value replication), 268–269
Lync, integration with, 3, 18
LZ77 compression, 506
LZX compression, 361

M
%m variable, 339
mail-enabled contacts, 179, 250–251
mail-enabled security groups, 284–285
mail-enabling public folders, 779–783
Mail Recipient Creation role, 146–147
Mail Recipients role, 146–147, 156
mail users, 252–253
mailbox assistants, 375–376, 454–455
–Mailbox cmdlet parameter, 783
Mailbox Import Export role, 133, 140, 159, 174, 624
mailbox management. See also specific mailboxes
about, 169–170
activating database copies, 533–534
applying retention policy to, 677–681
attributes and names, 185
auditing mailbox access, 239, 754–762
AutoDatabaseMountDial setting, 539–542
automating settings, 205–206
bulk mailbox creation, 195
common problems, 191
completing mailboxes, 193–194
creating mailboxes, 187–209
custom attributes, 207–209
custom filter to address mailboxes in databases, 310–313
data retention states, 745
default folders, 200–201
disabling mailboxes, 218–220, 223
enabling mailboxes for auditing, 757–758
language support, 196–200
mail-enabled contacts, 250–251
mail users, 252–253
mailbox assistants, 375–376, 454–455
mailbox databases, 410–416
mailbox quotas, 213–218, 391–392, 396, 606
mailbox repair cmdlets, 445–448
managing recipients, 180–183
manipulating settings, 202–206
message quotas, 128
MFCMAPI utility and, 38–40
migrating DAGs, 464–465
moderated recipients, 243–250
moving mailboxes, 59–60, 567–640
naming mailboxes, 185–187
need for mailboxes, 183–184
OAB generation mailboxes, 366–367
operational flexibility and, 4
perfection and progress, 170–180
placing mailboxes on hold, 19
provisioning mailboxes, 607
quarantined mailboxes, 439–441
recalling messages, 241–243
reconnecting mailboxes, 220–223
removing mailboxes, 218–220, 223
resources provisioning management agent, 209–213
role assignment policies and, 164–166
setting mailbox permissions, 229–240
setting mailbox quotas, 213–218
sizing mailboxes, 391–394
transport service, 27–28
viewing properties, 176
Mailbox Replication Proxy service (MRSPProxy), 612–614
Mailbox Replication Service (MRS)
about, 567–568
archive mailboxes and, 211
autosuspended state for moves, 587
configuration file, 621–623
corrupt item detection and isolation, 440–441
creating mailbox databases, 413–414
ensuring high availability, 620–621
improvements in, 8
mailbox import and export, 623
managing individual mailbox moves, 605–611
migration batches, 584–590, 593
Migration service and, 574
MSExchangeRepl versus, 465, 567
planning import of PST data, 627
planning mailbox moves, 615
preventing loss of data, 580–581
processing overview, 575–578
removing databases, 418
reporting mailbox moves, 599–604
suspending mailbox moves, 616–618
system resources and, 579–580
transaction log replay, 480
Mailbox Search role, 159, 704
Mailbox server role (backend), 1, 314
mailbox tables, 400
Mailbox Transport Submission service, 243
MailboxDatabaseFailureItems channel, 503
MailboxGuid property, 108
MailboxReplicationService policy, 579, 607
MailboxReplicationServiceHighPriority policy, 579, 607
MailboxType property, 593
MailTips
  about, 373–376
  auto-reply message, 203
  client interaction, 376–377
  configuring, 377–378
  customizing, 380–381
  distribution groups, 187
  DLP and, 18
  mail delivery indication, 251
  moderated groups, 245–246
  moderated mailboxes, 250
  multilingual, 381
  OAB support, 371, 381–382
  setting up, 253, 275
  user experience, 378–379
–MailTipsAllTipsEnabled cmdlet parameter, 377
–MailTipsExternalRecipientTipsEnabled cmdlet parameter, 378
–MailTipsGroupMetricsEnabled cmdlet parameter, 378
–MailTipsLargeAudienceThreshold cmdlet parameter, 378
–MailTipsMailboxSourcedTipsEnabled cmdlet parameter, 378
–MailTipTranslations cmdlet parameter, 381
MailUniversalDistributionGroup, 267
MailUniversalSecurityGroup, 267
maintenance, background. See background maintenance
Majority Node Set clusters, 493
Managed Availability (MSEExchangeHWWorker)
  Active Manager and, 469–472, 536
  database redundancy, 522
  health mailboxes and, 226–227
  Managed Store and, 407
  tracking actions, 472
Managed Folder Assistant (MFA)
  about, 454
  background maintenance, 390
  cumulative updates, 65
  Email Lifecycle Assistant, 726–727
  expired items, 673
  PST data and, 629
  public folder mailboxes, 773
  retention policies and, 653, 655, 665–666, 684–685, 689–696
  retention settings for mailboxes, 513
  writing policy information into mailbox, 678
managed folders, 688–689
Managed Store. See Exchange Store
–ManagedBy cmdlet parameter, 149, 151, 276
ManagedBy property, 113
–ManagedFolderMailboxPolicy cmdlet parameter, 193
ManageGroupManagementRole.ps1 script, 297
Management Committee group, 245
management interfaces, 21–22, 170
management roles. See roles
managing databases. See database management
–ManualDAGNetworkConfiguration cmdlet parameter, 509
MAPI (Messaging Application Programming Interface)
  about, 8
  applying retention policies, 689
  DAG networks and, 507
  LZ77 algorithm and, 506
  MFCMAPI utility, 38–40
  OAB support, 366
  properties supported, 692–693
MaxActiveMovesPerSourceMDB property, 622
MaxActiveMovesPerSourceServer property, 622
MaxActiveMovesPerTargetMDB property, 622
MaxActiveMovesPerTargetServer property, 622
MaxConcurrentMigrations property, 614
maximum database size, 389–392, 614
maximum number of active databases, 402–403
maximum number of open site mailboxes, 800
MaximumActiveDatabases property, 402–404
MaxMoveHistoryLength property, 623
MaxReceiveSize property, 803
MaxRetries property, 623
–MaxSendSize cmdlet parameter, 584–585, 605, 627
MaxTotalMovesPerMRS property, 622
Medium diagnostic level, 331
meeting requests
  meeting room conflicts, 260
  processing, 261–265
  room lists, 270, 298–299
MemberDepartRestriction property, 272, 284
MemberJoinRestriction property, 272, 284, 294
memory management
  Exchange Store, 402–407
  I/O operations and, 463
  mailbox searches and, 726
  PowerShell memory limits, 111
Message Tracking role, 137–138, 141, 147, 323
MessageBind action, 756–757
messages. See also addressing messages
  approving, 274
  auto-reply, 187
  changes in submission, 489
  compressing, 398
  FAI, 207
  health mailboxes, 227, 229
  journaling, 249
  message-subject logging, 324
  moderated groups and, 245
  processing moderation requests, 247–249
  recalling, 241–243
  recovering, 643
  sending on behalf of other users, 237–239
  troubleshooting, 443
MessageTrackingReportId property, 328
MessageTypes property, 713
Messaging Application Programming Interface (MAPI) about, 8
applying retention policies, 689
DAG networks and, 507
LZ77 algorithm and, 506
MFCMAPI utility, 38–40
OAB support, 366
properties supported, 692–693
messaging connectors, 179
messaging records management (MRM) about, 18, 641, 657
archive mailboxes and, 650
creating retention policies, 673–677
creating retention tags, 668–673
designing retention policies, 663–665
Managed Folder Assistant, 665–666
naming retention tags, 666–668
system tags, 662–663
types of retention tags, 657–662
metadata, 578, 712
MFA (Managed Folder Assistant) about, 454
background maintenance, 390
cumulative updates, 65
Email Lifecycle Assistant, 726–727
expired items, 673
PST data and, 629
public folder mailboxes, 773
retention policies and, 653, 655, 665–666, 684–685, 689–696
retention settings for mailboxes, 513
writing policy information into mailbox, 678
MFCMAPI utility about, 38–40
accessing mailbox audit data, 759
deferred action messages, 319
quarantined mailboxes and, 439–440
validating dates, 696
viewing properties, 693, 796–797
Microsoft Business Productivity Online Service, 6
Microsoft Exchange Replication service
Active Manager and, 465–479
handling storage failures, 549–550
MRS versus, 465, 567
verifying transaction logs, 482
Microsoft Exchange Security Groups OU, 73, 75, 148
Microsoft Exchange Server 2013. See Exchange Server
Microsoft Exchange Service Host process, 572–573
Microsoft Exchange System Objects OU, 781
Microsoft Exchange Transport Log Search service, 326
Microsoft Federation Gateway, 313
Microsoft Internet Information Services, 53, 85, 88–90
Microsoft Lync, integration with, 3, 18
Microsoft Management Console (MMC), 21, 36, 170, 382
Microsoft SharePoint, 3, 18, 389
Microsoft System Center Data Protection Manager, 527
Microsoft System Center Operations Manager, 442–443, 470, 520
Microsoft Technology Adoption Program, 14, 68
Microsoft Unified Communications Managed API, 53
migrating DAGs, 464–465
Migration account, 76
migration arbitration mailbox, 573
Migration Batch Wizard, 588
migration batches clearing, 584
completing, 590
controlling processing, 615–616
with EAC, 574, 581–592
with EMS, 574, 592–595
logging, 592
removing, 584
removing move requests, 591, 610–611
viewing details, 589
viewing properties of, 595–596
migration endpoints, 611–612
Migration role, 148
Migration service about, 570, 572–575
logging activity, 592
managing individual mailbox moves, 605–611
managing migration batches with EAC, 581–590
managing migration batches with EMS, 592–595
planning mailbox moves, 614–615
MMC (Microsoft Management Console), 21, 36, 170, 382
mobile access to public folder mailboxes, 786
mobile devices, 786
MobilePhone property, 154
moderated groups, 244–246, 270, 283
moderated mailboxes, 249–250
moderated recipients, 243–250
moderation requests, 247–249
modern public folders, 20–21, 420–421, 787–790
monitoring database copies, 517–522
Mount-Database cmdlet, 411–412, 512, 546
-MountDialOverride cmdlet parameter, 539
mounted databases, 407, 426–427, 462–463, 546
Move action, 756
Move-ActiveMailboxDatabase cmdlet, 538–541
Move-DatabasePath cmdlet, 415, 545–546
Move-Mailbox cmdlet, 606
Move Mailboxes role, 142, 148, 152
move reports about, 597–598
accessing histories, 599–606
outputting to files, 600–601
move requests checking and altering status, 609–611
clearing, 610–611
enforcing age limits for, 610–611
handling errors, 607–609
removing, 591, 610–611
viewing properties, 598
move tags, 672
MoveToArchive action, 672
MoveToDeleteItems action, 756
moving databases
moving locations within DAGs, 544–546
potential issues, 539–542
server health and, 536
transaction logs and, 480
moving mailboxes
about, 567–571
alerts and, 638–639
assigning priority, 607
asynchronous moving, 571–572
checking and altering move request status, 609–610
creating mailbox databases, 413–414
enabling MRSProxy, 612–614
ensuring high availability, 619–621
handling move request errors, 607–609
import and export processes, 623–637
managing individual moves, 605–611
migration batches with EAC, 581–592
migration batches with EMS, 592–595
migration endpoints, 611–612
Migration service, 572–575
MRS and, 567–568, 575–581, 621–623
planning, 614–619
removing databases, 418
reporting moves, 595–604
scheduling, 617
suspending, 615–619
transaction log replay, 480
transaction logs, 578
variations in speed, 619
moving public folders, 783–785
MRM (messaging records management)
about, 18, 641, 657
archive mailboxes and, 650
creating retention policies, 673–677
creating retention tags, 668–673
designing retention policies, 663–665
Managed Folder Assistant, 665–666
naming retention tags, 666–668
system tags, 662–663
types of retention tags, 657–662
MRS (Mailbox Replication Service)
about, 567–568
archive mailboxes and, 211
autosuspended state for moves, 587
configuration file, 621–623
corrupt item detection and isolation, 440–441
creating mailbox databases, 413–414
ensuring high availability, 620–621
improvements in, 8
mailbox import and export, 623
managing individual mailbox moves, 605–611
migration batches, 584–590, 593
Migration service and, 574
MSExchangeRepl versus, 465, 567
planning import of PST data, 627
planning mailbox moves, 615
preventing loss of data, 580–581
processing overview, 575–578
removing databases, 418
reporting mailbox moves, 599–604
suspending mailbox moves, 616–618
system resources and, 579–580
transaction log replay, 480
MRSProxy (Mailbox Replication Proxy service), 612–614
MsExchange Database counter, 435
MsExchange Database/Instances/DB Maintenance IO Reads/sec counter, 449
MsExchangeDAGMgmt.exe (Database Availability Group management service), 503
MsExchangeHMHost.exe (Health Manager Service process), 226
MsExchangeHMWorker.exe (Health Manager Worker process), 226
MsExchangeHomeServerName property, 44
MsExchangeHWWorker (Managed Availability)
Active Manager and, 469–472, 536
database redundancy, 522
health mailboxes and, 226–227
Managed Store and, 407
tracking actions, 472
MsExchangeRepl (Replication service)
Active Manager and, 465–479
handling storage failures, 549–550
MRS versus, 465, 567
verifying transaction logs, 482
MsExchangeServicesAppPool setting, 614
MsExchDataPath property, 236
MsExchDelegateLinkList property, 236–237
MsExchESEParamCacheSizeMax property, 405
MsExchESEparamCacheSizeMin property, 406
MsExchExtensionAttribute property, 385
MsExchGroupExternalMemberCount property, 374
MsExchGroupMemberCount property, 374
MsExchMailboxMoveBatchName property, 577
MsExchMailboxMoveFlags property, 577
MsExchMailboxMoveRemoteHostName property, 577
MsExchMailboxMoveSourceMDBLink property, 577
MsExchMailboxMoveStatus property, 577
MsExchMailboxMoveTargetMDBLink property, 577
MsExchMaxActiveMailboxDatabases property, 34
MsExchMDBAvailabilityGroupLink property, 496
MSEchQueryFilter property, 301
multi-domain forest, 122–123
multi-forest design, 45–46, 172
Multivalued Listbox control type, 385
MyBaseOptions role, 164
MyContactInformation role, 164
MyDistributionGroupMembership role, 165, 350
MyDistributionGroups role, 165, 290, 295–296
MyMarketPlaceApps role, 165
MyProfileInformation role, 164
MyRetentionPolicies role, 165, 682
MyTeamMailboxes role, 165
MyTextMessaging role, 165
MyVoiceMail role, 164

N
–Name cmdlet parameter, 192, 251, 253, 307, 348, 731
Name property, 185
name resolution, 365–366
Name Service Provider interface (NSPI), 365
named parameters, 97
namespace planning
about, 77
handling connections with CAS, 79–80
protocol-specific, 80–81
self-signed certificates, 77–78
naming conventions
about, 43–44
applying different, 186
EAC, 178–180
mailboxes, 185–187
public folder mailboxes, 768
retention tags, 666–668
retroactive, 180
underscores, 26
user-maintained groups, 289–295
NDRs (nondelivery reports), 420
–ne comparison operator, 110
.NET Framework, installing, 44
network interface cards (NICs), 458, 499, 506–508
NetworkCompression property, 505
NetworkEncryption property, 505
networks
APIPA addressing, 492
discovery mailboxes, 225
enforcing age limits for move requests, 610–611
immediate processing, 440
mailbox provisioning and, 607
migration processing, 573, 575, 587
suspending mailbox moves, 616, 618
New-OfflineAddressBook cmdlet, 354
New-PSSession cmdlet, 89–90
New-PublicFolder cmdlet, 777, 783
New-PublicFolderMoveRequest cmdlet, 789
New-PSUser cmdlet, 145, 148, 158
New-SiteMailbox cmdlet, 148, 801
New-TestCASConnectivityUser.ps1 script, 183
New-ThrottlingPolicy cmdlet, 725
NICs (network interface cards), 458, 499, 506–508
node runner processes, 406
New-FederationTrust cmdlet, 313
New-GlobalAddressList cmdlet, 353
New-InboxRule cmdlet, 319–320
New-Item cmdlet, 120
New-Mailbox cmdlet
about, 206
archive mailboxes, 648
database scope, 159
discovery mailboxes, 225
equipment mailboxes, 195
OAB generation mailboxes, 367
properties supported, 192–193
remote PowerShell, 91
room mailboxes, 195
New-MailboxAuditLogSearch cmdlet, 755, 759–760
New-MailboxDatabase cmdlet, 129, 212, 412
New-MailboxExportRequest cmdlet, 160, 625, 636, 722
New-MailboxFolder cmdlet, 202, 204
New-MailboxImportRequest cmdlet
about, 625, 632
importing PST data, 635
role assignments and, 160
tickles, 576
New-MailboxRepairRequest cmdlet, 446–447
New-MailboxRestoreRequest cmdlet, 581, 785
New-MailboxSearch cmdlet, 702–703, 731–732
New-MailContact cmdlet, 251
New-MailUser cmdlet, 253
New-ManagementRole cmdlet, 156, 161
New-ManagementRoleAssignment cmdlet, 157, 160–162, 323, 625
New-ManagementScope cmdlet, 143, 158–159
New-MigrationBatch cmdlet, 592–594, 611, 639, 790
New-MigrationEndpoint cmdlet, 612, 614
New-MoveRequest cmdlet
about, 148, 605–606
database scope and, 159
discovery mailboxes, 226
enforcing age limits for move requests, 610–611
immediate processing, 440
mailbox provisioning and, 607
migration processing, 573, 575, 587
suspending mailbox moves, 616, 618
New-OfflineAddressBook cmdlet, 354
New-PSSession cmdlet, 89–90
New-PublicFolder cmdlet, 777, 783
New-PublicFolderMoveRequest cmdlet, 789
New-PSUser cmdlet, 145, 148, 158
New-SiteMailbox cmdlet, 148, 801
New-TestCASConnectivityUser.ps1 script, 183
New-ThrottlingPolicy cmdlet, 725
NICs (network interface cards), 458, 499, 506–508
node runner processes, 406
nondelivery reports (NDRs), 420
nonsystem tags, 662–663
notification, email, 595–596
–NotificationEmails cmdlet parameter, 594
–notlike operator, 110
NSPI (Name Service Provider interface), 365
NTFS file system, 389
$Null variable, 101, 680

O
OAB generation mailboxes, 366–367
OAB (Offline Address Book)
about, 357–358
address book policies, 349, 354–355
contents of, 365–366
creating customized, 367–371
creating mailboxes, 366–367
default operation, 363–365
e-mail address policies, 340
generating, 362–367
group metrics, 375
mailbox databases and, 412–413
MailTips support, 378, 381–382
mixed environment, 360–361
Outlook and, 359–362
user-maintained groups, 288
OABGeneratorAssistant, 363–365
OABGeneratorWorkCycle property, 363
OABGeneratorWorkCycleCheckPoint property, 363
Oab.xml file, 361
object versions, 69–70
ObjectModified property, 752
OffCAT (Office Configuration Analyzer Tool), 37
Office 365
about, 349
archive and, 649
influence of cloud service, 9
Managed Availability, 469
motivation for upgrading, 12
public folder mailboxes, 773
site mailboxes and, 800
Office Configuration Analyzer Tool (OffCAT), 37
Office property, 351–352
Offline Address Book (OAB)
about, 357–358
address book policies, 349, 354–355
contents of, 365–366
creating customized, 367–371
creating mailboxes, 366–367
default operation, 363–365
e-mail address policies, 340
generating, 362–367
group metrics, 375
mailbox databases and, 412–413
MailTips support, 378, 381–382
moderated groups, 283
Offline Address Book and, 359–362
retention policies, 684–686
retention tags, 666–667, 676–677
room list groups, 299
site mailboxes and, 792, 799–800
troubleshooting, 443
upgrading versions, 11, 13, 16

Outlook Web App (OWA)
about, 11
accessing personal tags, 662–663
address book policies, 356
archive mailboxes, 649
delegating access, 239–240
delivery reports, 321–322
EAC and, 177
expired items and, 673
group membership and, 281
language support, 196–200
MailTips settings, 377
MailTips support, 378–379
maintaining rules, 319
managing groups, 276, 287
opening another user’s mailbox, 237
Options section, 154–156, 269, 276, 287, 331
Organize Email section, 321
People section, 348
retention policies, 684–686
retention tags, 666–667, 676
room list groups, 299
security groups, 284
Windows PowerShell and, 84

PagePreRead metric, 453
PageRead metric, 453
PAM (Primary Active Manager) role, 466–467
parameters. See also specific cmdlet parameters
named, 97
positional, 97, 105
Partial Attribute Set (PAS), 35
PAS (Partial Attribute Set), 35
passive database copies, 460, 544
–Password cmdlet parameter, 192
passwords, mailbox, 190–191
Patches system registry setting, 69
Performance Monitor, 435, 449, 517, 520
–Permanent cmdlet parameter, 220
PermanentlyDelete action, 672
permissions
discovery mailboxes, 226
EAC, 143–144, 233–234
mailbox import and export, 624–626
preparing Active Directory, 48
public folders, 775
revoking, 232
roles and, 139
setting for mailboxes, 229–240
split permissions model, 166–167
testing operational processes, 29–30
personal archive. See archive mailboxes
personal tags
about, 658, 660
accessing through OWA, 682
applying to items, 685–686
creating, 671–672
naming, 666–668
pipelines, 101–102, 109, 678
PKI (public key infrastructure), 315
planning
import of PST data, 626–629
mailbox moves, 614–619
policies. See also retention policies
address book, 348–357
EAC, 313
e-mail address, 333–345
execution, 118–119
group naming, 287, 289–295
mailbox creation, 190–191
MRS, 579
processing meeting requests, 261–265
removing tags from, 687–688
Resource Booking Attendee and, 256–261
role assignment, 133–134, 153–155, 163–166
sharing, 313
site mailboxes, 803–804
user-maintained groups, 289–295
portability, database, 459–464
positional parameters, 97, 105
PowerShell. See Windows PowerShell
PR_ARCHIVE_DATE property, 693
PR_ARCHIVE_PERIOD property, 693
PR_ARCHIVE_TAG property, 693
pre-staging Cluster Name Object, 497–499
precoded filters, 310
preparing for Exchange Server
about, 26–27
Active Directory, 47–48
installation considerations, 63–73
namespace planning, 77–81
test plans, 28–29
testing for operational processes, 29–30
testing for programming and customizations, 30–31
transport considerations, 27–28
updating versions, 31–32
–PreserveMailboxSizeLimit cmdlet parameter, 606
preserving information
about, 696–698
creating searches, 708–716
examining search results, 720–724
Exchange searches, 733–736
groups with searches, 728–730
in-place holds, 703–708
putting mailboxes on litigation hold, 699–701
putting mailboxes on retention hold, 698–699
removing searches, 730–731
resource throttling for searches, 724–726
retrieving discovered content, 716–720
search syntax, 736–737
searching mailbox content, 701–703
PreviewResultsLink property, 714
Primary Active Manager (PAM) role, 466–467
PrimarySMTPAddress property, 185
priorities
assigning for moves, 607
development, 2–5
email policy, 334–336, 340
privacy
challenges of, 4–5
complying with laws, 208
ProcessExternalMeetingMessages property, 262, 264
Products system registry setting, 69
$Profile variable, 120
profiles, 119–120
ProhibitSendQuota property, 115, 214
ProhibitSendReceiveQuota property, 214, 771, 803
properties. See specific properties
protected distribution groups, 278–281
protocols and required certificates, 314
ProvisionedFolder corruption type, 446
provisioning
mailboxes, 607
servers, 63
site mailboxes, 803–804
ProxyAddresses property, 385
PR_POLICY_TAG property, 693
PR_RETENTION_DATE property, 692
PR_RETENTION_FLAGS property, 693
PR_RETENTION_PERIOD property, 692
PR_ROAMING_XMLSTREAM property, 691
PR_START_DATE_ETC property, 692, 694
PST Capture Tool, 628
PST files
about, 201
archive mailboxes and, 644–647
importing data directly into archives, 635
legitimate movement, 443
mailbox import and export, 623–624, 629–637
planning import, 626–629
public folder mailboxes
about, 766
calculating size, 772–773
controlling the root, 774–776
creating, 767–770
mobile access, 786
naming conventions, 768
number needed, 770–772
redirecting content, 785–786
writeable hierarchy, 770
Public Folder Management role group, 75, 135, 774
public folders
about, 765–766, 804
accessing, 778
content indexing and, 776
creating, 776–779
creating mailboxes, 767–770
hierarchy in, 766–767
importing-exporting data in, 624
mail-enabling, 779–783
modern, 20–21, 420–421, 787–790
moving, 783–785
naming conventions, 179
new concepts, 766–767
organizational forms library, 786
permissions, 775
removing, 786
transaction logs and, 420–421
Public Folders container, 345
public groups, 270
public key infrastructure (PKI), 315
PublicFolderMigrationComplete flag, 790
PublicFolderToMailboxMapGenerator.ps1 script, 789
Purges folder, 746
$pwd variable, 101
Q

QBDG suffix, 179
quarantined mailboxes, 439–441
queries
  OPATH, 268, 270, 300–302
  validating results, 305–306
queue lengths, 481, 517, 529
queues, MRS, 576
quorums
  dynamic, 494
  FSW role and, 499
QuotaNotificationSchedule property, 215
quotas
  accommodating imported data, 626
  dealing with I/O, 396
  ensuring sufficiency of, 606
  Recoverable Items, 746–747
  setting, 213–218
  site mailboxes, 803
  sizing mailboxes, 391–392
quotation marks ("), 102–103

R

RBAC Address Lists role, 346
RBAC Discovery Management role group, 704
RBAC (role-based access control)
  about, 85, 131–136
  Active Directory, 35
  execution policies and, 118
  figuring out, 167–168
  mailbox import and export, 624–626
  remote Windows PowerShell, 86–92
  role assignment, 133–134, 152–166
  role entries and, 134
  role groups and, 133, 139–141
  role scope and, 134, 141–143, 409
  roles and, 133, 136–139
  security groups, 284
  split permissions model, 166–167
  testing for operational processes, 29–30
  universal security groups, 73
  working with, 143–152
RCA (Remote Connectivity Analyzer), 37
read-status tracking, 324
rebalancing database copies, 534–536
rebuilding databases, 448–451
Recall Message feature, 241–243
Recipient Management role group
  about, 75
  administrator searches and, 322
  creating security groups, 284
  linking to tasks, 135
  remote PowerShell and, 91
Recipient Policies role, 148
Recipient read scope, 141
Recipient write scope, 142
  –RecipientContainer cmdlet parameter, 307–308
RecipientContainer property, 301
  –RecipientFilter cmdlet parameter, 309–311, 347
RecipientFilter property, 301
  –RecipientOrganizationalUnitScope cmdlet parameter, 157
  –RecipientPathFilter cmdlet parameter, 328
recipients
  about, 180
  hidden, 357–358
  moderated, 243–250
Recipients property, 713
RecipientType property, 284
RecipientTypeDetails property, 228, 284, 802
reconnecting mailboxes, 220–223
Records Management role group, 75, 136, 322
Recoverable Items structure
  about, 737
  asynchronous moving, 571
  function of, 737–739
  improvements, 739–742
  managing quotas for, 746–747
  retention periods and, 671
  single-item recovery, 743–745
  viewing items in, 745–746
RecoverableItemsQuota setting, 757
recovering failed servers, 61–63
recovering messages, 643
  –Recurse cmdlet parameter, 769
recycling hardware, 23
Redirect-Message cmdlet, 564
redirecting content to public folder mailboxes, 785–786
RedistributeActiveDatabases.ps1 script, 535–536
redundancy, database, 411, 522
registry, system. See system registry
regulatory compliance. See compliance management
released to manufacturer (RTM), 13–14
relocating databases, 480, 536, 539–542, 544–546
Remote Connectivity Analyzer (RCA), 37
remote management, 492
remote procedure call (RPC)
  avoiding access errors, 492
  delivering messages, 28
  MAPI, 8, 506
  retrieving mailbox information, 128
remote Windows PowerShell
  about, 86–88
  connecting to, 88–91
  limiting user functionality, 91–92
RemoteExchange.ps1 script, 90, 119–120
RemoteSigned execution policy, 118
Remove-ADPermission cmdlet, 232
Remove-DatabaseAvailabilityGroupNetwork cmdlet, 509
Remove-DatabaseAvailabililtyGroupServer cmdlet, 494, 548–549
Remove-DistributionGroup cmdlet, 297
Remove-Mailbox cmdlet, 155, 219–220, 223
Remove-Mailbox Permission cmdlet, 205
Remove-MailboxDatabase cmdlet, 418, 548
Remove-MailboxDatabaseCopy cmdlet, 544, 547
Remove-MailboxExportRequest cmdlet, 626
Remove-MailboxImportRequest cmdlet, 625, 635
Remove-MailboxSearch cmdlet, 730–731
Remove-ManagementRoleAssignment cmdlet, 165
Remove-MigrationBatch cmdlet, 594
Remove-MigrationUser cmdlet, 591
Remove-MoveRequest cmdlet
  about, 591
  clearing move requests, 610
  migration batches with EAC, 584–585, 605
  removing databases, 418
  suspending mailbox moves, 619
Remove-PublicFolder cmdlet, 786
Remove-PublicFolderMoveRequest cmdlet, 785
Remove-RetentionPolicy cmdlet, 688
Remove-RetentionPolicy cmdlet, 671–672
Remove-RetentionPolicyTag cmdlet, 152
Remove-RestoreMailbox cmdlet, 219–220
RemovePrivateProperty property, 258
removing
  database copies, 546–548
  databases, 416–418
  mailboxes, 218–220, 223
  migration batches, 584
  move requests from migration batches, 591
  public folders, 786
  retention policies, 688
  searches, 730–731
  servers, 548–549
  tags from policies, 687–688
renaming databases, 414–415
repair cmdlets, 445–448
repairing Exchange Server, 61
Replace log record type, 425
replay, transaction log. See transaction log replay
ReplayLagManagerNumAvailableCopies system registry setting, 531
ReplayLagPlayDownPercentDiskFreeSpace system registry setting, 531
ReplayLagStatus property, 522
ReplayLagTime property, 527–528
ReplayQueueLength property, 521
replication
  block mode, 484–486
  database, 390, 411, 459
  public folders and, 767
queue lengths, 481, 517
transaction log replay and, 479–489
Replication service (MSExchangeRepl)
  Active Manager and, 465–479
  handling storage failures, 549–550
  MRS versus, 465, 567
  verifying transaction logs, 482
ReplicationPort property, 505
reporting licenses, 70–73
reporting mailbox moves
  about, 595–596
  accessing move report histories, 599–604
  getting more information about moves, 597–598
  troubleshooting, 599
reports
  delivery, 320–329
  HTML, 127–128
representation state transfer (REST), 792
REQ file, 317
RequestInPolicy property, 262–263
RequestOutOfRangePolicy property, 262, 264
  --RequireSenderAuthenticationEnabled cmdlet parameter, 280
Reread Logon Quotas interval registry setting, 217, 235
reseeding database copies, 523–525
reserved logs, 408, 431–432
Reset Password role, 137
  --ResetPasswordOnNextLogon cmdlet parameter, 192
resilience equation, 558–559
Resource Booking Assistant, 454
Resource Booking Attendant, 256–262, 265, 299
resource forest design, 46
resource mailboxes
  about, 188, 254–255
  creating, 195
  defining custom properties, 255–256
  naming conventions, 178–179
resource throttling for searches, 724–726
ResourceCapacity property, 255
ResourceCustom property, 255
resources provisioning management agent, 209–213
REST (representation state transfer), 792
Restore-MailboxExportRequest cmdlet, 576
Restricted execution policy, 118
  --ResultSize cmdlet parameter, 109–111, 328
Resume-ClusterNode cmdlet, 565
Resume-MailboxDatabaseCopy cmdlet, 529, 546
Resume-MailboxExportRequest cmdlet, 626
Resume-MailboxImportRequest cmdlet, 625
Resume-MoveRequest cmdlet, 605, 608, 617
Resume-PublicFolderMigrationRequest cmdlet, 790
Resume-PublicFolderMoveRequest cmdlet, 784
resynchronization
  after moving mailboxes, 587
  incremental, 487–488
retention date calculation, 693–696
retention hold (mailboxes), 19, 697–699
retention policies
  applying to mailboxes, 677–681
  compliance management and, 644, 646
  creating, 663, 673–677
  customizing for specific mailboxes, 681–683
  designing, 663–665
  Managed Folder Assistant and, 653, 655, 665–666, 689–696
  managing in hybrid environment, 681
  modifying, 681
  removing, 688
  setting on folders, 685–687
  user interaction with, 684–685
retention policy tags (RPT), 658–659, 666
retention tags
  changing, 688
  creating, 668–673
  disabling, 687
  managing in hybrid environment, 681
  MRM and, 657
  naming, 666–668
  removing from policies, 687–688
  troubleshooting, 671
  types of, 657–662
RetentionAction property, 670
RetentionComment property, 680, 700
–RetentionEnabled cmdlet parameter, 672
RetentionEnabled property, 666
RetentionURL property, 678, 680, 700
RetryDelay property, 623
revoking permissions, 232
Rich Text Format (RTF), 397–398
Richard, Pat, 124
role assignment policies, 133–134, 153–155, 163–166
role assignments
  about, 133, 152–163
  controlling, 153
  default, 141
  delegated, 160
role-based access control (RBAC)
  about, 85, 131–136
  Active Directory, 35
  execution policies and, 118
  figuring out, 167–168
  mailbox import and export, 624–626
  remote Windows PowerShell, 86–92
  role assignment, 133–134, 152–166
  role entries and, 134
  role groups and, 133, 139–141
  role scope and, 134, 141–143, 409
  roles and, 133, 136–139
  security groups, 284
  split permissions model, 166–167
  testing for operational processes, 29–30
universal security groups, 73
working with, 143–152
role entries, 134
role groups. See also specific role groups
  about, 133
  adding roles to, 147–149
  creating, 145–149
  default, 141
  identifying membership, 162–163
  linking to tasks, 135–136
  maintaining membership, 149–150
  managing, 151–152
  specific scopes for, 157–158
universal security groups and, 139–141
role scope, 134, 141–143, 157–158, 409
roles
  about, 133, 136–139
  adding to role groups, 147–149
  cmdlets and, 139
  creating for specific tasks, 155–157
  delegating, 140, 163, 625
  EAC and, 175
  end-user, 164–165
  scopes and, 141–143
  unscoped, 160–162
–Room cmdlet parameter, 195
room lists, 270, 298–299, 349
room mailboxes
  about, 188, 254–255
  creating, 195
  defining custom properties, 255–256
  naming conventions, 178–179
  viewing delegates, 263
–RoomList cmdlet parameter, 298
ROPCount metric, 453
RPC (remote procedure call)
  avoiding access errors, 492
  delivering messages, 28
  MAPI, 8, 506
  retrieving mailbox information, 128
RPT (retention policy tags), 658–659, 666
RTF (Rich Text Format), 397–398
RTM (released to manufacturer), 13–14
rule processing, 318–320, 492
S
S/MIME (Secure Multipurpose Internet Mail Extensions), 734
%s variable, 339
Safety Net, 459, 503, 531–532
SAM (Standby Active Manager), 190
SAM (Standby Active Manager) role, 466
SAN (storage area networks), 557
Sarbanes–Oxley Act, 641
SCC (single copy cluster), 29
schedules, maintenance, 434
Scheduling Assistant, 454
scheduling mailbox moves, 617
scheduling threads, 550
Schema Administrators group, 48
schemas
Active Directory information, 35–36
database, 398–400
preparing, 47–48
SCOM (System Center Operations Manager), 442–443, 470, 520
SCR (standby continuous replication), 29
scripting agent, 205
ScriptingAgentConfig.xml file, 205–206
scripts. See also specific scripts
calling, 117
cmdlets in, 93, 117
conditional checking, 103
initialization, 90, 100
outputting CSV files, 125–127
pipelines and, 101–102
writing, 93
Search-ADAccount cmdlet, 122
Search-AdminAuditLog cmdlet, 751–752
Search Foundation, 578, 701, 733–735
Search-Mailbox cmdlet, 703, 731
Search-MailboxAuditLog cmdlet, 755, 759
Search-MessageTrackingReport cmdlet, 326–328
–SearchDumpster cmdlet parameter, 731
SearchDumpster property, 713
searches
concurrent, 733
creating, 708–716
delivery reports, 321–329
eDiscovery, 224–225, 233, 702, 726
examining results, 720–724
Exchange support, 733–736
executing, 731–732
groups with, 728–730
mailbox content, 701–703
memory limitations and, 726
refining, 712–716
removing, 730–731
resource throttling for, 724–726
Search Foundation processes, 578
syntax for, 736–737
SearchFolder corruption type, 446
–SearchQuery cmdlet parameter, 731
SearchQuery property, 713
security. See also permissions
administrators and, 319–320
challenges of, 4–5
cmdlets and, 89
installation and, 73–77
mailboxes and, 184
PST files and, 646
self-signed certificates, 78
security groups
about, 284–285
mail users and, 252
security support provider (SSP), 506
seeding databases, 489
Select-Object cmdlet, 98, 282
self-maintained lagging, 530–532
self-signed certificates, 77–78, 314, 317
Send As permission, 229–231, 275
Send-NewUserWelcome.ps1 script, 194
Send On Behalf Of permission, 229–231, 241, 275
SendAs action, 756
Senders property, 713
SendOnBehalf action, 756
Sent Items folder, 393, 658
separate rules functionality, 318
–Server cmdlet parameter, 86, 207
Server Management role group, 75, 135
server message block (SMB), 500
server-side filters, 110–113
server-side rules, 318
server switchovers, 536–539
ServerManager PowerShell module, 52
ServerManagerCmd cmdlet, 52
ServerManagerCmd PowerShell module, 52
servers
edge, 48
maintaining, 562–565
mounted databases, 463
provisioning, 63
recovering failed, 61–63
removing, 548–549
stressed, 565–566
virtual, 24–25
witness, 490–491
Service (Exchange Online), 5–9
service level agreement (SLA), 9
service packs (SPs), 64
Set-AdminAuditLogConfig cmdlet, 748–749
Set-ADServerSettings cmdlet, 59, 122, 364
Set-AuthenticodeSignature cmdlet, 118
Set-CalendarProcessing cmdlet, 202, 204, 260–261, 264
Set-CASMailbox cmdlet, 84
Set-Contact cmdlet, 98, 381
Set-DatabaseAvailabilityGroup cmdlet
adding IP addresses, 493
DAC and, 552
editing properties, 492, 505–506
exerting manual control over networks, 509
lag copy self-maintenance, 531
Set-DatabaseAvailabilityGroupNetwork cmdlet, 509
Set-DistributionGroup cmdlet
- group expansion, 277–278
- mail-enabled groups, 285
- mailbox delegation permissions, 232
- management lists for groups, 276
- marking existing groups, 298
- moderated groups, 245, 248–249
- multilingual custom MailTips, 381
- protected groups, 278–279
- users maintaining groups, 294

Set-DynamicDistributionGroup cmdlet, 246, 301, 306, 313

Set-EmailAddressPolicy cmdlet, 335, 343

Set-EventLogLevel cmdlet, 331

Set-ExchangeServer cmdlet, 71, 411

Set-ExecutionPolicy cmdlet, 118–119

Set-Group cmdlet, 98, 372–373

Set-InboxRule cmdlet, 320

Set-Mailbox cmdlet
- about, 84, 192, 194
- address book policies, 351–352, 355
- applying retention policies, 678
- assigning quota values, 216
- bulk updates, 115
- custom MailTips, 380
- defining custom properties, 256
- EAC and, 175
- email address policies, 333
- limiting user functionality, 91–92
- mailbox delegation, 231–232
- managing archive properties, 651
- multilingual custom MailTips, 381
- OAB generation mailboxes, 367
- putting mailboxes on litigation hold, 699–701
- putting mailboxes on retention hold, 698–699
- RBAC and, 137
- read tracking, 325
- remote PowerShell and, 91
- removing retention policies, 680
- role assignment policies, 164–165
- setting language, 196–197
- shared mailboxes, 241

Set-MailboxAuditByPassAssociation cmdlet, 758

Set-MailboxAutoReplyConfiguration cmdlet, 202–203

Set-MailboxCalendarConfiguration cmdlet, 202–204, 206, 261

Set-MailboxCalendarSettings cmdlet, 261

Set-MailboxDatabase cmdlet
- activation performance numbers, 535
- backup settings, 416
- controlling checksum scans, 433
- handling move request errors, 609
- maintenance settings, 434
- OAB assignments, 370
- provisioning settings, 211
- reconnecting mailboxes, 220–221

Set-MailboxDatabaseCopy cmdlet, 159, 525, 527, 531

Set-MailboxExportRequest cmdlet, 625, 637

Set-MailboxFolderPermission cmdlet, 239

Set-MailboxImportRequest cmdlet, 625

Set-Mailbox JunkE MailConfiguration cmdlet, 202, 204–205

Set-MailboxMessageConfiguration cmdlet, 202, 204

Set-MailboxRegionalConfiguration cmdlet, 199, 202–203, 206

Set-MailboxSearch cmdlet, 730–731

Set-MailboxServer cmdlet
- activation blocks, 542–544
- building DAGs, 561
- database activation restrictions, 565
- maximum number of databases, 402
- OAB updates, 364
- server switchovers, 538–539
- work cycle information, 455

Set-MailboxSpellingConfiguration cmdlet, 202

Set-MailContact cmdlet, 251

Set-MailPublicFolder cmdlet, 381

Set-MailUser cmdlet, 253

Set-ManagementRoleAssignment cmdlet, 158

Set-MigrationBatch cmdlet, 594, 639

Set-MigrationEndpoint cmdlet, 614

Set-MoveRequest cmdlet, 608, 610–611

Set-Notification cmdlet, 639

Set-OfflineAddressBook cmdlet, 366, 370

Set-OrganizationConfig cmdlet
- configuring MailTips, 377–378
- creating site mailboxes, 802
- group creation settings, 288, 292
- hierarchical address books, 372
- locking down public files, 790
- read-status tracking setting, 324

Set-OrganizationConfig command, 790

Set-OutlookAnywhere cmdlet, 79

Set-PublicFolder cmdlet, 784, 786

Set-ReceiveConnector cmdlet, 92

Set-ResourceConfig cmdlet, 255–256

Set-RetentionPolicyTag cmdlet, 671, 681–682, 688

Set-RoleAssignmentPolicy cmdlet, 165

Set-RoleGroup cmdlet, 151, 158

Set-ServerComponentState cmdlet, 563–566

Set-SiteMailbox cmdlet, 803

Set-SiteMailboxProvisioningPolicy cmdlet, 804

Set-ThrottlingPolicy cmdlet, 725

Set-TransportConfig cmdlet, 531, 627

Set-User cmdlet
- about, 98
- address book policies and, 351–352
- hierarchical address books, 373
- remote PowerShell and, 88
- updating mailbox properties, 194
Set-WebServicesVirtualDirectory cmdlet, 613
Set-WorkloadPolicy cmdlet, 580
Setup command
   about, 49–51
   /ActiveDirectorySplitPermissions parameter, 167
   /ADToolsNeeded switch, 52
dele gated setup, 63
   /IAcceptExchangeServerLicenseTerms parameter, 49
   /PrepareAD parameter, 52, 167
   /PrepareSchema parameter, 52
re covering failed servers, 62
   running, 53–56
   split permissions model, 167
   uninstalling Exchange Server, 58
Windows PowerShell and, 84
setup logs, 56–57
shared mailboxes, 188, 240–241
SharePoint
   integration with, 3, 18
   site mailboxes and, 791–792
   SQL support, 389
   synchronization with Exchange and, 795–799
sharing policies, 313
Shift+Delete key combination, 738
Shift+Tab key combination, 97
   –ShowDetails cmdlet parameter, 760
Simple Mail Transfer Protocol (SMTP)
certificate management, 313
e-mail address policies, 343–344
message routing and, 35
selective output example, 100
site mailboxes and, 794
single copy cluster (SCC), 29
single forest design, 45
single-instance storage, 393
dingle-item recovery, 743–745
SinglePageRestore property, 521
site mailboxes
   about, 18, 791–795, 804
   accessing information from Outlook, 799–800
   importing-exporting data in, 624
   life cycle of, 800–803
   Office 365 and, 800
   opening maximum number of, 800
   provisioning policy, 803–804
   sizing, 803
sizing
   databases, 389–392
   mailboxes, 391–394
   public folder mailboxes, 772–773
   site mailboxes, 803
   transaction logs, 420
   –SkipActiveCopyChecks cmdlet parameter, 541
   –SkipClientExperienceChecks cmdlet parameter, 540
   –SkipHealthChecks cmdlet parameter, 541
   –SkipLagChecks cmdlet parameter, 541
   SLA (service level agreement), 9
   SMB (server message block), 500
   SMTP (Simple Mail Transfer Protocol)
      certificate management, 313
e-mail address policies, 343–344
      message routing and, 35
      selective output example, 100
      site mailboxes and, 794
   SoftDelete action, 756, 759
   SoftDeleted state, 581, 604
   Sort-Object cmdlet, 92–93
   Source property, 713
   –SourceMailboxes cmdlet parameter, 731
   SourceMailboxes property, 713
   –SourceRootFolder cmdlet parameter, 636
   split brain syndrome, 551
split permissions model, 166–167
Split-PublicFolderMailbox.ps1 script, 773
SPs (service packs), 64
SQL Server, 388–389
SQL Server, 388–389
SSL (Secure Sockets Layer), 30, 77, 313–314
SSP (security support provider), 506
StalledDueToHA status, 481
Standard edition, 32–33, 407
Standby Active Manager (SAM), 190
Standby Active Manager (SAM) role, 466
standby continuous replication (SCR), 29
Start-MailboxSearch cmdlet, 702, 731
Start-ManagedFolderAssistant cmdlet, 678, 690
Start-MigrationBatch cmdlet, 593
Start-Transcript cmdlet, 114
StartDagServerMaintenance.ps1 script, 563
State or Province property, 305, 346, 351
statistics, database usage, 451–453
   –Status cmdlet parameter, 99, 481, 493, 634
Status property, 520, 713–714
   –StatusDetail cmdlet parameter, 579
   –StatusMailRecipients cmdlet parameter, 731, 762
   StatusRetrievedTime property, 521
Stop-MailboxSearch cmdlet, 702
StopDagServerMaintenance.ps1 script, 563
storage area networks (SAN), 557
storage management
   Active Directory, 35–36
   DAG design, 557
eDiscovery searches, 225
   handling storage failures, 549–551
   setting mailbox quotas, 213–218
   single-instance, 393
   upgrading, 15–22
   virtualized servers, 24–25
Store (Exchange Store). See Exchange Store
   –StoreMailboxIdentity cmdlet parameter, 220, 223
   streaming backups, 29, 389
stressed servers, 565–566
submission changes in messages, 489
Super Help Desk Users USG, 140
Support Diagnostics role, 160
Suspend-ClusterNode cmdlet, 564
–Suspend cmdlet parameter, 616, 618
Suspend-MailboxDatabaseCopy cmdlet, 529, 531, 545
Suspend-MailboxExportRequest cmdlet, 626
Suspend-MailboxImportRequest cmdlet, 625, 635
Suspend-MoveRequest cmdlet, 605
suspending mailbox moves, 615–619
–SuspendWhenReadyToComplete cmdlet parameter, 616–617
SuspendWhenReadyToComplete property, 606
swelling databases, 444–445
switchovers
database copies, 466, 533–534
server, 536–539
targetless, 538
synchronization
incremental, 575, 615
public folder hierarchy, 772
with SharePoint and Exchange Server, 795–799
site mailboxes, 791–792
syntax for searches, 736–737
System Attendant process, 363
System Center Operations Manager (SCOM), 442–443, 470, 520
system registry
CrashCount entry, 438
execution policy, 119
LastCrashTime entry, 438
Patches entry, 69
Products entry, 69
ReplayLagManagerNumAvailableCopies entry, 531
ReplayLagPlayDownPercentDiskFreeSpace entry, 531
RereadLogonQuotasInterval entry, 217, 235
version numbers, 68
system resources, MRS and, 579–580
system tags, 662–663
SystemMailbox account, 76, 224, 363

T
Tab key, 95–97
TAP (Technology Adoption Program), 14, 68
templates
display, 382–386
OAB, 361
temporary log files, 430–431
Test-AssistantHealth cmdlet, 454–455
test plans, 28–29
virtual machines and, 25
Thawte certificate vendor, 314
tickles (signals), 576
time formats, 199
time to live (TTL) settings, 462
time zone property, 261
TimeInCPU metric, 453
TimeInServer metric, 452–453
timestamps, 425
TLS (Transport Layer Security), 313
TMG (Threat Management Gateway), 16
tombstone maintenance, 437
totalMailboxSize property, 597
transaction log replay
about, 479–483
block mode replication, 484–486
incremental resynchronization, 487–488
seeding databases, 489
transaction log compression, 483–484
transaction log truncation, 486–487
transaction logs
about, 388, 408, 419
block mode replication and, 484–486
checksums, 427
circular logging, 429–431
compressing, 483–484
database redundancy and, 411
debugging swelling databases, 445
excessive growth, 443–444
I/O operations, 427–429
log sets, 419–427
moving mailboxes, 578
validating
checksums, 427
query results, 305–306
transaction logs, 482
variables
common and user-defined, 101–103
e-mail address policies, 339
–Verbose cmdlet parameter, 114
verbose PowerShell, 129
VeriSign certificate vendor, 118, 314
versions and version numbers
about, 67–69
DAG servers, 494–495
evolving from earlier, 12–13
groups and, 268
installation and, 63
object, 69–70
selecting for deployment, 22–23
updating, 31–32
Versions folder, 746
View–Only Organization Management role group, 75, 135
View–Only Recipients role, 146
–ViewEntireForest cmdlet parameter, 122, 364
viewing group members, 268–269, 281–283
views (secondary indexes), 393, 395, 437, 578
VIPs (virtual IPs), 558
virtual IPs (VIPs), 558
virtual servers, 24–25
virtualization, 24–25
VMware, 24–25
voice mail, default tag, 661
Volume Shadow Copy Services (VSS)
backup considerations, 458
restoring databases, 390
streaming backups, 29, 389
VSS (Volume Shadow Copy Services)
backup considerations, 458
restoring databases, 390
streaming backups, 29, 389

watermarks, 55–56
Watson dumps, 551
WCF (Windows Communications Foundation), 568
WebDAV API, 16, 30
Where-Object cmdlet, 92–93, 107
white space in databases, 444, 449–450, 597
wildcard characters, 94, 110, 138, 737
Windows CA (Windows Certificate Services), 314–315
Windows Certificate Services (Windows CA), 314–315
Windows Communications Foundation (WCF), 568
Windows Failover Clustering, 463–464, 492, 495–496, 553

Windows PowerShell. See also cmdlets
about, 83–86
Active Directory for, 120–123
automation and, 170
command editing, 95–97
deployment overview, 50–53
development priorities, 4
execution policies, 118–119
hyphens and, 103
influence of cloud service, 6
installing, 44
Integrated Scripting Environment, 103–105
memory limits, 111
profiles, 119–120
quotation marks and, 102–103
remote, 86–92
ServerManager module, 52
ServerManagerCmd module, 52
starting sessions, 53
testing programming and customizations, 30
verbose output, 129
Windows Server Backup, 389
Windows Server Manager, 51
Windows Unified Communications Managed API, 44
Windows User Account Control (UAC), 54
witness servers, 490–491
WitnessDirectory property, 505
WitnessServer property, 505
worker processes
health mailboxes, 226–227
Managed Availability, 402–407, 469–472, 522, 536
WorkerProcessId property, 402, 521
working directory, 117
Workload Management, 579
Write-AdminAuditLog cmdlet, 750
write smoothing, 395
writing scripts, 93

X
X.400 standard, 343–344
X.509 certificates, 26, 313
%xg variable, 339
XML configuration files, 51–52, 205–206
%xs variable, 339
XSO API, 568

Z
zeroing, page, 433, 436