Quest

Metalogix® StoragePoint 6.0.

PowerShell and API Reference Guide

© 2021 Quest Software Inc. ALL RIGHTS RESERVED.

This guide contains proprietary information protected by copyright. The software described in this guide is furnished under a software license or nondisclosure agreement. This software may be used or copied only in accordance with the terms of the applicable agreement. No part of this guide may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of Quest Software Inc.

The information in this document is provided in connection with Quest Software products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Quest Software products. EXCEPT AS SET FORTH IN THE TERMS AND CONDITIONS AS SPECIFIED IN THE LICENSE AGREEMENT FOR THIS PRODUCT, QUEST SOFTWARE ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL QUEST SOFTWARE BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF QUEST SOFTWARE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Quest Software makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Quest Software does not make any commitment to update the information contained in this document.

If you have any questions regarding your potential use of this material, contact:

Quest Software Inc.
Attn: LEGAL Dept.
4 Polaris Way
Aliso Viejo, CA 92656

Refer to our Web site (https://www.quest.com) for regional and international office information.

Patents

Quest Software is proud of our advanced technology. Patents and pending patents may apply to this product. For the most current information about applicable patents for this product, please visit our website at https://www.quest.com/legal.

Trademarks

Quest, the Quest logo, and Metalogix are trademarks and registered trademarks of Quest Software Inc. and its affiliates. For a complete list of Quest marks, visit https://www.quest.com/legal/trademark-information.aspx. All other trademarks and registered trademarks are property of their respective owners.

Legend

CAUTION: A caution icon indicates potential damage to hardware or loss of data if instructions are not followed.

IMPORTANT, NOTE, TIP, MOBILE OR VIDEO: An information icon indicates supporting information.

Metalogix® StoragePoint
Updated October 2021

Contents

PowerShell Guide 6
StoragePoint PowerShell Overview 6
Getting Started 6
Profile and Endpoint Management Cmdlets 7
Add-Endpoint 7
Add-Profile 9
Add-EndpointToProfile 11
Add-BackupEndpointToProfile 14
Set-ProfileEndPointSequence 14
Set-ProfileArchiving 14
Set-ArchiveRulesToProfile 15
Set-RMToProfile 17
Set-HMToProfile 18
Get-EndpointExists 18
Get-Endpoint 19
Get-AllEndpoints 19
Get-Profile 20
Get-AllProfiles 21
Get-AllEndpointsByProfile 22
Get-DecryptUncompressBLOBFile 22
Set-EndpointConnection 23
Set-SystemCacheConnection 23
Remove-ProfileEndpoint 24
Remove-Profile 24
Remove-Endpoint 25
Reset-DefaultValues 26
Timer Job Scheduling Cmdlets 26
Set-BLOBHealthAnalyzerJob 27
Set-UnusedBLOBCleanupJob 28
Set-BLOBExternalizationJob 29
Set-BLOBRecallJob 30
Set-BLOBMigrateJob 30
Set-ArchivingAgingJob 32
Set-ArchivingMetadataJob 32
Set-MigrateRecordsJob 33
Set-MigrateHoldsJob 34
Set-BackupSyncJob 34
Set-ContentMigratorJob 35
Set-EndPointCapacityMonitorJob 36
Set-PerServerMaintenanceJob 37
BLOB Information and Migration Cmdlets 38
Get-AllBLOBs 39

Move-BLOB 39
Invoke-RecallItem 39
Invoke-ExternalizeItem 40
Miscellaneous SharePoint Utility Cmdlets 40
Get-SiteCollectionId 40
Get-ContentDBId 40
Get-WebApplicationId 41
Find-SharePointItem 41
Set-LargeFileUpload 41
PowerShell Script Examples 42
Creating a Site Collection Profile with No Encryption or Compression 42
Creating a Content Database Profile with Encryption and Compression 42
Creating a Content Database Profile Using RBS 42
Creating Multiple Profiles Using the Same Endpoint 43
Creating a Profile with Asynchronous Endpoint Selection 44
Displaying All Endpoints Meeting Certain Criteria 44
Displaying All BLOB Files Associated with a SharePoint Document 44
Enabling App Management service Application and App Catalog 45
Altering MasterKey Encryption Password on Existing Profiles 45
StoragePoint API Reference 48
Using the API Objects in Visual Studio 48
Profile Creation API 49
ProfileAPI Object Reference 49
Profile Creation Code Examples 54
Archive API 56
Archive API Object Reference 56
Archive Rule Code Examples 57
Timer Job API 60
Timer Job API Object Reference 60
Timer Job API Object Reference Code Examples 61
TimerJobStatusAPI 68
Timer Job Status API Object Reference 69
Timer Job Status API Object Reference Code Examples 69
Blob Migration API 72
BlobAPI Object Reference 72
BlobAPI Object Reference Code Examples 73
Blob Reference API 73
BlobReferenceAPI Object Reference 73
BlobReferenceAPI Code Example 75
Upload Large File API 76
UploadLargeFileAPI Object Reference 76
Upload Large File API Code Example 77
Validator API 78
Validator API Examples 78
StoragePoint APIs in Visual Studio 80
About Us 84

Contacting Quest 84
Technical Support Resources 84

1

PowerShell Guide

StoragePoint PowerShell Overview

StoragePoint provides a number of PowerShell cmdlets to automate setup and configuration tasks. While it is also possible to use the StoragePoint API directly in PowerShell, these cmdlets simplify common tasks and provide a basis for more complex API-oriented scripts.

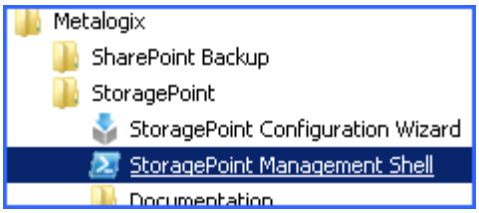
The StoragePoint cmdlets can be grouped into the following categories:

- Profile and endpoint management
- Timer job scheduling
- BLOB information and migration
- Miscellaneous SharePoint utility commands

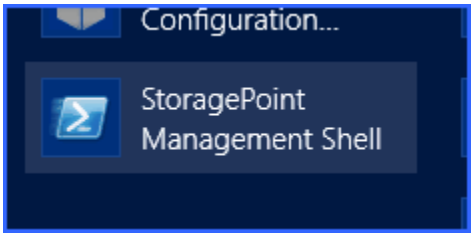
Getting Started

The StoragePoint PowerShell cmdllets require PowerShell version 2.0 or higher. Windows 2008 R2 includes PowerShell 2.0. For other operating systems, PowerShell version 2.0 is available in the Windows Management Framework download (see the Microsoft downloads site).

For Windows Server 2008, go to the start menu>Metalogix>StoragePoint>StoragePoint Management Shell.



For Windows Server 2012 go to Search and type StoragePoint Management Shell.

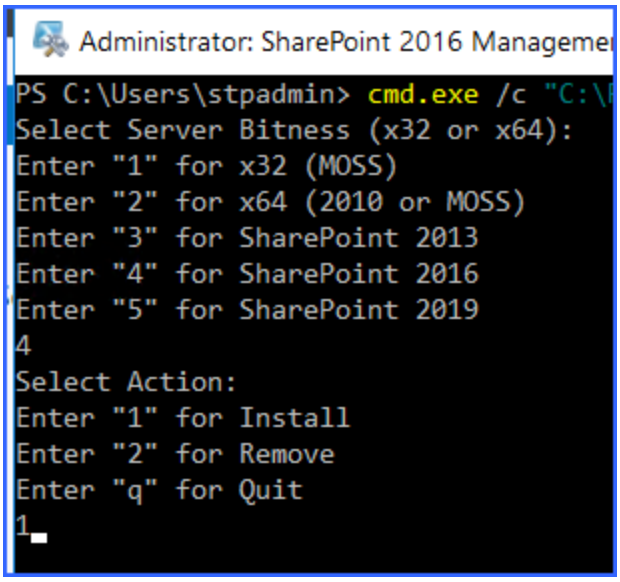


Installing StoragePoint Cmdlets for use with SharePoint PowerShell Console

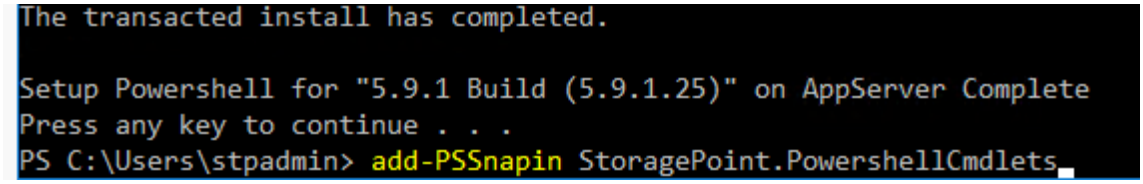
Open the SharePoint Management Shell as Administrator.

Enter cmd.exe /c "<path to PowerShell installer>". For example, cmd.exe /c "C:\Program Files\Metalogix\StoragePoint\AppServer\PowershellInstaller.bat"

Then select which version of SharePoint is being used, and then which action to take.



When that has finished, run add-PSSnapin StoragePoint.PowershellCmdlets



Profile and Endpoint Management Cmdlets

The StoragePoint PowerShell cmdlet library contains a number of cmdlets to simplify the creation and management of profiles and endpoints. In many cases these cmdlets return StoragePoint API objects (i.e. ProfileAPI, EndpointAPI, ProfileEndpointAPI). Please refer to the StoragePoint API Reference for more information on properties available in these objects.

Add-Endpoint

This command will create and save a new storage endpoint in StoragePoint.

Parameters

- EndpointName (epn): The name of the new Endpoint. REQUIRED.

- Connection (conn): The connection string for the Endpoint. REQUIRED.
- AdapterName (adn): The adapter name; e.g. FileSystem, Azure, etc. REQUIRED.
- Foldering: When specified (-Foldering), folders will be created within the endpoint BLOB store. Use -Foldering:$false to disable foldering.
- FolderingLevel: A number associated with the needed level of foldering within the BLOB store:
  - 1 = Year
  - 2 = Year and Month
  - 3 = Year, Month, and Day
  - 4 = Year, Month, Day, and Hour
  - 5 (default) = Year, Month, Day, Hour, and Minute
  - 6 = Year, Month, Day, Hour, Minute, and Second
- UseCompression: When specified (-UseCompression), indicates the endpoint should compress content written to it.
- UseEncryption: When specified (-UseEncryption), indicates the endpoint should encrypt content.
- EncryptionPassphrase: Encyrption passphrase used to generate an encryption key. REQUIRED IF -UseEncryption is specified.
- EncryptionProviderId: The Guid Id associated with the encryption provider that will be used to encryption content. The default is AES 256-bit encryption which is appropriate for most scenarios.
- Status: Endpoint status (Online = 1, Offline = 0).
- EndpointType: Endpoint Type (0=Normal, 1=System Cache, 2=Default Template, 3=Backup).
- IsWORMDevice: True or False; endpoint is ignored for Unused BLOB Cleanup if True.
- ErrorCountThreshold: Number of consecutive write errors before endpoint is automatically taken offline.
- ErrorCountWarningThreshold: Number of consecutive write errors before a warning email is sent.
- FreespaceThreshold: Amount of freespace (in MB) below which endpoint is automatically taken offline.
- FreespaceWarningThreshold: Amount of freespace (in MB) below which a warning email is sent.
- FreespacePercentThreshold: Percentage of freespace below which endpoint is automatically taken offline.
- FreespacePercentWarningThreshold: Percentage of freespace below which a warning email is sent.

- UseDefaultContacts: Send freespace warning emails to default contact list (maintained in General Settings).
- OtherNotify: Other email addresses (separate multiple with ;) to send freespace warnings to.

Examples

The following example will create a new StoragePoint endpoint with only the required parameters applied. All other values will be assigned default values when the endpoint is created. This endpoint will be on the file system and use folder \\server\blobShare\EP2.

```
Add-Endpoint -epn MainEndpoint2 -conn "PATH=\
\Server\blobShare\EP2;AUDITDELETE=False;BUFFERSIZE=8192;SHREDONDELETE=False;" -adn
FileSystem
```

This example creates an endpoint with compression and encryption turned on, and declares it a WORM device.

```
Add-Endpoint -epn MainEndpoint2 -conn "PATH=\
\Server\blobShare\EP2;AUDITDELETE=False;BUFFERSIZE=8192;SHREDONDELETE=False;" -adn
FileSystem -IsWormDevice –UseCompression –UseEncryption –EncryptionPassphrase
"encryptionpwd"
```

Add-Profile

This command will create a new StoragePoint profile. It creates one endpoint but others can be added afterward by the Add-EndpointToProfile command (see below).

Parameters

- ProfileName (pn): The name of the new Profile. REQUIRED.
- ProfileType (ptype): The type of profile to create, e.g. SiteCollection, WebApplication, or ContentDB. REQUIRED.
- ScopeId (sid): The GUID Id value of the scope of the profile. For ProfileType of SiteCollection, this is the SiteId. For a ProfileType of ContentDb, this is the Id of the content database. For a ProfileType of WebApplication, this is the Id of the web application. REQUIRED.
- Externalization (extrn): Whether externalization is enabled or not on this new profile. The default is yes; use 0 for No.
- EndpointId (eid): The Id or Name of the first endpoint to create on the profile. REQUIRED.
- BlobRetentionDays: Number of days to retain BLOB files after they have been removed from SharePoint. Default is 30 days and it cannot be 0.
- UseRBS: If specified (-UseRBS) then RBS will be used if a Content Database profile is being created. By default EBS is used for SharePoint 2010. REQUIRED to use RBS, even on SharePoint 2013. If EBS is preferred, it must be enabled on the SharePoint 2010 farm.

- SelectEndpointAsync: If specified (-SelectEndpointAsync) then the profile will evaluate endpoints asynchronously. This allows the use of file type and scope filters, as well as SharePoint properties promotion. It is also required for enabling Large File Uploads, performing backup and restore functions, Records and Holds Management options, Bulk Migration and Archiving. Make sure a system cache has been configured to use asynchronous operations.
- Externalization: Indicate whether externalization is enabled, disabled, or Archive Only.
- EndpointStartFolder: Specifies a start sub-folder to use on the endpoint to write content from the profile.
- EndpointAsyncPromoteFilename: If specified (-EndpointAsyncPromoteFilename) then the SharePoint filename will be used as the basis for the filename of the externalized BLOB file. (Requires SelectEndpointAsync.)
- EndpointAsyncPromoteFolder: If specified (-EndpointAsyncPromoteFolder) then the SharePoint folder structure will be used as the foldering on the BLOB store for a given externalized BLOB. (Requires -SelectEndpointAsync.)
- EndpointAsyncPromoteExtension: If specified (-EndpointAsyncPromoteExtension) then the extension of the SharePoint filename will be used as the extension on the externalized BLOB file. This is ignored if AsyncPromoteFilename is also true. (Requires -SelectEndpointAsync.)
- EndpointFileSizeOp: Specifies the operator to use if a file size filter is specified. 0 for <= or 1 for >=.
- EndpointFileSize: The file size in KB if a file size filter is specified. Default is -1 (no size filter).
- EndpointFileTypeOp: Specifies the operator to use if a file type filter is specified. 0 for Exclude or 1 for Include.
- EndpointFileTypes: A comma separated list of file extensions to filter on (DOCX, PPTX, TIF, etc.). If the SharePoint filename contains the extension, then it will be included. (Requires -SelectEndpointAsync.)
- EndpointScopeOp: Specifies the operator to use if a Scope filter is used. 0 for Include and 1 for Exclude.
- EndpointScopes: Specifies a comma separated list of scope filter values. See the StoragePoint API Reference for more information on the syntax of scope filters. (Requires -SelectEndpointAsync.)
- BackupEndpoint (be): On new profile creation, the backup endpoint can be designated. It can also be added later with Add-BackupEndpointToProfile.
- BackupRetentionDays (brd): How long files must remain on the backup endpoint.
- LastAccessDateRetention (ladr): Indicate whether this profile should retain Last Access Date for BLOBs. The default is No; use 1 for Yes.
- SetMasterKeyEncryption (setmk): Create the master key encryption for the content database in cases where the profile scope is a content database. It will need to be entered during profile creation by adding the following line to beginning of the script: $SecurePassword = Read-Host -Prompt "Enter Password" -AsSecureString

Examples

This simple example creates a basic profile with an endpoint, for a Site Collection. The endpoint must have been created previously (see the Add-Endpoint command).

```
$scid = Get-SiteCollectionId –s http://s3-sp16-wfe1:33472/sites/testSC2

Add-Profile –pn MainProfile2 –ptype SiteCollection –sid  $scid.ToString() –eid SANEndpoint
```

This next example creates an RBS ContentDb profile with an asynchronous endpoint:

```
$cId = Get-ContentDbId -s http://s3-sp16-wfe1:33472/sites/testSC2

$newprofile = Add-Profile -pn MainProfile2 -ptype ContentDb -sid $cid.tostring() -UseRBS -
Externalization "Yes" –eid NASEndpoint –EndpointAsyncPromoteFilename –
EndpointAsyncPromoteFolder LastAccessDateRetention:$true -BackupEndpoint "BkEp" -
BackupRetentionDays 180
```

This final example creates an EBS ContentDb profile and sets it for asynchronous endpoint selection so a file type filter can be applied to the endpoint.It also sets the BLOB retention period to 180 days and sets the SharePoint filename and folder promotion options. It also creates the master key encryption.

```
 $SecurePassword = Read-Host -Prompt "Enter Password" -AsSecureString

$cId = Get-ContentDbId -s http://s3-sp10-wfe1:33472/sites/testSC2

$newprofile = Add-Profile -pn MainProfile2 -ptype ContentDb -sid $cid.tostring() –
BlobRetentionDays 180  –eid NASEndpoint –SelectEndpointAsync –
EndpointAsyncPromoteFilename –EndpointAsyncPromoteFolder  -EndpointFileTypeOp 1 –
EndpointFileTypes “DOC,DOCX,XLS,XLSX” -setmk $SecurePassword
```

If the scope ID is known:

```
Add-Profile -pn MainProfile2 -ptype ContentDb -sid aad5c5f3-8d0b-43ae-a078-57cc15c34b4f -
Externalization 1 –BlobRetentionDays 180 -UseRBS –eid EP1 -EndpointStartFolder "Wss_Content" –
SelectEndpointAsync –EndpointAsyncPromoteFilename –EndpointAsyncPromoteFolder -be
"Backup EP2" -brd 25 -ladr 1
```

Add-EndpointToProfile

The command allows you to add an endpoint to an existing profile.

Parameters

- Endpoint (ep): The Id or name of the endpoint. REQUIRED
- Profile (p): The Id or name of the profile. REQUIRED
- StartFolder: Specifies a start sub-folder to use on the endpoint to write content from the profile.

- AsyncPromoteFilename: If specified (-AsyncPromoteFilename) then the SharePoint filename will be used as the basis for the filename of the externalized BLOB file. (Requires WriteMode to be set to Asynchronous.)
- AsyncPromoteFolder: If specified (-AsyncPromoteFolder) then the SharePoint folder structure will be used as the foldering on the BLOB store for a given externalized BLOB. (Requires WriteMode to be set to Asynchronous.)
- AsyncPromoteExtension: If specified (-AsyncPromoteExtension) then the extension of the SharePoint filename will be used as the extension on the externalized BLOB file. This is ignored if AsyncPromoteFilename is also true. (Requires WriteMode to be set to Asynchronous.)
- FileSizeOp: Specifies the operator to use if a file size filter is specified. 0 for <= or 1 for >=.
- FileSize: The file size in KB if a file size filter is specified. Default is -1 (no size filter).
- FileTypeOp: Specifies the operator to use if a file type filter is specified. 0 for Exclude or 1 for Include.
- FileTypes: A comma separated list of file extensions to filter on (DOCX, PPTX, TIF, etc.). If the SharePoint filename contains the extension, then it will be included. (Only works on profiles created with the -SelectEndpointAsync option.)
- ScopeOp: Specifies the operator to use if a Scope filter is used. 1 for Include and 0 for Exclude.
- Scopes: Specifies a comma separated list of scope filter values. (Only works on profiles created with the -SelectEndpointAsync option.)

  [Target=Site Collection]
  s:<Site Collection ID>:<Root Web Title>:<Site Collection ID>

  [Target=Web]
  w:<Web ID>:<Web Title>:<Site Collection ID>

  [Target=Web CT]
  ct:<Web CT ID>:Document:<Site Collection ID>

  [Target=List]
  l:<List ID>:<List Title>:<Site Collection ID>:<List ID>

  [Target=List CT]
  lct:<List CT ID>:<List CT Name>:<Site Collection ID>:<List ID>
- SizeFilterType: Specifies the size filter type. '-1' (default value) means no size filter and '1' means file size filter. This parameter is only valid in SharePoint 2013.
- MetadataRules: Specifies a metadata rule for externalization to the endpoint. Metadata rules must be expressed in xml.

> NOTE: For Operator, values (>=,<=,>,<, <>) cannot be entered directly; it will not work with the xml format. Instead, use the following syntax:

< (<), > (>).

- PromoteImpRetention: Specifies whether retention settings created in IMP for externalized content will be promoted to the endpoint.
- If more than one externalization filter is added to the endpoint, the rules can be configured to work in an AND/OR configuration. AND (default) requires that all conditions are met. OR means that the content needs to meet any of the conditions to be externalized to that endpoint. Values are 0 (AND) or 1 (OR).
  - FileTypeFilterOp
  - ScopesFilterOp
  - MetadataRulesFilterOp

Examples

The following example shows how to add an endpoint to a profile. This example adds the endpoint using all of the default values (no filename/folder promotion, no endpoint filters).

```
Add-EndpointToProfile -ep MainEndpoint1 -p MainProfile1
```

The next example looks up the profile using the Get-Profile cmdlet and then adds the endpoint to it using the ProfileId property on the Profile object returned. This example shows the use of Profile objects in PowerShell scripts.

```
$profile = Get-Profile –ProfileType SiteCollection –SiteUrl http://sharepoint

Add-EndpointToProfile –ep MainEndpoint1 –p $profile.ProfileId.ToString()
```

This example shows adding an endpoint with several advanced features including SharePoint filename/folder promotion, a file type filter (to include only Word and Excel documents), and a metadata filter. (Note that the file type filter in this example requires that the profile was created with the SelectEndpointAsync option.)

```
Add-EndpointToProfile -ep MainEndpoint1 -p MainProfile1 –AsyncPromoteFilename –
AsyncPromoteFolder –FileTypeOp 1 –FileTypes “DOC,DOCX,XLS,XLSX” -MetadataRules
"<MetadataRule><PropertyId>8553196d-ec8d-4564-9861-
3dbe931050c8</PropertyId><PropertyName>Name</PropertyName><Operator>contains</Operat
or><Value>TestText</Value></MetadataRule>"
```

Adding an endpoint with more than one metadata rule and IMP retention promotion.

```
 Add-EndpointToProfile -p p1 -Endpoint "XXX1" -MetadataRules
"<MetadataRule><PropertyId>8553196d-ec8d-4564-9861-
3dbe931050c8</PropertyId><PropertyName>Name</PropertyName><Operator>=</Operator><Val
ue>TestName</Value></MetadataRule>,<MetadataRule><PropertyId>fa564e0f-0c70-4ab9-b863-
0177e6ddd247</PropertyId><PropertyName>Title</PropertyName><Operator>&lt;&gt;</Operator
><Value>Title</Value></MetadataRule>,<MetadataRule><PropertyId>28cf69c5-fa48-462a-b5cd-
27b6f9d2bd5f</PropertyId><PropertyName>Modified</PropertyName><Operator>&lt;</Operator
><Value>9/18/2015 12:00:00 AM</Value></MetadataRule>" -PromoteImpRetention:$true
```

Add-BackupEndpointToProfile

The command allows you to add a backup endpoint to an existing profile. This endpoint will be used to configure the BLOB Backup timer job. Backup endpoints are configured using Add-Endpoint, EndpointType=3.

Parameters

- Endpoint (e): The Id or name of the endpoint. REQUIRED
- Profile (p): The Id or name of the profile. REQUIRED

Examples

The following example shows how to add a backup endpoint to a profile. Backup Endpoints don't use filters or other options.

```
Add-BackupEndpointToProfile -e BackupEndpoint1 -p MainProfile1
```

Set-ProfileEndPointSequence

This command allows users to change the sequence of endpoints associated to a profile.

Parameters

- ProfileName (p): The Id or name of the Profile. REQUIRED
- EndpointName (e): The Id or name of the endpoint. REQUIRED
- ProfileEndpointSequence (seq): Endpoint sequence or index number. The endpoint sequence start from 1 until n. REQUIRED

Examples

Move the Endpoint “EP1” to the first position

```
Set-ProfileEndpointSequence –p “ProfileName” –e “EP1” –seq 1
```

Move the Endpoint “EP1” to last position

```
Set-ProfileEndpointSequence –p “ProfileName” –e “EP1” –seq 1000
```

Set-ProfileArchiving

This command sets the Archiving feature to Yes on an existing profile. This is necessary to be able to use the Set-ArchivingRulesToProfile cmdlet, which is necessary for creating archiving conditions.

Parameters

- ProfileName (p): The Id or name of the Profile. REQUIRED
- Archiving (arch): Set to enable archiving. Use false to disable archiving.

Examples

Enable archiving on the profile

```
Set-ProfileArchiving –p “ProfileName”
```

Disable archiving on the profile

```
Set-ProfileArchiving –p “ProfileName” -arch:$false
```

Set-ArchiveRulesToProfile

This command allows archiving rules to be added to an existing profile. It can also be used to edit existing rules. Scopes and rules can be defined according to archiving restrictions.

- If archiving isn't enabled on a profile, use Set-ProfileArchiving before adding rules to the profile.
- After the rules are added, use Set-ArchvingMetadataJob or Set-ArchivingAgingJob to process existing content.

Parameters

- ProfileName (p): The Id or name of the profile. REQUIRED
- RuleScope (rs): The scope of the archive rule. This cannot be broader than the scope of the profile. REQUIRED
  - wa:<Web Application Id>/cdb:<Content DB Id>/s:<site collection Id>/w:<web page id>/l:<list Id:>/c:<content type name string>
  - The profile scope ID along with the rule scope ID must be included. See examples below.
- ArchiveRuleType (art) : The type of archive rule - "age" or "metadata". REQUIRED
- RuleDefinitions (rds): Conditions for the type of archiving rule specified, in an XML format.
  - Age - "<rules><age><DateProperty>Date Property ={created,modified} </DateProperty><Duration>integer value</Duration><Interval>interval value={day,month,year} </Interval><MetadataProperty>Metadata Property Name</MetadataProperty><Operator>Operator Value</Operator><Value>Metadata Value</Value><DestinationEndpoint>Endpoint Name</DestinationEndpoint></age><rules>”
  - Metadata - "<rules><metadata><MetadataProperty>Metadata Property Name</MetadataProperty><Operator>Operator Value={=,<>,Contains,Begins,>=,<=} </Operator><Value>Metadata Value</Value><DestinationEndpoint>Endpoint Name</DestinationEndpoint></metadata></rules>”

NOTE: For Operator, values (>=,<=,>,<, <>) cannot be entered directly; it will not work with the xml format. Instead, use the following syntax:

< (<), > (>).

NOTE:For optional metadata criteria on Age rules, <MetadataProperty>, <Operator> and <Value> need to be set as None if not being used.

Examples

The following examples shows how to add archive rules to a profile. .

Age rule (one condition). This example shows how to create an age rule with a web application scope.

```
Set-ArchiveRulesToProfile –p "SharePoint-30293" -RuleScope "wa:af97b4e9-7ef8-4d38-8df4-ec6082a10ad7" -ArchiveRuleType "age" –RuleDefinitions "<rules><age><DateProperty>Created</DateProperty><Duration>1</Duration><Interval>day</Interval><MetadataProperty>Name</MetadataProperty><Operator>=</Operator><Value>value metadata</Value><DestinationEndpoint>EP1</DestinationEndpoint></age></rules>"
```

Age rule (many conditions). This example shows how to create an age rule with a web application scope.

```
Set-ArchiveRulesToProfile –p "SharePoint-30293" -RuleScope "wa:af97b4e9-7ef8-4d38-8df4-ec6082a10ad7" -ArchiveRuleType "age" –RuleDefinitions "<rules><age><DateProperty>Created</DateProperty><Duration>1</Duration><Interval>day</Interval><MetadataProperty>Name</MetadataProperty><Operator>=</Operator><Value>value metadata</Value><DestinationEndpoint>EP1</DestinationEndpoint></age><age><DateProperty>Select</DateProperty><Duration>4</Duration><Interval>month</Interval><MetadataProperty>Name</MetadataProperty><operator>=</Operator><Value>value metadata</Value><DestinationEndpoint>EP2</DestinationEndpoint></age></rules>"
```

Age rule (without metadata). This example shows how to create an age rule with a web application scope.

```
Set-ArchiveRulesToProfile –p "SharePoint-30293" -RuleScope "wa:af97b4e9-7ef8-4d38-8df4-ec6082a10ad7" -art "age" –RuleDefinitions "<rules><age><DateProperty>Created</DateProperty><Duration>1</Duration><Interval>day</Interval><MetadataProperty>Name</MetadataProperty><Operator>=</Operator><Value>value metadata</Value><DestinationEndpoint>EP1</DestinationEndpoint></age><age><DateProperty>Select</DateProperty><Duration>4</Duration><Interval>month</Interval><MetadataProperty>None</MetadataProperty><Operator>None</Operator><Value>None</Value><DestinationEndpoint>EP2</DestinationEndpoint></age></rules>"
```

Metadata Rule (one condition). This example shows how to a create metadata rule with a web application scope:

```
Set-ArchiveRulesToProfile –p "SharePoint-30293" -RuleScope "wa:af97b4e9-7ef8-4d38-8df4-ec6082a10ad7/cdb:51764237-af67-4f92-a570-a8abac87627b" -art "metadata" –RuleDefinitions
```

"<rules><metadata><MetadataProperty>Name</MetadataProperty><Operator>=</Operator><Valu
e>test metadata</Value><DestinationEndpoint>EP1</DestinationEndpoint></metadata></rules>"

Metadata Rule (one condition). This example shows how to a create metadata rule with a Web Application scope and using operator <>.

Set-ArchiveRulesToProfile –p "SharePoint-30293" -rs "wa:af97b4e9-7ef8-4d38-8df4-ec6082a10ad7" -art "metadata" –RuleDefinitions "<rules><metadata><MetadataProperty>Name</MetadataProperty><Operator><></Operator><Value>test metadata</Value><DestinationEndpoint>EP1</DestinationEndpoint></metadata></rules>"

Metadata Rule (many conditions). This example shows how to a create metadata rule with a Web Application scope and Content DataBase:

Set-ArchiveRulesToProfile –p "SharePoint-30293" -rs "wa:af97b4e9-7ef8-4d38-8df4-ec6082a10ad7/cdb:51764237-af67-4f92-a570-a8abac87627b" -art "metadata" –RuleDefinitions "<rules><metadata><MetadataProperty>Name</MetadataProperty><Operator>=</Operator><Value>test metadata</Value><DestinationEndpoint>EP1</DestinationEndpoint></metadata><metadata><MetadataProperty>Name</MetadataProperty><Operator>=</Operator><Value>test metadata</Value><DestinationEndpoint>EP1</DestinationEndpoint></metadata></rules>"

Set-RMToProfile

This command configures Record Management rules for migrating BLOBs based on declaring or undeclaring a record. After these are configured, existing records can be migrated using Set-MigrateRecordsJob.

Parameters

- ProfileName (p): The Id or name of the profile. REQUIRED
- Enabled : True or false, to enable or disable records management on the profile.
- Scope (s) : The scope of the records management rule. This cannot be broader than the scope of the profile.
    - wa:<Web Application Id>/cdb:<Content DB Id>/s:<site collection Id>
- DeclaredAsRecord : True or False, whether to migrate a BLOB when the item is declared a record.
- DeclaredAsRecordEndpoint (de) : Name of the endpoint where BLOBs should be stored when declared a record.
- UndeclaredAsRecord : True or False, whether to migrate the BLOB when the item is no longer declared a record.
- UndeclaredAsRecordEndpoint : Name of the endpoint where BLOBs should be stored when no longer declared a record.

Examples

The following example shows how to enable and add Records Management rules to a profile.

```
Set-RMToProfile -p "Main Profile" -Enabled:$true -Scope "cdb:c54827c2-ff44-4569-86c0-d15e348ac71c/s:2dac99fe-b477-451a-bb1e-8f5279694a7d" -DeclaredAsRecord:$true -DeclaredAsRecordEndpoint "EP2" -UndeclaredAsRecord:$true -UndeclaredAsRecordEndpoint "EP1"
```

Set-HMToProfile

This command configures Hold Management rules for migrating BLOBs based on placing a file in a Hold. After these are configured, existing content can be migrated using Set-MigrateHoldsJob.

Parameters

- ProfileName (p): The Id or name of the profile. REQUIRED
- Enabled : True or false, to enable or disable holds management on the profile.
- Scope : The scope of the holds management rule. This cannot be broader than the scope of the profile. REQUIRED
    - wa:<Web Application Id>/cdb:<Content DB Id>/s:<site collection Id>
- OnHold : True or False, whether to migrate a BLOB when the item is put in a hold.
- OnHoldEndpoint : Name of the endpoint where BLOBs should be stored when placed in a hold.
- RemovedFromHold : True or False, whether to migrate the BLOB when the item is no longer in a hold.
- RemovedFromHoldEndpoint : Name of the endpoint where BLOBs should be stored when no longer in hold.

Examples

The following examples show how to enable and add Holds Management rules to a profile.

```
Set-HMToProfile -p "Main Profile" -Enabled:$true -Scope "cdb:c54827c2-ff44-4569-86c0-d15e348ac71c/s:2dac99fe-b477-451a-bb1e-8f5279694a7d" -OnHold:$true -OnHoldEndpoint "EP2" -RemovedFromHold:$true -RemovedFromHoldEndpoint "EP1"
```

Get-EndpointExists

Determines whether the specified endpoint exists.

Parameters

- EndpointId (ep): The endpoint name or Id. REQUIRED.

Example

Get-EndpointExists -ep MainEndpoint1

Result

True if endpoint exists. False if not.

Get-Endpoint

This command will retrieve an endpoint object. An EndpointAPI object is returned. See the StoragePoint API Reference for more information on properties of this object.

Parameters

- Endpoint (ep): The name or Id of the endpoint to retrieve. REQUIRED.

Examples

This example will retrieve the endpoint object and list all properties of the object for viewing.

```
Get-Endpoint -ep MainEndpoint1
```

This example shows how to assign an Endpoint object to a variable within the PowerShell pipeline and then use this object later on in the script.

```
$endpoint = Get-Endpoint -ep MainEndpoint1

Add-EndpointToProfile –ep $endpoint.EndpointId.ToString() –p MainProfile1
```

Get-AllEndpoints

This command will retrieve all StoragePoint endpoints into a list and display all the properties of each endpoint found.

Example

Get-AllEndpoints

Result

```
EndpointId          : <GUID>
Name                : MainEndpoint1
AdapterName         : FileSystem
Connection          :PATH=\\Server\blobShare;BUFFERSIZE=8192;SHREDONDELETE=False;
Foldering                   : True
```

FolderingLevel : 5

StoreSourceMetadata :

HandleMissingMetadata :

IsNew : False

UseCompression : False

UseEncryption : False

EncryptionProviderId :

EncryptionPassphrase :

ProfileState : 1

Get-Profile

This command will retrieve a profile object.

Parameters

If looking up a profile for a given scope (site coll, content db, web app):

- ProfileType (ptype): The profile type, e.g. SiteCollection, WebApplication, ContentDB to find. *Must also specify either SiteUrl or ScopeId*
- SiteUrl (s): The URL of the SharePoint site in which a profile is associated with. If ProfileType is SiteCollection, this will cause the command to find the profile for the exact site collection. If ProfileType is ContentDb or WebApplication, then the command will find the profile covering the content database or web application, respectively, of the site collection url specified.
- ScopeId (sid): The scope id of the siteCollection, WebApplication, or ContentDB to find the profile for.

If looking up a profile for an externalized document:

- DocUrl (doc): Url of an externalized document in SharePoint to retrieve the profile for.

If looking up a specific profile by name or profile id:

- ProfileId (id): The Id (GUID) of the StoragePoint profile.
- ProfileName (name): The Name of the StoragePoint profile.

Example

This example retrieves the profile object and list all properties of the object for view.

```
Get-Profile -name MainProfile1
```

The following example shows to assign a Profile object to a variable within the PowerShell pipeline for later use.

```
$profile = Get-Profile -name MainProfile1
```

The following example retrieves the profile object that a document was externalized under.

```
Get-Profile -DocUrl http://moss/docs/documents/123.tif
```

The following example looks up a site collection's profile.

```
Get-Profile –ProfileType SiteCollection –SiteUrl http://sharepoint
```

The final example shows how to retrieve a profile using a scope id value (site id, content db id or web app id). The scope id can be determined in a number of ways but this example uses the StoragePoint cmdlet helper command Get-ContentDbId.

```
$cid = Get-ContentDbId –s http://sharepoint

Get-Profile –ProfileType ContentDb –ScopeId $cid.Id.ToString()
```

Get-AllProfiles

This command will retrieve all active StoragePoint profiles into a list and display all properties for the profiles.

Example

```
Get-AllProfiles
```

Result

```
Type                : SiteCollection
ProfileId           : <GUID>
IsNew               : False
IsActive            : True
ScopeId             : <GUID>
BlobRetentionDays   : 30
CabinetOn           : False
CabinetLevel        : 5
ProfileState        : 1
ProfileEndpoints    : {MainEndpoint1, MainEndpoint2}
Name                : MainProfile1
```

AdapterName : FileSystem

Get-AllEndpointsByProfile

This command will show all endpoints that are associated with the specified profile.

Parameters

- Profile (p): The name or Id of the profile. REQUIRED.

Example

The following example will list all endpoints that are associated with the profile 'MainProfile1'

```
Get-AllEndpointsByProfile -p MainProfile1
```

Result

EndpointType : Synchronous

Name : MainEndpoint1

EndpointId : <GUID>

IsActive : True

Get-DecryptUncompressBLOBFile

The command allows you to decrypt and/or uncompress a BLOB file that is no longer referenced in SharePoint.

Parameters

- InputFile: The name of the encrypted and/or compressed BLOB file.
- OutputFile: The filename and path to write the decrypted and/or uncompressed BLOB file to. *Should be different than InputFile parameter.*
- Uncompress: Switch parameter that indicates if input file is compressed and should be uncompressed.
- Decrypt: Switch parameter that indicates if input file is encrypted and should be decrypted.
- Passphrase: If decryption is specified (-Decrypt parameter), this parameter specifies the passphrase that was used as the basis for the encryption key. This is *required* if decryption is desired.
- KeySize: Strength of AES encryption used to encrypt. 256 is the default. 128 is the other option.

Examples

The following example shows decrypting a BLOB:

```
Add-PSSnapin StoragePoint.PowershellCmdlets –erroraction SilentlyContinue

Get-DecryptUncompressBLOBFile -InputFile "\\fs\sh\Blob1.docx.blob" -OutputFile "c:\blob1.docx" -Decrypt -Passphrase "myencryptionpassword"
```

The following example shows just uncompressing a BLOB:

```
Add-PSSnapin StoragePoint.PowershellCmdlets –erroraction SilentlyContinue

Get-DecryptUncompressBLOBFile -InputFile "\\fs\sh\Blob2.docx.blob" -OutputFile "c:\blob2.docx" -Uncompress
```

The following example shows decrypting and uncompressing a BLOB:

```
Add-PSSnapin StoragePoint.PowershellCmdlets –erroraction SilentlyContinue

Get-DecryptUncompressBLOBFile -InputFile "\\fs\sh\Blob3.docx.blob" -OutputFile "c:\blob3.docx" -Decrypt -Passphrase "myencryptionpassword" -Uncompress
```

Set-EndpointConnection

This command will update the storage endpoint connection string in StoragePoint.

Parameters

- EndpointName (e): The name of the new Endpoint. REQUIRED.
- Connection (conn): The connection string for the Endpoint. REQUIRED.

Examples

The following example will update the connection string of an existing StoragePoint endpoint with a file system adapter.

```
Set-EndpointConnection –e “EP1” –conn “PATH=\\SERVERNAME\Endpoints\EP10”
```

Set-SystemCacheConnection

This commandlet updates the System Cache connection string for the adapter.

Parameters

- Connection (conn): Connection string for the adapter. REQUIRED.

Example

The following example will update the system cache endpoint connection string, using the File System Adapter.

```
Set-SystemCacheConnection –conn “PATH=\
\SERVERNAME\EndpointName;AUDITDELETE=False;BUFFERSIZE=8192;SHREDONDELETE=False;USE
META=False;”
```

Remove-ProfileEndpoint

The command allows you to remove an endpoint from an existing profile based on index position or endpoint name.

Parameters

- Profile (p): The Id or name of the profile. REQUIRED
- Endpoint (e): The Id or name of the endpoint.
- Sequence (seq): Endpoint position or index.

Example

The following example shows how to remove an endpoint from a profile using the index position.

```
Remove-ProfileEndpoint -p "MainProfile1" -seq 2
```

- If both fields are used, and there is a conflict, i.e. 'EP2' is not in position 2, sequence will be used.

Remove-Profile

This command allows the removal of a profile from StoragePoint. If a broader profile exists, the profile can be retired. If a profile is removed, and not retired, the bulk recall job will be run automatically.

Parameters

- Profile (p): The Id or name of the profile. REQUIRED
- Retire: Flag to Retire the profile. Include this parameter to Retire the Profile.
- RetireBackupedUpBLOBs: Flag to indicate that the Backup Sync job will run. If Retire option is set, this option can be used.
- BackupEndpoint (be): Backup endpoint name to include in the Retire Profile. If RetireBackupedUpBLOBs option is set, this option it is REQUIRED.
- DeleteContentDatabaseBackups: Delete any backups of the content database(s) associated with the profile.

- NumberOfThreads (threads): Number of threads to run the job with.
- EmailDefault: If specified (-EmailDefault), send status email to the default notification group setup in General Settings.
- NotificationEmailOther (email): Other email addresses (not in default notification group) to send status emails to.
- JobServer (jobsvr): Name of server machine on which to run the job.

Examples

Deletion when the profile does not have successor profile.

```
Remove-Profile –p ”CDB-20390”
```

Deletion when the profile has successor profile, but without retiring the backed up BLOBs.

```
Remove-Profile –p ” CDB-20390” –Retire
```

Deletion when the profile has successor profile and includes retiring the backed up BLOBs.

```
Remove-Profile –p “CDB-20390” –Retire –RetireBackupedUpBLOBs –BackupEndpoint “Backup1”
```

Deletion with optional arguments.

```
Remove-Profile –p “CDB-20390” –Retire –RetireBackupedUpBLOBs –BackupEndpoint “Backup1” -
DeleteConentDatabaseBackups –EmailDefault –threads 5 –NotificationEmailOther
“test.example@dominio.com” –jobsvr “SERVER”
```

Remove-Endpoint

This command allows the removal of an endpoint from StoragePoint. If a successor endpoint can be selected, the endpoint can be retired.

Parameters

- EndpointName (e): The Id or name of the endpoint. REQUIRED
- Retire: Flag to indicate that the endpoint will be Retired.
- SuccessorEndpoint (se): Name or ID of the successor endpoint to which the blobs will be migrated. If Retire option is set, this option it is REQUIRED.
- NumberOfThreads (threads): Number of threads to run the job.
- EmailDefault: If specified (-EmailDefault), send status email to the default notification group setup in General Settings.
- NotificationEmailOther (email): Other email addresses (not in default notification group) to send status emails to.

Examples

Delete an endpoint that is not associated to any profile

Remove-Endpoint –e “EP1”

Delete an endpoint with Retire option

Remove-Endpoint –e “EP1” –Retire –se “EP2”

Delete an endpoint with Retire option and additional parameters

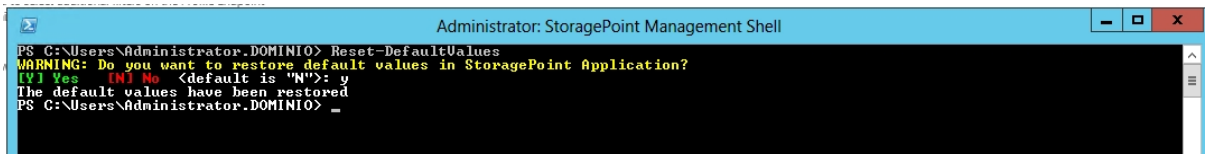
Remove-Endpoint –e “EP1” –Retire –se “EP2” –threads 5 –EmailDefault –email “test.example@dominio.com”

Reset-DefaultValues

This command will reset the default settings to the initial state of the StoragePoint configuration.

Example

Reset-DefaultValues



Timer Job Scheduling Cmdlets

The StoragePoint cmdlet library includes a number of commands to schedule the various StoragePoint timer jobs. The commands cover the BLOB Health Analyzer, Unused BLOB Cleanup, Externalization, Recall, (Bulk) Migrate, Backup, Records & Holds, Aging, Versioning and Metadata archiving jobs.

Scope Parameter Definition

When selecting scopes for select timer jobs, the following format needs to be used for -JobScope. Not all parameters are available for all timer jobs.

[Target=WebApp]
```
-JobScope 'wa:<WebApp ID>'
```

[Target=CDB]
```
-JobScope 'wa:<WebApp ID>/cdb:<CDB ID>'
```

[Target=Site Collection]
```
-JobScope 'wa:<WebApp ID>/cdb:<CDB ID>/s:<Site Collection ID>'
```

[Target=Site (Web)]
```
-JobScope 'wa:<WebApp ID>/cdb:<CDB ID>/s:<Site Collection
ID>/w:<site (Web) ID>'
```

[Target=List]
```
-JobScope 'wa:<WebApp ID>/cdb:<CDB ID>/s:<Site Collection
ID>/w:<site (Web) ID>/l:<List ID>'
```

Set-BLOBHealthAnalyzerJob

This command schedules a BLOB Health Analyzer timer job. This timer job will analyze and optionally try to restore profile content. Analyzer will fix from the following:

- Missing BLOB files.
- Restore BLOBs if RepoulateIfMissing is used.
- BLOB references with mismatched size.
- BLNK (link) files with invalid content.
- BLNK files optimized and updated.
- Various system cache metrics.

Parameters

- Profile (p): The name or Id of the profile in which to run the job under. REQUIRED
- Type (schedtype): The type of schedule to use when setting up this job. Valid values are OneTime, Daily, and Weekly. If not specified, the timer job will run immediately.
- DayOfWeek (dow): The day of the week to run the job when the schedule type is set to Weekly.
- JobStartDate (startdate): The start date/time. The time portion is used to set the start time for Daily and Weekly jobs.
- JobEndDate (enddate): The end date/time. Not used for OneTime jobs. The time portion is used to set the end time for Daily and Weekly jobs.
- EmailDefault: If specified (-EmailDefault), send status email to the default notification group setup in General Settings.
- NotificationEmailOther (email): Other email addresses (not in default notification group) to send status emails to.
- RunNow: If specified (-RunNow), then run the job immediately. Ignores any scheduling options provided.
- EmailOnErrorOnly: If specified (-EmailOnErrorOnly), then send status email only if an error occurs when running the job.
- RepopulateIfMissing: Repopulates missing blobs, if available. Default is false. Set value to 1 to enable this feature, 0 to disable.
- JobScope: See parameter definitions in Timer Job Scope Parameter.

Example

This example schedules BLOB Health Analyzer job to run immediately.

```
Set-BlobHealthAnalyzerJob –p MainProfile1 –EmailDefault –RunNow
```

This example schedules BLOB Health Analyzer job to run weekly on Wednesday between 2 and 5 am..

```
Set-BlobHealthAnalyzerJob –p MainProfile1 -schedtype Weekly -dow Wednesday -startdate "08-15-15 2:00" -enddate "08-15-15 5:00" –EmailDefault
```

Set-UnusedBLOBCleanupJob

This command schedules an Unused BLOB clean-up job for a specified profile.

Parameters

- Profile (p): The name or Id of the profile in which to run the job under. REQUIRED.
- Type (schedtype): The type of schedule to use when setting up this job. Valid values are OneTime, Daily, and Weekly. If not specified, the timer job will run immediately.
- DayOfWeek (dow): The day of the week to run the job when the schedule type is set to Weekly.
- JobStartDate (startdate): The start date/time. The time portion is used to set the start time for Daily and Weekly jobs.
- JobEndDate (enddate): The end date/time. Not used for OneTime jobs. The time portion is used to set the end time for Daily and Weekly jobs.
- EndpointFilter (ef): Specify specific endpoints which the timer job should analyze.
- RunNow: If specified (-RunNow), then run the job immediately. Ignores any scheduling options provided.
- EmailDefault: If specified (-EmailDefault), send status email to the default notification group setup in General Settings.
- NotificationEmailOther (email): Other email addresses (not in default notification group) to send status emails to.
- EmailOnErrorOnly: If specified (-EmailOnErrorOnly), then send status email only if an error occurs when running the job.
- IncludeLargeFiles (ilf): If indicated, the Large File Uploads will also be removed by the unused BLOB cleanup job.
- OnlyLargeFiles (olf): Only large files will be analyzed and removed by the unused BLOB cleanup job.
- JobServer (jobsrv): Server name to run the job. REQUIRED for scheduled jobs.
- JobScope: See parameter definitions in Timer Job Scope Parameter.

Examples

The following example will run the Unused BLOB Cleanup Job immediately.

```
Set-UnusedBLOBCleanupJob -p MainProfile1 -RunNow
```

The following example schedules the job on a recurring weekly schedule, and includes large files.

```
Set-UnusedBLOBCleanupJob -p MainProfile1 -schedtype Weekly -dow Saturday -startdate
“12/26/2009 3:00” –enddate “12/26/2009 6:00” –EmailDefault -email someone@example.com -
IncludeLargeFiles
```

The following example runs the job immediately and has LIST scope.

```
Set-UnusedBLOBCleanupJob -p MainProfile1 -RunNow -JobScope 'wa:61993f69-a1fa-47ae-a037-
f19bb65e1341/cdb:8e3b4665-672c-4655-a41b-861c8f4e86b5/s:054de887-db61-4407-8322-
564531dbefc4/w:74b8d00f-f6f5-4233-b372-05a0f267533a/l:c4dfbdba-21c6-44d1-8e5e-
41be2b11079f'
```

Set-BLOBExternalizationJob

This command schedules externalization job for a specified profile. The job may be run immediately (-RunNow) or at a specified date/time (-JobStartDate).

Parameters

- Profile (p): The name or Id of the profile in which to run the job under. REQUIRED.
- JobStartDate (startdate): The start date/time to run the job.
- NumberOfThreads (threads): Number of threads to run the job with.
- RunNow: If specified (-RunNow), then run the job immediately. Ignores any scheduling options provided.
- JobRole (JobServerRole): Values allowed: Controller, Standalone
- Workers (ws): Servers that will act in the worker role of a controller-worker configuration.
- EmailDefault: If specified (-EmailDefault), send status email to the default notification group setup in General Settings.
- NotificationEmailOther (email): Other email addresses (not in default notification group) to send status emails to.
- EmailOnErrorOnly: If specified (-EmailOnErrorOnly), then send status email only if an error occurs when running the job.

Example

This example schedules an externalization job to run immediately.

```
Set-BLOBExternalizationJob –p MainProfile1 –threads 20 –EmailDefault –RunNow
```

Set-BLOBRecallJob

This command allows a user to schedule a recall job for a particular profile scope.

Parameters

- Profile (p): The name or Id of the profile in which to run the job under. REQUIRED.
- JobStartDate (startdate): The start date/time to run the job.
- NumberOfThreads (threads): Number of threads to run the job with.
- RunNow: If specified (-RunNow), then run the job immediately. Ignores any scheduling options provided.
- JobRole (JobServerRole): Values allowed: Controller, Standalone
- Workers (ws): Servers that will act in the worker role of a controller-worker configuration.
- EmailDefault: If specified (-EmailDefault), send status email to the default notification group setup in General Settings.
- NotificationEmailOther (email): Other email addresses (not in default notification group) to send status emails to.
- EmailOnErrorOnly: If specified (-EmailOnErrorOnly), then send status email only if an error occurs when running the job.
- EndpointFilter (ef): Filter on which BLOBs to recall by endpoint (*=All Endpoints or EP1,EP2).
- SizeFilter (sf): Filter on which BLOBs to recall by size (>= or <=). Use 'Default' to clear this filter.
- JobScope: See parameter definitions in Timer Job Scope Parameter.

Example

This example schedules a recall job to run immediately.

```
Set-BLOBRecallJob –p MainProfile1 –threads 20 –EmailDefault –RunNow
```

This example schedules a recall job and has filters applied.

```
Set-BLOBRecallJob –p MainProfile1 –threads 20 –EmailDefault -ef EP1 -sf ">=50"
```

Set-BLOBMigrateJob

This command allows a user to schedule a migrate job for a specified profile. When run, the job will migrate BLOBs from the source endpoint to the target endpoint.

Parameters

- Profile (p): The name or Id of the profile in which to run the job under. REQUIRED.
- SourceEndpoint: The source endpoint's name/Id. REQUIRED.
- TargetEndpoint: The destination endpoint's name/Id. REQUIRED.
- JobStartDate (startdate): The start date/time to run the job.
- NumberOfThreads (threads): Number of threads to run the job with.
- RunNow: If specified (-RunNow), runs the job immediately.
- JobRole (JobServerRole): Values allowed: Controller, Standalone
- Workers (ws): Servers that will act in the worker role of a controller-worker configuration.
- EmailDefault: If specified (-EmailDefault), send status email to the default notification group setup in General Settings.
- NotificationEmailOther (email): Other email addresses (not in default notification group) to send status emails to.
- EmailOnErrorOnly: If specified (-EmailOnErrorOnly), then send status email only if an error occurs when running the job.
- FilterScopes: add the scope to migrate BLOB job. This is a site collection, content database or web application ID. Get-siteCollectionId or Get-ContentDBId can be used in conjunction.
- FilterScopesOp : Include or Exclude Option.
- JobScope: See parameter definitions in Timer Job Scope Parameter.

Example

This example schedules a migration job to run immediately.

```
Set-BLOBMigrateJob –p MainProfile1 -SourceEndpoint Endpoint1 –TargetEndpoint Endpoint2 –threads 20 –EmailDefault –RunNow
```

This example schedules a migration job to run at a specified time.

```
Set-BLOBMigrateJob –p MainProfile1 -SourceEndpoint Endpoint1 –TargetEndpoint Endpoint2 -startdate "08/20/2015 10:10" –threads 20 –EmailDefault
```

This example schedules a migration job to run at a specified time with filters enabled.

```
Set-BLOBMigrateJob –p MainProfile1 -SourceEndpoint EP1 –TargetEndpoint EP2 -startdate "09-09-2015 11:55"  -FilterScopes "s:42154fb7-176d-4ff6-b16f-3cb3554a4118" -FilterScopesOp include
```

Set-ArchivingAgingJob

This command schedules an Aging job for a specified profile. Archive rules must exist on the profile for this job to run. Use Set-ArchiveRulesToProfile to create the rules.

Parameters

- Profile (p): The name or Id of the profile in which to run the job under. REQUIRED.
- ScheduleType (schedtype): The type of schedule to use when setting up this job. Valid values are OneTime, Daily, and Weekly. If not specified, the timer job will run immediately.
- DayOfWeek (dow): The day of the week to run the job when the schedule type is set to Weekly.
- JobStartDate (startdate): The start date/time. The time portion is used to set the start time for Daily and Weekly jobs.
- JobEndDate (enddate): The end date/time. Not used for OneTime jobs. The time portion is used to set the end time for Daily and Weekly jobs.
- JobServer (jobsrv): Server on which the process will be run.
- NumberOfThreads (threads): Number of threads to run the job with.
- RunNow: If specified (-RunNow), then run the job immediately. Ignores any scheduling options provided.
- EmailDefault: If specified (-EmailDefault), send status email to the default notification group setup in General Settings.
- NotificationEmailOther (email): Other email addresses (not in default notification group) to send status emails to.
- EmailOnErrorOnly: If specified (-EmailOnErrorOnly), then send status email only if an error occurs when running the job.

Example

The following example schedules the job on a recurring weekly schedule.

```
Set-ArchivingAgingJob -p MainProfile1 -schedtype Weekly -dow Saturday -startdate “12/26/2009 3:00” –enddate “12/26/2009 6:00” -jobsrv SP2010WFE1 –EmailDefault -email someone@example.com -threads 2
```

Set-ArchivingMetadataJob

This command schedules an archiving metadata processing job for a specified profile. Archive rules must exist on the profile for this job to run. Use Set-ArchiveRulesToProfile to create the rules.

Parameters

- Profile (p): The name or Id of the profile in which to run the job under. REQUIRED.
- JobStartDate (startdate): The start date/time.
- NumberOfThreads (threads): Number of threads to run the job with.
- RunNow: If specified (-RunNow), then run the job immediately. Ignores any scheduling options provided.
- EmailDefault: If specified (-EmailDefault), send status email to the default notification group setup in General Settings.
- NotificationEmailOther (email): Other email addresses (not in default notification group) to send status emails to.
- EmailOnErrorOnly: If specified (-EmailOnErrorOnly), then send status email only if an error occurs when running the job.

Example

This example schedules an archiving metadata processing job to run immediately.

```
Set-ArchivingMetadataJob –p MainProfile1 –threads 20 –EmailDefault –RunNow
```

Set-MigrateRecordsJob

This command schedules a Records processing job for a specified profile.

Parameters

- Profile (p): The name or Id of the profile in which to run the job under. REQUIRED.
- JobStartDate (startdate): The start date/time.
- NumberOfThreads (threads): Number of threads to run the job with.
- RunNow: If specified (-RunNow), then run the job immediately. Ignores any scheduling options provided.
- EmailDefault: If specified (-EmailDefault), send status email to the default notification group setup in General Settings.
- NotificationEmailOther (email): Other email addresses (not in default notification group) to send status emails to.
- EmailOnErrorOnly: If specified (-EmailOnErrorOnly), then send status email only if an error occurs when running the job.

Example

This example schedules an archiving metadata processing job to run immediately.

```
Set-MigrateRecordsJob –p MainProfile1 –threads 20 –EmailDefault –RunNow
```

Set-MigrateHoldsJob

This command schedules a Holds processing job for a specified profile.

Parameters

- Profile (p): The name or Id of the profile in which to run the job under. REQUIRED.
- JobStartDate (startdate): The start date/time.
- NumberOfThreads (threads): Number of threads to run the job with.
- RunNow: If specified (-RunNow), then run the job immediately. Ignores any scheduling options provided.
- EmailDefault: If specified (-EmailDefault), send status email to the default notification group setup in General Settings.
- NotificationEmailOther (email): Other email addresses (not in default notification group) to send status emails to.
- EmailOnErrorOnly: If specified (-EmailOnErrorOnly), then send status email only if an error occurs when running the job.

Example

This example schedules an archiving metadata processing job to run immediately.

```
Set-MigrateHoldsJob –p MainProfile1 –threads 20 –EmailDefault –RunNow
```

Set-BackupSyncJob

This command schedules a backup Sync Job for a specified profile. This job backs up the endpoint(s) of the profile.

Parameters

- Profile (p): The name or Id of the profile in which to run the job under. REQUIRED
- SchedType : The type of schedule to use when setting up this job. Valid values are OneTime, Minute, Daily, and Weekly. If not specified, the timer job will run immediately.
- DayOfWeek (dow): The day of the week to run the job when the schedule type is set to Weekly.
- Minutes: A number between 1 and 59 that represents how often the timer job should be run.

- JobStartDate (startdate): The start date/time. The time portion is used to set the start time for Daily and Weekly jobs.
- JobEndDate (enddate): The end date/time. Not used for OneTime jobs. The time portion is used to set the end time for Minute, Daily and Weekly jobs.
- JobRole (JobServerRole): Values allowed: Controller, Standalone
- Workers (ws): Servers that will act in the worker role of a controller-worker configuration.
- EmailDefault: If specified (-EmailDefault), send status email to the default notification group setup in General Settings.
- NotificationEmailOther (email): Other email addresses (not in default notification group) to send status emails to.
- NumberOfThreads (threads): Number of threads to run the job with.
- RunNow: If specified (-RunNow), then run the job immediately. Ignores any scheduling options provided.
- EmailOnErrorOnly: If specified (-EmailOnErrorOnly), then send status email only if an error occurs when running the job.

Example

This example schedules an backup sync Job to run immediately.

```
Set-BackupSyncJob –p MainProfile1 –threads 20 –EmailDefault –RunNow
```

This examples sets the job up to run every 30 minutes

```
Set-BackupSyncJob –p MainProfile1 -schedtype minute -minutes 30
```

Set-ContentMigratorJob

This command allows the Content Migrator Job to run immediately or be scheduled. When run, it will move blobs to the endpoint as expected. The Content Migrator job is scheduled to run every 5 minutes, by default. This can be used to update various parameters.

Parameters

- RunEveryMinutes (runevery): A number between 1 and 59 that represents how often the timer job should be run.
- ScheduledTime : Set scheduled timeframe in format time. For example: “15:00-22:00”.
- JobServer (jobsrv): Server name to run the job.
- NumberOfThreads (threads): Number of threads to run the job.
- EmailDefault: If specified (-EmailDefault), status email will be sent to the default notification group setup in General Settings.

- NotificationEmailOther: Other email addresses (not in default notification group) to receive status emails. (optional)
- RunNow: If specified (-RunNow), then the job runs immediately. This setting ignores any scheduling options provided. This may be necessary to override current scheduled jobs.
- JobRole: Values allowed: Controller, Standalone
- Workers (ws): Servers that will act in the worker role of a controller-worker configuration.
- IncludeLargeFiles (ilf): Set the flag to include the large files options.

Example

Example to run the job immediately.

```
Set-ContentMigratorJob –RunNow
```

Set job Properties as RunEveryMinutes, ScheduleTime

```
Set-ContentMigratorJob –runevery 45 –scheduledtime “15:00-20:00” -ilf yes
```

Set job Properties as EmailDefault, threads, NotificationEmailOther

```
Set-ContentMigratorJob –jobsrv “SERVERNAME” –threads 15 –EmailDefault –
NotificationEmailOther “test.example@dominio.com”
Set-ContentMigratorJob –jobsrv “SERVERNAME” –threads 15 –EmailDefault –
NotificationEmailOther “test.example@dominio.com” -JobRole Controller
```

Run Content Migrator Job with additional Parameters

```
Set-ContentMigratorJob –jobsrv “SERVERNAME” –threads 15 –EmailDefault –
NotificationEmailOther “test.example@dominio.com” -RunNow
```

Set-EndPointCapacityMonitorJob

This command allows the Endpoint Capacity Monitor Job to run immediately or be scheduled. When run, it will check the status of the existing endpoints. The Endpoint Capacity Monitor job is scheduled to run every minute, by default.

Parameters

- RunEveryMinutes (runevery): A number between 1 and 59 that represents how often the timer job should be run.
- ScheduledTime : Set scheduled timeframe in format time. For example: “15:00-22:00”.
- JobServer (jobsrv): Server name to run the job.
- EmailDefault: If specified (-EmailDefault), status email will be sent to the default notification group setup in General Settings.
- NotificationEmailOther: Other email addresses (not in default notification group) to receive status emails. (optional)

- RunNow: If specified (-RunNow), then the job runs immediately. This setting ignores any scheduling options provided. This may be necessary to override current scheduled jobs.
- BringOfflineEndpointsOnline (bringonline): Yes will bring endpoints back online that were taken offline because they went over the free space threshold that was set in the endpoint section of StoragePoint.

Example

Example to run the job immediately.

Set-EndpointCapacityMonitorJob –RunNow

Set job Properties as RunEveryMinutes, ScheduleTime

Set-EndpointCapacityMonitorJob –runevery 45 –ScheduledTime “15:00-20:00”

Set job Properties as EmailDefault, threads, NotificationEmailOther

Set-EndpointCapacityMonitorJob –RunEveryMinutes 45 –ScheduledTime “16:00-18:00” –NotificationEmailOther “test.example@dominio.com” –EmailDefault –NumberOfThreads 5

Set job Properties as BringOfflineEndpointsOnline, JobServer

Set-EndpointCapacityMonitorJob –BringOnline “yes” –jobsrv “SERVERNAME”

Run Endpoint Capacity Monitor with email Default and Notification to additional contacts

Set-EndpointCapacityMonitorJob –EmailDefault - NotificationEmailOther “test@dominio.com” -RunNow

Set-PerServerMaintenanceJob

This command schedules the StoragePoint Per Server Maintenance Job, which cleans up unused files from temp folders used by the servers in the SharePoint farm where StoragePoint is installed. It is scheduled to run hourly, by default. This command can be used to set which servers run the timer job.

Parameters

- ScheduleType: Designate the type of schedule. Allowable values: EveryMinute, Hourly, Daily, Weekly, Monthly.
- EveryMinute: used to indicate the number of minutes between runs. Values are a numeral with the value 1-59. Used with ScheduleType EveryMinute.
- StartMinuteHourly, -EndMinutehourly: Used to indicate the number of minutes past the top of the hour to start and stop, respectively, the Hourly timer job. Values are 1-59. Used with ScheduleType Hourly.
- JobStartDate, -JobEndDate: Used to indicate the start and stop times, respectively, of the Daily timer job. Values are represented by 24 hour format, "10:00", "15:00", etc. Used with -ScheduleType Daily. It is also used with -ScheduleType Weekly.

- BeginDayOfWeek, -EndDayOfWeek: Used to indicate the start and stop day of the week, respectively, of the Weekly timer job. Values are Sunday, Monday, Tuesday, Wednesday, Thursday, Friday or Saturday. Used with -ScheduleType Weekly.
- MonthlyEveryType: Day or date of the month to run the timer job. Values are Day or Date. Used with -ScheduleType Monthly
- StartingDayEveryMonth: Used with -MonthlyEveryType Date to say which day of the month (1-31) the job should start. Can be used with EndingDayEveryMonth.
- WeekEveryMonth: Used with -MonthlyEveryType Day, to indicate which occurrence (First, Second, Third, Fourth, or Last) of the day of the week to run the job.

Examples

Example to run the job every 45 minutes.

```
Set-PerServerMaintenanceJob -ScheduleType EveryMinute -EveryMinute 45
```

Example to run the job every hour at 45 minutes past the hour.

```
Set-PerServerMaintenanceJob -ScheduleType Hourly -StartMinuteHourly 45 -EndMinutehourly 55
```

Example to run the job daily between 3:00 PM and 8:00 PM.

```
Set-PerServerMaintenanceJob -ScheduleType Daily -JobStartDate "15:00" -JobEndDate "20:00"
```

Example to run the job weekly between Monday and Thursday.

```
Set-PerServerMaintenanceJob -ScheduleType Weekly -BeginDayOfWeek Monday -EndDayOfWeek
Thursday -JobStartDate "12:00" -JobEndDate "15:00"
```

Example to run the job Monthly by Date of the month.

```
Set-PerServerMaintenanceJob -ScheduleType Monthly -MonthlyEveryType Date -
StartingDayEveryMonth 5 -EndingDayEveryMonth 30 -JobStartDate "17:00" -JobEndDate "20:00"
```

Example to run the job Monthly by Day of the month, ie. every fourth Tuesday at 9 am.

```
Set-PerServerMaintenanceJob -ScheduleType Monthly -MonthlyEveryType Day  -BeginDayOfWeek
Tuesday -WeekEveryMonth fourth -JobStartDate "9:00"
```

BLOB Information and Migration Cmdlets

The StoragePoint cmdlet library contains functions to query for BLOB file information and also to migrate individual BLOBs.

Get-AllBLOBs

This command returns a list of BLOB Reference objects based on a SharePoint CAML query run against a specific SharePoint document library.

Parameters

- SiteUrl (s): The URL of the SharePoint site with which a profile is associated. REQUIRED.
- List (l): The SharePoint document library to query.
- Query (q): The CAML query in order to find the needed BLOB reference(s).

Example

The following runs a CAML query, finding all documents that have a file name which contains "tif". The BLOB Reference objects for the results of the query will be returned in a list.

```
Get-AllBLOBs -s http://sharepoint/docs -l Shared Documents -q "<Where><Contains><FieldRef Name=FileLeafRef /><Value Type=File>tif</Value></Contains></Where>
```

Move-BLOB

This command migrates an externalized SharePoint document's BLOB to another endpoint.

Parameters

- DocUrl (doc): Absolute URL of the document BLOB to migrate. REQUIRED.
- EndpointId (ep): The Id (GUID) or name of the target endpoint. REQUIRED.
- IncludeVersions (iv): If specified (-IncludeVersions), command will migrate all versions of the document as well.

Example

The following example moves a BLOB to the endpoint MainEndpoint2.

```
Move-BLOB -doc "http://sharepoint/docs/documents/123.tif" -ep MainEndpoint2
```

Invoke-RecallItem

This command recalls one SharePoint item to the content database. If any of the shreds (chunks) of the BLOB are shared, they will be recalled.

Parameters

- -Url : Full URL of the document you wish to recall. REQUIRED.

Example

Invoke-RecallItem -Url "http://lcamachoc-dv22:18764/sites/testSiteCollection/Shared%20Documents/Externalization1.pdf"

Invoke-ExternalizeItem

This command externalizes the BLOB or shreds of one SharePoint item to the StoragePoint Endpoint.

Parameters

- -Url : Full URL of the document, folder, list or library you wish to externalize. REQUIRED.

Example

Invoke-ExternalizeItem -Url "http://lcamachoc-dv22:18764/sites/testSiteCollection/Shared%20Documents/Externalization1.pdf"

Miscellaneous SharePoint Utility Cmdlets

The StoragePoint cmdlet library contains some miscellaneous functions to help lookup values in SharePoint that may be needed by the other commands in the library. It is important to note that there are other methods of finding these values, especially in SharePoint 2010 which has an extensive PowerShell cmdlet library of its own.

Get-SiteCollectionId

This command gets the Site Collection Id for the site collection associated with the site URL

Parameters

- SiteUrl (s): The URL of the SharePoint site. REQUIRED.

Get-ContentDBId

This command gets the Content DB Id for the content database of the specified site collection.

Parameters

- SiteUrl (s): The URL of the SharePoint site. REQUIRED.

Get-WebApplicationId

This command gets the Web Application Id for the web application of the specified site collection.

Parameters

- SiteUrl (s): The URL of the SharePoint site with which a profile is associated. REQUIRED.

Find-SharePointItem

This command will return a list of SharePoint list items based on the query submitted.

Parameters

- SiteUrl (s): The URL of the SharePoint site to query. REQUIRED.
- List (l): The SharePoint list to query. If not specified, the query will be run on the site.
- Query (q): The CAML query to run on the site or list. Can be left out if List parameter is specified (to get all items in the list).

Set-LargeFileUpload

This command allows the uploading of a large file to a library. It only processes one file at a time, and the destination must be a library; this command doesn't work on sites or subsites. Please see Large File Uploads for more information.

Parameters

- ffn: Full File Name, Full UNC path of the file to be uploaded.
- lu: List URL, Destination URL of the Document Library where the file will be uploaded. This needs to be a basic URI+list name.

Example

Example to upload a text file to the 'Doc Lib 1' document library.

```
Set-LargeFileUpload -ffn "C:\Users\Public\Documents\1.4 GB file.txt" -lu "http://sp2013a-wfe1:20264/sites/monet/doc%20lib%201"
```

PowerShell Script Examples

The following are examples of PowerShell scripts that utilize various StoragePoint cmdlets. These scripts should serve as templates for many common scenarios and can be customized as needed.

Creating a Site Collection Profile with No Encryption or Compression

This example creates a basic profile for a site collection. It uses the included FileSystem adapter and does not enable compression and encryption support. It creates a new endpoint which is then added to the profile.

```
$siteid = Get-SiteCollectionId –s "http://sharepoint/site"

Add-Endpoint –epn "NAS Endpoint 1" –adn "FileSystem" –conn "path=\\NAS\FILESTORE"

Add-Profile –pn "Site Collection 1 Profile" –ptype SiteCollection –sid $siteid –eid "NAS Endpoint 1"
```

Creating a Content Database Profile with Encryption and Compression

This example creates a basic profile with a content database scope. Also note that the endpoint is set to enable compression and encryption.

```
$cid = Get-ContentDbId –s "http://sharepoint/site"

Add-Endpoint –epn "NAS Endpoint 1" –adn "FileSystem" –conn "path=\\NAS\FILESTORE" –UseCompression –UseEncryption –EncryptionPassphrase "anypassword"

Add-Profile –pn "Content Database Profile" –ptype ContentDb –sid $cid –eid "NAS Endpoint 1"
```

Creating a Content Database Profile Using RBS

This example creates a basic content database profile using the RBS interface. RBS requires SQL Server 2008 Enterprise Edition and the profile creation will error out if trying to activate an RBS profile on any previous edition of SQL Server.

```
$cid = Get-ContentDbId –s "http://sharepoint/site"

Add-Endpoint –epn "NAS Endpoint 1" –adn "FileSystem" –conn "path=\\NAS\FILESTORE" –UseCompression –UseEncryption –EncryptionPassphrase "anypassword"
```

```
Add-Profile –pn "Content Db Profile" –ptype ContentDb –sid $cid –UseRBS –eid "NAS Endpoint 1"
```

Creating Multiple Profiles Using the Same Endpoint

This example iterates all of the site collections in a web application and creates a profile for each one. Each profile uses the same endpoint but specifies a different Start Folder. Using different Start Folder values will keep the content separated by profile even though the same endpoint is being used.

```
# Add the common endpoint

Add-Endpoint –epn "Common Endpoint 1" –adn "FileSystem" –conn "path=\\NAS\COMMON"


# Get web app - there are many other ways to do this.

$site = New-Object Microsoft.SharePoint.SPSite("http://spsite")

$webapp = $site.WebApplication


# Iterate through all site collections in web app

foreach($sc in $webapp.Sites)

{

        $startfolder = $sc.RootWeb.Title

        $profilename = $startfolder + " Profile"

        Write-Output "Creating profile for: $startfolder"


        # Create profile

Add-Profile –pn $profilename –ptype SiteCollection –sid $sc.ID.ToString() –eid "Common Endpoint 1" -
EndpointStartFolder $startfolder


        # Cleanup

        $sc.Dispose()

}


# Cleanup
```

```
$site.Dispose()
```

Creating a Profile with Asynchronous Endpoint Selection

This example creates a profile that will select endpoints asynchronously. This allows the profile to support enhanced filters including file extension/type and multiple additional scope options including web, list and content type. This sample sets up a filter to externalize only Microsoft Word and Excel documents.

```
$siteid = Get-SiteCollectionId –s "http://sharepoint/site"

Add-Endpoint –epn "NAS Endpoint 1" –adn "FileSystem" –conn "path=\\NAS\FILESTORE"

Add-Profile –pn "Site Collection 1 Profile" –ptype SiteCollection –sid $siteid –SelectEndpointAsync –eid "NAS Endpoint 1" –EndpointFileTypeOp 1 –EndpointFileTypes "DOCX,DOC,XLS,XLSX"
```

Displaying All Endpoints Meeting Certain Criteria

This example displays all endpoints in the system whose name contains the word Test. This illustrates the use of WHERE filters in PowerShell as well as use of StoragePoint API object properties (EndpointAPI.Name and .Connection in this case).

```
$Endpoints = Get-AllEndpoints | Where-Object {$_.Name -Like "*Test*"}

ForEach ($ep in $Endpoints) {$ep.Name + " : " + $ep.Connection}
```

Displaying All BLOB Files Associated with a SharePoint Document

This example displays all BLOB files associated with a given document in SharePoint. This includes versions and drafts/checked out versions, etc.

```
$blobs = Get-BLOB –doc "http://spsite/documents/123.tif"

ForEach ($b in $blobs) {$b.Endpoint.Name + " : " + $b.FilePath}
```

Enabling App Management service Application and App Catalog

The following scripts can be used to provision the service and app catalog required for enabling the StoragePoint SPFX Site Collection feature. This feature is required for full StoragePoint functionality in the SharePoint 2019 Modern User Interface.

- Provide and configure App Management service application:

```
$accountName = "<domain\ID>"
$svcAppPoolName = "AppManagementServiceApplicationPool"
$svcAppName = "App Management Service Application"
$dbServer = "<server>"
$dbName = "AppManagement_" +  [System.Guid]::NewGuid()
$svcAppProxyName = "App Management Server Application Proxy"

$account = Get-SPManagedAccount $accountName
$appPool = New-SPServiceApplicationPool -Name $svcAppPoolName -
Account $account
$svcApp = New-SPAppManagementServiceApplication -Name $svcAppName -
DatabaseServer $dbServer -DatabaseName $dbName -ApplicationPool
$appPool
New-SPAppManagementServiceApplicationProxy -Name $svcAppProxyName -
UseDefaultProxyGroup -ServiceApplication $svcApp
```

- Provide and configure AppCatalog site collection (for specific web application):

```
$appCatalogUrl = '<URL>'
New-SPSite -Url $appCatalogUrl -OwnerAlias "<domain\ID>" -Name "App
Catalog" -Template "APPCATALOG#0"
Update-SPAppCatalogConfiguration -Site $appCatalogUrl -Force:$true -
SkipWebTemplateChecking:$true
```

Altering MasterKey Encryption Password on Existing Profiles

The following script can be used to modify or create the master key encryption password that would normally be done if a web app or content database scoped profile was created using the user interface.

```
param (
    [Parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [string]
    $ContentDatabaseName,
```

```
    [Parameter(Mandatory=$true)]
    [SecureString]
    [ValidateNotNullOrEmpty()]
    $MasterkeyPassword,

    [Parameter(Mandatory=$false)]
    [Switch]
    $Force
)

Add-PSSnapin Microsoft.SharePoint.PowerShell -ErrorAction SilentlyContinue

try {
    # get SP CDB
    $cdb = Get-SPContentDatabase $ContentDatabaseName
    if ($cdb) {
        #$cdb.Name
        #$cdb.Properties["NeedEncryptionPassword"]
        #$cdb.Properties | Format-List
        if (-not $Force) { # skipping validation
            if (-not $cdb.Properties.Contains("NeedEncryptionPassword")) { # no flag present to regenerate CDB
                Write-Warning "'NeedEncryptionPassword' flag was not present to regenerate Content Database. If needed, use -Force switch to force regeneration.`n"
                exit
            }
            else {
                if (-not [string]::IsNullOrEmpty($cdb.Properties["NeedEncryptionPassword"]) -and [bool]$cdb.Properties["NeedEncryptionPassword"] -eq $false) {
                    Write-Warning ("Content Database Master Key has been already regenerated. If needed, use -Force switch to force regeneration. Detail: NeedEncryptionPassword={0}`n" -f [bool]$cdb.Properties["NeedEncryptionPassword"])
```

```
                exit
            }
        }
    }
    else { # Proceed CDB master key regeneration
        try {
            $sqlConnection = New-Object System.Data.SqlClient.SqlConnection
$cdb.DatabaseConnectionString
            $sqlConnection.Open()

            $sqlCommand = $sqlConnection.CreateCommand()
            $BSTR =
[System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($MasterkeyPassword)
            $unsecurePassword = [System.Runtime.InteropServices.Marshal]::PtrToStringAuto($BSTR)
            $sqlCommand.CommandText=[string]::Format("ALTER MASTER KEY REGENERATE WITH
ENCRYPTION BY PASSWORD = '{0}'", $unsecurePassword)
            $sqlCommand.ExecuteNonQuery() | Out-Null
            $cdb.Properties.Remove("NeedEncryptionPassword")
            $cdb.Update();
            Write-Host ("Master key for content database: {0} has been successfully regenerated.`n" -f
$ContentDatabaseName) -ForegroundColor Green
        }
        catch [Exception] {
            Write-Error ("Unable to regenerate content database master key. Detail: Content database
name={0}, Exception={1}" -f $cdb.Name, $Error[0].Exception)
        }
        finally {
            if ($sqlConnection) {
                $sqlConnection.Close()
            }
        }
    }
}
```

```
}
catch { # unexpected exceptions
   $_.Exception | format-list -force
}
```

StoragePoint API Reference

StoragePoint API Overview

StoragePoint exposes APIs for performing actions from custom Microsoft .net framework code. A Profile Creation api (ProfileAPI) is exposed as well as a Blob Migration API (BlobAPI), a Blob Reference API (BlobReferenceAPI), Archive API (ArchiveAPI), Validator API (ValidatorAPI) and Timer Job API (TimerJobAPI). These APIs are callable only from Microsoft .net framework languages (i.e. C#, VB.net) and can only run on a Web Front End server in a SharePoint farm.

Using the API Objects in Visual Studio

A Visual Studio project must reference the BlueThread.SharePoint.StoragePoint.StoragePointAPI assembly in order to use the ProfileAPI class. This assembly is installed by the core StoragePoint installation in the SharePoint bin directory (ex. C:\Program Files\Common Files\Web Server Extension\12\bin if SharePoint is installed on the C drive). To set the reference in Visual Studio:

1) Right click on the References node under the custom project.
2) Select the Browse tab and browse to the SharePoint bin directory (see above).
3) Select the file Bluethread.SharePoint.StoragePoint.StoragePointAPI.dll and click OK.

Also note the following limitations on solutions developed using the classes:

- The solution must run on a SharePoint WFE server in the farm. No provision is made for access by machines outside of the farm.
- Use outside of a SharePoint context is not supported.
- The solution should run under the SharePoint service account so that access to SharePoint resources and databases is possible.

Profile Creation API

The ProfileAPI object allows the creation of StoragePoint externalization profiles outside of the StoragePoint UI. This object is usable only from Microsoft .net code.

ProfileAPI Object Reference

Constructor Reference

The ProfileAPI object is created by one of three constructors. The first in the table below is the constructor to use to create a new profile. The other two are used only for loading existing profiles.

Constructor Signature	Description
`ProfileAPI(ProfileType type, string scopeId)`	Used to create a profile on a given site collection, content database or web application (or a no-scope profile). If type is SiteCollection, then scopeId should be the ID value of the SPSite object for the site collection. If type is WebApplication then scopeId should be the Id value of the SPWebApplication object for the web application. If type is NoScope, then scopeId should be null.
`ProfileAPI(string ProfileId)`	Used to load a profile given its unique ProfileId identifier. Cannot be used to create a new profile.
`ProfileAPI(SPListItem listItem)`	Used to load the profile for a given document (identified by its SPListItem object). Cannot be used to create a new profile.

Property Reference

The ProfileAPI object has many properties that can be set to enable various profile options. Most of these options correspond to items presented in the Add/Edit Profile page in StoragePoint's management console.

Property Name	Data Type	Required	Description
IsActive	bool	Yes	Specifies whether the profile should externalize content or not. Should generally set to True.
AdapterName	string	Yes	The storage adapter to use for the profile. Possible values include FileSystem, Atmos, AmazonS3, Azure, EMCCentera, HcapRest, and Rackspace (note that all but FileSystem require a

			separate purchase of the given adapter). FileSystem should be used to store content on file shares or most SAN devices.
Connection	string	Yes	The adapter specific connection string.
UseCompression	bool	No	Specifies whether compression is turned on for the profile. Set to true to turn on compression. Default is false.
UseEncryption	bool	No	Specifies whether encryption is turned on for the profile. Set to true to enable (default is false). By default, when enabling encryption, the profile will be automatically set to use 256-bit AES encryption. This can be overridden via the EncryptionProviderId property.
EncryptionPassphrase	string	No (unless UseEncryption set)	A password that will be used to generate an encryption key. Must be set if UseEncryption property is set to true.
EncryptionProviderId	string	No	The guid-style identifier of the encryption profile to use. This does not need to be set if the default AES 256-bit provider is desired. This id value can be obtained by examining the EncryptionProviders table in the StoragePoint database.
ProfileId	string	No	The internal guid-style identifier of the profile. Will be NULL for any new profile (until Save is called).
Type	ProfileType	Yes – in constructor	The type of the profile. Can be: SiteCollection for site collection profiles. WebApplication for web app profiles. NoScope for scope-less profiles. This property is read-only. The type of the profile being created is specified in the constructor of the ProfileAPI class.

SiteId	string	Yes – in constructor	The id of site collection that the profile is attached to (for SiteCollection type profiles). This is set in the constructor of the ProfileAPI class.
WebApplicationId	string	Yes – in constructor	The id of web application that the profile is attached to (for WebApplication type profiles). This is set in the constructor of the ProfileAPI class.
BlobRetentionDays	int	No	Number of days after a BLOB file has been orphaned that it should be kept.
Foldering	Bool	No	Specifies if content should be auto-foldered on the external file store (not in SharePoint – see CabinetOn for that). Default is true. Leaving this setting on true is highly recommended.
FolderingLevel	int	No	Specifies the format of the auto foldering scheme. Default value is 5. Possible values are: 1 for YYYY 2 for YYYY/MM 3 for YYYY/MM/DD 4 for YYYY/MM/DD/HH 5 for YYYY/MM/DD/HH/MM 6 for YYYY/MM/DD/HH/MM/SS
FileSizeOp	int	No	Specifies the operator to use if a file size filter is specified. Possible values are: 0 for less than or equal to (<=) 1 for greater than or equal to (>=)
FileSize	int	No	The file size in KB if a file size filter is specified. -1 (negative one) means no file size filter is in place and is the default value.
FileTypeOp	int	No	Specifies the operator to use if a file type filter is specified. Possible values are: 0 for Exclude specified file types

			1 for Include specified file types
FileTypes	string	No	A comma separated list of file types in the filter. The file type must be UPPER case and be the correct extension for the file type. For example, TIF for TIFF files or PPTX for PowerPoint files. Separate multiple file types by commas (,). The value should be null (the default) if no file type filter is desired.
RMEnabled	bool	No	Specifies whether Records Management is enabled.
RMScope	string	No	Scope of the Records Management rules.
RMDeclaredRecord	bool	No	Whether declaring a record moves the BLOB.
RMDeclaredRecordEndpoint	EndpointAPI	No	The endpoint which stores the BLOB of a record.
RMUndeclaredRecord	bool	No	Whether undeclaring a record moves the BLOB.
RMUndeclaredRecordEndpoint	EndpointAPI	No	The endpoint which stores the BLOB that is no longer a record.
HMEnabled	bool	No	Specifies whether Holds Management is enabled.
HMScope	string	No	Scope of the Holds Management rules.
HMOnHold	bool	No	Whether putting an item in hold moves the BLOB.
HMOnHoldEndpoint	EndpointAPI	No	The endpoint which stores the BLOB of an item in hold.
HMRemovedFromHold	bool	No	Whether removing a file from hold moves the BLOB.
HMRemovedFromHoldEndpoint	EndpointAPI	No	The endpoint which stores the BLOB that is no longer in a hold.

Method Reference

There are a few methods provided in the ProfileAPI object to accomplish certain tasks. The following methods are available:

Method Name	Parameters	Description
Save	(none)	Saves the profile and activates if (if necessary) on the specified site collection or web application.
Test	(none)	Tests the profile by writing out a small test file and then deleting it. Will throw an exception if anything goes wrong.
Delete	int recallJobThreads -> Number of threads to allocate to the recall job that is scheduled by the delete. string notificationEmailAddress -> Email address to send completion email to when delete and recall are complete..	Schedules a deletion of the profile. Deletion of a profile requires a recall job to run first. Thus, this method returns before the deletion is complete.
CreateOrphanBlobCleanupJob	SPSchedule schedule -> Specifies the schedule of when to run the job. Can be any of the SharePoint SPSchedule derived types such as SPDailySchedule or SPWeeklySchedule. string emailNotificationAddress -> Specifies the email address to send the job completion email to. Pass null if no email is desired. bool emailOnErrorOnly -> true to send completion email on error only and false to send always.	Creates an Orphan BLOB Cleanup Job (garbage collector) for the profile.

Profile Creation Code Examples

The following are examples in Microsoft C# of creating various types of StoragePoint profiles. A good strategy for creating Profile API code is to take one of these samples and customize it as needed. For example, to create a profile with encryption but not compression support, simply take the Creating a Profile with Encryption and Compression example and delete the line profile.UseCompression = true;.

> NOTE: All examples assume the presence of an SPSite object named site (for site collection profile examples) or an SPWebApplication object named webapp (for web application examples). These are used solely to obtain the Id value of the given site or web application. The Id value may be obtained in other ways or even hard-coded if desired.

Creating a Site Collection Profile with No Encryption or Compression (C#)

This example creates a basic profile on a site collection. It uses the included FileSystem adapter and does not enable compression and encryption support.

```
ProfileAPI profile =
        new ProfileAPI(ProfileAPI.ProfileType.SiteCollection,
                                site.ID.ToString());
profile.IsActive = true;       // Profile should externalize content
profile.AdapterName = "FileSystem";
profile.Connection = "path=e:\\ExternalFileStore";
profile.Save();
```

Creating a Web Application Profile with No Encryption or Compression (C#)

This example creates a basic profile (similar to the last example) on a Web Application.

```
ProfileAPI profile =
        new ProfileAPI(ProfileAPI.ProfileType.WebApplication,
                                webapp.Id.ToString());
profile.IsActive = true;       // Profile should externalize content
profile.AdapterName = "FileSystem";
profile.Connection = "path=e:\\ExternalFileStore";
profile.Save();
```

Creating a Profile with Encryption and Compression (C#)

This example builds on the previous examples to include compression and encryption support. This example may be used on a web application by modifying the first line to create a ProfileType.WebApplication type profile (see the last example).

```
ProfileAPI profile =
        new ProfileAPI(ProfileAPI.ProfileType.SiteCollection,
                                site.ID.ToString());
profile.IsActive = true;       // Profile should externalize ntent
profile.AdapterName = "FileSystem";
```

```
profile.Connection = "path=e:\\ExternalFileStore";
profile.UseCompression = true;
profile.UseEncryption = true;
profile.EncryptionPassphrase = "your_unique_passphrase_here";
profile.Save();
```

Creating a Web Application Profile with a File Type and File Size Filter (C#)

This example creates a basic profile (similar to the last example) on a Web Application.

```
ProfileAPI profile =
        new ProfileAPI(ProfileAPI.ProfileType.WebApplication,
                                webapp.Id.ToString());
profile.IsActive = true;       // Profile should externalize content
profile.AdapterName = "FileSystem";
profile.Connection = "path=e:\\ExternalFileStore";
profile.FileSizeOp = 1;        // >=
profile.FileSize = 40;         // 40k
profile.FileTypeOp = 1;        // Include
profile.FileTypes = "TIF,PPTX"; // TIFF and powerpoint files
profile.Save();
```

Creating a Scope-less Profile with No Encryption or Compression (C#)

This example creates a scope-less profile that stores externalized content on an EMC Atmos cloud account (requires the Atmos cloud adapter for StoragePoint and an EMC Atmos online account). A scope-less profile can only be used to externalize content from a workflow or other custom code (like a custom event receiver).

```
ProfileAPI profile = new ProfileAPI(ProfileAPI.ProfileType.NoScope,null);
profile.IsActive = true;
profile.AdapterName = "Atmos";
profile.Connection ="UID=atmos user id;Key=atmos secret key;";
profile.UseCompression = true;
profile.UseEncryption = true;
profile.EncryptionPassphrase = "your_unique_passphrase_here";
profile.Save();
```

Deleting a Profile (C#)

This example loads an existing profile and schedules it for deletion.

```
ProfileAPI p =
            new ProfileAPI("81d2fa85-365d-4e56-8820-295f9cdd1aa5");
p.Delete(4, "admin@storagepoint.local");
```

Archive API

The ArchiveAPI object allows the creation of StoragePoint archiving rules to existing StoragePoint profiles. This object is usable only from Microsoft .net code.

Archive API Object Reference

Property Reference

The ArchiveAPI object has many properties that can be set to enable various archiving options. These options correspond to fields presented in the Archiving section of the Profile page in StoragePoint's management console.

AgeRuleFormatAPI crates a rule for archiving content based on elapsed time since the content was created or modified.

Property Name	Data Type	Required	Description
DateProperty	string	yes	Valid values are Created or Modified. Specifies the date field to be used to measure elapsed time.
Duration	int	yes	Duration of time that has elapsed.
Interval	string	yes	Units of timer duration. Valid entries are Day, Days, Month, Months, Year, Years.

MetadataRuleFormatAPI can be used to archive content based on a field that has met archiving conditions.

Property Name	Data Type	Required	Description
Property	string	yes	The metadata column name used to evaluate the content.
Operator	string	yes	The analysis used to evaluate the property. This will vary depending on the property used, but usually it will be =, <, >, etc.
Value	string	yes	Used to compare against the property.

Common fields for all formats.

Property Name	Data Type	Required	Description

DestinationEndpointName	string	yes	The endpoint name or ID for archiving the BLOB when conditions are met.

Archive Rule Code Examples

The ArchiveAPI can be used to add archiving rules to a profile, as seen in the examples below.

```
using Bluethread.SharePoint.StoragePoint;
using Bluethread.SharePoint.StoragePoint.ArchiveAPI;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {

             ProfileAPI profile = new ProfileAPI("CDB-1111"); //Creates a APIProfile based on
       existent profile name.
            profile.ArchivingSettings.Enabled = true;         //To enable the Archiving
            MetadataRuleTest(profile);                              //Calls to MetadataRules
            AgeRuleTest(profile);                             //Calls to AgeRuleTest
             profile.Save();                                   //Needed to save changes to the
       profile



       //Creates AgeRuleTest with only one condition
        private static void AgeRuleTest(ProfileAPI profile)
        {
            AgeRuleAPI ageRule = new AgeRuleAPI(profile, "EP1", "cdb:74d30558-d550-4678-8f5c-
b75b97855e0f/s:50777858-f8e3-417a-8ca6-973ac0a47209/w:728c538c-8275-43c0-a6c6-5067c6279b7a");
            AgeRuleFormatAPI ageRuleInput = new AgeRuleFormatAPI();
            ageRuleInput.DateProperty = "Created";
            ageRuleInput.Duration = 5;
            ageRuleInput.Interval = "Day";
            ageRuleInput.DestinationEndpointName = "EP2";
            ageRuleInput.Property = "Name";
            ageRuleInput.Operator = "Contains";
            ageRuleInput.Value = "metalogix";

            bool isValidAge = ageRule.Validate(ageRuleInput.GetArchiveRule());
            ageRule.Save();
        }

        //Creates MetadataRuleTest with only one condition
        private static void MetadataRuleTest(ProfileAPI profile)
        {
            MetadataRuleAPI metadata = new MetadataRuleAPI(profile, "EP2", "cdb:74d30558-
d550-4678-8f5c-b75b97855e0f/s:50777858-f8e3-417a-8ca6-973ac0a47209/w:728c538c-8275-43c0-a6c6-
5067c6279b7a");
```

```
            MetadataRuleFormatAPI metadataInput = new MetadataRuleFormatAPI();
            metadataInput.Property = "Name";
            metadataInput.Operator = "Begins";
            metadataInput.Value = "metalogix";
            metadataInput.DestinationEndpointName = "EP1";

            bool isValid = metadata.Validate(metadataInput.GetArchiveRule());
            metadata.Save();
        }
    }
}
```

Example Two

```
using Bluethread.SharePoint.StoragePoint;
using Bluethread.SharePoint.StoragePoint.ArchiveAPI;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace StoragePointAPI
{
    class ArchiveExamples
    {

        public static void EnableArchiving()
        {
            // enable the Archive in the profile
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");
            profileInstance.ArchivingSettings.Enabled = true;
            profileInstance.Save();
        }

        public static void CreateAgeRulewithMetadata()
        {
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");

            AgeRuleFormatAPI ageRuleinput = new AgeRuleFormatAPI();
            ageRuleinput.DateProperty = "created"; // it can be : created , modified
            ageRuleinput.DestinationEndpointName = "EP1"; // endpoint Name
            ageRuleinput.Duration = 15; // duration of the age rule
            ageRuleinput.Interval = "day"; // it can be : day,month, year
            // metadata section
            ageRuleinput.Property = "Name"; //metadata property name
            ageRuleinput.Operator = "=" ; // operator it depends of the Metadata Name
            ageRuleinput.Value = "Document"; //metadata value

            AgeRuleAPI ageRuleAPI = new AgeRuleAPI(profileInstance, "EP1", "wa:ec8d5b10-ccc6-
4fea-9fad-7e42bd362ab1");
            ageRuleAPI.Validate(ageRuleinput.GetArchiveRule()); // this validate the rule and
check it if the any other rule exists for the same scope
            ageRuleAPI.Save(); // it save the rule
        }


        public static void CreateAgeRuleWithoutMetadata()
        {
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");

            AgeRuleFormatAPI ageRuleinput = new AgeRuleFormatAPI();
            ageRuleinput.DateProperty = "modified"; // it can be : created , modified
```

```
            ageRuleinput.DestinationEndpointName = "EP1"; // endpoint Name destination
            ageRuleinput.Duration = 2; // duration of the age rule
            ageRuleinput.Interval = "month"; // it can be : day,month, year
            // metadata section
            ageRuleinput.Property = "None"; //metadata property name
            ageRuleinput.Operator = "None"; // operator it depends of the Metadata Name
            ageRuleinput.Value = "None"; //metadata value

            AgeRuleAPI metadataRule = new AgeRuleAPI(profileInstance, "EP1", "wa:ec8d5b10-
ccc6-4fea-9fad-7e42bd362ab1/cdb:b3589d08-70da-4081-a31c-ab52552df86f");
            metadataRule.Validate(ageRuleinput.GetArchiveRule()); // this validate the rule
and check it if the any other rule exists for the same scope
            metadataRule.Save(); // it save the rule
        }


        public static void CreateAgeRuleWithManyConditions()
        {
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");

            AgeRuleFormatAPI ageRuleinput = new AgeRuleFormatAPI();
            ageRuleinput.DateProperty = "modified"; // it can be : created , modified
            ageRuleinput.DestinationEndpointName = "EP1"; // endpoint Name destination
            ageRuleinput.Duration = 2; // duration of the age rule
            ageRuleinput.Interval = "month"; // it can be : day,month, year
            // metadata section
            ageRuleinput.Property = "None"; //metadata property name
            ageRuleinput.Operator = "None"; // operator it depends of the Metadata Name
            ageRuleinput.Value = "None"; //metadata value


            AgeRuleFormatAPI ageRuleinputWithMetadata = new AgeRuleFormatAPI();
            ageRuleinputWithMetadata.DateProperty = "modified"; // it can be : created ,
modified
            ageRuleinputWithMetadata.DestinationEndpointName = "EP3"; // endpoint Name
destination
            ageRuleinputWithMetadata.Duration = 2; // duration of the age rule
            ageRuleinputWithMetadata.Interval = "month"; // it can be : day,month, year
            // metadata section
            ageRuleinputWithMetadata.Property = "Name"; //metadata property name
            ageRuleinputWithMetadata.Operator = "begins"; // operator it depends of the
Metadata Name
            ageRuleinputWithMetadata.Value = "important"; //metadata value

            AgeRuleAPI metadataRule = new AgeRuleAPI(profileInstance, "EP1,EP3",
"wa:ec8d5b10-ccc6-4fea-9fad-7e42bd362ab1/cdb:b3589d08-70da-4081-a31c-ab52552df86f");
            metadataRule.Validate(ageRuleinput.GetArchiveRule() + "?" +
ageRuleinputWithMetadata.GetArchiveRule()); // this validate the rule and check it if the any
other rule exists for the same scope
            metadataRule.Save(); // it saves the rule
        }

        public static void CreateMetadataRule()
        {
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");
            MetadataRuleFormatAPI metadataFormatAPI = new MetadataRuleFormatAPI();
            metadataFormatAPI.DestinationEndpointName = "EP1";
            metadataFormatAPI.Operator = "=";
            metadataFormatAPI.Property = "Modified";
            metadataFormatAPI.Value = "12/10/2015";
            MetadataRuleAPI metadataInstance = new MetadataRuleAPI(profileInstance, "EP1",
"wa:ec8d5b10-ccc6-4fea-9fad-7e42bd362ab1");
            metadataInstance.Validate(metadataFormatAPI.GetArchiveRule());
            metadataInstance.Save();
```

```
        }

        public static void CreateMetadataRuleWithManyConditions()
        {
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");
            MetadataRuleFormatAPI metadataFirstAPI = new MetadataRuleFormatAPI();
            metadataFirstAPI.DestinationEndpointName = "EP1";
            metadataFirstAPI.Operator = "=";
            metadataFirstAPI.Property = "Select";
            metadataFirstAPI.Value = "James";

            MetadataRuleFormatAPI metadataSecondtAPI = new MetadataRuleFormatAPI();
            metadataSecondtAPI.DestinationEndpointName = "EP3";
            metadataSecondtAPI.Operator = "=";
            metadataSecondtAPI.Property = "Title";
            metadataSecondtAPI.Value = "head first";


            MetadataRuleAPI metadataInstance = new MetadataRuleAPI(profileInstance,
"EP1,EP3", "wa:ec8d5b10-ccc6-4fea-9fad-7e42bd362ab1/cdb:b3589d08-70da-4081-a31c-
ab52552df86f");
            metadataInstance.Validate(metadataFirstAPI.GetArchiveRule() + '?' +
metadataSecondtAPI.GetArchiveRule());
            metadataInstance.Save();

        }

    }
}
```

Timer Job API

The TimerJobAPI object allows for the configuration of timer jobs and for scheduling and running the jobs. It also has commands for monitoring timer jobs and getting statuses.

Timer Job API Object Reference

The TimerJobAPI classes can be used to configure StoragePoint timer jobs.

Methods

Method Name	Description
public static ContentMigratorJobSettings GetInstance(string server)	
public void RunJob(string server, string webRole, int NumberOfThreads, bool emailDefault, string NotificationEmailOther, string includeLargeFiles, string workers)	Used for running the timer job.
public void SetThreads(int numberOfThreads)	Set the number of threads for a job.

public void SetIncludeLargeFiles(string includeLargeFilesOption)	Set whether or not the timer job should include the processing of Large File Uploads.
public void Suspend()	Suspend a currently running job.
public void Abort()	Abort a currently running job.
public void Resume()	Resume a suspended job.

Properties

Use these to pull current information about configured jobs. TimeJobStatusAPI

Property name
public DateTime Started { get; }
public DateTime Stopped { get; }
public DateTime HeartBeat { get; }
public int ThreadCount { get; }
public string Status { get; }
public string LastErr { get; }

Timer Job API Object Reference Code Examples

Below is a listing of the profile timer jobs set up with variations on schedule and options.

```
using Bluethread.SharePoint.StoragePoint;
using Microsoft.SharePoint.Administration;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace StoragePointAPI
{
    public class TimerJobExamples
    {
        public static void BLOBHealthAnalyzerExample()
```

```
        {
            //create  one-time scheduled job to run immediately.
            Microsoft.SharePoint.SPOneTimeSchedule oneTimeSchedule = new
Microsoft.SharePoint.SPOneTimeSchedule(DateTime.Now);
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");
            profileInstance.CreateBLOBRefScanJob(oneTimeSchedule, true, "5", true,
"othermail@gmail.com", false);
        }

        public static void BLOBHealthAnalyzerWithServerExample()
        {
            //create  one-time scheduled job to run immediately.
            Microsoft.SharePoint.SPOneTimeSchedule oneTimeSchedule = new
Microsoft.SharePoint.SPOneTimeSchedule(DateTime.Now);
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");
            profileInstance.CreateBLOBRefScanJob(oneTimeSchedule, true, "5", true,
"othermail@gmail.com", false, SPServer.Local);

            // create daily scheduled job definition
            Microsoft.SharePoint.SPDailySchedule dailySchedule = new
Microsoft.SharePoint.SPDailySchedule();
            dailySchedule.BeginHour = 13;
            dailySchedule.EndHour = 18;
            dailySchedule.BeginMinute = 10;
            dailySchedule.EndMinute = 50;
            profileInstance.CreateBLOBRefScanJob(dailySchedule, true, "5", true,
"othermail@gmail.com", false);

        }


        public static void UnusedBLOBCleanupExample()
        {
            //create  one-time scheduled job to run immediately.
            Microsoft.SharePoint.SPOneTimeSchedule oneTimeSchedule = new
Microsoft.SharePoint.SPOneTimeSchedule(DateTime.Now);
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");
            profileInstance.CreateUnusedBlobCleanupJob(oneTimeSchedule, true,
"othermail@gmail.com", true, SPServer.Local, false);

            // create daily scheduled job definition
            Microsoft.SharePoint.SPDailySchedule dailySchedule = new
Microsoft.SharePoint.SPDailySchedule();
            dailySchedule.BeginHour = 13;
            dailySchedule.EndHour = 18;
            dailySchedule.BeginMinute = 10;
            dailySchedule.EndMinute = 50;
            profileInstance.CreateUnusedBlobCleanupJob(dailySchedule, true,
"othermail@gmail.com", true, SPServer.Local, false);

        }

        public static void BLOBExternalizationExample()
        {
            //create  one-time scheduled job to run immediately.
            Microsoft.SharePoint.SPOneTimeSchedule oneTimeSchedule = new
Microsoft.SharePoint.SPOneTimeSchedule(DateTime.Now);
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");
            profileInstance.CreateBLOBExternalizationJob(oneTimeSchedule, "5", true,
"othermail@gmail.com", true);

            // create daily scheduled job definition
            Microsoft.SharePoint.SPDailySchedule dailySchedule = new
Microsoft.SharePoint.SPDailySchedule();
```

```
            dailySchedule.BeginHour = 13;
            dailySchedule.EndHour = 18;
            dailySchedule.BeginMinute = 10;
            dailySchedule.EndMinute = 50;
            profileInstance.CreateBLOBExternalizationJob(oneTimeSchedule, "5", true,
"othermail@gmail.com", true, SPServer.Local);
        }

        public static void BLOBRecallJobExample()
        {
            //create  one-time scheduled job to run immediately.
            Microsoft.SharePoint.SPOneTimeSchedule oneTimeSchedule = new
Microsoft.SharePoint.SPOneTimeSchedule(DateTime.Now);
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");
            profileInstance.CreateBLOBRecallJob(oneTimeSchedule, "5", true,
"othermail@gmail.com", true);

            // create daily scheduled job definition
            Microsoft.SharePoint.SPDailySchedule dailySchedule = new
Microsoft.SharePoint.SPDailySchedule();
            dailySchedule.BeginHour = 13;
            dailySchedule.EndHour = 18;
            dailySchedule.BeginMinute = 10;
            dailySchedule.EndMinute = 50;
            profileInstance.CreateBLOBRecallJob(oneTimeSchedule, "5", true,
"othermail@gmail.com", true, SPServer.Local);
        }

        public static void BLOBMigrateJobExample()
        {

            // get the endpoint source and endpoint target'
            EndpointAPI source = new EndpointAPI("EP1");
            EndpointAPI target = new EndpointAPI("EP3");

            //create  one-time scheduled job to run immediately.
            Microsoft.SharePoint.SPOneTimeSchedule oneTimeSchedule = new
Microsoft.SharePoint.SPOneTimeSchedule(DateTime.Now);
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");
            profileInstance.CreateBLOBMigrateJob(oneTimeSchedule, "5", source, target, true,
"othermail@gmail.com", true);

            // create daily scheduled job definition
            Microsoft.SharePoint.SPDailySchedule dailySchedule = new
Microsoft.SharePoint.SPDailySchedule();
            dailySchedule.BeginHour = 13;
            dailySchedule.EndHour = 18;
            dailySchedule.BeginMinute = 10;
            dailySchedule.EndMinute = 50;
            profileInstance.CreateBLOBMigrateJob(oneTimeSchedule, "5", source, target, true,
"othermail@gmail.com", true, "s:4dd13983-aac1-4a98-8e83-a0abf61d11e1", "include");

        }


        public static void ArchivingAgingJob()
        {
            //create  one-time scheduled job to run immediately.
            Microsoft.SharePoint.SPOneTimeSchedule oneTimeSchedule = new
Microsoft.SharePoint.SPOneTimeSchedule(DateTime.Now);
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");
            profileInstance.CreateAgingJob(oneTimeSchedule, "5", true, "othermail@gmail.com",
true);
```

```
            // create daily scheduled job definition
            Microsoft.SharePoint.SPDailySchedule dailySchedule = new
Microsoft.SharePoint.SPDailySchedule();
            dailySchedule.BeginHour = 13;
            dailySchedule.EndHour = 18;
            dailySchedule.BeginMinute = 10;
            dailySchedule.EndMinute = 50;
            profileInstance.CreateAgingJob(dailySchedule, "5", true, "othermail@gmail.com",
true, SPServer.Local);
        }

        public static void ArchivingMetadataJob()
        {
            //create  one-time scheduled job to run immediately.
            Microsoft.SharePoint.SPOneTimeSchedule oneTimeSchedule = new
Microsoft.SharePoint.SPOneTimeSchedule(DateTime.Now);
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");
            profileInstance.CreateMetadataJob(oneTimeSchedule, "5", true,
"othermail@gmail.com", true);

            // create daily scheduled job definition
            Microsoft.SharePoint.SPDailySchedule dailySchedule = new
Microsoft.SharePoint.SPDailySchedule();
            dailySchedule.BeginHour = 13;
            dailySchedule.EndHour = 18;
            dailySchedule.BeginMinute = 10;
            dailySchedule.EndMinute = 50;
            profileInstance.CreateMetadataJob(dailySchedule, "5", true,
"othermail@gmail.com", true, SPServer.Local);
        }

        public static void ArchivingVersioningJob()
        {
            //create  one-time scheduled job to run immediately.
            Microsoft.SharePoint.SPOneTimeSchedule oneTimeSchedule = new
Microsoft.SharePoint.SPOneTimeSchedule(DateTime.Now);
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");
            profileInstance.CreateVersionsJob(oneTimeSchedule, "5", true,
"othermail@gmail.com", true);

            // create daily scheduled job definition
            Microsoft.SharePoint.SPDailySchedule dailySchedule = new
Microsoft.SharePoint.SPDailySchedule();
            dailySchedule.BeginHour = 13;
            dailySchedule.EndHour = 18;
            dailySchedule.BeginMinute = 10;
            dailySchedule.EndMinute = 50;
            profileInstance.CreateVersionsJob(dailySchedule, "5", true,
"othermail@gmail.com", true, SPServer.Local);
        }

        public static void BLOBMigrateRecordsJob()
        {
            //create  one-time scheduled job to run immediately.
            Microsoft.SharePoint.SPOneTimeSchedule oneTimeSchedule = new
Microsoft.SharePoint.SPOneTimeSchedule(DateTime.Now);
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");
            profileInstance.CreateMigrateRecordsJob(oneTimeSchedule, true, "5", true,
"othermail@gmail.com", true);

            // create daily scheduled job definition
            Microsoft.SharePoint.SPDailySchedule dailySchedule = new
Microsoft.SharePoint.SPDailySchedule();
            dailySchedule.BeginHour = 13;
```

```
            dailySchedule.EndHour = 18;
            dailySchedule.BeginMinute = 10;
            dailySchedule.EndMinute = 50;
            profileInstance.CreateMigrateRecordsJob(dailySchedule, "5", true,
"othermail@gmail.com", true, SPServer.Local);
        }

        public static void BLOBMigrateHoldJob()
        {
            //create  one-time scheduled job to run immediately.
            Microsoft.SharePoint.SPOneTimeSchedule oneTimeSchedule = new
Microsoft.SharePoint.SPOneTimeSchedule(DateTime.Now);
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");
            profileInstance.CreateMigrateHoldsJob(oneTimeSchedule, true, "5", true,
"othermail@gmail.com", true);

            // create daily scheduled job definition
            Microsoft.SharePoint.SPDailySchedule dailySchedule = new
Microsoft.SharePoint.SPDailySchedule();
            dailySchedule.BeginHour = 13;
            dailySchedule.EndHour = 18;
            dailySchedule.BeginMinute = 10;
            dailySchedule.EndMinute = 50;
            profileInstance.CreateMigrateHoldsJob(dailySchedule, true, "5", true,
"othermail@gmail.com", true, SPServer.Local);
        }

        public static void BLOBBackupSyncJob()
        {
            //create  one-time scheduled job to run immediately.
            Microsoft.SharePoint.SPOneTimeSchedule oneTimeSchedule = new
Microsoft.SharePoint.SPOneTimeSchedule(DateTime.Now);
            ProfileAPI profileInstance = new ProfileAPI("SharePoint-15548");
            profileInstance.CreateBackupSyncJob(oneTimeSchedule, "5", true,
"othermail@gmail.com", true);

            // create daily scheduled job definition
            Microsoft.SharePoint.SPDailySchedule dailySchedule = new
Microsoft.SharePoint.SPDailySchedule();
            dailySchedule.BeginHour = 13;
            dailySchedule.EndHour = 18;
            dailySchedule.BeginMinute = 10;
            dailySchedule.EndMinute = 50;
            profileInstance.CreateBackupSyncJob(dailySchedule, "5", true,
"othermail@gmail.com", true, SPServer.Local);
        }

    }
}
```

TimerJobStatusAPI Reference - Content Migrator Example

The following example is specific to the Content Migrator Job.

```
using Bluethread.SharePoint.StoragePoint.TimerJobAPI;
using Microsoft.SharePoint.Administration;
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;

namespace StoragePointAPI
{
    public class ContentMigratorExample
    {
        public static void SetRunEveryMinutes_Example()
        {
            int minutes = 50; //Run every 50 minutes
            string serverName = Environment.MachineName;
            ContentMigratorJobSettings contentMigratorSettings =
ContentMigratorJobSettings.GetInstance(serverName);
            contentMigratorSettings.SetRecurrenceDuration(minutes);
            contentMigratorSettings.CommitChanges();
        }

        public static void SetNumberOfThreads_Example()
        {
            int numberOfThreads = 5; // number of Threads to run the job
            string serverName = Environment.MachineName;
            ContentMigratorJobSettings contentMigratorSettings =
ContentMigratorJobSettings.GetInstance(serverName);
            contentMigratorSettings.SetRecurrenceDuration(numberOfThreads);
            contentMigratorSettings.CommitChanges();
        }

        public static void SetJobRole_Example()
        {
            string jobRole = "standAlone";
            string serverName = Environment.MachineName;
            ContentMigratorJobSettings contentMigratorSettings =
ContentMigratorJobSettings.GetInstance(serverName);
            contentMigratorSettings.SetJobRole(jobRole);
            contentMigratorSettings.CommitChanges();
        }

        public static void SetEmailDefault_Example()
        {
            string emailDefault = "True";
            string serverName = Environment.MachineName;
            ContentMigratorJobSettings contentMigratorSettings =
ContentMigratorJobSettings.GetInstance(serverName);
            contentMigratorSettings.SetNotificationEmailDefault(emailDefault);
            contentMigratorSettings.CommitChanges();
        }

        public static void SetNotificationEmailOther_Example()
        {
            string emailother = "otherMail@hotmail.com";
            string serverName = Environment.MachineName;
            ContentMigratorJobSettings contentMigratorSettings =
ContentMigratorJobSettings.GetInstance(serverName);
            contentMigratorSettings.SetNotificationEmailOther(emailother);
            contentMigratorSettings.CommitChanges();
        }

        public static void SetManyPropertiesOfCMQ_Example()
        {
            string emailother = "otherMail@hotmail.com";
            string serverName = Environment.MachineName;
            ContentMigratorJobSettings contentMigratorSettings =
ContentMigratorJobSettings.GetInstance(serverName);
            contentMigratorSettings.SetNotificationEmailOther(emailother);
```

```
            contentMigratorSettings.SetJobRole("standAlone");
            contentMigratorSettings.SetThreads(5);

            contentMigratorSettings.CommitChanges();
        }


        public static void RunJob_Example()
        {
            string serverName = Environment.MachineName;
            ContentMigratorJobSettings contentMigratorSettings =
ContentMigratorJobSettings.GetInstance(serverName);
            contentMigratorSettings.RunJob(SPServer.Local.Name, "standAlone", 5, true,
"otherMail@hotmail.com", "yes", string.Empty);
        }

        public static void SetScheduleTime()
        {
            string serverName = Environment.MachineName;
            string startHour = "15:15";
            string endHour = "20:15";
            ContentMigratorJobSettings contentMigratorSettings =
ContentMigratorJobSettings.GetInstance(serverName);
            contentMigratorSettings.SetStartHour(startHour);
            contentMigratorSettings.SetEndHour(endHour);
            contentMigratorSettings.CommitChanges();
        }
    }
}
```

TimerJobStatusAPI Reference - Capacity Monitor Example

This is an example for the Capacity Monitor Job.

```
using Bluethread.SharePoint.StoragePoint.TimerJobAPI;
using Microsoft.SharePoint.Administration;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace StoragePointAPI
{
    public class EndpointCapacityMonitorExamples
    {

        public static void SetRunEveryMinutes_Example()
        {
            string serverName = Environment.MachineName;
            int runEveryMinutes = 45;
            EndPointCapacityMonitorJobSettings endPointCapacityMonitorInstance =
EndPointCapacityMonitorJobSettings.GetInstance(serverName);
            endPointCapacityMonitorInstance.SetRecurrenceDuration(runEveryMinutes);
            endPointCapacityMonitorInstance.CommitChanges();
        }

        public static void SetBringOfflineEndPointsOnline_Example()
        {
            string serverName = Environment.MachineName;
```

```
            string bringOfflineEndpoint = "yes";
            EndPointCapacityMonitorJobSettings endPointCapacityMonitorInstance =
EndPointCapacityMonitorJobSettings.GetInstance(serverName);
            endPointCapacityMonitorInstance.SetBringOfflineEndpointsBackOnline(bringOfflineEn
dpoint);
            endPointCapacityMonitorInstance.CommitChanges();
        }

        public static void SetEmailDefault_Example()
        {
            string serverName = Environment.MachineName;
            string emailDefault = true.ToString();
            EndPointCapacityMonitorJobSettings endPointCapacityMonitorInstance =
EndPointCapacityMonitorJobSettings.GetInstance(serverName);
            endPointCapacityMonitorInstance.SetNotificationEmailDefault(emailDefault);
            endPointCapacityMonitorInstance.CommitChanges();
        }

        public static void setNotificationEmailOther()
        {
            string serverName = Environment.MachineName;
            string emailOther = "otherMail@hotmail.com";
            EndPointCapacityMonitorJobSettings endPointCapacityMonitorInstance =
EndPointCapacityMonitorJobSettings.GetInstance(serverName);
            endPointCapacityMonitorInstance.SetNotificationEmailOther(emailOther);
            endPointCapacityMonitorInstance.CommitChanges();
        }

        public static void SetScheduleTime()
        {
            string serverName = Environment.MachineName;
            string startHour = "15:15";
            string endHour = "20:15";
            EndPointCapacityMonitorJobSettings endPointCapacityMonitorInstance =
EndPointCapacityMonitorJobSettings.GetInstance(serverName);
            endPointCapacityMonitorInstance.SetStartHour(startHour);
            endPointCapacityMonitorInstance.SetEndHour(endHour);
            endPointCapacityMonitorInstance.CommitChanges();
        }

        public static void RunJob()
        {
            string serverName = Environment.MachineName;
            EndPointCapacityMonitorJobSettings endPointCapacityMonitorInstance =
EndPointCapacityMonitorJobSettings.GetInstance(serverName);
            endPointCapacityMonitorInstance.RunJob(SPServer.Local.Name, "yes", true,
"otherMail@hotmail.com");
        }


    }
}
```

TimerJobStatusAPI

The TimerJobStatusAPI retuns lists of timer jobs in their current state (running, completed, etc.)

Timer Job Status API Object Reference

The following can be used to list jobs in the following statuses.

public static List<TimerJobStatusAPI> GetAvailableJobs()
public static List<TimerJobStatusAPI> GetRunningJobs()
public static List<TimerJobStatusAPI> GetCompleteJobs()
public static List<TimerJobStatusAPI> GetAbandonedJobs()
public static List<TimerJobStatusAPI> GetAbortingJobs()
public static List<TimerJobStatusAPI> GetAbortedJobs()
public static List<TimerJobStatusAPI> GetSuspendingJobs()
public static List<TimerJobStatusAPI> GetSuspendedJobs()
public static List<TimerJobStatusAPI> GetResumingJobs()
public static List<TimerJobStatusAPI> GetExceptionJobs()

Timer Job Status API Object Reference Code Examples

```
using Bluethread.SharePoint.StoragePoint.TimerJobAPI;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace StoragePointAPI
{
    public class TimerJobStatusAPIExample
    {
        public static void GetAvaliableJobsExamples()
        {
            List<TimerJobStatusAPI> avaliableJobs = TimerJobStatusAPI.GetAvailableJobs();
            foreach (TimerJobStatusAPI avaliableJob in avaliableJobs)
            {
                Console.WriteLine(string.Format("Job Name: {0}", avaliableJob.Name));
                Console.WriteLine(string.Format("Job Type: {1}",avaliableJob.Type));
                Console.WriteLine(string.Format("Job Started Time: {0}",
avaliableJob.Started));
                Console.WriteLine(string.Format("Job Server Name: {0}", avaliableJob.Name));
                Console.WriteLine(string.Format("Job Status: {0}", avaliableJob.Status));
                Console.WriteLine(string.Format("Job State: {0}", avaliableJob.State));
            }
        }


        public static void GetRunningJobsExamples()
        {
            List<TimerJobStatusAPI> runningJobs = TimerJobStatusAPI.GetRunningJobs();
            foreach (TimerJobStatusAPI runningJob in runningJobs)
            {
                Console.WriteLine(string.Format("Job Name: {0}", runningJob.Name));
                Console.WriteLine(string.Format("Job Type: {1}", runningJob.Type));
                Console.WriteLine(string.Format("Job Started Time: {0}",
runningJob.Started));
                Console.WriteLine(string.Format("Job Server Name: {0}", runningJob.Name));
                Console.WriteLine(string.Format("Job Status: {0}", runningJob.Status));
```

```
                Console.WriteLine(string.Format("Job State: {0}", runningJob.State));
            }
        }

        public static void GetCompleteJobsExample()
        {
            List<TimerJobStatusAPI> completedJobs = TimerJobStatusAPI.GetCompleteJobs();
            foreach (TimerJobStatusAPI completeJob in completedJobs)
            {
                Console.WriteLine(string.Format("Job Name: {0}", completeJob.Name));
                Console.WriteLine(string.Format("Job Type: {1}", completeJob.Type));
                Console.WriteLine(string.Format("Job Started Time: {0}",
completeJob.Started));
                Console.WriteLine(string.Format("Job Server Name: {0}", completeJob.Name));
                Console.WriteLine(string.Format("Job Status: {0}", completeJob.Status));
                Console.WriteLine(string.Format("Job State: {0}", completeJob.State));
                Console.WriteLine(string.Format("Job Duration : {0}",
completeJob.ElapsedTime));
                Console.WriteLine(string.Format("Las error Job : {0}", completeJob.LastErr));
            }
        }

        public static void GetAbandonedJobsExample()
        {
            List<TimerJobStatusAPI> abandonedJbos = TimerJobStatusAPI.GetAbandonedJobs();
            foreach (TimerJobStatusAPI abandonedJob in abandonedJbos)
            {
                Console.WriteLine(string.Format("Job Name: {0}", abandonedJob.Name));
                Console.WriteLine(string.Format("Job Type: {1}", abandonedJob.Type));
                Console.WriteLine(string.Format("Job Started Time: {0}",
abandonedJob.Started));
                Console.WriteLine(string.Format("Job Server Name: {0}", abandonedJob.Name));
                Console.WriteLine(string.Format("Job Status: {0}", abandonedJob.Status));
                Console.WriteLine(string.Format("Job Error: {0}", abandonedJob.LastErr));
                Console.WriteLine(string.Format("Job Completion: {0}",
abandonedJob.Completion));
            }
        }

        public static void GetAbortingJobsExample()
        {
            List<TimerJobStatusAPI> abortedJobs = TimerJobStatusAPI.GetAbortingJobs();
            foreach (TimerJobStatusAPI abortJob in abortedJobs)
            {
                Console.WriteLine(string.Format("Job Name: {0}", abortJob.Name));
                Console.WriteLine(string.Format("Job Type: {1}", abortJob.Type));
                Console.WriteLine(string.Format("Job Started Time: {0}", abortJob.Started));
                Console.WriteLine(string.Format("Job Server Name: {0}", abortJob.Name));
                Console.WriteLine(string.Format("Job Status: {0}", abortJob.Status));
                Console.WriteLine(string.Format("Job State: {0}", abortJob.State));
                Console.WriteLine(string.Format("Job Error: {0}", abortJob.LastErr));
                Console.WriteLine(string.Format("Job Completion: {0}", abortJob.Completion));
                Console.WriteLine(string.Format("Job Stopped Date: {0}", abortJob.Stopped));

            }
        }

        public static void GetAbortedJobsExample()
        {
            List<TimerJobStatusAPI> abortedJobs = TimerJobStatusAPI.GetAbortedJobs();
            foreach (TimerJobStatusAPI abortedJob in abortedJobs)
            {
                Console.WriteLine(string.Format("Job Name: {0}", abortedJob.Name));
                Console.WriteLine(string.Format("Job Type: {1}", abortedJob.Type));
```

```
                Console.WriteLine(string.Format("Job Started Time: {0}",
abortedJob.Started));
                Console.WriteLine(string.Format("Job Server Name: {0}", abortedJob.Name));
                Console.WriteLine(string.Format("Job Status: {0}", abortedJob.Status));
                Console.WriteLine(string.Format("Job State: {0}", abortedJob.State));
                Console.WriteLine(string.Format("Job Stopped Date: {0}",
abortedJob.Stopped));
            }
        }

        public static void GetSuspendingJobsExample()
        {
            List<TimerJobStatusAPI> supendingJobs = TimerJobStatusAPI.GetSuspendingJobs();
            foreach (TimerJobStatusAPI suspendingJob in supendingJobs)
            {
                Console.WriteLine(string.Format("Job Name: {0}", suspendingJob.Name));
                Console.WriteLine(string.Format("Job Type: {1}", suspendingJob.Type));
                Console.WriteLine(string.Format("Job Started Time: {0}",
suspendingJob.Started));
                Console.WriteLine(string.Format("Job Server Name: {0}", suspendingJob.Name));
                Console.WriteLine(string.Format("Job Status: {0}", suspendingJob.Status));
                Console.WriteLine(string.Format("Job State: {0}", suspendingJob.State));
                Console.WriteLine(string.Format("Job Stopped Date: {0}",
suspendingJob.Stopped));
                Console.WriteLine(string.Format("Job ElapsedTime Date: {0}",
suspendingJob.ElapsedTime));
            }
        }

        public static void GetSuspendedJobsExample()
        {
            List<TimerJobStatusAPI> suspendedJobs = TimerJobStatusAPI.GetSuspendedJobs();
            foreach (TimerJobStatusAPI suspendedJob in suspendedJobs)
            {
                Console.WriteLine(string.Format("Job Name: {0}", suspendedJob.Name));
                Console.WriteLine(string.Format("Job Type: {1}", suspendedJob.Type));
                Console.WriteLine(string.Format("Job Started Time: {0}",
suspendedJob.Started));
                Console.WriteLine(string.Format("Job Server Name: {0}", suspendedJob.Name));
                Console.WriteLine(string.Format("Job Status: {0}", suspendedJob.Status));
                Console.WriteLine(string.Format("Job State: {0}", suspendedJob.State));
            }
        }

        public static void GetResumingJobsExample()
        {
            List<TimerJobStatusAPI> runningJobs = TimerJobStatusAPI.GetResumingJobs();
            foreach (TimerJobStatusAPI runningJob in runningJobs)
            {
                Console.WriteLine(string.Format("Job Name: {0}", runningJob.Name));
                Console.WriteLine(string.Format("Job Type: {1}", runningJob.Type));
                Console.WriteLine(string.Format("Job Started Time: {0}",
runningJob.Started));
                Console.WriteLine(string.Format("Job Server Name: {0}", runningJob.Name));
                Console.WriteLine(string.Format("Job Status: {0}", runningJob.Status));
                Console.WriteLine(string.Format("Job State: {0}", runningJob.State));

            }
        }

        public static void GetJobsWithExceptionExample()
        {
            List<TimerJobStatusAPI> jobWithException = TimerJobStatusAPI.GetExceptionJobs();
            foreach (TimerJobStatusAPI jobWithError in jobWithException)
```

```
            {
                Console.WriteLine(string.Format("Job Name: {0}", jobWithError.Name));
                Console.WriteLine(string.Format("Job Type: {1}", jobWithError.Type));
                Console.WriteLine(string.Format("Job Started Time: {0}",
jobWithError.Started));
                Console.WriteLine(string.Format("Job Server Name: {0}", jobWithError.Name));
                Console.WriteLine(string.Format("Job Status: {0}", jobWithError.Status));
                Console.WriteLine(string.Format("Job State: {0}", jobWithError.State));
                Console.WriteLine(jobWithError.Name);
            }
        }
    }
}
```

Blob Migration API

The BlobAPI object supports migrating content from one content store to another. This API is callable only from custom .net code such as a SharePoint workflow or event handler. This API should also only be used in a SharePoint context such as an event handler or workflow. Use outside of a SharePoint context is not supported.

BlobAPI Object Reference

Method Reference

The following methods are provided by the BlobAPI object:

Method Name	Parameters	Description
void MigrateBlob (SPListItem listItem, Guid targetProfileId)	*listItem* – An SPListItem representing the document to migrate. *targetProfileId* – Id of the profile to migrate document to	Migrates the document to the specified profile (represented by a Guid profile id).
void MigrateBlob(SPListItem listItem, string targetProfileId)	*listItem* – An SPListItem representing the document to migrate. *targetProfileId* – Id of the profile to migrate document to	Migrates the document to the specified profile. This overload of the method takes a string representation of the Guid profile id. String must still be in Guid form.

BlobAPI Object Reference Code Examples

The MigrateBlob method requires a profile id to represent the profile of the content store to migrate to. This can be obtained in couple of different ways:

- Create a No Scope profile in StoragePoint. The ID will be displayed by StoragePoint on the Profiles screen. No Scope profiles are designed specifically for use with the MigrateBlob API.
- Use the ProfileAPI class to look up a profile by site collection id or web app id and then obtain the ID from the ProfileAPI object (i.e. the ProfileId property).

The following is an example of using the BlobAPI object to migrate content from one profile to another.

```
BlobAPI blobAPI = new BlobAPI();
blobAPI.MigrateBlob(properties.ListItem,
                    new Guid("FBCA353D-2BF9-4d50-8827-D0760420D86D"));
```

The following is an example of using the BlobAPI object to retrieve large file size in Kb.

```
// Method to validate GetLargeFileSize it returns LargeFile (kbs) size in case exists, or -1
other case.

            var _docUrl = "http://win-710b6vpt0vj:1111/Shared%20Documents/alas%20de%
20ensueño.flv.aspx";    //Url to the large file document
            using (Microsoft.SharePoint.SPSite site = new
Microsoft.SharePoint.SPSite(_docUrl))
            {
                using (Microsoft.SharePoint.SPWeb web = site.OpenWeb())
                {
                    Microsoft.SharePoint.SPListItem item = web.GetListItem(_docUrl);
                    BlobAPI blob = new BlobAPI();
                    Console.WriteLine(blob.GetLargeFileSize(item));
                }
            }
```

Blob Reference API

The Blob Reference API exposes the ability to determine the BLOB file corresponding to a given SharePoint document. This API exposes functionality similar to the Details screen in StoragePoint.

The BlobReferenceAPI object represents an external BLOB file on the blob store. There is a method for obtaining all of the blob references for a given SharePoint document (by url). This API must be called on a server in the SharePoint farm and the user running the code must have administrative rights in SharePoint.

BlobReferenceAPI Object Reference

Method Reference

The following methods are provided by the BlobReferenceAPI object:

Method Name	Parameters	Description
static List<BlobReferenceAPI> GetBlobRefsForDoc(string DocUrl)	*DocUrl* – An absolute URL representing a document in SharePoint	Given an absolute URL representing a document, this method returns a list of BlobReferenceAPI objects representing all of the external BLOB files associated with this document. A list is returned because a given document can have multiple versions and multiple publishing states (published, checked out copy, etc.). The DocUrl parameter is an absolute URL respresenting a document in SharePoint. An example is: http://server/sites/web1/Pages/default.aspx

Property Reference

The BlobReferenceAPI object has many properties that provide information about the exposed BLOB file.

Property Name	Property Data Type	Description
FilePath	string	Returns the path to the BLOB file. If the file was externalized using the FileSystem adapter, this will be a complete path (ex. \\filer\filestore\folder\folder\filename). For other adapters, this will be only the relative folder + the filename of the file.
Folder	string	The folder on the file store containing the file. For the FileSystem adapter this will include the absolute path from the connection string of the profile.
BlobId	string	The filename of the BLOB file.
BlobSize	Int	The uncompressed size of the original file. This will differ from the actual file on the file store if compression is enabled on the profile.
ProfileId	String	The ID of the profile the file was externalized under.

Profile	ProfileAPI object	A ProfileAPI object representing the profile the file was externalized under.
DocUrl	string	The absolute URL of the doc that the BLOB file belongs to.
DocId	Guid	The internal SharePoint ID of the doc that the BLOB file belongs to.
DocType	RefDocType	Can be Doc if the BLOB file belongs to a document or Version if the BLOB file belongs to a version of the document.
DocLevel	RefDocLevel	Documents in SharePoint can have multiple publishing states. Values for this property include Published for a published doc, Draft for a draft copy and Checkout for a working version of a checked out document.
Version	string	The version of the doc or version that the BLOB file is attached to. For example, 1.5 would mean that the BLOB file belongs to version 1.5 of the document.

BlobReferenceAPI Code Example

The static GetBlobRefsForDoc method of the BlobReferenceAPI class will return all of the external BLOB files associated with a given document URL in SharePoint. Since SharePoint can have multiple publishing versions and regular versions associated with a single document URL, this method returns a list of all of the relevant BLOBs

The following is an example of using the BlobReferenceAPI object to retrieve all of the BLOB file references associated with an URL and iterating through the results :

```
        List<BlobReferenceAPI> blobRefs =
        BlobReferenceAPI.GetBlobRefsForDoc("http://site1/web1/lib/file.docx");

foreach (BlobReferenceAPI blobRef in blobRefs)
{
  MessageBox.Show("Blob Found: " + blobRef.DocType.ToString() +
    ", Size=" + blobRef.BlobSize +
    ", Version: " + blobRef.Version +
    ", Publishing Level: " + blobRef.DocLevel.ToString() +
    ", File Path: " + blobRef.FilePath);
}
```

Upload Large File API

The Large File Upload API exposes the ability to upload a large file, similar to the Large File Upload interface on the SharePoint document library ribbon. Only one file at a time can be uploaded with this method. Large File Support must be enabled on the SharePoint farm, on StoragePoint General Settings. An active Storage Profile must exist that covers the destination of the uploaded file.

This API must be called on a server in the SharePoint farm and the user running the code must have administrative rights in SharePoint.

UploadLargeFileAPI Object Reference

Method Reference

The following method is provided by the UploadLargeFileAPI object:

Method Name	Parameters	Description
void PerformUploadLargeFile(string fullFilename, string listUrl)	*ListURL* - Destination URL of the Document Library where the file will be uploaded. An example is: http://server/sites/web1/Pages/default.aspx *FullFileName* - Full UNC path of the file to be uploaded	Uploads a file larger than the Web Application upload limit to a document library and externalizes that large file to the profile endpoint. An aspx file is placed in the document library that points to the large file.

Property Reference

The UploadLargeFileAPI object has properties that can be set to track the upload and log the progress.

Property Name	Data Type	Required	Description
WrapLogger	log	No	Can be used to set up logging of the large file upload. `void SetLoggers(Action <string> Info, Action<string> Error)`

Action	TrackProgress	No	Delegate that could be used for tracking the upload progress, the delegate receive two parameters: bytes already uploaded, and total bytes to be uploaded. `Action<long, long>`

Upload Large File API Code Example

The LargeFileUploaderAPI allows a way to upload large files (those greater than the max upload limit for the Web Application) into a SharePoint Document Library, where a StoragePoint profile is configured.

Basic example, without logs and tracking set:

```
static void Main(string[] args)
{
   var listUrl = "http://server:6003/Shared%20Documents";
   var fullFilename = @"c:\Uploads\smFile10M.txt";
   LargeFileUploaderAPI uploader = new LargeFileUploaderAPI();
   uploader.PerformUploadLargeFile(fullFilename, listUrl);
}
```

Example with logs set:

```
static void Main(string[] args)
{
   LargeFileUploaderAPI uploader = new LargeFileUploaderAPI();
   uploader.log.SetLoggers(Console.WriteLine, Console.WriteLine);
   var listuri = "http://server:3265/sites/siteCollection/Shared%20Documents/Forms/AllItems. aspx?
     RootFolder=%2Fsites%2FsiteCollection%2FShared%20Documents%
     2FFolderTest&FolderCTID=0x012000A5FC1D1D7E870B4A97A9E2704833E271&View=%
     7B835F52A6%2D5514%2D45AD%2D901B%2D8CC94564B3E5%7D";
   var file = @"C: \documents\windows-8-start-screen. png";
   uploader.PerformUploadLargeFile(file, listuri);
}
```

Example with logs and tracking usage:

```
static void Main(string[] args)
{
   var listUrl = "http://server:6003/Shared%20Documents";
   var fullFilename = @"c:\Uploads\smFile10M.txt";
   LargeFileUploaderAPI uploader = new LargeFileUploaderAPI();
   uploader.log.SetLoggers(Console.WriteLine, Console.WriteLine);
   uploader.TrackProgress = (long bytesSent, long total) =>
   {
      var percentage = total > 0 ? ((float)bytesSent / (float)total) * 100 : 0;
      Console.WriteLine("{0} -- {1} :  {2}%", bytesSent, total, percentage);
   };
   uploader.PerformUploadLargeFile(fullFilename, listUrl);
}
```

Validator API

The ValidatorAPI object can be used to verify the identities of endpoints and profiles is correct, before executing commands using that information.

Method Name	Parameters	Description
EndpointIdentityValidator (string NameOrID)	*NameORID* - The endpoint name or ID to be validated.	Verifies that the name or ID is valid.
ProfileIdentityValidator (string profileNameOrID	*NameORID* - The profile name or ID to be validated.	Verifies that the name or ID is valid.

Validator API Examples

Example:

```
using Bluethread.SharePoint.StoragePoint;
using Bluethread.SharePoint.StoragePoint.ValidatorAPI;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace StoragePointAPI
{
    public class ValidatorExamples
    {
        public static void ValidatorProfile()
        {
            EndpointIdentityValidator endPointValidator = new
EndpointIdentityValidator("EP1");
            Guid endPointId;
            endPointValidator.IsValidString(out endPointId); // return true and the Guid of
Endpoint if it exists inside STP database in other case it throws a exception
        }

        public static void ValidatorEndpoint()
        {
            ProfileIdentityValidator profileValidator = new
ProfileIdentityValidator("SharePoint-15548");
            Guid profileId;
            profileValidator.IsValidString(out profileId); // return true and the Guid of
Profile if it exists inside STP database in other case it throws a exception

        }
       public static void ValidatorEndpointNameExample()
        {
            EndpointIdentityValidator endPointValidator = new
EndpointIdentityValidator("EP1");
```

```
            Guid endpointId;
            if (endPointValidator.IsValidString(out endpointId))// return true and the Guid
of Endpoint if it exists inside STP database in other case it throws a exception
            {
                EndpointAPI EP1 = new EndpointAPI(endpointId);
                Console.WriteLine(string.Format("Endpoint AdapterName : {0} Connection String
:{1} Endpoint Status: {2} , ", EP1.AdapterName.ToString(), EP1.Connection, EP1.Status));
            }
        }

        public static void ValidatorProfileNameExample()
        {
            ProfileIdentityValidator profileValidator = new
ProfileIdentityValidator("SharePoint-15548");
            Guid profileId;
            if (profileValidator.IsValidString(out profileId))// return true and the Guid of
Profile if it exists inside STP database in other case it throws a exception
            {
                ProfileAPI webbApp_15548 = new ProfileAPI(profileId);
                Console.WriteLine(string.Format("Profile Type: {0}   Retain Unused BLOBs:
{1}", webbApp_15548.Type.ToString(), webbApp_15548.BlobRetentionDays));
            }
        }


        public static void ValidatorEndpointGuidExample()
        {
            EndpointIdentityValidator endPointValidator = new
EndpointIdentityValidator("51F544CA-A32A-4DC4-ADE5-D8BA60DC5638");
            Guid endpointId;
            if (endPointValidator.IsValidString(out endpointId))// return true and the Guid
of Endpoint if it exists inside STP database in other case it throws a exception
            {
                EndpointAPI endpointInstance = new EndpointAPI(endpointId);
                Console.WriteLine(string.Format("Endpoint Name: {0} has Compression: {1}, has
Encryption: {2} ", endpointInstance.Name, endpointInstance.UseCompression,
endpointInstance.UseEncryption));
            }

        }

        public static void ValidatorProfileGuidExample()
        {
            ProfileIdentityValidator profileValidator = new
ProfileIdentityValidator("4CF245C8-0C4B-4344-A4AB-0578C64C6503");
            Guid profileId;
            if (profileValidator.IsValidString(out profileId)) // return true and the Guid of
Profile if it exists inside STP database in other case it throws a exception
            {
                ProfileAPI profileInstance = new ProfileAPI(profileId);
                string backupEndpoint =  profileInstance.BackupEndpoint == null? "None" :
profileInstance.BackupEndpoint.Name;
                string holdEndpoint = profileInstance.HMOnHoldEndpoint == null? "None" :
profileInstance.HMOnHoldEndpoint.Name;
                string recordEndpoint = profileInstance.RMDeclaredRecordEndpoint == null ?
"None" : profileInstance.RMDeclaredRecordEndpoint.Name;
                Console.WriteLine(string.Format("Profile Name: {0}, Profile Scope: {1},
Profile Type: {2}", profileInstance.Name, profileInstance.ScopeId,
profileInstance.Type.ToString()));
                Console.WriteLine(string.Format("Endpoint Backup: {0}, Endpoint Hold: {1}
Endpoint Record: {2}", backupEndpoint, holdEndpoint, recordEndpoint));

                foreach (ProfileEndpointAPI profileEndpoint in
profileInstance.ProfileEndpoints)
```

```
                {
                    Console.WriteLine(string.Format("Endpoint Name: {0}, Endpoint Status:
{1}, Endpoint Adapter: {2} , Endpoint Connection string: {3}",
                        profileEndpoint.Name, profileEndpoint.Status,
profileEndpoint.AdapterName, profileEndpoint.Connection));
                }

            }
        }
    }
}
```

StoragePoint APIs in Visual Studio

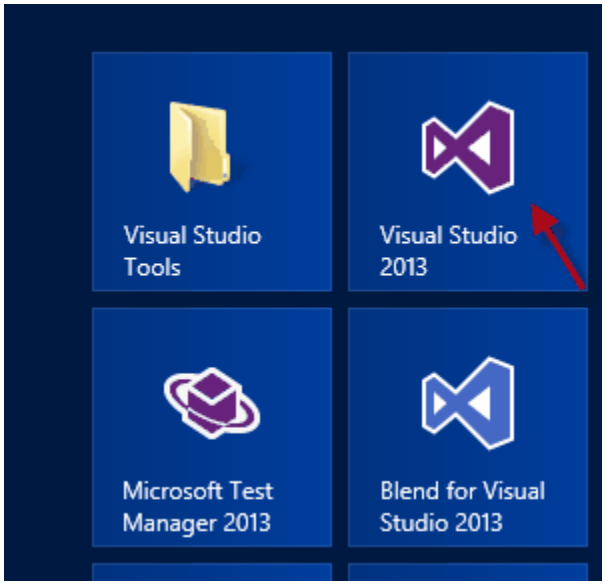
The following steps will help you to create a project in Visual Studio and manage the StoragePoint APIs.

Prerequisites

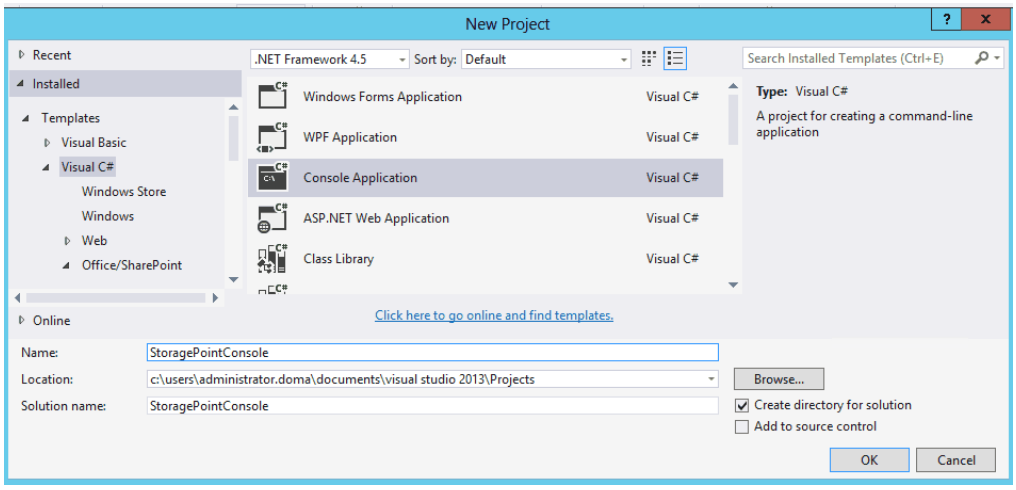
- StoragePoint installed.
- Visual Studio 2013 installed on the WFE where SharePoint is hosted.
- See other prerequisites for each API.

Steps

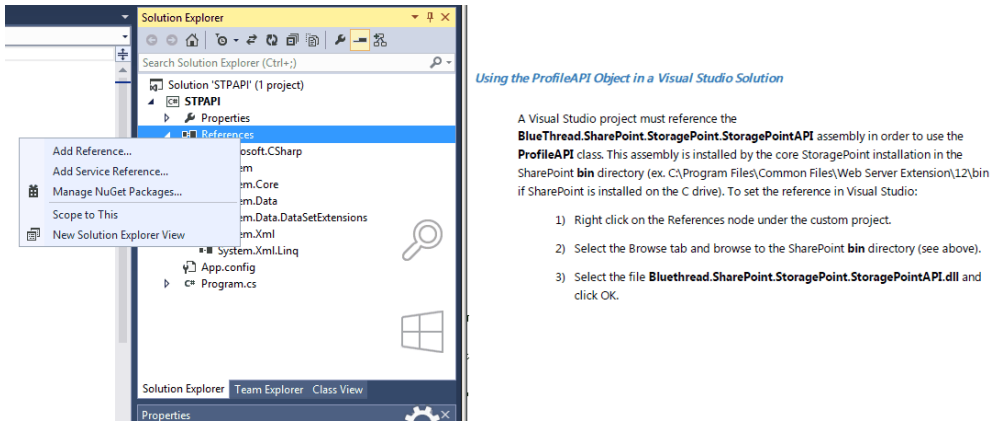
1. Open Visual Studio.



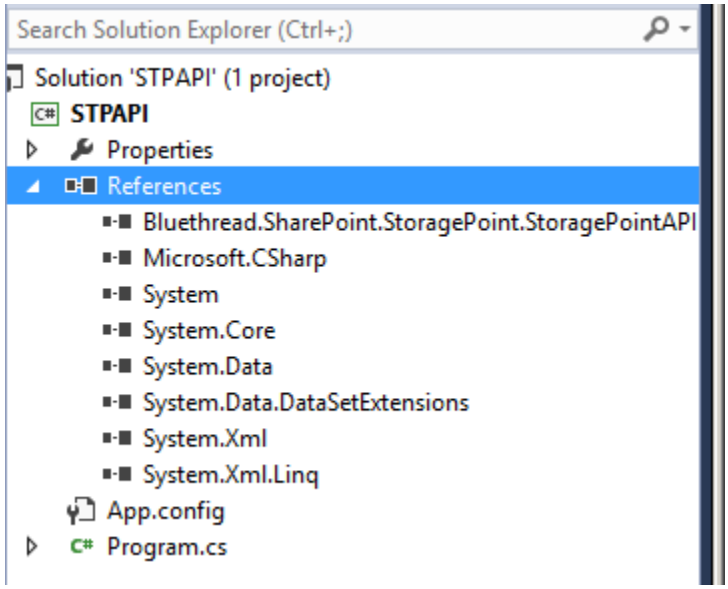
2. File >Create New Project>C# Console.



3. Go to Project>Add Reference and select BlueThread.SharePoint.StoragePointAPI assembly.



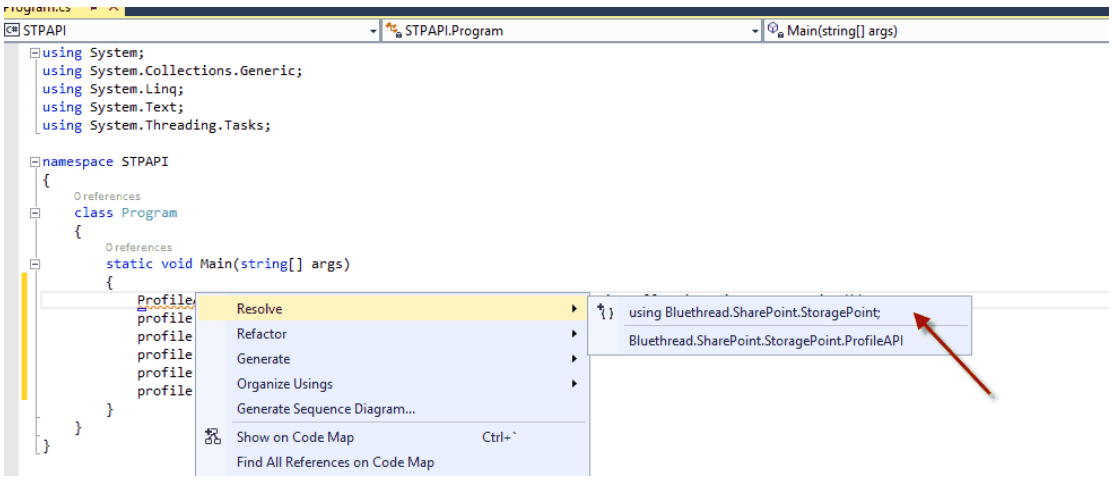
As a result the following reference should be listed



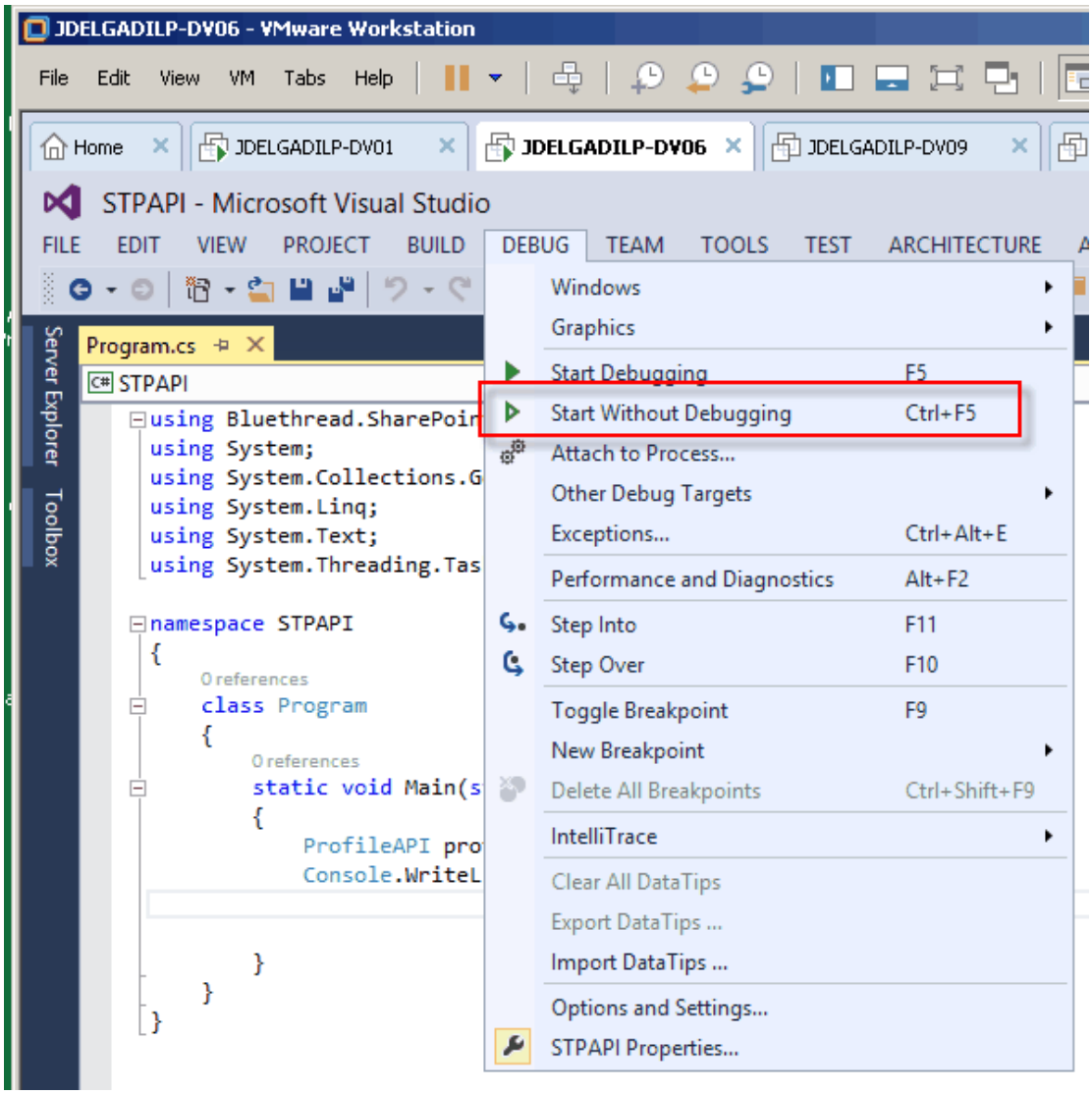
4. Copy the needed code into the c# class

```
using Bluethread.SharePoint.StoragePoint;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace STPAPI
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            ProfileAPI profile = new ProfileAPI("ProfileContentDB");
            Console.WriteLine(profile.ProfileId.ToString());


        }
    }
}
```

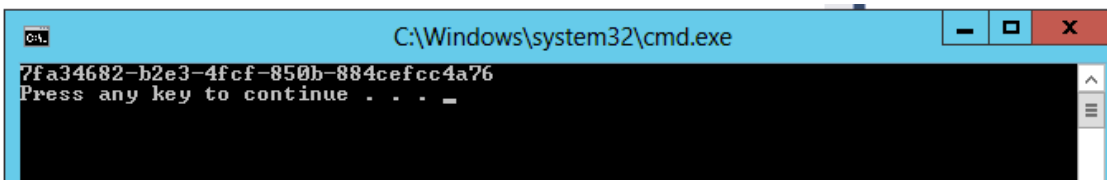
5. Resolve the libraries.



6. Save the changes and run the project.



7. As a result you will have the following results



Do the same with all the commands that are listed in this guide.

2

About Us

Quest creates software solutions that make the benefits of new technology real in an increasingly complex IT landscape. From database and systems management, to Active Directory and Office 365 management, and cyber security resilience, Quest helps customers solve their next IT challenge now. Around the globe, more than 130,000 companies and 95% of the Fortune 500 count on Quest to deliver proactive management and monitoring for the next enterprise initiative, find the next solution for complex Microsoft challenges and stay ahead of the next threat. Quest Software. Where next meets now. For more information, visit www.quest.com.

Contacting Quest

For sales or other inquiries, visit www.quest.com/contact.

Technical Support Resources

Technical support is available to Quest customers with a valid maintenance contract and customers who have trial versions. You can access the Quest Support Portal at https://support.quest.com

The Support Portal provides self-help tools you can use to solve problems quickly and independently, 24 hours a day, 365 days a year. The Support Portal enables you to:

- Submit and manage a Service Request
- View Knowledge Base articles
- Sign up for product notifications
- Download software and technical documentation
- View how-to-videos
- Engage in community discussions
- Chat with support engineers online
- View services to assist you with your product