File Service

FileService is a utility application that can be used to remotely manage files in a Windows server-based folder via a RESTful Web Service API.

The service runs as a Windows Service and, by default, can be accessed at the following URL:

http://<server_name_or_ip>:8080/FileService

The service supports the following operations:

- Ping
- List all files
- Check if a file exists
- Download a text file
- Create a file
- Create a file (chunked)
- Update a file
- Delete a file

Synergy Licensing

FileService is a Windows service written in Synergy .NET and as such requires that Synergy/DE core components are installed and configured on the system, and requires the dedicated use of one Synergy/DE Windows runtime license (RUN10).

You should verify that these prerequisites are met BEFORE attempting to run the FileService installation.

Installation

The easiest way to install file service on a Windows system is to download the installer from the GitHub repository and then execute it on the desired system or systems system.

Because FileService runs as a Windows service, you must be logged into a user account that has administrative rights for the system in order to successfully run the installation.

The installer makes the following changes to the system:

1. Installs the FileService application code in C:\Program Files (x86)\Synergex\FileService
2. Creates a start menu folder containing shortcuts to the documentation and uninstall.
3. Registers a windows service named FileService and sets the service to start automatically when the system boots.
4. Starts the service.

Configuration

The service exposes the files in a single storage folder. By default that storage folder is C:\Users\Public\Documents\FileService.

You can change the location used by editing the configuration file C:\Program Files (x86)\Synergex\FileService\FileService.exe.config and specifying an alternate location via the **StorageFolder** setting.

The HTTP port that the server listens on can also be altered by changing the value of the **HttpListenerPort** setting.

```
<FileService.Properties.Settings>
   <setting name="StorageFolder" serializeAs="String">
      <value>C:\some\other\folder</value>
   </setting>
   <setting name="HttpListenerPort" serializeAs="String">
      <value>8080</value>
   </setting>
```

</FileService.Properties.Settings>

If you make any changes to this file then you must stop and restart the service.

Starting and Stopping the Service

The Windows service may be started and stopped in several ways:

1. From the Services tab in Task Manager.
2. From the Services control panel (services.msc).
3. From the command line

   net stop FileService

   net start FileService

License Agreement

COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Web Service Operations

The service supports the following operations:

Ping	GET	/FileService/ping
List all files	GET	/FileService
Check if a file exists	GET	/FileService/exists/filename.ext
Download a text file	GET	/FileService/text/filename.ext
Create a file	POST	/FileService/filename.ext
Update a file	PUT	/FileService/filename.ext
Delete a file	DELETE	/FileService/filename.ext

Ping the Server

This operation allows you to ping the server to verify that it is operating.

HTTP method	GET
URI	/FileService/ping
Request headers	Host: <server_dns_or_ip>[:<port>]
Request body	n/a
Response headers	Content-Length: 0 Date: <date_time> Server: Microsoft-HTTPAPI/2.0
Response body	None
HTTP result codes	HTTP 204 (no content) indicates a successful ping

Listing All Files

This operation allows you to retrieve a list of all of the files that currently exist in the servers storage folder.

HTTP method	GET
URI	/FileService
Request headers	Host: <server_dns_or_ip>[:<port>] Accept: <mime_type>
Request body	n/a
Response headers	Content-Length: <int> Content-Type: <mime_type> Date: <date_time> Server: Microsoft-HTTPAPI/2.0
Response body	A JSON or XML array containing the names of the files present in the server folder.
HTTP result code(s)	HTTP 200 (OK) indicates a successful response.

Response Data Format

The format of the returned data is determined by the value of the HTTP request Accept header that you pass.

Response Format	MIME Type
JSON	application/json
XML	application/xml

If you do not pass an Accept header then the default response will be a JSON array.

Checking if a File Exists

This operation allows you to determine if a file with a specified name exists in the servers storage folder.

HTTP method	GET
URI	/FileService/exists/<filename>
Request headers	Host: <server_dns_or_ip>[:<port>]
Request body	n/a
Response headers	Content-Length: 0 Date: <date_time> Server: Microsoft-HTTPAPI/2.0
Response body	None
HTTP result codes	HTTP 204 (no content) indicates that the file exists. HTTP 404 (not found) indicates that the file does not exist.

Download a Text File

This operation allows you to download a copy of a named file from the servers storage folder.

HTTP method	GET
URI	/FileService/text/<filename>
Request headers	Host: <server_dns_or_ip>[:<port>] Accept: <mime_type>
Request body	n/a
Response headers	Content-Length: <int> Content-Type: <mime_type> Date: <date_time> Server: Microsoft-HTTPAPI/2.0
Response body	A JSON or XML array of strings containing the records from the text file.
HTTP result codes	HTTP 200 (OK) that the file exists and its content was returned in the response body. HTTP 404 (not found) indicates that the file does not exist.

Response Data Format

The format of the returned data is determined by the value of the HTTP request Accept header that you pass.

Response Format	MIME Type
JSON	application/json
XML	application/xml

If you do not pass an Accept header then the default response will be a JSON array.

Create a File

This operation allows you to create a new named file in the servers storage folder. If the file already exists the operation will fail.

HTTP method	POST
URI	/FileService/<filename>
Request headers	Host: <server_dns_or_ip>[:<port>] Accept: <mime_type> Content-Length: <int> Content-Type: text/plain
Request body	A string containing the records from the text file.
Response headers	Content-Length: 0 Date: <date_time> Location: <url_to_retrieve_file> Server: Microsoft-HTTPAPI/2.0 ServerFileSpec: <file_spec>
Response body	None
HTTP result codes	HTTP 201 (created) indicates that the file was created. HTTP 403 (forbidden) indicates that the file already existed and was NOT updated.

Create a File (chunked)

This operation allows you to create a new named file in the servers storage folder by uploading the file in several "chunks" which are appended together on the server to make a single file. This is useful when uploading very large files that would otherwise not be possible to process because of memory or network constraints.

A chunked upload takes place during a series of at least two web service calls.

1. A mandatory call to start the chunked upload.
2. Optionally, any number of calls to continue the upload.
3. A mandatory call to finish the chunked upload.

Starting a Chunked Upload

This operation starts a new chunked upload. The first part of the files data is passed and is written to a new file on the server. If the file already exists then the operation will fail.

HTTP method	POST
URI	/FileService/chunked/start/<filename>
Request headers	Host: <server_dns_or_ip>[:<port>] Accept: <mime_type> Content-Length: <int> Content-Type: text/plain
Request body	A string containing the first set of records from the text file.
Response headers	Content-Length: <int> Date: <date_time> Server: Microsoft-HTTPAPI/2.0
Response body	None
	HTTP 204 (no content) indicates that the first part of the file was created.

HTTP result codes	HTTP 403 (forbidden) indicates that the file already existed and was NOT updated.

Continuing a Chunked Upload

This operation continues a chunked upload. The next part of the files data is passed and is appended to the existing file on the server.

HTTP method	POST
URI	/FileService/chunked/continue/<filename>
Request headers	Host: <server_dns_or_ip>[:<port>] Accept: <mime_type> Content-Length: <int> Content-Type: text/plain
Request body	A string containing the first set of records from the text file.
Response headers	Content-Length: 0 Date: <date_time> Server: Microsoft-HTTPAPI/2.0
Response body	None
HTTP result codes	HTTP 204 (no content) indicates that the first part of the file was created. HTTP 400 (bad request) indicates that the file does not exist on the server.

Finishing a Chunked Upload

This operation finishes a chunked upload. The final part of the files data is passed and is appended to the existing file on the server.

HTTP method	POST

URI	/FileService/chunked/finish/<filename>
Request headers	Host: <server_dns_or_ip>[:<port>] Accept: <mime_type> Content-Length: <int> Content-Type: text/plain
Request body	A string containing the first set of records from the text file.
Response headers	Content-Length: 0 Date: <date_time> Location: <url_to_retrieve_file> Server: Microsoft-HTTPAPI/2.0 ServerFileSpec: <file_spec>
Response body	None
HTTP result codes	HTTP 201 (created) indicates that the file was created. HTTP 403 (forbidden) indicates that the file already existed and was NOT updated.

Update a File

This operation allows you to update an existing file in the servers storage folder. If the file already exists it will be overwritten.

HTTP method	PUT
URI	/FileService/<filename>
Request headers	Host: <server_dns_or_ip>[:<port>] Content-Type: text/plain Content-Length: <length>
Request body	A string containing the records from the text file.
Response headers	Content-Length: 0 Date: <date_time> Server: Microsoft-HTTPAPI/2.0
Response body	None
HTTP result codes	HTTP 200 (OK) indicates that the file was created or updated.

Delete a File

This operation allows you to delete a file from the servers storage folder.

HTTP method	DELETE
URI	/FileService/<filename>
Request headers	Host: <server_dns_or_ip>[:<port>]
Request body	None
Response headers	Content-Length: 0 Date: <date_time> Server: Microsoft-HTTPAPI/2.0
Response body	None
HTTP result codes	HTTP 204 (no content) indicates that the file was deleted.

Change Log

V1.1

- Added chunked upload.
- The create a file operation was altered such that it no longer returns the server file spec in the response body. Rather it now follows REST best practices by returning the URL that can be used to retrieve the new file via the Location response header, and also returns the server file spec via the ServerFileSpec response header. The success status code was changed from 200 (OK) to 201 (created).
- Released 2/21/2018

V1.0

- Initial release with basic functionality.
- Released 2/20/2018