API Kit Usage Guidelines

For further information about this manual or other FileHold Systems products, contact us at 3010 Boundary Road Office #2H, Burnaby, BC, Canada V5M 4A1, via email sales@filehold.com, our website https://www.filehold.com, or call 604-734-5653.

Proprietary Notice

CONTENTS

1. API OVERVIEW

Access to the FileHold API is included with all FileHold licenses at no additional cost. It consists of a series of web services on the FileHold application server. In addition, a DLL is provided for dot net programs to simplify interfacing to the web services. There is a DLL for 32 bit applications and one for 64 bit applications.

The API is provided AS IS and is not directly supported under the FileCare program beyond what is supported when used by the standard clients. Users of the API must take appropriate precautions to ensure they do not lose any information. For a complete list of disclaimers and limitations of liability, please see the web site.

https://www.filehold.com/help/developer/web-services-api-overview

A help file is provided to give details on the API methods and parameters. The help does not describe how FileHold operates. A reader of the help file should have basic knowledge of operating FileHold before attempting to program with the API. The normal FileHold clients use the same API that is available to developers. A good rule of thumb for the capability of the API is to consider that if you can do something with the web client, you will also be able to do it with the API.

Three examples are provided in the API kit for using the API. Additional examples may be available from the FileHold blog or directly from FileHold.

- APIDemo – demonstrate the basic principles for creating API programs.
- BatchMetadataUpdater – Update metadata in FileHold from a CSV file. Demonstrates searching and updating metadata.
- BatchMetadataExtractor – Export metadata from FileHold to a CSV file. Demonstrates accessing saved searches and metadata from search results.

Further examples may be available on www.filehold.com and may include a variety of code snippets written in C Sharp or Powershell.

1.1. Client Connections

API programs must follow the same rules as any client for connecting to FileHold to ensure they have a correct license for using the system. For example, each user may have only one connection to the server at a time for any given client type in the standard connection pool. If your client seems to be disconnecting for no reason, check to make sure the same registered user is not being used in more than one place.

The “custom” client type must be used for API programs in your call to **StartSession**. You will be able to clearly see your API programs in the user activity report by looking for this client type.

FileHold has four connection pools: full, limited, Capture, and Sharepoint. API users can use the first two connection pools. If anonymous users are required the limited connection pool must be used. This is an optional feature with very low-cost connections. These users have a limited number of features available. They are slightly less powerful than a user with a read-only role.

The same limited registered user can have many connections to the limited pool. All activities will be logged for the same registered user regardless of who is actually executing web service.

After a session has been created, **SessionManager.CheckApiVersionAndLogClientInfo** should be called to update the user activity log. Although this call is not technically necessary it will help FileHold administrators to manage the system. Although the **clientVersion** parameter is a string, it must follow the Microsoft convention for version numbering as it will be cast to a **System.Version** object for comparison.

The compatibility check is largely for standard FileHold clients as they present a warning if the version of the client is lower than the server and an error if the client version is higher. There is no reason that a custom client could not span many FileHold server versions and continue to work correctly.

The **clientBuildNumber** and **clientAddress** are unvalidated but used in the user activity report. If there will be more than one type of custom client used in a system, it would be good if the build number contained something that would identify the specific client in use. The address is typically the IPv4 or IPv6 address.

Full registered users are allowed to have exactly one session per client type. If a start session call is made when a session already exists, the first session will be forced off. You can reuse an existing session at any time and you can check if it is valid by calling **IsSessionValid**. There is also the option to keep a session alive. Use this with caution as it will override the system administration settings for the inactivity timer. If it is known that the user will not need the session in the immediate future, they can be safely logged off. This will free up sessions from the concurrent session pool for other users. This can be helpful in scenarios where there is not a one-to-one relationship between full registered user and concurrent session licenses.

Avoid rapid closing and opening of sessions. Each of the four applications on the FileHold application server keep their own cache of valid sessions for efficiency reasons. The cache life is approximately 60 seconds, so a rapid start, stop, and restart of sessions for a single user could result in one application having an old cached session when the others have updated their caches. This could prevent a user's transaction from completing successfully.

1.2. Session management

When designing applications to create sessions in FileHold it is important to ensure a seamless user interface by avoiding scenarios where one session is started and then a second session is started before the user has finished interacting with FileHold in the first session. As mentioned above, FileHold will only allow a single session per user on a single type of client. This means starting a second session before the first completes would cause the first to fail.

The best way to minimize unnecessary session creation or potential multi-session conflicts is to cache the session id in the client and reuse it for all cases as long as it remains valid. The cache should contain the session id and its expected lifetime. When a session id is needed, it can be retrieved from the cache. If the expected lifetime has been exceeded the cache can be refreshed by verifying the id with **IsSessionValid**. If the session is no longer valid a new session can be created and added to the cache. This method minimizes the complexity and cost of maintaining the cache and avoiding making calls with likely invalid session ids.

By default, FileHold has a 30 minute inactivity timer. The local cache expected lifetime should be less than the inactivity timer. The minimum inactivity timer is 1 minute, so there should never be a reason for the expected lifetime of the cache to be less than 1 minute.

Regardless of how the session id cache is maintained, the application should be prepared for a http 401 error as this is theoretically possible at any time. As with other critical server errors such as an http 503 error, the best response for the application is to present the user with an error message and allow them to attempt the operation at a later time.

Deadlock exceptions are expected depending on the interactions of different clients in the multi-user system. A reasonable resolution is to retry the transaction after a random number of seconds of delay. If the deadlock errors do not resolve after a small number of retries, the client should consider abandoning the attempt and reporting an error.

1.3. Troubleshooting

- Incorrectly set permissions can cause functional errors that appear strange for a new API programmer. When first developing a program use a system administration role. When the basic functionality is in place you can downgrade the role as needed. Any errors that occur after the downgrade can be assumed to be related to the permission configuration instead of the client logic.
- If a function does not seem to work correctly, make sure you can perform the function using the FileHold web client. If you cannot do it with the standard web client, it will not work with the API. Make sure you choose the same user for your tests with the standard and custom client.
- Tracing the HTTP communication between the FileHold Desktop Client and the server can be very helpful in understanding how the normal FileHold clients use the API. A network sniffer like Wireshark can be used to capture the packets. The SOAP XML will be contained in the HTTP packets. Where practical to integrate into a communication stream, a tool like Fiddler is very effective for understanding issues at a higher level than Wireshark.
- If you are using the DLL provided, make sure you have included the correct bit size with your project. The 32 bit DLL will only work with a 32 bit project, etc.

2. APPLICATION SERVER INTEGRATION SCRIPTS

Most API usage is client based, however, there are a couple of scenarios where end users can integrate custom API programs into the application server. This code is unlike other API programs. It is only intended for the purpose of selecting the library location where documents will be filed, but it runs in the context of the application server itself so high attention should be paid to safe coding practices and correct code. Poorly created code could effectively disable the application server, damage your data or open you to security risks.

2.1. Auto-Filing Scripts

Auto-filing scripts allow automating the selection of a destination storage location in the library. With the introduction of auto-filing templates there is less need for auto-filing scripts for typical cases, but there are still many cases not possible with an auto-filing template. There are five example auto-filing scripts included with FileHold. The source for each is provided with the API kit.

Documentation for using and customizing auto-filing scripts is available on the web site.

https://www.filehold.com/help/developer/auto-filing-scripts

2.2. Export Scripts

Export scripts provide a way to integrate custom data export logic into the application server. These scripts are a combination of executable code in DLL and administrative configuration. They can be selected for use inside of workflow activities. For example, at the end of an invoice approval workflow it is necessary to send the invoice details to an ERP system for payment.

Documentation for creating export scripts was is not available at the time of publishing, but the www.filehold.com will be updated to include this information.