[MS-FSSAC]:
Search Authorization Connector Protocol Specification

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft Open Specification Promise or the Community Promise. If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
11/06/2009	0.1	Major	Initial Availability
02/19/2010	1.0	Major	Updated and revised the technical content
03/31/2010	1.01	Editorial	Revised and edited the technical content
04/30/2010	1.02	Editorial	Revised and edited the technical content
06/07/2010	1.03	Editorial	Revised and edited the technical content
06/29/2010	1.04	Editorial	Changed language and formatting in the technical content.
07/23/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
09/27/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
11/15/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
12/17/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
03/18/2011	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
06/10/2011	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
01/20/2012	1.5	Minor	Clarified the meaning of the technical content.
04/11/2012	1.5	No change	No changes to the meaning, language, or formatting of the technical content.
07/16/2012	1.5	No change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This document specifies the Search Authorization Connector Protocol used between a protocol client and the protocol server, which is the FSA Manager Service. This protocol enables a protocol client to make changes to the users, groups, and user-to-user mappings stored on the protocol server.

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Sections 1.5 and 1.9 are also normative but cannot contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Active Directory
Domain Name System (DNS)
GUID

The following terms are defined in [MS-OFCGLOS]:

local cache user store
managed property
principal aliaser
principal aliaser identifier
principal aliasing
security principal identifier
user security filter
user store
user store identifier
XML principal aliaser

The following terms are specific to this document:

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

References to Microsoft Open Specifications documentation do not include a publishing year because links are to the latest version of the technical documents, which are updated frequently. References to other documents include a publishing year when one is available.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[MS-FSSACFG] Microsoft Corporation, "Search Authorization Configuration File Format".

[MS-FSSADFF] Microsoft Corporation, "Search Authorization Data File Format".

[MS-FSSAS] Microsoft Corporation, "Search Authorization Synchronization Protocol Specification".

[RFC1952] Deutsch, P., "GZIP file format specification version 4.3", RFC 1952, May 1996, http://www.rfc-editor.org/rfc/rfc1952.txt

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

[RFC2616] Fielding, R., Gettys, J., Mogul, J., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.ietf.org/rfc/rfc2616.txt

[XMLSCHEMA1] Thompson, H.S., Ed., Beech, D., Ed., Maloney, M., Ed., and Mendelsohn, N., Ed., "XML Schema Part 1: Structures", W3C Recommendation, May 2001, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

1.2.2 Informative References

[ISO/IEC-29500-4] International Organization for Standardization, "Information technology -- Document description and processing languages -- Office Open XML File Formats -- Part 4: Transitional Migration Features", ISO/IEC 29500-4:2008, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51462

[MS-FSO] Microsoft Corporation, "FAST Search System Overview".

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

[MS-OFCGLOS] Microsoft Corporation, "Microsoft Office Master Glossary".

1.3 Protocol Overview (Synopsis)

Search authorization ensures that only secure search results are returned. This means that search results are predicated on the querying user or user group's authorization to receive that particular information.

Secure search has two phases. Initially, content repositories are traversed to create indexes. Authorization **managed properties** that identify the users and groups that are granted or denied access are added to the indexes for each item.

Subsequently, a query sent by a user, the indexes quickly identify search results; secure search consisting of authenticating the user and the Query and Result Service rewriting the user's query by intersecting the user's original query with the user's **user security filter** so that the indexes return only authorized search results. That is, the user's security filter uses the authorization managed properties to limit the query results to items the user is entitled to see.

To generate the user's security filter, the FSA worker component needs to know of which groups the user is a member. For users in a **local cache user store**, the FSA worker component uses that **user store** for the groups. Protocol clients use this protocol to communicate with the FSA Manager Service to change the users and groups of a local cache user store, and the FSA Manager Service in turn disseminates this information to each FSA worker component.

Some users have identities in multiple user stores. For example, an **Active Directory** user may have a corresponding account under Lotus Notes. To generate the user's security filter, the FSA worker component requires the user's identities and groups in all user stores. Because the **security principal identifier** may not be the same in all user stores, the FSA worker component uses **principal aliasing** to map users and groups from one user store into another. Protocol clients use

this protocol to communicate with the FSA Manager Service to change the mappings in an **XML principal aliaser**.

For whichever reason a protocol client communicates with the protocol server, the FSA Manager Service administers the FSA worker components ensuring that each processing node of the Query and Result Service has the correct configuration, as described in [MS-FSO]. The protocol server responds to the protocol client with a **GUID** that is used to in subsequent requests to query the completion status of the initial request, as shown in the following diagram.

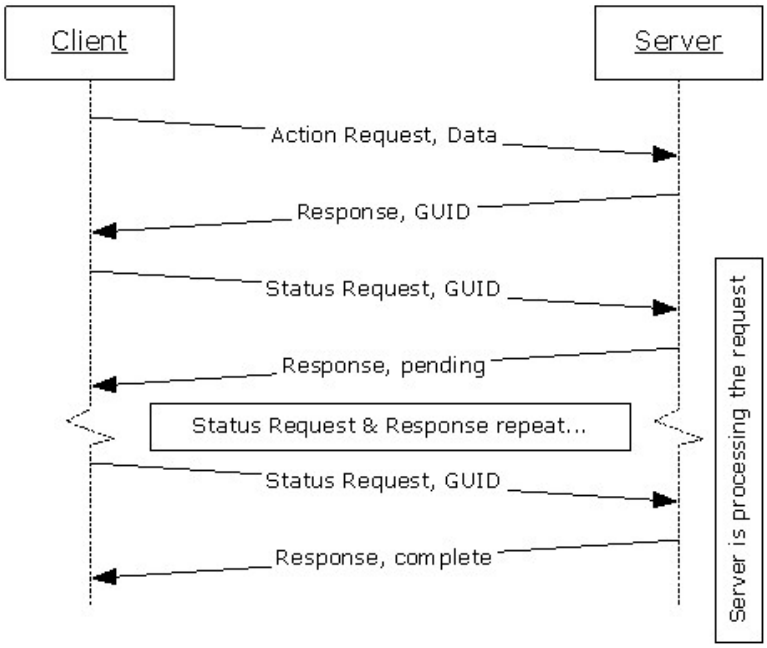

Figure 1: Search Authorization Connector Protocol request/response

1.4 Relationship to Other Protocols

This protocol uses the HTTP protocol to transmit messages as described in [RFC2616].

The following diagram shows the underlying messaging and transport stack that this protocol uses:

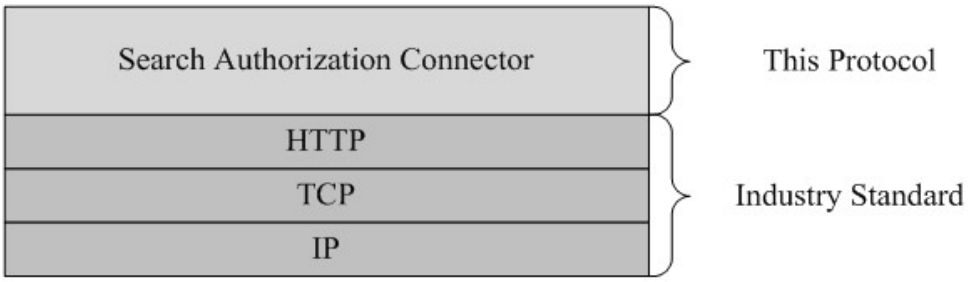

Figure 2: This protocol in relation to other protocols

1.5 Prerequisites/Preconditions

The protocol client has the address, both the URL and port number, of the protocol server.

The protocol client has both the user store identifiers of the user stores it modifies using this protocol, and the **principal aliaser** identifiers of the principal aliasers it modifies using this protocol.

The protocol server only accepts requests from authenticated protocol clients.

1.6 Applicability Statement

This protocol is appropriate for changing the user and group information in a local cache user store, and for changing the user mappings in an XML principal aliaser.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

All messages in this protocol are transported over HTTP as specified in [RFC2616].

2.2 Message Syntax

The messages in this protocol are HTTP requests and responses. The URL path name determines the message type. The path name MUST match the following:

```
/id/requestType/
```

The trailing slash (/) is optional.

The id is replaced with a **user store identifier** or a **principal aliaser identifier** as specified in sections 2.2.1 through 2.2.4.

If the URL path name does not match or requestType is not replaced with one of the requests as specified in sections 2.2.1 through 2.2.4, the message is malformed.

All of the messages MUST meet the following requirements:

- In the requests that follow, *serverAndPort* MUST be replaced with the **DNS** name or IP address and port number of the protocol server as specified in [RFC2616] section 3.2.2.
- The **Content-Type** header MUST be `text/xml; charset="UTF-8"`. If the **Content-Type** header is different, then the message is malformed. The **Content-Type** header is specified in [RFC2616] section 14.17.
- The Content-Encoding header MUST be "UTF-8" or "gzip" or MUST NOT be set. If the **Content-Encoding** header is not set, then it is assumed to be "UTF-8". If the **Content-Encoding** header is "gzip", then the payload MUST be compressed as specified in [RFC1952]. If the **Content-Encoding** header is "gzip", but the payload cannot successfully be decompressed, then the message is malformed. The **Content-Encoding** header is specified in [RFC2616] section 14.11.
- The request-method MUST be "POST"; otherwise the message is malformed. The **POST** request method is specified in [RFC2616] section 9.5.
- All headers and their values MUST be according to HTTP as specified in [RFC2616] section 4.2.
- If the protocol server is not able to process a request or the message is malformed, the protocol server MUST return an HTTP response, the status code of which does not have a value of 200. HTTP status codes are specified in [RFC2616] section 6.1.1.
- If the protocol server is able to process a request, the protocol server MUST return an HTTP response, the status code of which has a value of 200. HTTP status codes are specified in [RFC2616] section 6.1.1.

2.2.1 Request: uploadusercomplete

This request initiates processing of the request payload on the protocol server and returns a GUID, as specified in section 2.2.6, identifying this **upload user payload processing**. The GUID is used with the **uploaduserstatus** request, specified in section 2.2.3, to determine the processing status of this request. The URL format is as follows:

URL: http://*serverAndPort*/*userStoreId*/uploadusercomplete

The *userStoreId* in the URL path name MUST be the user store identifier of an existing user store on the protocol server.

The uncompressed payload MUST be a Local Cache Upload User file, as specified in [MS-FSSADFF] section 2.1.

The response is specified in section 2.2.6 or section 2.2.8.

The upload user payload processing replaces the information of the local cache user store given by **userStoreId** with the uncompressed payload.

The following figure depicts the relationships between the **upload user** messages and the Local Cache User files. The local cache user upload file and local cache user store file are specified in [MS-FSSADFF] sections 2.1 and 2.2. The Search Authorization Synchronization **TransferFile** method is specified in [MS-FSSAS] section 3.1.4.27.

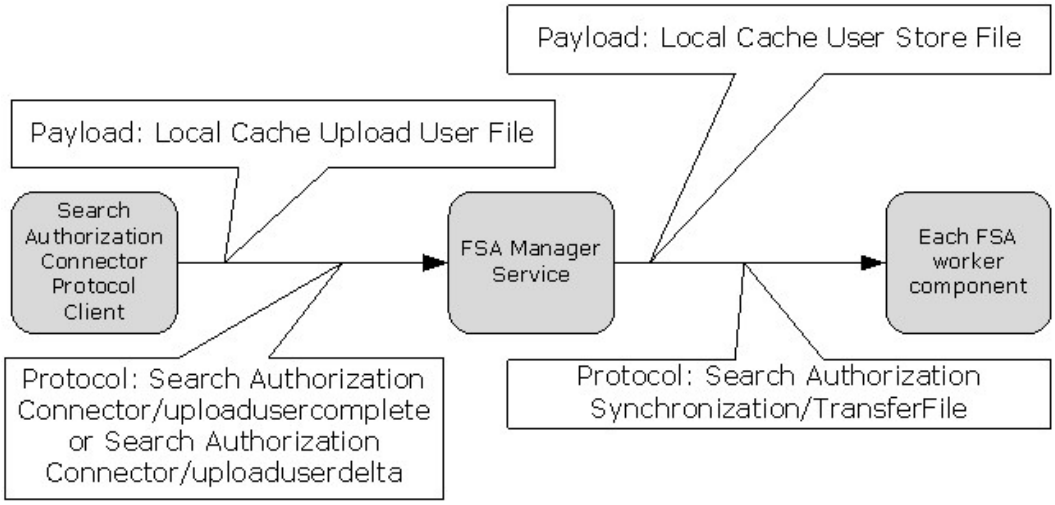

Figure 3: Local cache upload relationships

2.2.2 Request: uploaduserdelta

This request initiates processing of the request payload on the protocol server and returns a GUID, as specified in section 2.2.6, identifying this **upload user payload processing**. The GUID is used with the **uploaduserstatus** request, specified in section 2.2.3, to determine the status of this processing. The URL format is as follows:

URL: http://*serverAndPort*/*userStoreId*/uploaduserdelta

The *userStoreId* in the URL path name MUST be the user store identifier of an existing user store on the protocol server.

The uncompressed payload MUST be a Local Cache Upload User file as specified in [MS-FSSADFF] section 2.1.

The response is specified in section 2.2.7 or section 2.2.8.

The upload user payload processing applies changes specified in the uncompressed payload to the information of the local cache user store given by **userStoreId**.

2.2.3 Request: uploaduserstatus

This request returns the status of the upload user payload processing that was initiated by a previous **uploadusercomplete** request, as specified in section 2.2.1, or **uploaduserdelta** request, as specified in section 2.2.2. The URL format is as follows:

URL: http://*serverAndPort*/*userStoreId*/uploaduserstatus

The payload MUST be an XML document matching the following, *theGuid* of which is replaced by the GUID from the status response, as specified in section 2.2.7, of the previous **uploadusercomplete** or **uploaduserdelta** request.

```
<statusguid>theGuid</statusguid>
```

The **userStoreId** in the URL path name MUST match the **userStoreId** of the request that provided the GUID.

The response is specified in section 2.2.7 or section 2.2.8.

2.2.4 Request: uploadmappingfile

This request initiates processing of the request payload on the protocol server and returns a GUID, as specified in section 2.2.6, identifying this **upload mapping payload processing**. The GUID is used with the **uploadmappingstatus** request, specified in section 2.2.4, to determine the status of this processing. The URL format is as follows:

URL: http://*serverAndPort*/*principalAliaserId*/uploadmappingfile

The *principalAliaserId* in the URL path name MUST be the principal aliaser identifier of an existing XML principal aliaser on the protocol server.

The uncompressed payload MUST be an XML Principal Aliaser Mapping file as specified in [MS-FSSADFF] section 2.3.

The response is specified in section 2.2.7 or section 2.2.8.

The upload mapping payload processing replaces the information of the XML principal aliaser signified by **principalAliaserId** with the uncompressed payload.

The following figure depicts the relationships between the upload mapping messages and the XML Aliaser files. The XML Principal Aliaser Mapping file is specified in [MS-FSSADFF] section 2.3. The Search Authorization Synchronization **TransferFile** method is specified in [MS-FSSAS] section 3.1.4.27.

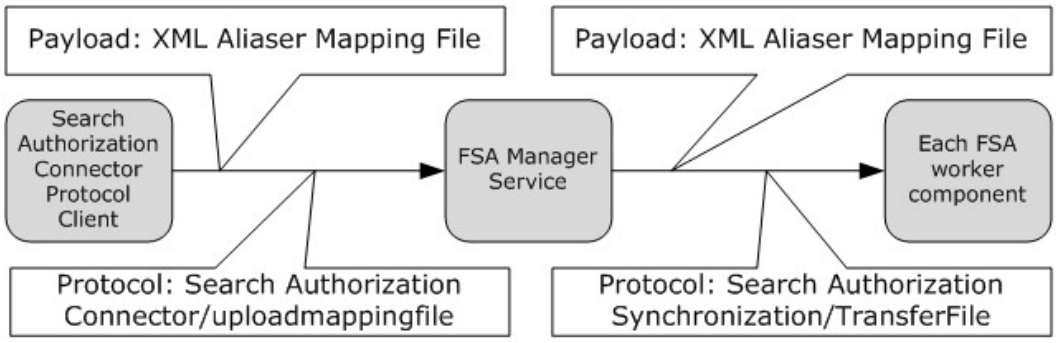

Figure 4: XML aliaser upload relationships

2.2.5 Request: uploadmappingstatus

This request returns the status of the upload mapping payload processing that was initiated by a previous **uploadmappingfile** as specified in section 3.2.5.4. The URL format is as follows:

URL: http://*serverAndPort*/*principalAliaserId*/uploadmappingstatus

The payload MUST be an XML document matching the following, *theGuid* of which is replaced by the GUID from the status response, as specified in section 2.2.7, of the previous **uploadmappingfile** request.

```
<statusguid>theGuid</statusguid>
```

The **principalAliaserId** in the URL path name MUST match the **principalAliaserId** of the request that provided the GUID.

The response is specified in section 2.2.7 or section 2.2.8.

2.2.6 Response: GUID

The **GUID** response reports the GUID associated with the upload user payload processing or upload mapping payload processing initiated by the respective request.

The response entity MUST be an XML document as follows, **theGuid** of which is replaced by the associated GUID.

```
<?xml version="1.0" encoding="utf-8"?>
<statusguid>theGuid</statusguid>
```

2.2.7 Response: status

The **status** response reports the status of upload user payload processing or upload mapping payload processing. See section 6 for the full XML Schema as specified in [XMLSCHEMA1] section 2.1.

If the payload processing indicated by the GUID from this request is still being processed, the response entity MUST be an XML document, as follows.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<status status="pending"/>
```

If the payload processing indicated by the GUID from this request finished successfully without warnings, the response entity MUST be an XML document, as follows.

```
<?xml version="1.0" encoding="utf-8"?>
<status status="complete"/>
```

If the payload processing indicated by the GUID from this request finished successfully but with warnings, the response entity MUST be an XML document as follows where each **warningTextN** is replaced by an opaque text message describing the warning. The **warning** element MUST be repeated as many times as needed.

```
<?xml version="1.0" encoding="utf-8"?>
<status status="complete">
  <warning>warningText1</warning>
  <warning>warningText2</warning>
</status>
```

If the payload processing indicated by the GUID from this request finished but with an error, the response entity MUST be an XML document as follows, where **errorMessage** is replaced by an opaque text message describing the error:

```
<?xml version="1.0" encoding="utf-8"?>
<status status="error"><error>errorMessage</error></status>
```

The payload processing returns an error message as a normal status message. The last option is not an error in this protocol, but rather a normal status message that reports the error message returned by payload processing.

2.2.8 Response: error

This response reports an error in using this protocol.

The entity returned MUST be an XML document as follows, the **errorMessage** of which is replaced by an opaque text message describing the error.

```
<?xml version="1.0" encoding="utf-8"?>
<error>errorMessage</error>
```

3 Protocol Details

3.1 Client Details

3.1.1 Abstract Data Model

This section describes a conceptual model of a possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

uploadUserRequests: A list of tuples, each consisting of a user store identifier and a GUID. The user store identifier is the **userStoreId** of the URL path name of an **uploadusercomplete** or **uploaduserdelta** request and the GUID is the **theGuid** in the **GUID** response from the same request. Each tuple is used in **uploaduserstatus** requests. The lifetime of each tuple never ends, but after the **uploaduserstatus** request returns something other than "pending", the returned status value MUST NOT change in subsequent **uploaduserstatus** requests.

uploadMappingRequests: A list of tuples, each consisting of a principal aliaser identifier and a GUID. The principal aliaser identifier is the **principalAliaserId** of the URL path name of an **uploadmappingfile** request and the GUID is the **theGuid** in the **GUID** response from the same request. Each tuple is used in **uploadmappingstatus** requests. The lifetime of each tuple never ends, but after the **uploadmappingstatus** request returns something other than "pending", the returned status value MUST NOT change in subsequent **uploadmappingstatus** requests.

The **uploadusercomplete**, **uploaduserdelta**, and **uploaduserstatus** requests are specified in sections 2.2.1, 2.2.2, and 2.2.3. The **uploadmappingfile** and **uploadmappingstatus** requests are specified in sections 2.2.4 and 2.2.5. The **GUID** response is specified in section 2.2.6.

3.1.2 Timers

None.

3.1.3 Initialization

The protocol client establishes a connection with the protocol server using HTTP.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Message Processing Events and Sequencing Rules

The **uploaduserstatus** request uses a specified GUID. It MUST be preceded by the **uploadusercomplete** or **uploaduserdelta** request that returned the specified GUID.

An **uploaduserstatus** request requires both a user store identifier and a GUID. The **uploadUserRequests** provide valid pairs of user store identifiers and **GUIDs** for an **uploaduserstatus** request.

The **uploadmappingstatus** request uses a specified GUID. It MUST be preceded by the **uploadmappingfile** request that returned the specified GUID.

An **uploadmappingstatus** request requires both a principal aliaser identifier and a GUID. The **uploadmappingRequests** provide valid pairs of principal aliaser identifiers and GUIDs for an **uploadmappingstatus** request.

For a specified user store there MUST be only one active upload user payload processing. The protocol client MUST wait for the payload processing of a previous **uploadusercomplete** or **uploaduserdelta** request to complete before sending another for the same user store. If payload processing is still active for a given user store and a new **uploadusercomplete** or **uploaduserdelta** request for the same user store is sent by the protocol client, then the protocol server returns an **error** response that the user store is currently being modified (sections 3.2.5.1 or 3.2.5.2). Otherwise, the protocol server returns a **GUID** response.

For a particular user store and GUID, if an **uploaduserstatus** request returns a **status** response of "pending", then the **upload user payload processing** indicated by the given GUID is still active. The **uploadUserRequests** will list each **upload user payload processing** that was initiated by this protocol client. There may be others initiated by other protocol clients.

For a specified principal aliaser there MUST be only one active upload mapping payload processing. The protocol client MUST wait for the payload processing of a previous **uploadmappingfile** request to complete before sending another request for the same principal aliaser. If payload processing is still active for a given principal aliaser and a new **uploadmappingfile** request for the same principal aliaser is sent by the protocol client, then the protocol server returns an **error** response that the principal aliaser is currently being modified (section 3.2.5.4). Otherwise, the protocol server returns a **GUID** response.

For a particular principal aliaser and GUID, if an **uploadmappingstatus** request returns a **status** response of "pending", then the upload mapping payload processing indicated by the given GUID is still active. The **uploadMappingRequests** will list each upload mapping payload processing that was initiated by this protocol client. There may be others initiated by other protocol clients.

The **uploadusercomplete**, **uploaduserdelta**, and **uploaduserstatus** requests are specified in sections 2.2.1, 2.2.2, and 2.2.3. The **uploadmappingfile** and **uploadmappingstatus** requests are specified in sections 2.2.4 and 2.2.5. The **GUID**, **status**, and **error** responses are specified in sections 2.2.6, 2.2.7, and 2.2.8.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Server Details

3.2.1 Abstract Data Model

This section describes a conceptual model of a possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

activeUploadUserRequests: A mapping of a user store identifier to a structure that contains a GUID and **upload user payload processing** information. Each successful

uploadusercomplete or **uploaduserdelta** request MUST create a new mapping entry. The user store identifier is the **userStoreId** of the URL path name of an **uploadusercomplete** or **uploaduserdelta** request, the GUID is the **theGuid** in the **GUID** response to the request, and the upload user payload processing information as initiated by that same request. Each new mapping entry MUST create a new GUID and new **upload user payload processing** information. This mapping is used by subsequent **uploadusercomplete** and **uploaduserdelta** requests to prevent multiple overlapping requests to the same user store. It is used by **uploaduserstatus** requests to determine that the upload user payload processing is still pending.

activeUploadMappingRequests: A mapping of a principal aliaser identifier to a structure that contains a GUID and upload mapping payload processing information. Each successful **uploadmappingfile** request MUST create a new mapping entry. The principal aliaser identifier is the **principalAliaserId** of the URL path name of an **uploadmappingfile** request, the GUID is the **theGuid** in the **GUID** response to the request, and the **upload mapping payload processing** information as initiated by that same request. Each new mapping entry MUST create a new GUID and new upload mapping payload processing information. This mapping is used by subsequent **uploadmappingfile** requests to prevent multiple overlapping requests to the same principal aliaser. It is used by **uploadmappingstatus** requests to determine that the **upload mapping payload processing** is still "pending".

completedUploadUserStatusInfo: A persistent mapping of a tuple (user store identifier, GUID) to the final upload user payload processing status information. The user store identifier is the **userStoreId** of the URL path name of an uploadusercomplete or **uploaduserdelta** request, the GUID is the **theGuid** in the GUID response to the request, and the upload user payload processing information as initiated by that same request. When any upload user payload processing terminates, the protocol server MUST create a new mapping entry in **completedUploadUserStatusInfo** and remove the corresponding entry from **activeUploadUserRequests**. When an **uploaduserstatus** request arrives that is not in **activeUploadUserRequests**, the **completedUploadUserStatusInfo** is used to determine the status response of "complete" or "error".

completedUploadMappingStatusInfo: A persistent mapping of a tuple (principal aliaser identifier, GUID) to the final **upload mapping payload processing** status information. The principal aliaser identifier is the **principalAliaserId** of the URL path name of an **uploadmappingfile** request, the GUID is the **theGuid** in the **GUID** response to the request, and the upload mapping payload processing information as initiated by that same request. When any upload mapping payload processing terminates, the protocol server MUST create a new mapping entry in **completedUploadMappingStatusInfo** and remove the corresponding entry from **activeUploadMappingRequests**. When an **uploadmappingstatus** request arrives that is not in **activeUploadMappingRequests**, the **completedUploadMappingStatusInfo** is used to determine the **status** response of "complete" or "error".

In addition, the conceptual model requires access to the configuration of the FSA Manager Service as described in [MS-FSO] to determine the current user stores and principal aliasers on the protocol server as specified in [MS-FSSACFG] sections 2.2 and 2.3. The upload user payload processing and upload mapping payload processing initiated by this protocol will need to modify the data used by the user stores and principal aliasers.

The **uploadusercomplete**, **uploaduserdelta**, and **uploaduserstatus** requests are specified in sections 2.2.1, 2.2.2, and 2.2.3. The **uploadmappingfile** and **uploadmappingstatus** requests are specified in sections 2.2.4 and 2.2.5. The **GUID** and **status** responses are specified in sections 2.2.6 and 2.2.7.

3.2.2 Timers

None.

3.2.3 Initialization

None.

3.2.4 Higher-Layer Triggered Events

3.2.4.1 Upload User Payload Processing Completion

When any upload user payload processing that was initiated by an **uploadusercomplete,** as specified in section 2.2.1, or **uploaduserdelta,** as specified in section 2.2.2, request completes, the protocol server MUST remove the corresponding entry in **activeUploadUserRequests** and it MUST add a new entry to **completedUploadUserStatusInfo**. The success or failure of the payload processing MUST be in the new entry added to **completedUploadUserStatusInfo**.

3.2.4.2 Upload Mapping Payload Processing Completion

When any upload mapping payload processing completes that was initiated by an **uploadmappingfile** as specified in section 2.2.4 request, the protocol server MUST remove the corresponding entry in **activeUploadMappingRequests** and it MUST add a new entry to **completedUploadMappingStatusInfo**. The success or failure of the payload processing MUST be in the new entry added to **completedUploadMappingStatusInfo**.

3.2.5 Message Processing Events and Sequencing Rules

Upon receipt of a malformed request, the protocol server MUST return an HTTP response as specified in [RFC2616] section 6.1.1. This response contains a status-code with a value of 500 and an opaque error message. The following conditions MUST cause a malformed request:

- A URL path name that cannot be parsed into **id** and **requestType** as specified in section 2.2
- An unrecognized **requestType**, as specified in section 2.2
- A **Content-Type** variable that does not contain "text/xml"
- A **Content-Encoding** variable that has a value of "gzip" in a payload that cannot be successfully decompressed

3.2.5.1 Receiving an uploadusercomplete Request

Upon receipt of an **uploadusercomplete** request, as specified in section 2.2.1, there are two possible, mutually exclusive responses. The possible responses are an error, or the GUID representing a new upload user payload processing.

If the specified user store is a key in **activeUploadUserRequests**, then an **upload user payload processing** is already active for that user store. In this case, protocol server MUST return an **error** response with an error message that the specified user store is currently being modified, as specified in section 2.2.8.

Otherwise, the protocol server MUST do the following:

- Generate a new GUID.

- Add the user store and GUID to **activeUploadUserRequests**.
- Initiate the upload user payload processing including the information that the payload is complete, and replacing any existing user store data.
- Return a **GUID** response as specified in section 2.2.6.

3.2.5.2 Receiving an uploaduserdelta Request

Upon receipt of an **uploaduserdelta** request, as specified in section 2.2.2, there are two possible, mutually exclusive responses. The possible responses are an error, or the GUID representing a new upload user payload processing.

If the specified user store is a key in **activeUploadUserRequests**, then an **upload user payload processing** is already active for that user store. In this case, protocol server MUST return an **error** response with an error message that the specified user store is currently being modified, as specified in section 2.2.8.

Otherwise, the protocol server MUST do the following:

- Generate a new GUID.
- Add the user store and GUID to **activeUploadUserRequests**.
- Initiate the **upload user payload processing** including the information that the payload is a delta, and modifying any existing user store data.
- Return a **GUID** response as specified in section 2.2.6.

3.2.5.3 Receiving an uploaduserstatus Request

Upon receipt of an **uploaduserstatus** (section 2.2.3) request, there are four possible, mutually exclusive responses depending on the state of the upload user payload processing. The possible responses are pending, not known, successfully finished, or finished with an error.

If the specified user store is a key in **activeUploadUserRequests**, then the upload user payload processing is active for that user store. If it is active then the protocol server MUST return a **status** response with the value "pending" as specified in section 2.2.7. The entity returned MUST be an XML document as follows.

```
<?xml version="1.0" encoding="utf-8"?>
<status status="pending"/>
```

Otherwise, when the upload user payload processing is not active, if the specified user store and GUID are found in **completedUploadUserStatusInfo**, then the upload user payload processing status is known.

If the upload user payload processing status is not known, then the protocol server MUST return an **error** response with an error message that the specified GUID or user store identifier is not recognized, as specified in section 2.2.8.

If the upload user payload processing status is known and it finished successfully, then the protocol server MUST return a **status** response with the value "complete". The entity returned MUST be an XML document as follows.

```
<?xml version="1.0" encoding="utf-8"?>
<status status="complete"/>
```

If the upload user payload processing status is known but it did not complete successfully, then the protocol server MUST return a **status** response with the error message from **completedUploadUserStatusInfo**. The entity returned MUST be an XML document as follows, the **errorMessage** of which is replaced by the error message from **completedUploadUserStatusInfo**.

```
<?xml version="1.0" encoding="utf-8"?><status status="error">
<error>errorMessage</error></status>
```

3.2.5.4 Receiving an uploadmappingfile Request

Upon receipt of an **uploadmappingfile** as specified in section 2.2.3, request, there are two possible, mutually exclusive responses. The possible responses are an error, or the GUID representing a new upload mapping payload processing.

If the specified principal aliaser is a key in **activeUploadMappingRequests**, then an upload mapping payload processing is already active for that principal aliaser. In this case, protocol server MUST return an **error** response with an error message that the specified principal aliaser is currently being modified, as specified in section 2.2.8.

Otherwise, the protocol server MUST do the following:

- Generate a new GUID.
- Add the principal aliaser and GUID to **activeUploadMappingRequests**.
- Initiate the upload mapping payload processing.
- Return a **GUID** response as specified in section 2.2.6.

3.2.5.5 Receiving an uploadmappingstatus Request

Upon receipt of an **uploadmappingstatus** request, as specified in section 2.2.5, there are four possible, mutually exclusive responses depending on the state of the upload mapping payload processing. The possible responses are pending, not known, successfully finished, or finished with an error.

If the specified principal aliaser is a key in **activeUploadMappingRequests**, then the upload mapping payload processing is active for that principal aliaser. If it is active then the protocol server MUST return a **status** response with the value "pending" as specified in section 2.2.7. The entity returned MUST be an XML document as follows.

```
<?xml version="1.0" encoding="utf-8"?>
<status status="pending"/>
```

Otherwise, when the upload mapping payload processing is not active, if the specified principal aliaser and GUID are found in **completedUploadMappingStatusInfo**, then the upload mapping payload processing status is known.

If the upload mapping payload processing status is not known, then the protocol server MUST return an **error** response with an error message that the specified GUID or principal aliaser is not recognized, as specified in section 2.2.8.

If the upload mapping payload processing status is known and it finished successfully, then the protocol server MUST return a **status** response with the value "complete". The entity returned MUST be an XML document as follows.

```
<?xml version="1.0" encoding="utf-8"?>
<status status="complete"/>
```

If the upload mapping payload processing status is known but it did not complete successfully, then the protocol server MUST return a **status** response with the error message from **completedUploadMappingStatusInfo**. The entity returned MUST be an XML document as follows, the **errorMessage** of which is replaced by the error message from **completedUploadMappingStatusInfo**.

```
<?xml version="1.0" encoding="utf-8"?><status status="error">
<error>errorMessage</error></status>
```

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

4 Protocol Examples

The content length of the following examples is not accurate.

4.1 uploadusercomplete Message

This is an example using the **uploadusercomplete** request as described in section 2.2.1 and the **GUID** response as described in section 2.2.6. It replaces the contents of the user store associated with the variable which value is "ln0" and returns the GUID 59615d70-8a3d-4c1b-a9e0-6abf5bd13a4e for use in subsequent **uploaduserstatus** requests as described in section 2.2.3.

The request payload creates two users and two groups, *user1*, *user2*, *group1*, and *group2*. The *user1* is a member of *group1* and *user2* is a member of both *group1* and *group2*. The semantics of the request payload is not relevant to this protocol, and are described in [MS-FSSADFF] section 2.1.

Request

```
POST /ln0/uploadusercomplete/ HTTP/1.1Content-Type: text/xml; charset="UTF-8"Content-
Encoding: UTF-8Host: 192.168.36.142:13271Content-Length: 394Connection: Keep-Alive<?xml
version="1.0"?><entities version="1.0">  <entity id="group1" name="group1" type="group" >
</entity>  <entity id="user1" name="user1" type="user" >    <memberof id="group1" />
</entity>  <entity id="group2" name="group2" type="group" />  <entity id="user2" name="user2"
type="user" >    <memberof id="group1" />    <memberof id="group2" />  </entity></entities>
```

Response

```
HTTP/1.1 200 OKContent-Length: 101Server: Microsoft-HTTPAPI/2.0Date: Thu, 02 Apr 2009
19:01:47 GMT<?xml version="1.0" encoding="utf-8"?><statusguid>59615d70-8a3d-4c1b-a9e0-
6abf5bd13a4e</statusguid>
```

4.2 uploaduserstatus Message Returning Pending

This is an example of an **uploaduserstatus** request as described in section 2.2.3 following the example in section 4.1. It uses a user store identifier with a value of "ln0", and that example's GUID, to request a **status** response as described in section 2.2.7. In this case the status has a value of "pending", indicating that the upload user payload processing has not finished.

Request

```
POST /ln0/uploaduserstatus/ HTTP/1.1Content-Type: text/xml; charset="UTF-8"Host:
192.168.36.142:13271Content-Length: 101<?xml version="1.0" encoding="utf-
8"?><statusguid>59615d70-8a3d-4c1b-a9e0-6abf5bd13a4e</statusguid>
```

Response

```
HTTP/1.1 200 OKContent-Length: 67Server: Microsoft-HTTPAPI/2.0Date: Thu, 02 Apr 2009 19:01:52
GMT<?xml version="1.0" encoding="utf-8"?><status status="pending" />
```

4.3 uploaduserstatus Message Returning Complete

This is an example of an **uploaduserstatus** request as described in section 2.2.3 following the examples in sections 4.1 or 4.2. It uses a user store identifier with a value of "ln0" and the GUID

from those examples to request a **status** response as described in section 2.2.7. In this case the status has the value "complete", indicating that the upload user payload processing has finished.

Request

```
POST /ln0/uploaduserstatus/ HTTP/1.1Content-Type: text/xml; charset="UTF-8"Host:
192.168.36.142:13271Content-Length: 101<?xml version="1.0" encoding="utf-
8"?><statusguid>59615d70-8a3d-4c1b-a9e0-6abf5bd13a4e</statusguid>
```

Response

```
HTTP/1.1 200 OKContent-Length: 68Server: Microsoft-HTTPAPI/2.0Date: Thu, 02 Apr 2009 19:01:57
GMT<?xml version="1.0" encoding="utf-8"?><status status="complete" />
```

4.4 uploaduserdelta Message

This is an example using the **uploaduserdelta** request as described in section 2.2.2 and the **GUID** response as described in section 2.2.6. It modifies the user store associated with the user store identifier which value is "ln0" and returns the GUID d4c4fa67-b792-49a7-bad4-34b0f2e51c53 for use in subsequent **uploaduserstatus** requests as described in section 2.2.3. Examples of **uploaduserstatus** requests that could follow this example are shown in sections 4.2 and 4.3.

The request payload creates and removes various users and groups and their memberships. The semantics of the request payload is not relevant to this protocol, and are described in [MS-FSSADFF] section 2.1.

Request

```
POST /ln0/uploaduserdelta/ HTTP/1.1Content-Type: text/xml; charset="UTF-8"Content-Encoding:
UTF-8Host: 192.168.36.142:13271Content-Length: 382Connection: Keep-Alive<?xml version="1.0"
encoding="UTF-8"?><entities version="1.0">  <removeentity id="User1" />  <entity id="User2"
name="UserTwo" >    <memberof id="Group2" />  </entity>  <entity id="Group2" name="GroupTwo"
>  </entity>  <entity id="User1" >    <removememberof id="Group1" />  </entity>
<removeentity id="User1" />  <removeentity id="Group1" /></entities>
```

Response

```
HTTP/1.1 200 OKContent-Length: 101Server: Microsoft-HTTPAPI/2.0Date: Thu, 02 Apr 2009
22:44:51 GMT<?xml version="1.0" encoding="utf-8"?><statusguid>d4c4fa67-b792-49a7-bad4-
34b0f2e51c53</statusguid>
```

4.5 uploadmappingfile Message

This is an example using the **uploadmappingfile** request as described in section 2.2.4 and the **GUID** response as described in section 2.2.6. It replaces the contents of the XML principal aliaser associated with the value "xmlAliaser1", and returns the GUID de420d05-ecb5-4dc3-a47a-90e0fb6307f9 for use in subsequent **uploadmappingstatus** requests as described in section 2.2.5.

The request payload creates mappings between a user "tom" within a user store specified in the configuration of **xmlAliaser1**, as described in [MS-FSSACFG] section 2.3, and the users "davidjones" of the user store "ora" and "djones" of the user store "dmt". The semantics of the request payload are not relevant to this protocol, and therefore are described in [MS-FSSADFF] section 2.3.

Request

```
POST /xmlAliaser1/uploadmappingfile/ HTTP/1.1Content-Type: text/xml; charset="UTF-8"Content-
Encoding: UTF-8Host: 192.168.36.142:13271Content-Length: 198Connection: Keep-Alive<?xml
version="1.0" encoding="UTF-8" ?><ssoMap ver="1.1">  <user name="tom">    <domain
prefix="ora" username="davidjones"/>    <domain prefix="dmt" username="djones"/>
</user></ssoMap>
```

Response

```
HTTP/1.1 200 OKContent-Length: 101Server: Microsoft-HTTPAPI/2.0Date: Thu, 02 Apr 2009
23:14:18 GMT<?xml version="1.0" encoding="utf-8"?><statusguid>de420d05-ecb5-4dc3-a47a-
90e0fb6307f9</statusguid>
```

4.6 uploadmappingstatus Message Returning Pending

This is an example of an **uploadmappingstatus** request, as described in section 2.2.5, that is processed after the example described in section 4.5. It uses the identifier "xmlAliaser1" for the XML principal aliaser and the GUID of that example to request a **status** response as described in section 2.2.7. In this case the status has the value "pending", indicating that the upload mapping payload processing has not finished.

Request

```
POST /xmlAliaser1/uploadmappingstatus/ HTTP/1.1Content-Type: text/xml; charset="UTF-8"Host:
192.168.36.142:13271Content-Length: 101<?xml version="1.0" encoding="utf-
8"?><statusguid>de420d05-ecb5-4dc3-a47a-90e0fb6307f9</statusguid>
```

Response

```
HTTP/1.1 200 OKContent-Length: 67Server: Microsoft-HTTPAPI/2.0Date: Thu, 02 Apr 2009 23:14:23
GMT<?xml version="1.0" encoding="utf-8"?><status status="pending" />
```

4.7 uploadmappingstatus Message Returning Complete

This is an example of an **uploadmappingstatus** request, as described in section 2.2.5, that is processed after the examples described in sections 4.5 and 4.6. It uses the identifier "xmlAliaser1" for the XML principal aliaser and the example GUID from section 4.5 to request a **status** response as described in section 2.2.7. In this case the status has the value "complete", indicating that the upload mapping payload processing has finished.

Request

```
POST /xmlAliaser1/uploadmappingstatus/ HTTP/1.1Content-Type: text/xml; charset="UTF-8"Host:
192.168.36.142:13271Content-Length: 101<?xml version="1.0" encoding="utf-
8"?><statusguid>de420d05-ecb5-4dc3-a47a-90e0fb6307f9</statusguid>
```

Response

```
HTTP/1.1 200 OKContent-Length: 68Server: Microsoft-HTTPAPI/2.0Date: Thu, 02 Apr 2009 23:14:29
GMT<?xml version="1.0" encoding="utf-8"?><status status="complete" />
```

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: Full XML Schema

For ease of implementation, this section provides the full W3C XML schema for the new elements, attributes, complex types, and simple types described in section 2. Any schema references to namespaces included in ISO/IEC-29500:2008 refer specifically to the transitional schemas as described in [ISO/IEC-29500-4].

The following is the schema for **status** responses.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="status">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="warning" type="xs:string" />
        <xs:element name="error" type="xs:string" />
      </xs:sequence>
      <xs:attribute name="status" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft® FAST™ Search Server 2010

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

8 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

9 Index

S

T

U

V

X