

Evernote API Overview

Version 1.18.1

April 26, 2011

Revision History

Date	Ver	Author	Changes Summary
6/4/08	0.1	Dave Engberg	Principal author / editor
9/18/08	1.10	Dave Engberg	Added OAuth web application documentation
10/27/08	1.10.1	Dave Engberg	Updated Formal EBNF grammar for parsing search expressions
12/4/08	1.11	Dave Engberg	Added changes to API key management in client authentication
3/17/09	1.12	Dave Engberg	Update test server host name
12/10/09	1.14	Dave Engberg	Explain default key permissions and durations
4/16/10	1.15	Seth Hitchings	Updated for ENML2 and note versioning
4/20/10	1.15.1	Seth Hitchings	Updated supported Resource MIME types
5/5/10	1.15.2	Seth Hitchings	Updated supported Resource MIME types
5/24/10	1.15.3	Seth Hitchings	Updated OAuth screenshots
9/21/10	1.16	Seth Hitchings	Updated to Thrift 0.4.0
11/5/10	1.16.1	Seth Hitchings	Updated to cover OAuth 1.0a support
11/10/10	1.17	Seth Hitchings	Added new shared notebook APIs
2/8/11	1.17.1	Seth Hitchings	Updated to reflect OAuth POST support
4/12/11	1.18	Seth Hitchings	Added mobile OAuth authorization layout
4/26/11	1.18.1	Seth Hitchings	Added guidance for storing passwords

Table of Contents

1. Background	4
2. EDAM Overview	5
3. EDAM Data Model	7
3.1. User.....	8
3.2. Accounting	8
3.3. Notebook.....	8
3.4. Note.....	9
3.5. Resource.....	10
3.6. Tag.....	10
3.7. SavedSearch	11
4. UserStore.....	12
5. Web Application Authentication via OAuth	14
5.1. Obtain client credentials.....	14
5.2. Obtain temporary credentials (Request Token)	15
5.3. Resource owner authorization	15
5.4. Obtain token credentials (Access Token)	17
5.5. Access the Evernote Account	18
6. NoteStore	20
7. Local Client API Usage Example.....	23
A. Evernote Markup Language (ENML)	25
A.1. EN-NOTE.....	26
A.2. EN-MEDIA	26
A.3. EN-CRYPT	26
A.4. EN-TODO	27
A.5. ENML Example.....	27
B. Evernote Recognition Index XML Format	29
C. Evernote Search Grammar	30
C.1. Search Terms	30
C.1.1. Scope modifiers.....	30
C.1.2. Matching literal terms.....	31
C.1.3. Matching Core Note Properties	32
C.1.4. Attribute Matching	34
C.1.5. Advanced Content Matching.....	36
C.2. Date/Time Arguments	36
C.2.1. Absolute Date/Time Arguments.....	36
C.2.2. Relative Date Arguments.....	37
C.3. Examples.....	37
C.4. Formal Search Grammar.....	38

1. Background

Evernote is a service that allows you to easily capture information in any environment using whatever device or platform you find most convenient, and makes this information accessible and searchable at any time, from anywhere. Evernote's goal is *ubiquitous access*: users should have a wide variety of ways to create and access their memories.

Evernote's accessibility is based on an Application Programming Interface (API) called the **Evernote Data Access and Management (EDAM)** protocol. EDAM allows secure access to account data via standard web protocols. This API is used internally by all of Evernote's own client applications, and we've made this available for third-party developers to integrate into their applications.

EDAM is designed to support both "thick" applications, such as stateful desktop clients which maintain a full local copy of user data, and "thin" applications, such as web services that only need to access a small amount of current information at a given time. These cases vary a bit in how they securely authenticate the user, but they share a common set of interfaces to use once authentication is completed.

For example, this API could be used by a Linux note-taking application to provide central, secure data storage that synchronizes to Evernote, or it could be used by a PHP web service that links calendaring events with Evernote notes.

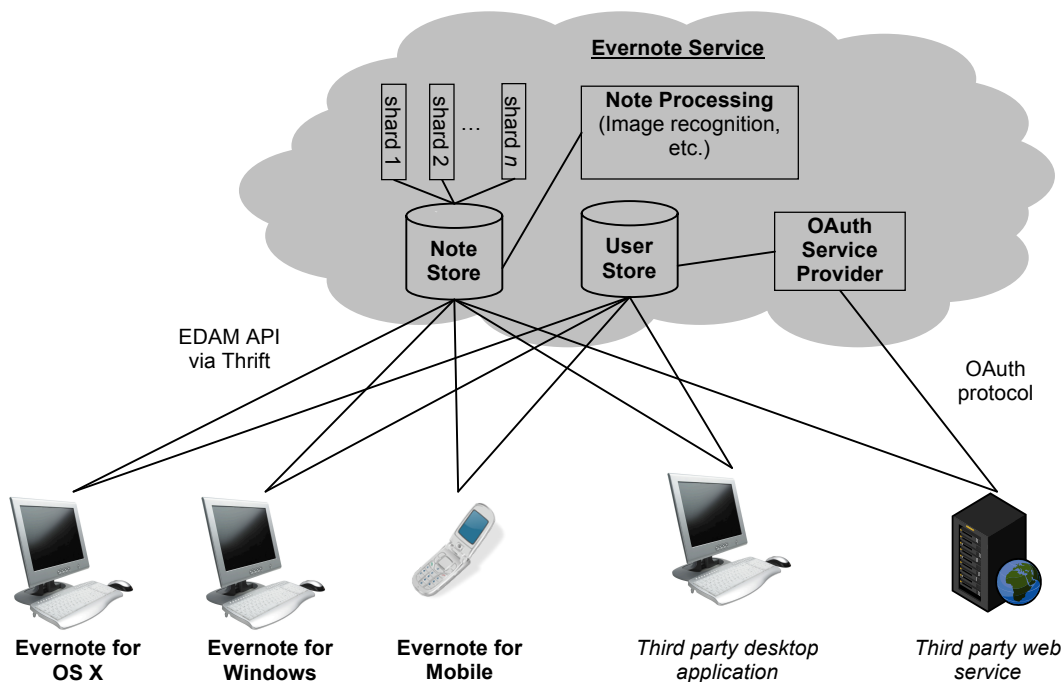
This document provides an introduction to the EDAM API for use by third party developers who wish to integrate with Evernote. This assumes a fairly technical background, so may not be particularly readable for non-programmers.

2. EDAM Overview

The EDAM API is made of two parts: a **data model**, which expresses the types of information that are managed by the Evernote service, and a set of **remote procedures** that retrieve information from the service or request changes to the data.

Both of these are expressed externally via the Thrift service definition language, which originated at Facebook but is now in incubation at Apache (<http://incubator.apache.org/thrift/>). Evernote uses Thrift (version 0.4.0) to specify the EDAM data structures and procedures using a simple Interface Definition Language, which can then be compiled down to language-specific interfaces for a wide variety of different programming platforms (Java, Java ME, PHP, C/C++, Cocoa, Perl, C#, Ruby, etc.). Thrift achieves similar goals to a number of other RPC systems like SOAP, XML-RPC, CORBA, ICE, etc., but Thrift offers a light-weight solution that supports a wide variety of platforms with good performance for the type of large binary data being used by Evernote. Evernote is able to use generated code from Thrift to provide language-specific internal client interfaces for Java service components, Win32/WinCE C++ clients, OS X Objective-C Cocoa clients, and Java Mobile phones. We recommend Facebook's Thrift Whitepaper for a quick introduction to the goals and design of Thrift.

EDAM's remote procedures are divided into two remote services: the **UserStore** service, which is used by local applications to authenticate users, and the **NoteStore**, which authorized applications use to access and change data in user accounts. EDAM clients communicate with these two services as shown in the following diagram:



Local applications (such as desktop utilities) gain authorization to access user accounts by authenticating with a username and password against the UserStore (see Section 4). Third-party web services should not have access to Evernote passwords, so they gain authorization via the OAuth protocol, as described on Section 5. Once authorization is complete using either method, communication with the NoteStore procedures is identical for both local and service applications.

The EDAM API is expressed through a set of Thrift IDL files, which are processed by the Thrift compiler to produce language-specific client and server bindings. Each of these files includes extensive internal documentation on each data structure and procedure call, and these files serve as the definitive reference for the service APIs. The EDAM API is split between the following IDL files:

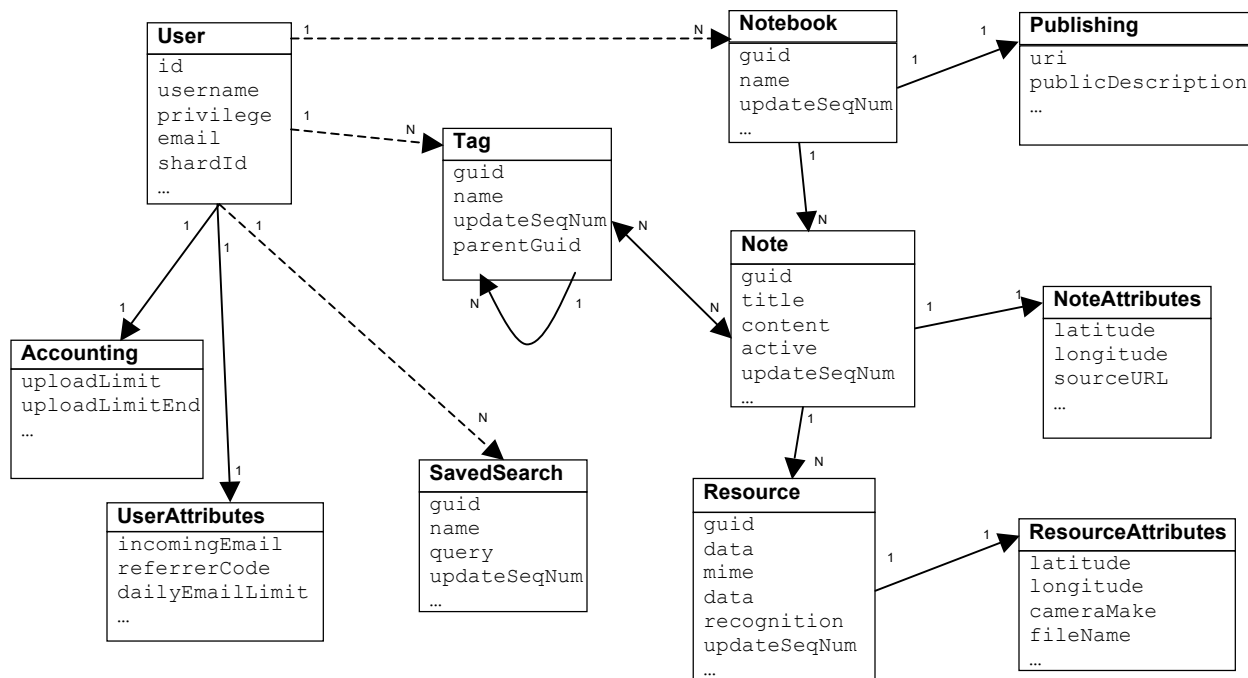
- **Types.thrift** – the declarations for all of the data structures that make up the core EDAM data model for users’ accounts. This includes both the structures that are exchanged with the UserStore (e.g. “UserAttributes” and “Accounting”) and structures that are exchanged with the NoteStore (e.g. “Note” and “Tag”).
- **UserStore.thrift** – the service interface which defines the functions for authenticating against a particular user account and receiving an “Authentication Token” that can be used to perform operations against the NoteStore.
- **NoteStore.thrift** – the service interface to query and modify the data stored in the user’s account.
- **Errors.thrift** – declarations of the exceptions that may be thrown by the UserStore and NoteStore when an error occurs on the service.
- **Limits.thrift** – declarations of the bounds permitted for the user account and various data elements within it. (E.g. the maximum length of a note title, the maximum number of Notebooks in an account, etc.)

The Thrift compiler converts these files to a language-specific library that can be used with the general Thrift runtime libraries in order to talk to the Evernote service.

Thrift represents the API as a set of structures and procedures on the client, which marshal into binary messages that are transmitted as HTTP requests. Since HTTP is used as the low-level transport for the EDAM messages over Thrift, these requests can take advantage of standard HTTP tricks like keep-alive, load balancing, etc.

3. EDAM Data Model

The following diagram shows the high-level relationships between the core data elements managed by the Evernote service for each subscriber. This shows the major data structures and key fields for each of those structures. The full documentation for each structure and field is found inline within the Thrift IDL files.



The structures on the left side of the diagram (User, Accounting, UserAttributes) are interchanged via the UserStore service, while the ones on the right are exchanged via the NoteStore service.

In this diagram, a one-sided arrow with a “1” and an “N” shows a relationship where one entity (e.g. a Notebook) strictly owns a set of another entity. For example, a Notebook owns zero or more Notes, and each Note only belongs in one Notebook. The two-sided arrow between “Tag” and “Note” indicates that any number of Tags may be assigned to any Note, and each Tag can be applied to multiple Notes (“many-to-many”). Dotted lines show relationships that cross from the UserStore to the NoteStore.

Each of the core NoteStore data elements has a globally unique identifier (guid) that is assigned by the service when the object is first created. This identifier does not change for the lifetime of the object, so it is used to refer to the object in later API calls.

Each of these objects also has an updateSequenceNumber (**USN**) that is changed whenever the object is modified in the service. Each updated object receives a new

USN whenever a change is committed. The USN values are unique and monotonically increasing within an account, so that a synchronization client can inspect any two objects in order to determine which was more recently modified.

3.1. User

Every Evernote subscriber will have one **User** account in the Evernote Service. Each user will have a single set of user information that will be managed by the service. This includes information for authenticating the user (username, password, email, etc.) as well as demographic attributes, preferences, etc.

3.2. Accounting

Each User has a set of associated **Accounting** information that allows clients to determine the level of service available for that User. For example, the Accounting structure includes the monthly “uploadLimit”, which specifies the number of bytes of new content that the user can upload into their account each month. Free accounts may have an uploadLimit of 40MB/month, while a Premium account may have a much higher limit. Clients may use the information from the Accounting structure to determine how much data can be transferred into an account.

3.3. Notebook

Each user account will have one or more **Notebooks**. A notebook has a name which must be unique within the owner’s account for clarity of navigation. The Notebook contains a single collection of **Notes** (which includes all of the **NoteAttributes** and **Resources** for those Notes). Each of these data elements will only exist within a single Notebook. It is possible to copy and move Notes from one Notebook to another, but it is not possible for (e.g.) a Note to exist in two different Notebooks. If the Note is copied, this creates a new Note. If the new Note is edited, the original will not be changed.

Each account always has a single Notebook that is marked as the “default” Notebook. This is the Notebook that will receive any Notes that are created without specifying a destination Notebook. For example, if a Note is added to an account via the SMTP email gateway, that Note will go into the default Notebook. The default Notebook can be changed by updating another Notebook to set its “default” flag. If the default Notebook is ever expunged from the account, the oldest remaining Notebook will be promoted to become the default.

The Evernote service does not permit the last Notebook to be deleted from a user’s account, which guarantees that there is always a Notebook that can be used to store new Notes. As a result, the only way to replace the last Notebook in an account is to create a new Notebook and then expunge the old one.

When a user publishes a Notebook for access by web browsers, the Notebook adds a **Publishing** structure which contains the information about accessing and displaying that Notebook. This includes the unique URI string that can be used (in conjunction with the username) to construct the full public notebook URL for use in web browsers.

3.4. Note

A Note is a single section of **hypertext** along with any associated embedded Resources (images, ink, audio, PDF, etc.). The hypertext “content” of the note is stored as a block of text in Evernote Markup Language (ENML) format. This format, which is largely a subset of XHTML, is described in a later section of this document.

The Note contains a block of ENML even when the Note does not appear to contain hypertext: e.g. a plaintext note is just a simple ENML document, and an “image” Note is just a small ENML placeholder with a reference to an embedded binary Resource containing the image bytes. The ENML content of each Note must include an “en-media” element that indicates the placement of each embedded Resource within the hypertext document.

Every Note also has a set of NoteAttributes that carry system-defined information such as creation time, size, origin, etc. These attributes will be usable for searching and filtering of notes.

The placement of a resource (e.g. image, audio or ink) within the hypertext should be represented using an “en-media” tag within the hypertext with a “hash” equal to the MD5 of the embedded resource. For example:

```
<en-media type="audio/wav" hash="ffd93f16876049265fbaef4da268dd0e"/>
```

```
<en-media type="image/jpeg" hash="9e107d9d372bb6826bd81d3542a419d6"/>
```

The MD5 checksum in these “hash” attributes is used as an unambiguous identifier that points from the hypertext to one of the Note’s attached Resources. The MD5 hash is only required to be unique across the Note’s own Resources. This identifier is used instead of the resource’s GUID so that the note (and all its resources) can be created and copied without needing to replace the identifier in the copied note.

Note that this HTML tag must be replaced with a display-specific alternative at some point before rendering the appropriate control via HTML. This replacement can happen at rendering time or when the data is first loaded/modified by a platform. The modified data is specific to each display platform and will not be exchanged via synchronization, etc.

Full documentation on the ENML format is available in Appendix A.

3.5. Resource

A Resource for a Note is a binary data block that is associated with the Note. This is similar to an attachment for an email message. Each Resource may have a set of ResourceAttributes that describe its system-defined properties. For example, meta-data from images may be extracted from a JPEG EXIF block and stored as searchable attributes on the Resource.

The Resource may also have a block of recognition data that can be used to find text within binary Resources such as images. The recognition block for each Resource is stored in the custom “recoData” XML format, which specifies the various recognized regions of text in an image along with the alternate words that may be found in that region.

The full content of the Resource is stored on the service with the Note ... to make sure that stored memories are available for years, Notes do not store “indirect” Resources that are stored on a particular desktop computer. This permits full synchronization across systems.

The service currently accepts Resources of the following types (defined in Limits.thrift) for all accounts, both Premium and free:

- image/gif
- image/jpeg
- image/png
- audio/wav
- audio/mpeg
- audio/amr
- application/vnd.evernote.ink
- application/pdf

For free accounts, the service will reject Notes containing Resources of other types. For Premium accounts, Notes may contain Resources of any type. The default type “application/octet-stream” should be used if a more specific type is not known.

3.6. Tag

Each user account also contains a set of zero or more **Tags**, which are an organizational tool to help users retrieve their memories. Each Tag has a name that is unique within that account. If a Note has the Tag named “food”, then this is the same Tag as “food” on any other Note in that account. Each Tag can be assigned to any number of Notes, and each Note may have any number of Tags.

While many users may be comfortable finding and using their Tags from a flat, alphabetized list, advanced users may want organization in order to help find a

particular Tag. The data model includes a simple organization scheme where a Tag can be moved to be a “child” of another Tag within the Tag list. This is intended as a simple layout option for the list of all Tags ... the location of a Tag underneath its “parent” does not imply any semantic relationship between these Tags. A Tag with no assigned parent is considered to be a “top level” Tag for purposes of display and navigation.

Each Tag can only have one parent, so it cannot be moved under two different Tags in the organizational scheme. The data model will not permit the creation of an “expenses” Tag underneath the “Evernote” Tag if an “expenses” Tag already exists elsewhere in that account.

Tags are shared across notebooks. If Notes in two different Notebooks have a Tag named “cooking”, then those Notes have the same Tag.

Tags are only assigned to a Note as part of an explicit user action. They are not automatically created based on Note attributes, and they are not created as part of automated search filters. Searching based on Note attributes (such as creation time and size) is supported, but this is independent from the “Tag” mechanism, which is reserved for user-assigned labels.

3.7. SavedSearch

Each account contains zero or more **SavedSearches**, which can be used to find a set of Notes meeting various criteria. These criteria can include a combination of keywords, tags and attributes. These criteria can be saved within the account for re-use at a later date, and each SavedSearch has a unique name within that account for ease of identification.

A SavedSearch includes a “query” string, which contains a search expression that is represented using the Evernote search grammar documented in Appendix C. The SavedSearch is just a stored query ... it is not directly associated with any of the Notes which would happen to match if this query were executed. This is different from a Tag, which has an explicit relationship with its Notes in the data model.

When the query SavedSearch is executed, Evernote will select all of the Notes that meet the search’s criteria, as if those criteria had been manually re-entered into the full search GUI. The SavedSearch includes a description of the criteria that should be applied, along with a name for the search to assist in displaying the list of SavedSearches.

4. UserStore

The account and authentication information for every user is bundled into a logical component called the UserStore. For local (i.e. desktop or mobile) client applications, the primary function of the UserStore is to authenticate a user in order to create an “**Authentication Token**”. These tokens are required for any requests to access private data via the NoteStore API. Authentication Tokens are short-lived, cryptographically protected strings that grant the bearer access to some set of operations within a single User account.

The service interface has been defined using a Thrift IDL (UserStore.thrift), which also contains the full documentation on each API function. This interface allows operations including:

Remote Procedure	Brief Description
<code>checkVersion</code>	Sends the client's EDAM version to the service, which responds with compatibility information.
<code>authenticate</code>	Sends a username and password to the service, which provides User information and an Authentication Token if successful.
<code>refreshAuthentication</code>	Sends an Authentication Token to the service which returns a newer token (with a later expiration).
<code>getUser</code>	Retrieves the User associated with a given Authentication Token.
<code>getPublicUserInfo</code>	Retrieve public information about the user with a given username.

The UserStore should always be accessed via SSL using a Thrift TBinaryProtocol wrapping a THttpClient transport through the following URL:

```
https://www.evernote.com/edam/user
```

Authenticating through the UserStore is appropriate for local (i.e. single user) applications that can directly take a username and password from an Evernote user. This means that the user's password is not transferred to any third parties, but is only used for direct authentication to Evernote's servers (over SSL).

Applications that take a username and password from an Evernote user and store the password in persistent storage MUST NOT store it in cleartext. The password MUST be protected using the appropriate mechanism for the platform. On MacOS and iOS, the Keychain should be used. Apple's Keychain sample code for iOS can be found [here](#). On Windows, CryptProtectData or an equivalent operating system API should be used. Microsoft's documentation for CryptProtectData can be found [here](#).

The 'authenticate' procedure call requires that the client identify itself via a unique "consumerKey" and "consumerSecret" pair of parameters. These client API keys are freely provided to developers by Evernote, and are used for general tracking and abuse prevention.

Applications that wish to access Evernote's EDAM API via a third-party service will not invoke the UserStore for authentication, since they should not have access to Evernote users' passwords. For example, a web service that mashes Evernote data with a calendaring database should not solicit Evernote passwords, but should rather retrieve an authenticationToken as described in Section 5. By default, Evernote API keys grant permission to create, read, and update content from accounts. Contact us if your application requires different permissions.

5. Web Application Authentication via OAuth

Web applications that need to access data from multiple Evernote accounts should not retrieve an Authentication Token via a username and password, but should instead receive authorization via the OAuth protocol (<http://oauth.net/>). Evernote has implemented an OAuth Service Provider that complies with a profile (subset) of the OAuth 1.0 Protocol as defined in [RFC 5849](#).

Note that the original OAuth specification used different terminology than RFC 5849. This document uses the terminology from RFC 5849, with the original terminology in parenthesis where applicable.

You will need the following information in order to build an OAuth client (Consumer) that works with Evernote:

- Temporary Credential Request URI:
 - <https://www.evernote.com/oauth>
- Resource Owner Authorization URI, full UI:
 - <https://www.evernote.com/OAuth.action>
- Resource Owner Authorization URI, embedded UI:
 - <https://www.evernote.com/OAuth.action?format=microclip>
- Resource Owner Authorization URI, mobile touch UI:
 - <https://www.evernote.com/OAuth.action?format=mobile>
- Token Request URI:
 - <https://www.evernote.com/oauth>
- Security:
 - HTTPS for all requests
- Supported HTTP methods:
 - GET
 - POST
- Supported signature methods:
 - PLAINTEXT
 - HMAC-SHA1
- Supported OAuth parameter locations:
 - HTTP Authorization header
 - Request URI query parameters

As described in section 2 of RFC 5849, your web service will need to perform the following steps to authenticate to Evernote via OAuth.

5.1. Obtain client credentials

When you request API access to Evernote, we will supply you with client credentials (formerly known as the Consumer Key and Consumer Secret), and a set of URLs to test on a “sandbox” system. When you are done with application testing, we will move these credentials into the production Evernote environment.

5.2. Obtain temporary credentials (Request Token)

As described in RFC 5849 section 2.1, your web service will make a direct request to Evernote’s servers to get temporary credentials (formerly known as the Request Token) that can be used to perform secure authentication for one of your users. This request will be sent over SSL and will be authenticated using your client credentials (Consumer Key and Consumer Secret).

The request parameter *oauth_callback* must contain the callback URL to be used in the following step. The response will contain the temporary credentials: the *oauth_token* and *oauth_token_secret*. If you used the signature method PLAINTEXT, the *oauth_token_secret* will be the empty string.

Sample temporary credentials request:

```
https://www.evernote.com/oauth?oauth_consumer_key=en_oauth_test&oauth_signature=1ca0956605acc4f2%26&oauth_signature_method=PLAINTEXT&oauth_timestamp=1288364369&oauth_nonce=d3d9446802a44259&oauth_callback=https%3A%2F%2Ffoo.com%2Fsettings%2Findex.php%3Faction%3DoauthCallback
```

Sample temporary credentials response:

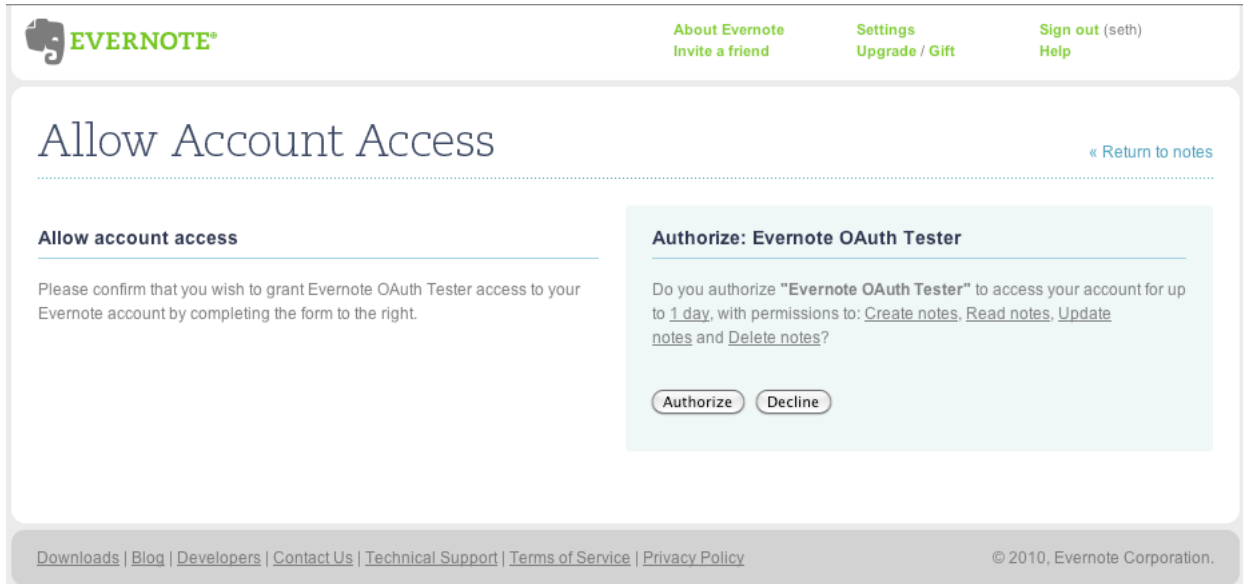
```
oauth_token=en_oauth_test.12BF8802654.687474703A2F2F6C6F63616C686F73742F7E736574682F4544414D576562546573742F696E6465782E7068703F616374696F6E3D63616C6C6261636B.1FFF88DC670B03799613E5AC956B6E6D&oauth_token_secret=&oauth_callback_confirmed=true
```

5.3. Resource owner authorization

Once you have temporary credentials, you should direct your user (the resource owner) to Evernote’s web site to approve access by your application as described in RFC 5849 section 2.2. Evernote provides two different web interfaces to perform this authorization. The simple version presents the authorization form in a full browser window. A second version permits completion of the authorization process in a smaller 500x240 frame (or iframe). In both cases, you must supply the mandatory OAuth parameter *oauth_token* containing the temporary credential identifier that you obtained in the previous step.

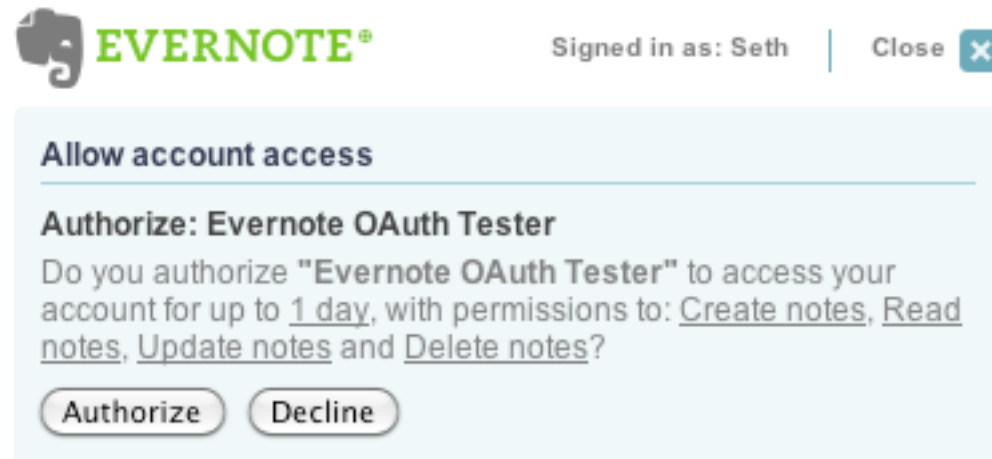
After the resource owner authorizes you to access their Evernote account, they will be redirected to the callback URL that you provided in the previous step. The request to your callback will include the *oauth_token* parameter containing the temporary credential identifier that was authorized and the *oauth_verifier* parameter that you will need in the following step.

The full web UI is available at the base authorization URL <https://www.evernote.com/OAuth.action>. After logging in the user (if they are not already logged in), this form prompts the user with the information about your service and a button to authorize access.



The screenshot shows the Evernote web interface. At the top left is the Evernote logo. To the right are links: "About Evernote", "Invite a friend", "Settings", "Upgrade / Gift", "Sign out (seth)", and "Help". The main heading is "Allow Account Access" with a link "Return to notes". Below the heading, there are two columns. The left column is titled "Allow account access" and contains the text: "Please confirm that you wish to grant Evernote OAuth Tester access to your Evernote account by completing the form to the right." The right column is titled "Authorize: Evernote OAuth Tester" and contains the text: "Do you authorize 'Evernote OAuth Tester' to access your account for up to 1 day, with permissions to: Create notes, Read notes, Update notes and Delete notes?". Below this text are two buttons: "Authorize" and "Decline". At the bottom of the page, there is a footer with links: "Downloads", "Blog", "Developers", "Contact Us", "Technical Support", "Terms of Service", "Privacy Policy", and a copyright notice: "© 2010, Evernote Corporation."

The embedded UI is available by adding the 'format=microclip' parameter to the authorization URL: <https://www.evernote.com/OAuth.action?format=microclip>. This prompts the user for the same approval. This embedded UI may require a bit more work by the originating site when the callback URL is invoked, since it will be processed within the same embedded frame.



The screenshot shows the embedded Evernote OAuth authorization UI. At the top left is the Evernote logo. To the right, it says "Signed in as: Seth" and "Close" with a close button. Below this is a light blue box containing the heading "Allow account access". Underneath is the heading "Authorize: Evernote OAuth Tester" and the text: "Do you authorize 'Evernote OAuth Tester' to access your account for up to 1 day, with permissions to: Create notes, Read notes, Update notes and Delete notes?". At the bottom of the box are two buttons: "Authorize" and "Decline".

The mobile UI is available by adding the 'format=mobile' parameter to the authorization URL: <https://www.evernote.com/OAuth.action?format=mobile>. This prompts the user for the same approval using a page layout suitable for display in a smartphone browser.



Allow account access

You are signed in as sethhitch. [Not sethhitch?](#)

Authorize: Evernote

Do you authorize "Evernote" to access your account for up to 1 day, with permissions to: Create notes, Read notes and Update notes?

Sample authorization URL:

```
https://www.evernote.com/OAuth.action?oauth_token=en_oauth_test.12BF8802654.687474703A2F2F6C6F63616C686F73742F7E736574682F4544414D576562546573742F696E6465782E7068703F616374696F6E3D63616C6C6261636B.1FFF88DC670B03799613E5AC956B6E6D
```

Sample callback URL:

```
https://foo.com/settings/index.php?action=oauthCallback&oauth_token=en_oauth_test.12BF88D95B9.687474703A2F2F6C6F63616C686F73742F7E736574682F4544414D576562546573742F696E6465782E7068703F616374696F6E3D63616C6C6261636B.AEDE24F1FAFD67D267E78D27D14F01D3&oauth_verifier=0D6A636CD623302F8D69DBB8DF76D86E
```

5.4. Obtain token credentials (Access Token)

After you receive a successful callback from the authorization service, you may exchange the temporary credentials for token credentials (formerly known as the Access Token) as described in RFC 5849 section 2.3. The OAuth term "token credentials" corresponds to the EDAM term "authentication token" – a valid token credential identifier is just a string containing an EDAM authentication token that can be used in EDAM API calls.

The URL to obtain token credentials is the same as the one to get temporary credentials (<https://www.evernote.com/oauth>). Evernote determines what you want by the presence of the mandatory *oauth_token* parameter in this second request. You must also include the mandatory *oauth_verifier* parameter that you obtained in the previous step.

The OAuth reply will contain the mandatory *oauth_token* and *oauth_token_secret* elements (URL encoded), and will also contain an *edam_shard* entry, which must be used to determine the correct URL for subsequent EDAM NoteStore calls, and the *edam_userId* entry, which identifies the authenticated user's Evernote user ID.

Sample token credentials request:

```
https://www.evernote.com/oauth?oauth_consumer_key=en_oauth_test&oauth_signature=1ca0956605acc4f2%26&oauth_signature_method=PLAINTEXT&oauth_timestamp=1288364923&oauth_nonce=755d38e6d163e820&oauth_token=en_oauth_test.12BF8888B3F.687474703A2F2F6C6F63616C686F73742F7E736574682F4544414D576562546573742F696E6465782E7068703F616374696F6E3D63616C6C6261636B.C3118B25D0F89531A375382BEEEDD421&oauth_verifier=DF427565AF5473BBE3D85D54FB4D63A4
```

Sample token credentials response:

```
oauth_token=S%3Ds4%3AU%3Da1%3AE%3D12bfd68c6b6%3AC%3D12bf8426ab8%3AP%3D7%3AA%3Den_oauth_test%3AH%3D3df9cf6c0d7bc410824c80231e64dbe1&oauth_token_secret=&edam_shard=s4&edam_userId=161
```

In this example, the OAuth token credential identifier is:

```
S=s4:U=a1:E=12bfd68c6b6:C=12bf8426ab8:P=7:A=en_oauth_test:H=3df9cf6c0d7bc410824c80231e64dbe1
```

The authenticated user account has the Evernote user ID 161 and should be accessed via Evernote shard “s4”, as described in Section 6. Evernote does not use the token credential secret, which will always be the empty string.

5.5. Access the Evernote Account

The token credential identifier from OAuth is an Evernote authentication token that can be used for calls to the EDAM NoteStore remote procedures. The token is provided directly in the function calls, so it is not used in any HTTP headers or query strings when accessing the NoteStore API.

Our testing keys are configured to grant your application permission to read, write, and modify content within an account. When a user authorizes access, your application will be able to manipulate the user's account for 24 hours. If you need a different set of access privileges for your application, or if you wish to request access

to accounts for longer than 24 hours (up to 365 days), you can contact us to request a change to your API key.

6. NoteStore

The user's Notebooks and all of their contents (Notes, Tags, Resources, SavedSearches, etc.) are maintained within the NoteStore component of the Evernote service. The NoteStore is responsible for maintaining the correct data model for each Notebook in a persistent, transactional database.

The NoteStore API includes a few remote procedures that would be primarily used by “thin” clients, which don't maintain a persistent copy of the user account, as well as procedures that would be used by a “thick” client that synchronizes all user data. The majority of routines, however, would be used by both types of clients.

Remote Procedure	Brief Description
listNotebooks	Returns a list of all of the Notebooks in the account.
getNotebook	Retrieves the state of a single Notebook.
getDefaultNotebook	Retrieves the Notebook that should receive new Notes which do not specify a destination.
createNotebook	Makes a new Notebook in the account.
updateNotebook	Changes an existing Notebook.
expungeNotebook	Permanently removes an existing Notebook. Notes within the notebook are moved to the current default Notebook and moved into the trash.
listTags	Returns a list of all of the Tags in the account.
listTagsByNotebook	Returns a list of all of the Tags that are applied to at least one note within a specified notebook.
getTag	Retrieves the state of a single Tag.
createTag	Makes a new Tag in the account.
updateTag	Changes an existing Tag.
untagAll	Removes a Tag from any Notes.
expungeTag	Permanently removes an existing Tag.
listSearches	Returns a list of all of the SavedSearches in the account.
getSearch	Retrieves the state of a single SavedSearch.
createSearch	Makes a new SavedSearch in the account.
updateSearch	Changes an existing SavedSearch.
expungeSearch	Permanently removes an existing SavedSearch.
findNotes	Performs a search of the Notes in the User's account based on a configurable filter, returning a paginated subset.

findNoteCounts	Performs a search based on a configurable filter, returning the number of Notes that would match this filter for each Notebook and Tag.
getNote	Retrieves the state of a single Note.
getNoteContent	Retrieves just the ENML hypertext content of a Note.
getNoteSearchText	Returns the plain text contents of a single note.
getNoteTagNames	Retrieves the names of the Tags for a single Note.
createNote	Makes a new Note in an existing Notebook.
updateNote	Changes the content or metadata of a single existing Note.
deleteNote	Moves a single existing Note to the trash.
expungeNote	Permanently removes an existing Note. In most cases, third party applications that wish to remove a Note should use deleteNote instead of expungeNote.
expungeNotes	Permanently removes a set of existing Notes.
expungeInactiveNotes	Permanently removes all of the notes that are currently not active (i.e. notes in the “Trash”)
copyNote	
listNoteVersions	Retrieves a list of the prior versions of a particular note that are saved within the service.
getNoteVersion	Retrieves a previous version of a Note after it has been updated within the service (for premium Users only).
getResource	Retrieves the state of a single Note attachment, optionally with its binary contents.
updateResource	Updates the metadata for a single Resource. (Not its binary contents.)
getResourceData	Retrieves the binary contents of a single Resource.
getResourceByHash	Retrieves one of the resources from a Note, via the MD5 checksum of its contents, not its GUID.
getResourceRecognition	Returns the XML recognition index file for a single Resource, which can be used to find words in the image.
getResourceAlternateData	Retrieves the binary contents of the Resource’s alternate data file.
getResourceAttributes	Returns the set of attributes for the Resource.
getResourceSearchText	Returns the plain text contents of a single Resource.
getPublicNotebook	Gets the information for one published Notebook from a user’s account, via its public

	URI.
<code>createSharedNotebook</code>	Make a new SharedNotebook object.
<code>listSharedNotebooks</code>	Returns a list of all of the SharedNotebooks in an account.
<code>expungeSharedNotebooks</code>	Permanently removes a set of existing SharedNotebooks.
<code>createLinkedNotebook</code>	Makes a new LinkedNotebook object.
<code>updateLinkedNotebook</code>	Changes an existing LinkedNotebook.
<code>listLinkedNotebooks</code>	Returns a list of all LinkedNotebooks in an account.
<code>expungeLinkedNotebook</code>	Permanently removes an existing LinkedNotebook.
<code>authenticateToSharedNotebook</code>	Returns an authentication token that can be used to access the contents of a notebook shared from someone else's account.
<code>getSharedNotebookByAuth</code>	Get extended information about a notebook shared from someone else's account.
<code>getSyncState</code>	Light-weight call for caching clients to "ping" the service to see whether the account has changed.
<code>getSyncChunk</code>	Core routine for full, synchronizing clients to retrieve the set of changes in an account since the last checkpoint.
<code>getLinkedNotebookSyncState</code>	Light-weight call for caching clients to "ping" the service to see whether shared notebooks linked from other accounts have changed.
<code>getLinkedNotebookSyncChunk</code>	Core routine for full, synchronizing clients to retrieve the set of changes in shared notebooks linked from other accounts since the last checkpoint.

The NoteStore should be accessed via the Thrift TBinaryProtocol using a THttpClient transport through the following URL:

`https://www.evernote.com/edam/note/<shardId>`

The "<shardId>" part at end of the URL should be replaced with the value from the "shardId" field on the User object, which was received as part of the authentication process. This allows the service to efficiently pass your requests to the appropriate NoteStore shard server.

7. Local Client API Usage Example

The following is a very simple example of a local (desktop) client snippet that logs in to the Evernote service and then prints a list of the titles of the first 100 Notes in each Notebook. This sample client was written in Java, but the same set of calls should work from any language supported by Thrift. Corresponding web API samples in Java and PHP are included with the EDAM API distribution.

```
import java.util.*;
import com.evernote.edam.type.*;
import com.evernote.edam.userstore.*;
import com.evernote.edam.notestore.*;
import com.facebook.thrift.*;
import com.facebook.thrift.protocol.TBinaryProtocol;
import com.facebook.thrift.transport.THttpClient;

public class EDAMDemo {

    public static void main(String args[]) throws Exception {
        String username = "dave";
        String password = "$ecr3T!";
        String consumerKey = "client-key-1";
        String consumerSecret = "0a1b2c3d4e5f";
        String userStoreUrl = "https://www.evernote.com/edam/user";
        String noteStoreUrlBase = "https://www.evernote.com/edam/note/";

        THttpClient userStoreTrans = new THttpClient(userStoreUrl);
        TBinaryProtocol userStoreProt = new TBinaryProtocol(userStoreTrans);
        UserStore.Client userStore =
            new UserStore.Client(userStoreProt, userStoreProt);

        boolean versionOk =
            userStore.checkVersion("Dave's EDAMDemo (Java)",
                com.evernote.edam.userstore.Constants.EDAM_VERSION_MAJOR,
                com.evernote.edam.userstore.Constants.EDAM_VERSION_MINOR);
        if (!versionOk) {
            System.err.println("Incomatible EDAM client protocol version");
            return;
        }

        AuthenticationResult authResult =
            userStore.authenticate(username, password, consumerKey, consumerSecret);
        User user = authResult.getUser();
        String authToken = authResult.getAuthenticationToken();
        System.out.println("Notes for " + user.getUsername() + ":" );

        String noteStoreUrl = noteStoreUrlBase + user.getShardId();
        THttpClient noteStoreTrans = new THttpClient(noteStoreUrl);
        TBinaryProtocol noteStoreProt = new TBinaryProtocol(noteStoreTrans);
        NoteStore.Client noteStore =
            new NoteStore.Client(noteStoreProt, noteStoreProt);

        List<Notebook> notebooks =
            (List<Notebook>) noteStore.listNotebooks(authToken);
        for (Notebook notebook : notebooks) {
            System.out.println("Notebook: " + notebook.getName());
        }
    }
}
```

```

    NoteFilter filter = new NoteFilter();
    filter.setNotebookGuid(notebook.getGuid());
    NoteList noteList = noteStore.findNotes(authToken, filter, 0, 100);
    List<Note> notes = (List<Note>) noteList.getNotes();

    for (Note note : notes) {
        System.out.println(" * " + note.getTitle());
    }
}
}

```


A. Evernote Markup Language (ENML)

The contents of Notes within the Evernote service must strictly comply with the XML definition found within the ENML DTD. The reference version of this DTD is located at:

<http://xml.evernote.com/pub/enml2.dtd>

The service will validate the Note content against this DTD before accepting any call to `NoteStore.createNote()` or `NoteStore.updateNote()`.

ENML was designed to provide a secure and portable document representation that could be consistently rendered on various clients and platforms and is largely based on a subset of XHTML. The current version of ENML, version 2, is intentionally broadened to reject less real-world HTML to reduce the likelihood of synchronization failures. This means that all attributes are defined as CDATA instead of more restrictive types, and every HTML element may embed every other HTML element.

Note: Version 1 of ENML, which is still accepted by the Evernote service, enforced a much stricter set of rules such as prohibiting all CSS-related elements and attributes. Notes created prior to the introduction of ENML version 2 will not be converted to ENML version 2, but since ENML version 2 is a superset of version 1, applications need only validate notes against the version 2 DTD.

Before submitting HTML content over the EDAM API the client application is expected to follow the following steps:

1. Convert the document into valid XML
2. Discard all tags that are not accepted by the ENML DTD
3. Convert tags to the proper ENML equivalent (e.g. BODY becomes EN-NOTE)
4. Validate against the ENML DTD
5. Validate *href* and *src* values to be valid URLs and protocols

ENML permits the use of the following standard XHTML elements:

A, ABBR, ACRONYM, ADDRESS, AREA, B, BDO, BIG, BLOCKQUOTE, BR, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIV, DL, DT, EM, FONT, H1, H2, H3, H4, H5, H6, HR, I, IMG, INS, KBD, LI, MAP, OL, P, PRE, Q, S, SAMP, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TFOOT, TH, THEAD, TITLE, TR, TT, U, UL, VAR, XMP

The following XHTML elements are not included in ENML:

APPLET, BASE, BASEFONT, BGSOUND, BLINK, BODY, BUTTON, DIR, EMBED, FIELDSET, FORM, FRAME, FRAMESET, HEAD, HTML, IFRAME, ILAYER, INPUT, ISINDEX, LABEL, LAYER, LEGEND, LINK, MARQUEE, MENU, META,

NOFRAMES, NOSCRIPT, OBJECT, OPTGROUP, OPTION, PARAM, PLAINTEXT, SCRIPT, SELECT, STYLE, TEXTAREA, XML

A number of attributes are also disallowed from the supported XHTML elements:

id, class, onclick, ondblclick, on* ..., accesskey, data, dynsrc, tabindex

URLs within ENML must use one of the following protocols: http, https, file

Other potentially dangerous protocols (e.g. javascript, vbscript, etc.) are disallowed, and documents using these protocols will be rejected by the service.

ENML introduces the following new XML elements for Evernote-specific entities:

A.1. EN-NOTE

This is the top-level element for all ENML documents. It replaces the HTML and BODY tags from an XHTML document.

The EN-NOTE element supports the following optional attributes:

- **bgcolor** – background color of the note
- **text** – text color
- **style, title, lang, xml:lang, dir** – optional attributes with the same semantics as the corresponding XHTML standard attributes.

A.2. EN-MEDIA

The EN-MEDIA element is used to mark the location that one of the Note's Resources should be embedded within the document. For example, this would be used (instead of an 'img' element) to mark the location where an image Resource should be displayed within the Note when it is displayed.

The EN-MEDIA element supports the following attributes:

- **hash** – MD5 checksum of the Resource body, in hexadecimal format. This is the primary unique identifier of the Resource. (required)
- **type** – the MIME type of the Resource. (required)
- **align, alt, longdesc, height, width, border, hspace, vspace, usemap** – optional attributes with the same semantics as the corresponding 'img' attributes. These should be used to help format the Resource when it is displayed.
- **style, title, lang, xml:lang, dir** – optional attributes with the same semantics as the corresponding XHTML standard attributes.

A.3. EN-CRYPT

The EN-CRYPT element contains encrypted and Base-64 encoded text which can be displayed in a full desktop client after the user has entered the appropriate decryption passphrase. The Base-64 encoded ciphertext is contained within the CDATA body of the element.

The EN-CRYPT element supports the following optional attributes:

- **hint** – the passphrase hint to present to the user during decryption.
- **cipher** – the name of the symmetric cipher that was used to encrypt the text. E.g. "RC2"
- **length** – the length of the symmetric cipher key, in bits. E.g. 64

A.4. EN-TODO

The EN-TODO element marks the location of a "To Do" item within the note contents. EN-TODO does not encompass any specific text, it is a self-closing tag that can include a single attribute:

- **checked** – "true" if the To Do item has been completed, or "false" if the item has not been completed. If absent, this defaults to "false".

A.5. ENML Example

The following is a sample ENML document that contains two Resources (an image and an audio file), an encrypted text region, and two To Do items:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE en-note SYSTEM "http://xml.evernote.com/pub/enml2.dtd">
<en-note>
  <b><font size="5">Here is a sample document:</font></b>
  <br/>
  Encrypted credit card number:
  <en-crypt cipher="RC2"
length="64">qo37rLw+x4eNnoaoII/OUN4fasfyauHhdsnq/2/QiA0=</en-crypt>
  <br/>
  <u>Things to work on:</u>
  <en-todo checked="true"/> Write up ENML API documentation
  <br/>
  <en-todo/> Set up ENML API wiki
  <br/>
  Important audio clip:
  <en-media type="audio/wav" hash="5d328e22b2a8593f265b61d05c37f0c6"/>
  <br/>
  Whiteboard picture:
  <en-media width="671" height="503" type="image/jpeg"
    hash="47aa2ac0e29962f3699abe50f1afa996"/>
  <br/>
</en-note>
```


B. Evernote Recognition Index XML Format

Evernote performs advanced processing on some types of Resources to extract content that can be used to later find the associated Notes. For example, images are processed to find printed and handwritten words, which are extracted into a text format for indexing and searching.

This recognition data is represented in a custom XML format that is expressed in the following XML DTD:

<http://xml.evernote.com/pub/recoIndex.dtd>

The recognition index contains a single top-level ‘recoIndex’ element, which contains zero or more items. Each item is associated with a rectangular region of the input image (via x, y, w, h). For each item, there are a list of possible terms (element ‘t’) which consist of a weighted score (attribute ‘w’) and a word in the CDATA body of the element. For example, the following is a recognition index file for an image with two discovered words. The first word is in a region at 151,219 with a size of 323x159, and the second is at 481,387 with a size of 266x120:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE recoIndex SYSTEM "http://xml.evernote.com/pub/recoIndex.dtd">
<recoIndex docType="handwritten" objType="image" objID="47aa2ac0e29962f3699abe50f1afa996"
engineVersion="1.9.4.6" recoType="client" lang="en" objWidth="1600" objHeight="1200">
  <item x="151" y="219" w="323" h="159">
    <t w="68">CLIPS</t>
    <t w="65">CHIPS</t>
    <t w="62">CERPS</t>
    <t w="32">CORPS</t>
    <t w="31">CREEPS</t>
    <t w="30">CHOPS</t>
  </item>
  <item x="481" y="387" w="266" h="120">
    <t w="40">NOTES</t>
    <t w="32">NOTTS</t>
    <t w="32">NOTE</t>
    <t w="27">NORTE</t>
    <t w="27">MOTTS</t>
    <t w="27">RIOTS</t>
  </item>
</recoIndex>
```

The top-scoring possibility for the first region is “CLIPS” and the top-scoring word for the second region is “NOTES”.

When a client submits a Note to the service that includes Resources with no recognition data, the service may queue those Resources for processing. Within a few minutes, the corresponding Resource may be updated with server-generated recognition data (which increases the Resource’s updateSequenceNumber). The client can retrieve this data via NoteStore.getResource or NoteStore.getResourceRecognition.

C. Evernote Search Grammar

Evernote translates all Note searching and filtering into a simple, text-based string representation. This internal search format is used for SavedSearch queries, and it may also be used directly by power users. This same search syntax is implemented on the Evernote service and all clients, which means that the same search should produce the same results on all systems.

Evernote's search grammar is a simple list of terms which are evaluated within a notebook (or "all notebooks") to find a match. By default, the search results are the intersection of the notes that match each individual search term. This behavior changes if the **"any:"** modifier is found in the search. In this case, the search is executed as a union of the matches of the individual terms, and notes will be returned that match any of the criteria terms. The results are obviously identical if there is only one search term.

String matches are case insensitive, and multiple spaces will compare as if they were a single space.

The search grammar includes a set of advanced search expressions in the form of "modifier:argument". A note will match this expression if the appropriate condition is met. The matching terms vary by the type of the modifier, so that a date may be compared differently than a string.

Any matching term may also be negated by adding a "-" character to the beginning. This means that the term will only match a note if the conditional is NOT true. Each term in the search may be one of the following:

C.1. Search Terms

This section documents the way that the search grammar will interpret search terms within the search expression.

C.1.1. Scope modifiers

notebook:[nb name] - will match notes in a notebook with the provided name. This must be the first term in the search. Name matching is case-insensitive. Since notebooks have exclusive relationships with notes, at most one notebook can be provided for the search. If no notebook is given, the search will go over all of the user's active notes. The notebook is not included in the "union" created by the "any:" operator. E.g.:

- notebook:"Bob's first notebook"
 - Matches all notes in this notebook
- notebook:"Hot Stuff" any: mexican italian
 - Matches all notes in the "Hot Stuff" notebook that have the word "mexican" or the word "italian" in them.

any: - If this expression is found at the beginning of the search (after the "notebook", if present), then the search will return a note that matches any of the other search terms. If this is not found, then the default behavior will be used: a note must match all of the search terms. This expression cannot be negated.

C.1.2. Matching literal terms

If no advanced search modifier is found in a search term, it will be matched against the note as a text content search. Words or quoted phrases must exactly match a word or phrase in the note contents, note title, tag name, or recognition index. Words in the content of the note are split by whitespace or punctuation. Words may end in a wildcard to match the start of a word. Searches are not case sensitive. (A wildcard is only permitted at the end of the term, not at the beginning or middle for scalability reasons on the service.) Multiple whitespace and/or punctuation characters in the quoted phrase or the note will be compared as if they were a single space. The backslash escape character ('\') may be used to escape a quotation mark within a quoted phrase. E.g.:

- potato
 - matches: "Sweet Potato Pie"
 - does not match: "Mash four potatoes together"
- Ever*
 - matches: "Evernote Corporation"
 - does not match: "forevernote"
- "San Francisco"
 - matches: "The hills of San Francisco"
 - does not match: "San Andreas fault near Francisco winery"
- -potato
 - matches: "Mash four potatoes together"
 - does not match: "Sweet Potato Pie"
- ham
 - matches: "green eggs&ham."
- "eggs ham"
 - matches: "green eggs&ham."

Punctuation is used to split the input query and document into words, but it is ignored for text matching. The behavior of a quoted search should behave as if the following operations were performed on both the search query and the target note:

1. All XML markup is removed from the document, leaving only the visible text as a string
2. The string is converted to a list of words which are separated by one or more whitespace and/or punctuation characters.
3. The case of each word in the list is normalized

4. The list of words in the query must match with the same sequence of words in the converted Note

For example, if a user searches for the phrase "Spatula! City! For Bargains..." against this ENML document:

```
<en-note>Come down to Spatula  
City - for bargains on spatulas</en-note>
```

The algorithm should convert the search phrase into a normalized list of words:

```
[ "spatula", "city", "for", "bargains" ]
```

And the document into:

```
[ "come", "down", "to", "spatula", "city", "for", "bargains", "on",  
"spatulas" ]
```

The search should match, since words from the target phrase are found in the list of words extracted from the document. (The same result could be implemented without literally converting each note into a list of words, but this gives the intended behavior that we see from major search engines like Google and MS.)

C.1.3. Matching Core Note Properties

tag:[tag name] - will match notes that have a tag with the literal name (word or quoted phrase). This requires an full case-insensitive match on the tag name. The tag name may end with a wildcard to match the beginning of a tag. The pattern will match from the beginning of the full tag name, and punctuation will be included. I.e. the tag and the search string are not tokenized by whitespace and/or punctuation. This can be used multiple times to specify all tags that must match the notes. E.g.:

- tag:cooking
 - Matches any note with the "cooking" tag
- tag:cook*
 - Matches any note with a tag that starts with "cook"
- tag:"hot stuff"
- -tag:cook*
 - Matches any note that does not have a tag that starts with "cook"
- tag:*
 - Matches any note that has at least one tag
- -tag:*
 - Matches any note that has no tags

intitle:[literal] - will match notes with a title that contains the literal word or quoted phrase. Can be used more than once. E.g.:

- intitle:chicken
- intitle:"tale of two"
- -intitle:beef
 - Matches notes that do not have the word "beef" in their title.

created:[datetime] - will match any note that has a 'created' timestamp that is equal to, or more recent than, the provided datetime. (See Section C.2 for details on the legal format of the datetime argument.) E.g.:

- created:20070704
 - Matches notes that were created on or after July 4th, 2007, based on the client's timezone.
- created:20070704T090000
 - Matches notes that were created on or after 9:00am on July 4th, 2007, based on the client's timezone.
- created:20070704T150000Z
 - Matches notes that were created on or after 3:00pm GMT on July 4th, 2007.
- -created:20070704
 - Matches notes that were created before July 4th, 2007, based on the client's timezone.
- created:day-1
 - Matches notes that were created yesterday or today
- -created:day
 - Matches notes that were created before today
- created:day-1 -created:day
 - Matches notes that were created yesterday (only)
- created:day-30
 - Matches notes that were created within the last 30 days (or today)
- created:week
 - Matches notes that were created in this calendar week (Sunday-Saturday)
- -created:month
 - Matches notes that were created before this month
- created:year-1
 - Matches notes that were created last year or this year

updated:[datetime] - will match any note that has a 'updated' timestamp that is equal to, or more recent than, the provided datetime. (See Section C.2 for details on the legal format of the datetime argument.)

resource:[MIME type string] - will match notes that have a resource with a MIME type that matches the argument. E.g.:

- resource:image/gif
 - Matches notes with at least one image/gif resource

- `resource:audio/*`
 - Matches notes with at least one audio resource
- `-resource:image/*`
 - Matches notes with no images
- `resource:application/vnd.evernote.ink`
 - Matches notes with one or more ink resources

C.1.4. Attribute Matching

The search expression may also contain a matching term for any attribute that is defined in the data model. These are defined in the `NoteAttributes` and `ResourceAttributes` structures in `Types.thrift`. This will match against the `Note` attribute, if `NoteAttributes` contains an attribute with that name, or else it will try to match against the `Resource` attribute if one exists with that name. The match is performed based on the type of the attribute.

String attributes will be compared using the standard string matching as above (case insensitive, normalized spacing, optional wildcard at the end of the argument).

Datetime attributes will be matched in the same manner as "created" and "updated", above.

Boolean attributes will be matched based on the argument of "true" or "false". A boolean attribute will match the wildcard ("*") argument if it has any value set for that attribute.

Double expressions will match notes where the attribute that is greater than or equal to the argument (see "latitude" for examples). Double comparisons are numeric, not lexical, so an argument of "99.9" is less than an argument of "100". A double attribute will match the wildcard ("*") argument if it has any value set for that attribute.

subjectDate:[datetime] - matches notes with a `subjectDate` attribute that is equal to or later than the argument datetime.

latitude:[double] - matches notes with a latitude that is greater than or equal to the argument. E.g.:

- `latitude:37 -latitude:38`
 - Matches notes with a latitude that is greater than or equal to 37, but do not have a latitude greater than or equal to 38. (I.e. $37 \leq \text{latitude} < 38$)

longitude:[double] - matches notes with a longitude that is greater than or equal to the argument.

altitude:[double] - matches notes with an altitude that is greater than or equal to the argument.

author:[string] - will match notes that have an "author" attribute set with a name that matches the argument string. E.g.:

- author:"robert parker"
- author:robert*
- -author:*
 - Matches notes that have no "author" attribute set
- author:"Phil \"Chef\" Libin"
 - Matches notes containing: Evernote's CEO is Phil "Chef" Libin, formerly of Cambridge.

source:[string] - matches notes that came from an application or data source that matches the argument string. Notes that were created directly in an Evernote client will not have a "source" attribute. Legal source attributes include: app.ms.word, app.ms.excel, app.ms.powerpoint, mail.clip, mail.smtp, web.clip, mobile.wm E.g.:

- source:app.ms.word
 - Matches notes that came from a Microsoft Word document
- source:app.ms.*
 - Matches notes that came from any Microsoft application
- source:web.clip
 - Matches notes that were locally clipped from a web page
- source:mail.clip
 - Matches notes that were clipped from a local mail client
- source:mail.smtp
 - Matches notes that were delivered to the service via the email gateway.
- source:mobile.*
 - Matches notes that were created on a mobile client of some form

sourceApplication:[string] - matches notes that have a source application string that matches the argument. This string is not guaranteed to be structured.

recoType:[string] - matches notes with a resource that has recognition data that specifies this recognition document type. If this attribute is set on a resource, it will have one of the following values: 'printed', 'speech', 'handwritten', 'picture', or 'unknown'. E.g.:

- recoType:handwritten
 - Matches notes with at least one resource that was recognized as handwritten
- recoType:*

- Matches notes that contain at least one resource that has recognition index data

C.1.5. Advanced Content Matching

The following search terms are expressed as attributes, but these do not correspond to literal attributes in the NoteAttributes data model. Instead, these perform advanced matching against the content of the notes. These do not match standard words in the notes, but rather match special elements embedded within the ENML document.

todo:[true|false]* - if the argument is "true", this will match notes that have ToDo checkboxes that are currently checked. If the argument is "false", this will match notes that have ToDo checkboxes that are not currently checked. If the argument is "", this will match notes that have a ToDo checkbox of any type.

- -todo:false todo:true
 - Matches notes that have completed ToDo items, but no uncompleted items.

encryption: - matches notes that have an encrypted region within them.

C.2. Date/Time Arguments

Various expressions (such as "created:...") take an argument that is interpreted as a date or a date and time. This date is translated into a universal time value for comparison against the timestamps on the notes. The search grammar includes datetime expressions using either an absolute specification (including year, month, day...) or an expression that is relative to the current day/week/month/year. The former is required to support searches in a specific date range, but the latter is particularly useful for saved searches which may return notes that are (e.g.) no more than 7 days old.

C.2.1. Absolute Date/Time Arguments

Absolute datetimes are specified using a compact profile of ISO 8601 (http://en.wikipedia.org/wiki/ISO_8601). An absolute datetime must fit one of the following three forms:

yyyyMMdd - Used to specify a date with no time component. Equivalent to "yyyyMMddT000000" for the same values. The date is converted to a universal time based on the client's desired timezone before comparing against the internal universal timestamps on the notes. E.g. "20071031" evaluates to 12:00am on 31 October 2007 in the client's timezone.

yyyyMMdd'T'HHmmss - Used to specify a date with a time component. The date and time are converted to a universal time based on the client's desired timezone before comparing against the universal timestamps on the notes. E.g. "20071031T093000" evaluates to 9:30am on 31 October 2007 in the client's timezone.

yyyyMMdd'T'HHmmss'Z' - Used to specify a date and time in absolute UTC (aka "GMT" or "Zulu") time. This time can be compared against notes as a universal time, which will produce the same results regardless of the client's current timezone preferences. E.g. "20071031T153000Z" evaluates 3:30pm UTC on 31 October 2007.

C.2.2. Relative Date Arguments

Relative date arguments are evaluated based on the client's notion of the beginning of the current "day", "week", "month", or "year". They may include an integer delta to indicate a previous day/week/month/year instead of the current one. If no delta is provided, the argument evaluates the the beginning of the current day/week/month/year. The following examples show how each expression would be evaluated by a client with a local date and time of: *Wednesday, 31 October 2007, 13:30:56*

- **day** - would evaluate to: Wednesday, 31 October 2007, 00:00:00
- **day-1** - would evaluate to: Tuesday, 30 October 2007, 00:00:00
- **day-14** - would evaluate to: Wednesday, 17 October 2007, 00:00:00
- **week** - would evaluate to: Sunday, 28 October 2007, 00:00:00
- **week-2** - would evaluate to: Sunday, 14 October 2007, 00:00:00
- **month** - would evaluate to: Monday, 1 October 2007, 00:00:00
- **month-1** - would evaluate to: Monday, 1 September 2007, 00:00:00
- **year** - would evaluate to: Monday, 1 January 2007, 00:00:00
- **year-1** - would evaluate to: Sunday, 1 January 2006, 00:00:00

C.3. Examples

Find notes containing the word "chicken", tagged with "cooking", and created this year:

```
chicken tag:cooking created:year
```

Find notes tagged with "cooking" but not "mexican" that include the word "beef" but not the word "carrots"

```
tag:cooking -tag:mexican beef -carrots
```

Find notes in my "Travel" notebook with San Francisco in the title:

```
notebook:Travel intitle:"San Francisco"
```

Find notes that either include the text "San Francisco" or are tagged with the "SFO" tag:

```
any: "San Francisco" tag:SFO
```

Find image notes from the Sunnyvale region:

```
resource:image/* latitude:37 -latitude:38  
longitude:-123 -longitude:-122
```

Find untagged audio notes that I edited in the last week or two:

```
-tag:* resource:audio/* updated:week-1
```

C.4. Formal Search Grammar

The following grammar is expressed using EBNF notation (http://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_form). This grammar assumes that recursive-descent parsing will be "greedy" to handle any ambiguity. The output of parsing this grammar is a set of terms and operators; separators are ignored. The parsing results of search expressions which do not match this grammar are undefined.

```
search = [ notebook , separator ] ,  
        [ "any:" , separator ] ,  
        term ,  
        { separator , term } ;  
  
notebook = "notebook:" , ( word | quoted ) ;  
  
word = wordchar , { wordchar } ;  
  
quoted = '"' , { ( any - '"' ) | '\"' } , '"' ;  
  
separator = { sepchar , separator } ;  
  
sepchar = any - ( wordchar | '"' | "-" ) ;  
  
term = "-" negatable  
      | negatable  
      ;  
  
negatable = literal  
          | expression  
          ;  
  
literal = word | quoted | prefix ;  
  
prefix = { wordchar } , "*" ;  
  
expression = label , ":" , value ;
```

```

(*)
* Expression labels must be:
* The name of a NoteAttribute field (e.g. "subjectDate")
* or a ResourceAttribute field (e.g. "fileName"),
* or one of: "tag", "intitle", "created", "updated", "resource",
*           "todo", or "encryption".
* Search results are undefined if an unknown label is used.
*)
label = word ;

(*)
* The expression value will be interpreted based on the type of the
* field or attribute.
* Search results are undefined if the value does not match the
* required format for this field.
*)
value = nonspace | quoted ;

nonspace = { any - space } ;

space = ? whitespace character (Unicode character class Z) ? ;

wordchar = ? any Unicode Letter or Number (Unicode character classes L
and N) or underscore '_' ? ;

any = ? any printable characters ? ;

```

The following additional production rules indicate the expected format of expression values to match various note fields and attribute types described above.

```

boolean = "true"
        | "false"
        | "*"
        ;

double = [ "-" ] , digit , { digit } , [ "." , { digit } ]
        | "*"
        ;

datetime = absolutedate
        | relativedate
        | "*"
        ;

absolutedate = year , month , day ,
              [ "T" , hour , minute , second ] , [ "Z" ] ;
year = digit , digit , digit , digit ;
month = ( "0" | "1" ) , digit ;
day = ( "0" | "1" | "2" | "3" ) , digit ;
hour = ( "0" | "1" | "2" ) , digit ;
minute = ( "0" | "1" | "2" | "3" | "4" | "5" ) , digit ;
second = ( "0" | "1" | "2" | "3" | "4" | "5" ) , digit ;

```

```
relativedate = relative
               | relative , "-" , { digit }
               ;

relative = "day" | "week" | "month" | "year" ;

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
```