QEverCloud

6.2.0

Generated on Sat Apr 8 2023 19:33:14 for QEverCloud by Doxygen 1.9.8

Sat Apr 8 2023 19:33:14

Chapter 1

QEverCloud

Unofficial Evernote Cloud API for Qt

1.1 What's this

This library presents the complete Evernote SDK for Qt. All the functionality that is described on Evernote site is implemented and ready to use. In particular OAuth authentication is implemented.

Read doxygen generated documentation for detailed info.

The documentation can also be generated in the form of a .qch file which you can register with your copy of Qt Creator to have context-sensitive help. See below for more details.

1.2 How to contribute

See contribution guide for detailed info.

1.3 Downloads

Prebuilt versions of the library can be downloaded from the following locations:

- Stable version:
  - Windows binaries:
    - MSVC 2019 32 bit Qt 5.15.2
    - MSVC 2019 64 bit Qt 5.15.2
  - Mac binary (built with Qt 5.15.2)
  - Linux binary built on Ubuntu 20.04 with Qt 5.12.8
- Unstable version:
  - Windows binaries:
    - MSVC 2019 32 bit Qt 5.15.2
    - MSVC 2019 64 bit Qt 5.15.2
  - Mac binary (built with Qt 5.15.2)
  - Linux binary built on Ubuntu 20.04 with Qt 5.12.8

1.4 How to build

QEverCloud uses CMake build system which can be used as simply as follows (on Unix platforms):

```
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=<...> ../
make
make install
```

The library can be built and shipped either as a static library or a shared library. Dll export/import symbols necessary for Windows platform are supported.

QEverCloud uses C++14 standard. CMake automatically checks whether the compiler is capable enough of building QEverCloud so if the pre-build configuration step was successful, the build step should be successful as well. Known capable compilers are g++ 9 or later and Visual Studio 2019 or later.

The recommended version of Qt5 for building the library is the latest Qt 5.15. However, it is also known to build and work with Qt 5.12.

QEverCloud depends on the following Qt components:

- Qt5Core
- Qt5Widgets
- Qt5Network
- (Optional) Qt5WebKit an Qt5WebKitWidgets
- (Optional) Qt5WebEngine and Qt5WebEngineWidgets

The dependencies on Qt5WebKit or Qt5WebEngine are only actual if the library is built with OAuth support. But even then there is an option to build the library with OAuth support but without the dependency on either of these components. More on this below.

By default the library is built with OAuth support and uses Qt5WebEngine for it. The following cmake parameters are available to alter this behaviour:

- `-DBUILD_WITH_OAUTH_SUPPORT=NO` would disable building with OAuth support entirely.
- `-DUSE_QT5_WEBKIT=ON` would build the library with OAuth using Qt5WebKit for web page rendering.
- `-DQEVERCLOUD_USE_SYSTEM_BROWSER=ON` would build the library with OAuth not using either Qt5WebKit or Qt5WebEngine but instead delegating some portion of OAuth procedure to the system browser.

If Qt5's Qt5Test module is found during the pre-build configuration step, the unit tests are enabled and can be run with `make test` and more verbose `make check` commands.

Other available CMake configurations options:

BUILD_DOCUMENTATION - when *ON*, attempts to find Doxygen and in case of success adds *doc* target so the documentation can be built using `make doc` command after the pre-build configuration step. By default this option is on.

BUILD_QCH_DOCUMENTATION - when *ON*, passes instructions on to Doxygen to build the documentation in *qch* format. This option only has any meaning if *BUILD_DOCUMENTATION* option is on. By default this option is off.

BUILD_SHARED - when *ON*, CMake configures the build for the shared library. By default this option is on.

BUILD_WITH_Q_NAMESPACE - when *ON*, `Q_NAMESPACE` and `Q_ENUM_NS` macros are used to add introspection capabilities to enumerations within `qevercloud` namespace. Qt >= 5.8 is required to enable this option. By default this option is enabled.

BUILD_TRANSLATIONS - when *ON*, builds and installs translation files for translatable strings from QEverCloud.

If *BUILD_SHARED* is *ON*, `make install` installs CMake module necessary for applications using CMake's `find_package` command to find the installation of QEverCloud.

It is possible to build the library with enabled sanitizers using additional CMake options:

- `-DSANITIZE_ADDRESS=ON` to enable address sanitizer
- `-DSANITIZE_MEMORY=ON` to enable memory sanitizer
- `-DSANITIZE_THREAD=ON` to enable thread sanitizer
- `-DSANITIZE_UNDEFINED=ON` to enable undefined behaviour sanitizer

1.5 Include files for applications using the library

Two "cumulative" headers - *QEverCloud.h* or *QEverCloudOAuth.h* - include everything needed for the general and OAuth functionality correspondingly. More "fine-grained" headers can also be used if needed.

1.6 Seeding random numbers generator for Qt < 5.10

QEverCloud requires random numbers generator for OAuth procedure. When QEverCloud is built against Qt >= 5.10, it uses `QRandomGenerator` which is cryptographically secure on supported platforms and is seeded by Qt internals. With Qt < 5.10 QEverCloud uses `qrand`. It requires the client application to call `qsrand` with seed value before using OAuth calls of QEverCloud. So if you are using QEverCloud built with Qt < 5.10, make sure to call `qsrand` before using QEverCloud's OAuth.

1.7 Related projects

- `QEverCloudGenerator` repository hosts code generating parser of `Evernote Thrift IDL files`. This parser is used to autogenerate a portion of QEverCloud's headers and sources.
- `libquentier` is a library for creating feature rich full sync Evernote clients built on top of QEverCloud
- `Quentier` is an open source desktop note taking app capable of working as Evernote client built on top of libquentier and QEverCloud

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 qevercloud Namespace Reference

Classes

- struct Accounting
- struct AccountLimits
- class AsyncResult

  Returned by asynchonous versions of functions.
- struct AuthenticationResult
- struct BootstrapInfo
- struct BootstrapProfile
- struct BootstrapSettings
- struct BusinessInvitation
- struct BusinessNotebook
- struct BusinessUserAttributes
- struct BusinessUserInfo
- struct CanMoveToContainerRestrictions
- struct Contact
- struct CreateOrUpdateNotebookSharesResult
- struct Data
- class EDAMInvalidContactsException
- class EDAMInvalidContactsExceptionData
- class EDAMNotFoundException
- class EDAMNotFoundExceptionData
- class EDAMSystemException
- class EDAMSystemExceptionAuthExpired
- class EDAMSystemExceptionAuthExpiredData
- class EDAMSystemExceptionData
- class EDAMSystemExceptionRateLimitReached
- class EDAMSystemExceptionRateLimitReachedData
- class EDAMUserException
- class EDAMUserExceptionData
- class EventLoopFinisher
- class EverCloudException
- class EverCloudExceptionData

  EverCloudException counterpart for asynchronous API.
- class EverCloudLocalData

The EverCloudLocalData class contains several data elements which are not synchronized with Evernote service but which are nevertheless useful in applications using QEverCloud to implement feature rich full sync Evernote clients. Values of this class' types are contained within QEverCloud types corresponding to actual Evernote API types.

- class EvernoteException
- class EvernoteExceptionData
- class EvernoteOAuthDialog

  Authorizes your app with the Evernote service by means of OAuth authentication.
- class EvernoteOAuthWebView

  The class is tailored specifically for OAuth authorization with Evernote.
- struct Identity
- class IDurableService
- class ILogger
- class InkNoteImageDownloader

  the InkNoteImageDownloader class is for downloading the images of ink notes which can be created with the official Evernote client on Windows (only with it, at least at the time of this writing).
- class INoteStore
- struct InvitationShareRelationship
- class IRequestContext
- struct IRetryPolicy
- class IUserStore
- struct LazyMap
- struct LinkedNotebook
- struct ManageNotebookSharesError
- struct ManageNotebookSharesParameters
- struct ManageNotebookSharesResult
- struct ManageNoteSharesError
- struct ManageNoteSharesParameters
- struct ManageNoteSharesResult
- struct MemberShareRelationship
- class NetworkException

  The NetworkException class represents QNetworkReply level errors.
- class NetworkExceptionData
- struct Note
- struct NoteAttributes
- struct Notebook
- struct NotebookDescriptor
- struct NotebookRecipientSettings
- struct NotebookRestrictions
- struct NotebookShareTemplate
- struct NoteCollectionCounts
- struct NoteEmailParameters
- struct NoteFilter
- struct NoteInvitationShareRelationship
- struct NoteLimits
- struct NoteList
- struct NoteMemberShareRelationship
- struct NoteMetadata
- struct NoteRestrictions
- struct NoteResultSpec
- struct NoteShareRelationshipRestrictions
- struct NoteShareRelationships
- struct NotesMetadataList
- struct NotesMetadataResultSpec
- class NoteStoreServer

The NoteStoreServer class represents customizable server for NoteStore requests. It is primarily used for testing of QEverCloud.

- struct NoteVersionId
- class Optional
- class Printable
- struct PublicUserInfo
- struct Publishing
- class QAssociativeContainerConstReferenceWrapper
- class QAssociativeContainerReferenceWrapper
- struct RelatedContent
- struct RelatedContentImage
- struct RelatedQuery
- struct RelatedResult
- struct RelatedResultSpec
- struct Resource
- struct ResourceAttributes
- struct SavedSearch
- struct SavedSearchScope
- struct SharedNote
- struct SharedNotebook
- struct SharedNotebookRecipientSettings
- struct SharedNoteTemplate
- struct ShareRelationshipRestrictions
- struct ShareRelationships
- struct SyncChunk
- struct SyncChunkFilter
- struct SyncState
- struct Tag
- class ThriftException
- class ThriftExceptionData
- class Thumbnail

  The class is for downloading thumbnails for notes and resources from Evernote servers.
- struct UpdateNoteIfUsnMatchesResult
- struct User
- struct UserAttributes
- struct UserIdentity
- struct UserProfile
- class UserStoreServer

  The UserStoreServer class represents customizable server for UserStore requests. It is primarily used for testing of QEverCloud.
- struct UserUrls

Typedefs

- using IRetryPolicyPtr = std::shared_ptr< IRetryPolicy >
- using IDurableServicePtr = std::shared_ptr< IDurableService >
- using EverCloudExceptionDataPtr = std::shared_ptr< EverCloudExceptionData >
- using INoteStorePtr = std::shared_ptr< INoteStore >
- using IUserStorePtr = std::shared_ptr< IUserStore >
- using InvalidationSequenceNumber = qint64
- using IdentityID = qint64
- using UserID = qint32
- using Guid = QString
- using Timestamp = qint64
- using MessageEventID = qint64
- using MessageThreadID = qint64
- using ILoggerPtr = std::shared_ptr< ILogger >
- using IRequestContextPtr = std::shared_ptr< IRequestContext >

Enumerations

- enum class EDAMErrorCode {
  UNKNOWN = 1 , BAD_DATA_FORMAT = 2 , PERMISSION_DENIED = 3 , INTERNAL_ERROR = 4 ,
  DATA_REQUIRED = 5 , LIMIT_REACHED = 6 , QUOTA_REACHED = 7 , INVALID_AUTH = 8 ,
  AUTH_EXPIRED = 9 , DATA_CONFLICT = 10 , ENML_VALIDATION = 11 , SHARD_UNAVAILABLE = 12 ,
  LEN_TOO_SHORT = 13 , LEN_TOO_LONG = 14 , TOO_FEW = 15 , TOO_MANY = 16 ,
  UNSUPPORTED_OPERATION = 17 , TAKEN_DOWN = 18 , RATE_LIMIT_REACHED = 19 ,
  BUSINESS_SECURITY_LOGIN_REQUIRED = 20 ,
  DEVICE_LIMIT_REACHED = 21 , OPENID_ALREADY_TAKEN = 22 , INVALID_OPENID_TOKEN = 23 ,
  USER_NOT_ASSOCIATED = 24 ,
  USER_NOT_REGISTERED = 25 , USER_ALREADY_ASSOCIATED = 26 , ACCOUNT_CLEAR = 27 ,
  SSO_AUTHENTICATION_REQUIRED = 28 }
- enum class EDAMInvalidContactReason { BAD_ADDRESS , DUPLICATE_CONTACT , NO_CONNECTION }
- enum class ShareRelationshipPrivilegeLevel { READ_NOTEBOOK = 0 , READ_NOTEBOOK_PLUS_ACTIVITY = 10 , MODIFY_NOTEBOOK_PLUS_ACTIVITY = 20 , FULL_ACCESS = 30 }
- enum class PrivilegeLevel {
  NORMAL = 1 , PREMIUM = 3 , VIP = 5 , MANAGER = 7 ,
  SUPPORT = 8 , ADMIN = 9 }
- enum class ServiceLevel { BASIC = 1 , PLUS = 2 , PREMIUM = 3 , BUSINESS = 4 }
- enum class QueryFormat { USER = 1 , SEXP = 2 }
- enum class NoteSortOrder {
  CREATED = 1 , UPDATED = 2 , RELEVANCE = 3 , UPDATE_SEQUENCE_NUMBER = 4 ,
  TITLE = 5 }
- enum class PremiumOrderStatus {
  NONE = 0 , PENDING = 1 , ACTIVE = 2 , FAILED = 3 ,
  CANCELLATION_PENDING = 4 , CANCELED = 5 }
- enum class SharedNotebookPrivilegeLevel {
  READ_NOTEBOOK = 0 , MODIFY_NOTEBOOK_PLUS_ACTIVITY = 1 , READ_NOTEBOOK_PLUS_ACTIVITY = 2 , GROUP = 3 ,
  FULL_ACCESS = 4 , BUSINESS_FULL_ACCESS = 5 }
- enum class SharedNotePrivilegeLevel { READ_NOTE = 0 , MODIFY_NOTE = 1 , FULL_ACCESS = 2 }
- enum class SponsoredGroupRole { GROUP_MEMBER = 1 , GROUP_ADMIN = 2 , GROUP_OWNER = 3 }
- enum class BusinessUserRole { ADMIN = 1 , NORMAL = 2 }
- enum class BusinessUserStatus { ACTIVE = 1 , DEACTIVATED = 2 }
- enum class SharedNotebookInstanceRestrictions { ASSIGNED = 1 , NO_SHARED_NOTEBOOKS = 2 }
- enum class ReminderEmailConfig { DO_NOT_SEND = 1 , SEND_DAILY_EMAIL = 2 }
- enum class BusinessInvitationStatus { APPROVED = 0 , REQUESTED = 1 , REDEEMED = 2 }
- enum class ContactType {
  EVERNOTE = 1 , SMS = 2 , FACEBOOK = 3 , EMAIL = 4 ,
  TWITTER = 5 , LINKEDIN = 6 }
- enum class EntityType { NOTE = 1 , NOTEBOOK = 2 , WORKSPACE = 3 }
- enum class RecipientStatus { NOT_IN_MY_LIST = 1 , IN_MY_LIST = 2 , IN_MY_LIST_AND_DEFAULT_NOTEBOOK = 3 }
- enum class CanMoveToContainerStatus { CAN_BE_MOVED = 1 , INSUFFICIENT_ENTITY_PRIVILEGE = 2 , INSUFFICIENT_CONTAINER_PRIVILEGE = 3 }
- enum class RelatedContentType { NEWS_ARTICLE = 1 , PROFILE_PERSON = 2 , PROFILE_ORGANIZATION = 3 , REFERENCE_MATERIAL = 4 }
- enum class RelatedContentAccess { NOT_ACCESSIBLE = 0 , DIRECT_LINK_ACCESS_OK = 1 , DIRECT_LINK_LOGIN_REQUIRED = 2 , DIRECT_LINK_EMBEDDED_VIEW = 3 }
- enum class UserIdentityType { EVERNOTE_USERID = 1 , EMAIL = 2 , IDENTITYID = 3 }
- enum class LogLevel {
  Trace = 0 , Debug , Info , Warn ,
  Error }

Functions

- QEVERCLOUD_EXPORT IRetryPolicyPtr newRetryPolicy ()
- QEVERCLOUD_EXPORT IRetryPolicyPtr nullRetryPolicy ()
- QEVERCLOUD_EXPORT IDurableServicePtr newDurableService (IRetryPolicyPtr={}, IRequestContextPtr={})
- uint qHash (EDAMErrorCode value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const EDAMErrorCode value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const EDAMErrorCode value)
- uint qHash (EDAMInvalidContactReason value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const EDAMInvalidContactReason value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const EDAMInvalidContactReason value)
- uint qHash (ShareRelationshipPrivilegeLevel value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const ShareRelationshipPrivilegeLevel value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const ShareRelationshipPrivilegeLevel value)
- uint qHash (PrivilegeLevel value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const PrivilegeLevel value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const PrivilegeLevel value)
- uint qHash (ServiceLevel value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const ServiceLevel value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const ServiceLevel value)
- uint qHash (QueryFormat value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const QueryFormat value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const QueryFormat value)
- uint qHash (NoteSortOrder value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const NoteSortOrder value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const NoteSortOrder value)
- uint qHash (PremiumOrderStatus value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const PremiumOrderStatus value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const PremiumOrderStatus value)
- uint qHash (SharedNotebookPrivilegeLevel value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const SharedNotebookPrivilegeLevel value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const SharedNotebookPrivilegeLevel value)
- uint qHash (SharedNotePrivilegeLevel value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const SharedNotePrivilegeLevel value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const SharedNotePrivilegeLevel value)
- uint qHash (SponsoredGroupRole value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const SponsoredGroupRole value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const SponsoredGroupRole value)
- uint qHash (BusinessUserRole value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const BusinessUserRole value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const BusinessUserRole value)
- uint qHash (BusinessUserStatus value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const BusinessUserStatus value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const BusinessUserStatus value)
- uint qHash (SharedNotebookInstanceRestrictions value)

- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const SharedNotebookInstanceRestrictions value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const SharedNotebookInstanceRestrictions value)
- uint qHash (ReminderEmailConfig value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const ReminderEmailConfig value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const ReminderEmailConfig value)
- uint qHash (BusinessInvitationStatus value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const BusinessInvitationStatus value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const BusinessInvitationStatus value)
- uint qHash (ContactType value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const ContactType value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const ContactType value)
- uint qHash (EntityType value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const EntityType value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const EntityType value)
- uint qHash (RecipientStatus value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const RecipientStatus value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const RecipientStatus value)
- uint qHash (CanMoveToContainerStatus value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const CanMoveToContainerStatus value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const CanMoveToContainerStatus value)
- uint qHash (RelatedContentType value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const RelatedContentType value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const RelatedContentType value)
- uint qHash (RelatedContentAccess value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const RelatedContentAccess value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const RelatedContentAccess value)
- uint qHash (UserIdentityType value)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const UserIdentityType value)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const UserIdentityType value)
- QEVERCLOUD_EXPORT INoteStore * newNoteStore (QString noteStoreUrl={}, IRequestContextPtr ctx={}, QObject *parent=nullptr, IRetryPolicyPtr retryPolicy={})
- QEVERCLOUD_EXPORT IUserStore * newUserStore (QString userStoreUrl={}, IRequestContextPtr ctx={}, QObject *parent=nullptr, IRetryPolicyPtr retryPolicy={})
- QEVERCLOUD_EXPORT QNetworkProxy evernoteNetworkProxy ()
- QEVERCLOUD_EXPORT void setEvernoteNetworkProxy (QNetworkProxy proxy)
- QEVERCLOUD_EXPORT void resetEvernoteNetworkProxy ()
- QEVERCLOUD_EXPORT int libraryVersion ()
- QEVERCLOUD_EXPORT void initializeQEverCloud ()
- template<class Container > QAssociativeContainerReferenceWrapper< Container > toRange (Container &container)
- template<class Container > QAssociativeContainerConstReferenceWrapper< Container > toRange (const Container &container)
- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &out, const LogLevel level)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &out, const LogLevel level)
- QEVERCLOUD_EXPORT ILoggerPtr logger ()
- QEVERCLOUD_EXPORT void setLogger (ILoggerPtr logger)
- QEVERCLOUD_EXPORT ILoggerPtr nullLogger ()
- QEVERCLOUD_EXPORT ILoggerPtr newStdErrLogger (LogLevel level=LogLevel::Warn)

- void setNonceGenerator (quint64(*nonceGenerator)())

  Sets the function to use for nonce generation for OAuth authentication.
- QEVERCLOUD_EXPORT IRequestContextPtr newRequestContext (QString authenticationToken={}, qint64 requestTimeout=DEFAULT_REQUEST_TIMEOUT_MSEC, bool increaseRequestTimeoutExponentially=DEFAULT_REQUEST_TIMEOUT_EXPONENTIAL_INCREASE, qint64 maxRequestTimeout=DEFAULT_MAX_REQUEST_TIMEOUT_MSEC, quint32 maxRequestRetryCount=DEFAULT_MAX_REQUEST_RETRY_COUNT, QList< QNetworkCookie > cookies={})

Variables

- class QEVERCLOUD_EXPORT EverCloudExceptionData
- QEVERCLOUD_EXPORT const qint32 EDAM_ATTRIBUTE_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_ATTRIBUTE_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_ATTRIBUTE_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_ATTRIBUTE_LIST_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_ATTRIBUTE_MAP_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_GUID_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_GUID_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_GUID_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_EMAIL_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_EMAIL_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_EMAIL_LOCAL_REGEX
- QEVERCLOUD_EXPORT const QString EDAM_EMAIL_DOMAIN_REGEX
- QEVERCLOUD_EXPORT const QString EDAM_EMAIL_REGEX
- QEVERCLOUD_EXPORT const QString EDAM_VAT_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_TIMEZONE_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_TIMEZONE_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_TIMEZONE_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_MIME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_MIME_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_MIME_REGEX
- QEVERCLOUD_EXPORT const QString EDAM_MIME_TYPE_GIF
- QEVERCLOUD_EXPORT const QString EDAM_MIME_TYPE_JPEG
- QEVERCLOUD_EXPORT const QString EDAM_MIME_TYPE_PNG
- QEVERCLOUD_EXPORT const QString EDAM_MIME_TYPE_TIFF
- QEVERCLOUD_EXPORT const QString EDAM_MIME_TYPE_BMP
- QEVERCLOUD_EXPORT const QString EDAM_MIME_TYPE_WAV
- QEVERCLOUD_EXPORT const QString EDAM_MIME_TYPE_MP3
- QEVERCLOUD_EXPORT const QString EDAM_MIME_TYPE_AMR
- QEVERCLOUD_EXPORT const QString EDAM_MIME_TYPE_AAC
- QEVERCLOUD_EXPORT const QString EDAM_MIME_TYPE_M4A
- QEVERCLOUD_EXPORT const QString EDAM_MIME_TYPE_MP4_VIDEO
- QEVERCLOUD_EXPORT const QString EDAM_MIME_TYPE_INK
- QEVERCLOUD_EXPORT const QString EDAM_MIME_TYPE_PDF
- QEVERCLOUD_EXPORT const QString EDAM_MIME_TYPE_DEFAULT
- QEVERCLOUD_EXPORT const QSet< QString > EDAM_MIME_TYPES
- QEVERCLOUD_EXPORT const QSet< QString > EDAM_INDEXABLE_RESOURCE_MIME_TYPES
- QEVERCLOUD_EXPORT const QSet< QString > EDAM_INDEXABLE_PLAINTEXT_MIME_TYPES
- QEVERCLOUD_EXPORT const qint32 EDAM_SEARCH_QUERY_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_SEARCH_QUERY_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_SEARCH_QUERY_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_HASH_LEN
- QEVERCLOUD_EXPORT const qint32 EDAM_USER_USERNAME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_USER_USERNAME_LEN_MAX

- QEVERCLOUD_EXPORT const QString EDAM_USER_USERNAME_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_USER_NAME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_USER_NAME_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_USER_NAME_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_TAG_NAME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_TAG_NAME_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_TAG_NAME_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_TITLE_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_TITLE_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_NOTE_TITLE_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_CONTENT_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_CONTENT_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_APPLICATIONDATA_NAME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_APPLICATIONDATA_NAME_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_APPLICATIONDATA_VALUE_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_APPLICATIONDATA_VALUE_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_APPLICATIONDATA_ENTRY_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_APPLICATIONDATA_NAME_REGEX
- QEVERCLOUD_EXPORT const QString EDAM_APPLICATIONDATA_VALUE_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTEBOOK_NAME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTEBOOK_NAME_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_NOTEBOOK_NAME_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTEBOOK_STACK_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTEBOOK_STACK_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_NOTEBOOK_STACK_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_WORKSPACE_NAME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_WORKSPACE_NAME_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_WORKSPACE_DESCRIPTION_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_WORKSPACE_NAME_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_PUBLISHING_URI_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_PUBLISHING_URI_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_PUBLISHING_URI_REGEX
- QEVERCLOUD_EXPORT const QSet< QString > EDAM_PUBLISHING_URI_PROHIBITED
- QEVERCLOUD_EXPORT const qint32 EDAM_PUBLISHING_DESCRIPTION_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_PUBLISHING_DESCRIPTION_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_PUBLISHING_DESCRIPTION_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_SAVED_SEARCH_NAME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_SAVED_SEARCH_NAME_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_SAVED_SEARCH_NAME_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_USER_PASSWORD_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_USER_PASSWORD_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_USER_PASSWORD_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_BUSINESS_URI_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_BUSINESS_MARKETING_CODE_REGEX_PATTERN
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_TAGS_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_RESOURCES_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_USER_TAGS_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_BUSINESS_TAGS_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_USER_SAVED_SEARCHES_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_USER_NOTES_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_BUSINESS_NOTES_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_USER_NOTEBOOKS_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_USER_WORKSPACES_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_BUSINESS_NOTEBOOKS_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_BUSINESS_WORKSPACES_MAX

- QEVERCLOUD_EXPORT const qint32 EDAM_USER_RECENT_MAILED_ADDRESSES_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_USER_MAIL_LIMIT_DAILY_FREE
- QEVERCLOUD_EXPORT const qint32 EDAM_USER_MAIL_LIMIT_DAILY_PREMIUM
- QEVERCLOUD_EXPORT const qint64 EDAM_USER_UPLOAD_LIMIT_FREE
- QEVERCLOUD_EXPORT const qint64 EDAM_USER_UPLOAD_LIMIT_PREMIUM
- QEVERCLOUD_EXPORT const qint64 EDAM_USER_UPLOAD_LIMIT_BUSINESS_FIRST_MONTH
- QEVERCLOUD_EXPORT const qint64 EDAM_USER_UPLOAD_LIMIT_BUSINESS_NEXT_MONTH
- QEVERCLOUD_EXPORT const qint64 EDAM_USER_UPLOAD_LIMIT_PLUS
- QEVERCLOUD_EXPORT const qint64 EDAM_USER_UPLOAD_SURVEY_THRESHOLD
- QEVERCLOUD_EXPORT const qint64 EDAM_USER_UPLOAD_LIMIT_BUSINESS
- QEVERCLOUD_EXPORT const qint64 EDAM_USER_UPLOAD_LIMIT_BUSINESS_PER_USER
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_SIZE_MAX_FREE
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_SIZE_MAX_PREMIUM
- QEVERCLOUD_EXPORT const qint32 EDAM_RESOURCE_SIZE_MAX_FREE
- QEVERCLOUD_EXPORT const qint32 EDAM_RESOURCE_SIZE_MAX_PREMIUM
- QEVERCLOUD_EXPORT const qint32 EDAM_USER_LINKED_NOTEBOOK_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_USER_LINKED_NOTEBOOK_MAX_PREMIUM
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTEBOOK_BUSINESS_SHARED_NOTEBOOK_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTEBOOK_PERSONAL_SHARED_NOTEBOOK_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_BUSINESS_SHARED_NOTE_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_PERSONAL_SHARED_NOTE_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_CONTENT_CLASS_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_CONTENT_CLASS_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_NOTE_CONTENT_CLASS_REGEX
- QEVERCLOUD_EXPORT const QString EDAM_HELLO_APP_CONTENT_CLASS_PREFIX
- QEVERCLOUD_EXPORT const QString EDAM_FOOD_APP_CONTENT_CLASS_PREFIX
- QEVERCLOUD_EXPORT const QString EDAM_CONTENT_CLASS_HELLO_ENCOUNTER
- QEVERCLOUD_EXPORT const QString EDAM_CONTENT_CLASS_HELLO_PROFILE
- QEVERCLOUD_EXPORT const QString EDAM_CONTENT_CLASS_FOOD_MEAL
- QEVERCLOUD_EXPORT const QString EDAM_CONTENT_CLASS_SKITCH_PREFIX
- QEVERCLOUD_EXPORT const QString EDAM_CONTENT_CLASS_SKITCH
- QEVERCLOUD_EXPORT const QString EDAM_CONTENT_CLASS_SKITCH_PDF
- QEVERCLOUD_EXPORT const QString EDAM_CONTENT_CLASS_PENULTIMATE_PREFIX
- QEVERCLOUD_EXPORT const QString EDAM_CONTENT_CLASS_PENULTIMATE_NOTEBOOK
- QEVERCLOUD_EXPORT const QString EDAM_SOURCE_APPLICATION_POSTIT
- QEVERCLOUD_EXPORT const QString EDAM_SOURCE_APPLICATION_MOLESKINE
- QEVERCLOUD_EXPORT const QString EDAM_SOURCE_APPLICATION_EN_SCANSNAP
- QEVERCLOUD_EXPORT const QString EDAM_SOURCE_APPLICATION_EWC
- QEVERCLOUD_EXPORT const QString EDAM_SOURCE_APPLICATION_ANDROID_SHARE_EXTENSION
- QEVERCLOUD_EXPORT const QString EDAM_SOURCE_APPLICATION_IOS_SHARE_EXTENSION
- QEVERCLOUD_EXPORT const QString EDAM_SOURCE_APPLICATION_WEB_CLIPPER
- QEVERCLOUD_EXPORT const QString EDAM_SOURCE_OUTLOOK_CLIPPER
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_TITLE_QUALITY_UNTITLED
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_TITLE_QUALITY_LOW
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_TITLE_QUALITY_MEDIUM
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_TITLE_QUALITY_HIGH
- QEVERCLOUD_EXPORT const qint32 EDAM_RELATED_PLAINTEXT_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_RELATED_PLAINTEXT_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_RELATED_MAX_NOTES
- QEVERCLOUD_EXPORT const qint32 EDAM_RELATED_MAX_NOTEBOOKS
- QEVERCLOUD_EXPORT const qint32 EDAM_RELATED_MAX_TAGS
- QEVERCLOUD_EXPORT const qint32 EDAM_RELATED_MAX_EXPERTS
- QEVERCLOUD_EXPORT const qint32 EDAM_RELATED_MAX_RELATED_CONTENT
- QEVERCLOUD_EXPORT const qint32 EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_LEN_MAX

- QEVERCLOUD_EXPORT const QString EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_BUSINESS_PHONE_NUMBER_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_PREFERENCE_NAME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_PREFERENCE_NAME_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_PREFERENCE_VALUE_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_PREFERENCE_VALUE_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_MAX_PREFERENCES
- QEVERCLOUD_EXPORT const qint32 EDAM_MAX_VALUES_PER_PREFERENCE
- QEVERCLOUD_EXPORT const qint32 EDAM_PREFERENCE_ONLY_ONE_VALUE_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_PREFERENCE_NAME_REGEX
- QEVERCLOUD_EXPORT const QString EDAM_PREFERENCE_VALUE_REGEX
- QEVERCLOUD_EXPORT const QString EDAM_PREFERENCE_ONLY_ONE_VALUE_REGEX
- QEVERCLOUD_EXPORT const QString EDAM_PREFERENCE_SHORTCUTS
- QEVERCLOUD_EXPORT const QString EDAM_PREFERENCE_BUSINESS_DEFAULT_NOTEBOOK
- QEVERCLOUD_EXPORT const QString EDAM_PREFERENCE_BUSINESS_QUICKNOTE
- QEVERCLOUD_EXPORT const qint32 EDAM_PREFERENCE_SHORTCUTS_MAX_VALUES
- QEVERCLOUD_EXPORT const qint32 EDAM_DEVICE_ID_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_DEVICE_ID_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_DEVICE_DESCRIPTION_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_DEVICE_DESCRIPTION_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_SEARCH_SUGGESTIONS_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_SEARCH_SUGGESTIONS_PREFIX_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_SEARCH_SUGGESTIONS_PREFIX_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 EDAM_FIND_CONTACT_DEFAULT_MAX_RESULTS
- QEVERCLOUD_EXPORT const qint32 EDAM_FIND_CONTACT_MAX_RESULTS
- QEVERCLOUD_EXPORT const qint32 EDAM_NOTE_LOCK_VIEWERS_NOTES_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_GET_ORDERS_MAX_RESULTS
- QEVERCLOUD_EXPORT const qint32 EDAM_MESSAGE_BODY_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_MESSAGE_BODY_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_MESSAGE_RECIPIENTS_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_MESSAGE_ATTACHMENTS_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_MESSAGE_ATTACHMENT_TITLE_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_MESSAGE_ATTACHMENT_TITLE_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_MESSAGE_ATTACHMENT_SNIPPET_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_MESSAGE_ATTACHMENT_SNIPPET_REGEX
- QEVERCLOUD_EXPORT const qint32 EDAM_USER_PROFILE_PHOTO_MAX_BYTES
- QEVERCLOUD_EXPORT const qint32 EDAM_PROMOTION_ID_LEN_MAX
- QEVERCLOUD_EXPORT const QString EDAM_PROMOTION_ID_REGEX
- QEVERCLOUD_EXPORT const qint16 EDAM_APP_RATING_MIN
- QEVERCLOUD_EXPORT const qint16 EDAM_APP_RATING_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_SNIPPETS_NOTES_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_CONNECTED_IDENTITY_REQUEST_MAX
- QEVERCLOUD_EXPORT const qint32 EDAM_OPEN_ID_ACCESS_TOKEN_MAX
- QEVERCLOUD_EXPORT const QString CLASSIFICATION_RECIPE_USER_NON_RECIPE
- QEVERCLOUD_EXPORT const QString CLASSIFICATION_RECIPE_USER_RECIPE
- QEVERCLOUD_EXPORT const QString CLASSIFICATION_RECIPE_SERVICE_RECIPE
- QEVERCLOUD_EXPORT const QString EDAM_NOTE_SOURCE_WEB_CLIP
- QEVERCLOUD_EXPORT const QString EDAM_NOTE_SOURCE_WEB_CLIP_SIMPLIFIED
- QEVERCLOUD_EXPORT const QString EDAM_NOTE_SOURCE_MAIL_CLIP
- QEVERCLOUD_EXPORT const QString EDAM_NOTE_SOURCE_MAIL_SMTP_GATEWAY
- QEVERCLOUD_EXPORT const qint16 EDAM_VERSION_MAJOR
- QEVERCLOUD_EXPORT const qint16 EDAM_VERSION_MINOR

6.1.1 Detailed Description

Original work: Copyright (c) 2014 Sergey Skoblikov Modified work: Copyright (c) 2015-2019 Dmitry Ivanov

This file is a part of QEverCloud project and is distributed under the terms of MIT license: https://opensource.org/licenses/MIT

Copyright (c) 2019 Dmitry Ivanov

This file is a part of QEverCloud project and is distributed under the terms of MIT license: https://opensource.org/licenses/MIT

Original work: Copyright (c) 2014 Sergey Skoblikov Modified work: Copyright (c) 2015-2020 Dmitry Ivanov

This file is a part of QEverCloud project and is distributed under the terms of MIT license: https://opensource.org/licenses/MIT

This file was generated from Evernote Thrift API

Original work: Copyright (c) 2014 Sergey Skoblikov Modified work: Copyright (c) 2015-2020 Dmitry Ivanov

This file is a part of QEverCloud project and is distributed under the terms of MIT license: https://opensource.org/licenses/MIT All the library lives in this namespace.

Copyright (c) 2019-2020 Dmitry Ivanov

This file is a part of QEverCloud project and is distributed under the terms of MIT license: https://opensource.org/licenses/MIT

Copyright (c) 2016-2020 Dmitry Ivanov

This file is a part of QEverCloud project and is distributed under the terms of MIT license: https://opensource.org/licenses/MIT

Original work: Copyright (c) 2014 Sergey Skoblikov Modified work: Copyright (c) 2015-2020 Dmitry Ivanov

This file is a part of QEverCloud project and is distributed under the terms of MIT license: https://opensource.org/licenses/MIT

6.1.2 Typedef Documentation

6.1.2.1 EverCloudExceptionDataPtr

```
using qevercloud::EverCloudExceptionDataPtr = typedef std::shared_ptr<EverCloudExceptionData>
```

6.1.2.2 Guid

```
using qevercloud::Guid = typedef QString
```

Most data elements within a user's account (e.g. notebooks, notes, tags, resources, etc.) are internally referred to using a globally unique identifier that is written in a standard string format. For example:

"8743428c-ef91-4d05-9e7c-4a2e856e813a"

The internal components of the GUID are not given any particular meaning: only the entire string is relevant as a unique identifier.

6.1.2.3 IdentityID

```
using qevercloud::IdentityID = typedef qint64
```

A type alias for the primary identifiers for Identity objects.

6.1.2.4 IDurableServicePtr

```
using qevercloud::IDurableServicePtr = typedef std::shared_ptr<IDurableService>
```

6.1.2.5 ILoggerPtr

```
using qevercloud::ILoggerPtr = typedef std::shared_ptr<ILogger>
```

6.1.2.6 INoteStorePtr

```
using qevercloud::INoteStorePtr = typedef std::shared_ptr<INoteStore>
```

6.1.2.7 InvalidationSequenceNumber

```
using qevercloud::InvalidationSequenceNumber = typedef qint64
```

A monotonically incrementing number on each shard that identifies a cross shard cache invalidation event.

6.1.2.8 IRequestContextPtr

```
using qevercloud::IRequestContextPtr = typedef std::shared_ptr<IRequestContext>
```

6.1.2.9 IRetryPolicyPtr

```
using qevercloud::IRetryPolicyPtr = typedef std::shared_ptr<IRetryPolicy>
```

6.1.2.10 IUserStorePtr

```
using qevercloud::IUserStorePtr = typedef std::shared_ptr<IUserStore>
```

6.1.2.11 MessageEventID

```
using qevercloud::MessageEventID = typedef qint64
```

A sequence number for the MessageStore subsystem.

6.1.2.12 MessageThreadID

```
using qevercloud::MessageThreadID = typedef qint64
```

A type alias for the primary identifiers for MessageThread objects.

6.1.2.13 Timestamp

```
using qevercloud::Timestamp = typedef qint64
```

An Evernote Timestamp is the date and time of an event in UTC time. This is expressed as a specific number of milliseconds since the standard base "epoch" of:

January 1, 1970, 00:00:00 GMT

NOTE: the time is expressed at the resolution of milliseconds, but the value is only precise to the level of seconds. This means that the last three (decimal) digits of the timestamp will be '000'.

The Thrift IDL specification does not include a native date/time type, so this value is used instead.

The service will accept timestamp values (e.g. for Note created and update times) between 1000-01-01 and 9999-12-31

6.1.2.14 UserID

```
using qevercloud::UserID = typedef qint32
```

Every Evernote account is assigned a unique numeric identifier which will not change for the life of the account. This is independent of the (string-based) "username" which is known by the user for login purposes. The user should have no reason to know their UserID.

6.1.3 Enumeration Type Documentation

6.1.3.1 BusinessInvitationStatus

```
enum class qevercloud::BusinessInvitationStatus  [strong]
```

An enumeration defining the possible states of a BusinessInvitation.

APPROVED: The invitation was created or approved by a business admin and may be redeemed by the invited email.

REQUESTED: The invitation was requested by a non-admin member of the business and must be approved by an admin before it may be redeemed. Invitations in this state do not count against a business' seat limit.

REDEEMED: The invitation has already been redeemed. Invitations in this state do not count against a business' seat limit.

Enumerator

APPROVED	
REQUESTED	
REDEEMED	

6.1.3.2 BusinessUserRole

`enum class qevercloud::BusinessUserRole [strong]`

Enumeration of the roles that a User can have within an Evernote Business account.

ADMIN: The user is an administrator of the Evernote Business account.

NORMAL: The user is a regular user within the Evernote Business account.

Enumerator

ADMIN	
NORMAL	

6.1.3.3 BusinessUserStatus

`enum class qevercloud::BusinessUserStatus [strong]`

The BusinessUserStatus indicates the status of the user in the business.

A BusinessUser will typically start as ACTIVE. Only ACTIVE users can authenticate to the Business.

ACTIVE

The business user can authenticate to and access the business.

DEACTIVATED

The business user has been deactivated and cannot access the business

Enumerator

ACTIVE	
DEACTIVATED	

6.1.3.4 CanMoveToContainerStatus

`enum class qevercloud::CanMoveToContainerStatus [strong]`

This enumeration defines the possible types of canMoveToContainer outcomes.

An outdated client is expected to signal a "Cannot Move, Please Upgrade To Learn Why" like response to the user if an unknown enumeration value is received.

CAN_BE_MOVED Can move Notebook to Workspace.

INSUFFICIENT_ENTITY_PRIVILEGE Can not move Notebook to Workspace, because either: a) Notebook not in Workspace and insufficient privilege on Notebook or b) Notebook in Workspace and membership on Workspace with insufficient privilege for move

INSUFFICIENT_CONTAINER_PRIVILEGE Notebook in Workspace and no membership on Workspace.

Enumerator

CAN_BE_MOVED	
INSUFFICIENT_ENTITY_PRIVILEGE	
INSUFFICIENT_CONTAINER_PRIVILEGE	

6.1.3.5 ContactType

`enum class qevercloud::ContactType [strong]`

What kinds of Contacts does the Evernote service know about?

Enumerator

EVERNOTE	
SMS	
FACEBOOK	
EMAIL	
TWITTER	
LINKEDIN	

6.1.3.6 EDAMErrorCode

`enum class qevercloud::EDAMErrorCode [strong]`

Numeric codes indicating the type of error that occurred on the service.

UNKNOWN No information available about the error

BAD_DATA_FORMAT The format of the request data was incorrect

PERMISSION_DENIED Not permitted to perform action

INTERNAL_ERROR Unexpected problem with the service

DATA_REQUIRED A required parameter/field was absent

LIMIT_REACHED Operation denied due to data model limit

QUOTA_REACHED Operation denied due to user storage limit

INVALID_AUTH Username and/or password incorrect

AUTH_EXPIRED Authentication token expired

DATA_CONFLICT Change denied due to data model conflict

ENML_VALIDATION Content of submitted note was malformed

SHARD_UNAVAILABLE Service shard with account data is temporarily down

LEN_TOO_SHORT Operation denied due to data model limit, where something such as a string length was too short

LEN_TOO_LONG Operation denied due to data model limit, where something such as a string length was too long

TOO_FEW Operation denied due to data model limit, where there were too few of something.

TOO_MANY Operation denied due to data model limit, where there were too many of something.

UNSUPPORTED_OPERATION Operation denied because it is currently unsupported.

TAKEN_DOWN Operation denied because access to the corresponding object is prohibited in response to a take-down notice.

RATE_LIMIT_REACHED Operation denied because the calling application has reached its hourly API call limit for this user.

BUSINESS_SECURITY_LOGIN_REQUIRED Access to a business account has been denied because the user must complete additional steps in order to comply with business security requirements.

DEVICE_LIMIT_REACHED Operation denied because the user has exceeded their maximum allowed number of devices.

OPENID_ALREADY_TAKEN Operation failed because the Open ID is already associated with another user.

INVALID_OPENID_TOKEN Operation denied because the Open ID token is invalid. Please re-issue a valid token.

USER_NOT_REGISTERED There is no Evernote user associated with this OpenID account, and no Evernote user with a matching email

USER_NOT_ASSOCIATED There is no Evernote user associated with this OpenID account, but Evernote user with matching email exists

USER_ALREADY_ASSOCIATED Evernote user is already associated with this provider using a different email address.

ACCOUNT_CLEAR The user's account has been disabled. Clients should deal with this errorCode by logging the user out and purging all locally saved content, including local edits not yet pushed to the server.

SSO_AUTHENTICATION_REQUIRED SSO authentication is the only type of authentication allowed for the user's account. This error is thrown when the user attempts to authenticate by another method (password, OpenId, etc).

Enumerator

UNKNOWN	
BAD_DATA_FORMAT	
PERMISSION_DENIED	
INTERNAL_ERROR	
DATA_REQUIRED	
LIMIT_REACHED	
QUOTA_REACHED	
INVALID_AUTH	
AUTH_EXPIRED	
DATA_CONFLICT	
ENML_VALIDATION	
SHARD_UNAVAILABLE	
LEN_TOO_SHORT	
LEN_TOO_LONG	
TOO_FEW	
TOO_MANY	
UNSUPPORTED_OPERATION	
TAKEN_DOWN	
RATE_LIMIT_REACHED	
BUSINESS_SECURITY_LOGIN_REQUIRED	

Enumerator

DEVICE_LIMIT_REACHED	
OPENID_ALREADY_TAKEN	
INVALID_OPENID_TOKEN	
USER_NOT_ASSOCIATED	
USER_NOT_REGISTERED	
USER_ALREADY_ASSOCIATED	
ACCOUNT_CLEAR	
SSO_AUTHENTICATION_REQUIRED	

6.1.3.7 EDAMInvalidContactReason

`enum class qevercloud::EDAMInvalidContactReason [strong]`

An enumeration that provides a reason for why a given contact was invalid, for example, as thrown via an EDAMInvalidContactsException.

BAD_ADDRESS The contact information does not represent a valid address for a recipient. Clients should be validating and normalizing contacts, so receiving this error code commonly represents a client error.

DUPLICATE_CONTACT If the method throwing this exception accepts a list of contacts, this error code indicates that the given contact is a duplicate of another contact in the list. Note that the server may clean up contacts, and that this cleanup occurs before checking for duplication. Receiving this error is commonly an indication of a client issue, since client should be normalizing contacts and removing duplicates. All instances that are duplicates are returned. For example, if a list of 5 contacts has the same e-mail address twice, the two conflicting e-mail address contacts will be returned.

NO_CONNECTION Indicates that the given contact, an Evernote type contact, is not connected to the user for which the call is being made. It is possible that clients are out of sync with the server and should re-synchronize their identities and business user state. See Identity.userConnected for more information on user connections.

Note that if multiple reasons may apply, only one is returned. The precedence order is BAD_ADDRESS, DUPLICATE_CONTACT, NO_CONNECTION, meaning that if a contact has a bad address and is also duplicated, it will be returned as a BAD_ADDRESS.

Enumerator

BAD_ADDRESS	
DUPLICATE_CONTACT	
NO_CONNECTION	

6.1.3.8 EntityType

`enum class qevercloud::EntityType [strong]`

Entity types

Enumerator

NOTE	
NOTEBOOK	
WORKSPACE	

6.1.3.9 LogLevel

`enum class qevercloud::LogLevel [strong]`

Enumerator

Trace	
Debug	
Info	
Warn	
Error	

6.1.3.10 NoteSortOrder

`enum class qevercloud::NoteSortOrder [strong]`

This enumeration defines the possible sort ordering for notes when they are returned from a search result.

Enumerator

CREATED	
UPDATED	
RELEVANCE	
UPDATE_SEQUENCE_NUMBER	
TITLE	

6.1.3.11 PremiumOrderStatus

`enum class qevercloud::PremiumOrderStatus [strong]`

This enumeration defines the possible states of a premium account

NONE: the user has never attempted to become a premium subscriber

PENDING: the user has requested a premium account but their charge has not been confirmed

ACTIVE: the user has been charged and their premium account is in good standing

FAILED: the system attempted to charge the was denied. We will periodically attempt to re-validate their order.

CANCELLATION_PENDING: the user has requested that no further charges be made but the current account is still active.

CANCELED: the premium account was canceled either because of failure to pay or user cancelation. No more attempts will be made to activate the account.

Enumerator

NONE	
PENDING	
ACTIVE	
FAILED	
CANCELLATION_PENDING	
CANCELED	

6.1.3.12 PrivilegeLevel

`enum class qevercloud::PrivilegeLevel [strong]`

This enumeration defines the possible permission levels for a user. Free accounts will have a level of NORMAL and paid Premium accounts will have a level of PREMIUM.

Enumerator

NORMAL	
PREMIUM	
VIP	
MANAGER	
SUPPORT	
ADMIN	

6.1.3.13 QueryFormat

`enum class qevercloud::QueryFormat [strong]`

Every search query is specified as a sequence of characters. Currently, only the USER query format is supported.

Enumerator

USER	
SEXP	

6.1.3.14 RecipientStatus

`enum class qevercloud::RecipientStatus [strong]`

This enumeration defines the possible states that a notebook can be in for a recipient. It encompasses the "inMyList" boolean and default notebook status.

NOT_IN_MY_LIST The notebook is not in the recipient's list (not "joined").

IN_MY_LIST The notebook is in the recipient's notebook list (formerly, we would say that the recipient has "joined" the notebook)

IN_MY_LIST_AND_DEFAULT_NOTEBOOK The same as IN_MY_LIST and this notebook is the user's default notebook.

Enumerator

NOT_IN_MY_LIST	
IN_MY_LIST	
IN_MY_LIST_AND_DEFAULT_NOTEBOOK	

6.1.3.15 RelatedContentAccess

`enum class qevercloud::RelatedContentAccess [strong]`

This enumeration defines the possible ways to access related content.

NOT_ACCESSIBLE: The content is not accessible given the user's privilege level, but still worth showing as a snippet. The content url may point to a webpage that explains why not, or explains how to access that content.

DIRECT_LINK_ACCESS_OK: The content is accessible directly, and no additional login is required.

DIRECT_LINK_LOGIN_REQUIRED: The content is accessible directly, but an additional login is required.

DIRECT_LINK_EMBEDDED_VIEW: The content is accessible directly, and should be shown in an embedded web view. If the URL refers to a secured location under our control (for example, https://www.evernote.com/<smth>), the client may include user-specific authentication credentials with the request.

Enumerator

NOT_ACCESSIBLE	
DIRECT_LINK_ACCESS_OK	
DIRECT_LINK_LOGIN_REQUIRED	
DIRECT_LINK_EMBEDDED_VIEW	

6.1.3.16 RelatedContentType

`enum class qevercloud::RelatedContentType [strong]`

This enumeration defines the possible types of related content.

NEWS_ARTICLE: This related content is a news article PROFILE_PERSON: This match refers to the profile of an individual person PROFILE_ORGANIZATION: This match refers to the profile of an organization REFERENCE_MATERIAL: This related content is material from reference works

Enumerator

NEWS_ARTICLE	
PROFILE_PERSON	
PROFILE_ORGANIZATION	
REFERENCE_MATERIAL	

6.1.3.17 ReminderEmailConfig

`enum class qevercloud::ReminderEmailConfig [strong]`

An enumeration describing the configuration state related to receiving reminder e-mails from the service. Reminder e-mails summarize notes based on their Note.attributes.reminderTime values.

DO_NOT_SEND: The user has selected to not receive reminder e-mail.

SEND_DAILY_EMAIL: The user has selected to receive reminder e-mail for those days when there is a reminder.

Enumerator

DO_NOT_SEND	
SEND_DAILY_EMAIL	

6.1.3.18 ServiceLevel

`enum class qevercloud::ServiceLevel [strong]`

This enumeration defines the possible tiers of service that a user may have. A ServiceLevel of BUSINESS signifies a business-only account, which can never be any other ServiceLevel.

Enumerator

BASIC	
PLUS	
PREMIUM	
BUSINESS	

6.1.3.19 SharedNotebookInstanceRestrictions

`enum class qevercloud::SharedNotebookInstanceRestrictions [strong]`

An enumeration describing restrictions on the domain of shared notebook instances that are valid for a given operation, as used, for example, in NotebookRestrictions.

ASSIGNED: The domain consists of shared notebooks that belong, or are assigned, to the recipient.

NO_SHARED_NOTEBOOKS: No shared notebooks are applicable to the operation.

Enumerator

ASSIGNED	
NO_SHARED_NOTEBOOKS	

6.1.3.20 SharedNotebookPrivilegeLevel

`enum class qevercloud::SharedNotebookPrivilegeLevel [strong]`

Privilege levels for accessing shared notebooks.

Note that as of 2014-04, FULL_ACCESS is synonymous with BUSINESS_FULL_ACCESS. If a user is a member of a business and has FULL_ACCESS privileges, then they will automatically be granted BUSINESS_FULL_↩ ACCESS for notebooks in their business. This will happen implicitly when they attempt to access the corresponding notebooks of the business. BUSINESS_FULL_ACCESS is therefore deprecated.

READ_NOTEBOOK: Recipient is able to read the contents of the shared notebook but does not have access to information about other recipients of the notebook or the activity stream information.

MODIFY_NOTEBOOK_PLUS_ACTIVITY: Recipient has rights to read and modify the contents of the shared notebook, including the right to move notes to the trash and to create notes in the notebook. The recipient can also access information about other recipients and the activity stream.

READ_NOTEBOOK_PLUS_ACTIVITY: Recipient has READ_NOTEBOOK rights and can also access information about other recipients and the activity stream.

GROUP: If the user belongs to a group, such as a Business, that has a defined privilege level, use the privilege level of the group as the privilege for the individual.

FULL_ACCESS: Recipient has full rights to the shared notebook and recipient lists, including privilege to revoke and create invitations and to change privilege levels on invitations for individuals. For members of a business, FULL↩ _ACCESS privilege on business notebooks also grants the ability to change how the notebook will appear when shared with the business, including the rights to share and unshare the notebook with the business.

BUSINESS_FULL_ACCESS: Deprecated. See the note above about BUSINESS_FULL_ACCESS and FULL_↩ ACCESS being synonymous.

Enumerator

READ_NOTEBOOK	
MODIFY_NOTEBOOK_PLUS_ACTIVITY	
READ_NOTEBOOK_PLUS_ACTIVITY	
GROUP	
FULL_ACCESS	
BUSINESS_FULL_ACCESS	

6.1.3.21 SharedNotePrivilegeLevel

`enum class qevercloud::SharedNotePrivilegeLevel [strong]`

Privilege levels for accessing a shared note. All privilege levels convey "activity feed" access, which allows the recipient to access information about other recipients and the activity stream.

READ_NOTE: Recipient has rights to read the shared note.

MODIFY_NOTE: Recipient has all of the rights of READ_NOTE, plus rights to modify the shared note's content, title and resources. Other fields, including the notebook, tags and metadata, may not be modified.

FULL_ACCESS: Recipient has all of the rights of MODIFY_NOTE, plus rights to share the note with other users via email, public note links, and note sharing. Recipient may also update and remove other recipient's note sharing rights.

Enumerator

READ_NOTE	
MODIFY_NOTE	
FULL_ACCESS	

6.1.3.22 ShareRelationshipPrivilegeLevel

`enum class qevercloud::ShareRelationshipPrivilegeLevel [strong]`

Privilege levels for accessing shared notebooks.

READ_NOTEBOOK: Recipient is able to read the contents of the shared notebook but does not have access to information about other recipients of the notebook or the activity stream information.

READ_NOTEBOOK_PLUS_ACTIVITY: Recipient has READ_NOTEBOOK rights and can also access information about other recipients and the activity stream.

MODIFY_NOTEBOOK_PLUS_ACTIVITY: Recipient has rights to read and modify the contents of the shared notebook, including the right to move notes to the trash and to create notes in the notebook. The recipient can also access information about other recipients and the activity stream.

FULL_ACCESS: Recipient has full rights to the shared notebook and recipient lists, including privilege to revoke and create invitations and to change privilege levels on invitations for individuals. If the user is a member of the same group, (e.g. the same business) as the shared notebook, they will additionally be granted permissions to update the publishing status of the notebook.

Enumerator

READ_NOTEBOOK	
READ_NOTEBOOK_PLUS_ACTIVITY	
MODIFY_NOTEBOOK_PLUS_ACTIVITY	
FULL_ACCESS	

6.1.3.23 SponsoredGroupRole

`enum class qevercloud::SponsoredGroupRole [strong]`

Enumeration of the roles that a User can have within a sponsored group.

GROUP_MEMBER: The user is a member of the group with no special privileges.

GROUP_ADMIN: The user is an administrator within the group.

GROUP_OWNER: The user is the owner of the group.

Enumerator

GROUP_MEMBER	
GROUP_ADMIN	
GROUP_OWNER	

6.1.3.24 UserIdentityType

```
enum class qevercloud::UserIdentityType  [strong]
```

Enumerator

EVERNOTE_USERID	
EMAIL	
IDENTITYID	

6.1.4 Function Documentation

6.1.4.1 evernoteNetworkProxy()

```
QEVERCLOUD_EXPORT QNetworkProxy qevercloud::evernoteNetworkProxy ( )
```

Getter for network proxy settings used by QEverCloud. If none were set explicitly, returns the same result as QNetworkProxy::applicationProxy. Hence, QEverCloud uses the same proxy settings as the application which uses QEverCloud by default.

This function is thread-safe although internally it operates on a static object containing proxy settings so it's not recommended to read and write proxy settings concurrently to avoid contention for static object.

WARNING: when QEverCloud is built with QtWebEngine and some proxy settings different from QNetworkProxy↩ ::applicationProxy are set, the OAuth call which loads the web page would not use them; instead it would use proxy settings from QNetworkProxy::applicationProxy. This limitation is imposed by Qt: https://doc.qt.↩ io/qt-5/qtwebengine-overview.html#proxy-support

6.1.4.2 initializeQEverCloud()

```
QEVERCLOUD_EXPORT void qevercloud::initializeQEverCloud ( )
```

Initialization function for QEverCloud, needs to be called once before using the library. There is no harm if it is called multiple times

6.1.4.3 libraryVersion()

```
QEVERCLOUD_EXPORT int qevercloud::libraryVersion ( )
```

QEverCloud library version.

6.1.4.4 logger()

```
QEVERCLOUD_EXPORT ILoggerPtr qevercloud::logger ( )
```

6.1.4.5 newDurableService()

```
QEVERCLOUD_EXPORT IDurableServicePtr qevercloud::newDurableService (
            IRetryPolicyPtr  = {},
            IRequestContextPtr  = {} )
```

6.1.4.6 newNoteStore()

```
QEVERCLOUD_EXPORT INoteStore * qevercloud::newNoteStore (
            QString noteStoreUrl = {},
            IRequestContextPtr ctx = {},
            QObject * parent = nullptr,
            IRetryPolicyPtr retryPolicy = {} )
```

6.1.4.7 newRequestContext()

```
QEVERCLOUD_EXPORT IRequestContextPtr qevercloud::newRequestContext (
            QString authenticationToken = {},
            qint64 requestTimeout = DEFAULT_REQUEST_TIMEOUT_MSEC,
            bool increaseRequestTimeoutExponentially = DEFAULT_REQUEST_TIMEOUT_EXPONENTIAL_↩
INCREASE,
            qint64 maxRequestTimeout = DEFAULT_MAX_REQUEST_TIMEOUT_MSEC,
            quint32 maxRequestRetryCount = DEFAULT_MAX_REQUEST_RETRY_COUNT,
            QList< QNetworkCookie > cookies = {} )
```

6.1.4.8 newRetryPolicy()

```
QEVERCLOUD_EXPORT IRetryPolicyPtr qevercloud::newRetryPolicy ( )
```

6.1.4.9 newStdErrLogger()

```
QEVERCLOUD_EXPORT ILoggerPtr qevercloud::newStdErrLogger (
            LogLevel level = LogLevel::Warn )
```

6.1.4.10 newUserStore()

```
QEVERCLOUD_EXPORT IUserStore * qevercloud::newUserStore (
            QString userStoreUrl = {},
            IRequestContextPtr ctx = {},
            QObject * parent = nullptr,
            IRetryPolicyPtr retryPolicy = {} )
```

6.1.4.11 nullLogger()

```
QEVERCLOUD_EXPORT ILoggerPtr qevercloud::nullLogger ( )
```

6.1.4.12 nullRetryPolicy()

```
QEVERCLOUD_EXPORT IRetryPolicyPtr qevercloud::nullRetryPolicy ( )
```

6.1.4.13 operator<<() [1/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const BusinessInvitationStatus value )
```

6.1.4.14 operator<<() [2/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const BusinessUserRole value )
```

6.1.4.15 operator<<() [3/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const BusinessUserStatus value )
```

6.1.4.16 operator<<() [4/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const CanMoveToContainerStatus value )
```

6.1.4.17 operator<<() [5/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const ContactType value )
```

6.1.4.18 operator<<() [6/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const EDAMErrorCode value )
```

6.1.4.19 operator<<() [7/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const EDAMInvalidContactReason value )
```

6.1.4.20 operator<<() [8/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const EntityType value )
```

6.1.4.21 operator<<() [9/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const LogLevel level )
```

6.1.4.22 operator<<() [10/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const NoteSortOrder value )
```

6.1.4.23 operator<<() [11/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const PremiumOrderStatus value )
```

6.1.4.24 operator<<() [12/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const PrivilegeLevel value )
```

6.1.4.25 operator<<() [13/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const QueryFormat value )
```

6.1.4.26 operator<<() [14/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const RecipientStatus value )
```

6.1.4.27 operator<<() [15/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const RelatedContentAccess value )
```

6.1.4.28 operator<<() [16/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const RelatedContentType value )
```

6.1.4.29 operator<<() [17/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const ReminderEmailConfig value )
```

6.1.4.30 operator<<() [18/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const ServiceLevel value )
```

6.1.4.31 operator<<() [19/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const SharedNotebookInstanceRestrictions value )
```

6.1.4.32 operator<<() [20/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const SharedNotebookPrivilegeLevel value )
```

6.1.4.33 operator<<() [21/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const SharedNotePrivilegeLevel value )
```

6.1.4.34 operator<<() [22/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const ShareRelationshipPrivilegeLevel value )
```

6.1.4.35 operator<<() [23/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const SponsoredGroupRole value )
```

6.1.4.36 operator<<() [24/48]

```
QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (
            QDebug & out,
            const UserIdentityType value )
```

6.1.4.37 operator<<() [25/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const BusinessInvitationStatus value )
```

6.1.4.38 operator<<() [26/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const BusinessUserRole value )
```

6.1.4.39 operator<<() [27/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const BusinessUserStatus value )
```

6.1.4.40 operator<<() [28/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const CanMoveToContainerStatus value )
```

6.1.4.41 operator<<() [29/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const ContactType value )
```

6.1.4.42 operator<<() [30/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const EDAMErrorCode value )
```

6.1.4.43 operator<<() [31/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const EDAMInvalidContactReason value )
```

6.1.4.44 operator<<() [32/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const EntityType value )
```

6.1.4.45 operator<<() [33/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const LogLevel level )
```

6.1.4.46 operator<<() [34/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const NoteSortOrder value )
```

6.1.4.47 operator<<() [35/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const PremiumOrderStatus value )
```

6.1.4.48 operator<<() [36/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const PrivilegeLevel value )
```

6.1.4.49 operator<<() [37/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const QueryFormat value )
```

6.1.4.50 operator<<() [38/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const RecipientStatus value )
```

6.1.4.51 operator<<() [39/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const RelatedContentAccess value )
```

6.1.4.52 operator<<() [40/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const RelatedContentType value )
```

6.1.4.53 operator<<() [41/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const ReminderEmailConfig value )
```

6.1.4.54 operator<<() [42/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const ServiceLevel value )
```

6.1.4.55 operator<<() [43/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const SharedNotebookInstanceRestrictions value )
```

6.1.4.56 operator<<() [44/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const SharedNotebookPrivilegeLevel value )
```

6.1.4.57 operator<<() [45/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const SharedNotePrivilegeLevel value )
```

6.1.4.58 operator<<() [46/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const ShareRelationshipPrivilegeLevel value )
```

6.1.4.59 operator<<() [47/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const SponsoredGroupRole value )
```

6.1.4.60 operator<<() [48/48]

```
QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (
            QTextStream & out,
            const UserIdentityType value )
```

6.1.4.61 qHash() [1/23]

```
uint qevercloud::qHash (
            BusinessInvitationStatus value )  [inline]
```

6.1.4.62 qHash() [2/23]

```
uint qevercloud::qHash (
            BusinessUserRole value )  [inline]
```

6.1.4.63 qHash() [3/23]

```
uint qevercloud::qHash (
            BusinessUserStatus value )  [inline]
```

6.1.4.64 qHash() [4/23]

```
uint qevercloud::qHash (
            CanMoveToContainerStatus value )  [inline]
```

6.1.4.65 qHash() [5/23]

```
uint qevercloud::qHash (
            ContactType value )  [inline]
```

6.1.4.66 qHash() [6/23]

```
uint qevercloud::qHash (
            EDAMErrorCode value )  [inline]
```

6.1.4.67 qHash() [7/23]

```
uint qevercloud::qHash (
            EDAMInvalidContactReason value )  [inline]
```

6.1.4.68 qHash() [8/23]

```
uint qevercloud::qHash (
            EntityType value )  [inline]
```

6.1.4.69 qHash() [9/23]

```
uint qevercloud::qHash (
            NoteSortOrder value )  [inline]
```

6.1.4.70 qHash() [10/23]

```
uint qevercloud::qHash (
            PremiumOrderStatus value )  [inline]
```

6.1.4.71 qHash() [11/23]

```
uint qevercloud::qHash (
            PrivilegeLevel value )  [inline]
```

6.1.4.72 qHash() [12/23]

```
uint qevercloud::qHash (
            QueryFormat value )  [inline]
```

6.1.4.73 qHash() [13/23]

```
uint qevercloud::qHash (
            RecipientStatus value )  [inline]
```

6.1.4.74 qHash() [14/23]

```
uint qevercloud::qHash (
            RelatedContentAccess value )  [inline]
```

6.1.4.75 qHash() [15/23]

```
uint qevercloud::qHash (
            RelatedContentType value )  [inline]
```

6.1.4.76 qHash() [16/23]

```
uint qevercloud::qHash (
            ReminderEmailConfig value )  [inline]
```

6.1.4.77 qHash() [17/23]

```
uint qevercloud::qHash (
            ServiceLevel value )  [inline]
```

6.1.4.78 qHash() [18/23]

```
uint qevercloud::qHash (
            SharedNotebookInstanceRestrictions value ) [inline]
```

6.1.4.79 qHash() [19/23]

```
uint qevercloud::qHash (
            SharedNotebookPrivilegeLevel value ) [inline]
```

6.1.4.80 qHash() [20/23]

```
uint qevercloud::qHash (
            SharedNotePrivilegeLevel value ) [inline]
```

6.1.4.81 qHash() [21/23]

```
uint qevercloud::qHash (
            ShareRelationshipPrivilegeLevel value ) [inline]
```

6.1.4.82 qHash() [22/23]

```
uint qevercloud::qHash (
            SponsoredGroupRole value ) [inline]
```

6.1.4.83 qHash() [23/23]

```
uint qevercloud::qHash (
            UserIdentityType value ) [inline]
```

6.1.4.84 resetEvernoteNetworkProxy()

```
QEVERCLOUD_EXPORT void qevercloud::resetEvernoteNetworkProxy ( )
```

Reset network proxy settings used by QEverCloud to those returned from QNetworkProxy::applicationProxy static method.

This function is thread-safe although internally it operates on a static object containing proxy settings so it's not recommended to read and write proxy settings concurrently to avoid contention for static object.

6.1.4.85 setEvernoteNetworkProxy()

```
QEVERCLOUD_EXPORT void qevercloud::setEvernoteNetworkProxy (
            QNetworkProxy proxy )
```

Setter for network proxy settings used by QEverCloud. If this function is never called, QEverCloud would use proxy settings returned from QNetworkProxy::applicationProxy static method.

This function is thread-safe although internally it operates on a static object containing proxy settings so it's not recommended to read and write proxy settings concurrently to avoid contention for static object.

WARNING: when QEverCloud is built with QtWebEngine and some proxy settings different from QNetworkProxy↩ ::applicationProxy are set, the OAuth call which loads the web page would not use them; instead it would use proxy settings from QNetworkProxy::applicationProxy. This limitation is imposed by Qt: https://doc.qt.↩ io/qt-5/qtwebengine-overview.html#proxy-support

6.1.4.86 setLogger()

```
QEVERCLOUD_EXPORT void qevercloud::setLogger (
            ILoggerPtr logger )
```

6.1.4.87 setNonceGenerator()

```
void qevercloud::setNonceGenerator (
            quint64(*)() nonceGenerator )
```

Sets the function to use for nonce generation for OAuth authentication.

The default algorithm uses qrand() so do not forget to call qsrand() in your application!

qrand() is not guaranteed to be cryptographically strong. I try to amend the fact by using QUuid::createUuid() which uses /dev/urandom if it's available. But this is no guarantee either. So if you want total control over nonce generation you can write you own algorithm.

setNonceGenerator is NOT thread safe.

6.1.4.88 toRange() [1/2]

```
template<class Container >
QAssociativeContainerConstReferenceWrapper< Container > qevercloud::toRange (
            const Container & container )
```

6.1.4.89 toRange() [2/2]

```
template<class Container >
QAssociativeContainerReferenceWrapper< Container > qevercloud::toRange (
            Container & container )
```

6.1.5 Variable Documentation

6.1.5.1 CLASSIFICATION_RECIPE_SERVICE_RECIPE

```
QEVERCLOUD_EXPORT const QString qevercloud::CLASSIFICATION_RECIPE_SERVICE_RECIPE  [extern]
```

A value for the "recipe" key in the "classifications" map in NoteAttributes that indicates the Evernote service has classified a note as being a recipe.

6.1.5.2 CLASSIFICATION_RECIPE_USER_NON_RECIPE

```
QEVERCLOUD_EXPORT const QString qevercloud::CLASSIFICATION_RECIPE_USER_NON_RECIPE  [extern]
```

A value for the "recipe" key in the "classifications" map in NoteAttributes that indicates the user has classified a note as being a non-recipe.

6.1.5.3 CLASSIFICATION_RECIPE_USER_RECIPE

```
QEVERCLOUD_EXPORT const QString qevercloud::CLASSIFICATION_RECIPE_USER_RECIPE  [extern]
```

A value for the "recipe" key in the "classifications" map in NoteAttributes that indicates the user has classified a note as being a recipe.

6.1.5.4 EDAM_APP_RATING_MAX

```
QEVERCLOUD_EXPORT const qint16 qevercloud::EDAM_APP_RATING_MAX  [extern]
```

6.1.5.5 EDAM_APP_RATING_MIN

```
QEVERCLOUD_EXPORT const qint16 qevercloud::EDAM_APP_RATING_MIN  [extern]
```

App Feedback Rating range

6.1.5.6 EDAM_APPLICATIONDATA_ENTRY_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_APPLICATIONDATA_ENTRY_LEN_MAX  [extern]
```

The total length of an entry in an applicationData LazyMap, which is the sum of the length of the key and the value for the entry.

6.1.5.7 EDAM_APPLICATIONDATA_NAME_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_APPLICATIONDATA_NAME_LEN_MAX  [extern]
```

Maximum length of an application name, which is the key in an applicationData LazyMap found in entities such as Resources and Notes.

6.1.5.8 EDAM_APPLICATIONDATA_NAME_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_APPLICATIONDATA_NAME_LEN_MIN  [extern]
```

Minimum length of an application name, which is the key in an applicationData LazyMap found in entities such as Resources and Notes.

6.1.5.9 EDAM_APPLICATIONDATA_NAME_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_APPLICATIONDATA_NAME_REGEX  [extern]
```

An application name must match this regex. An application name is the key portion of an entry in an applicationData map as found in entities such as Resources and Notes. Note that even if both the name and value regexes match, it is still necessary to check the sum of the lengths against EDAM_APPLICATIONDATA_ENTRY_LEN_MAX.

6.1.5.10 EDAM_APPLICATIONDATA_VALUE_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_APPLICATIONDATA_VALUE_LEN_MAX  [extern]
```

Maximum length of an applicationData value in a LazyMap, found in entities such as Resources and Notes. Note, however, that the sum of the size of hte key and value is constrained by EDAM_APPLICATIONDATA_ENTRY_↩ LEN_MAX, so the maximum length, in practice, depends upon the key value being used.

6.1.5.11 EDAM_APPLICATIONDATA_VALUE_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_APPLICATIONDATA_VALUE_LEN_MIN  [extern]
```

Minimum length of an applicationData value in a LazyMap, found in entities such as Resources and Notes.

6.1.5.12 EDAM_APPLICATIONDATA_VALUE_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_APPLICATIONDATA_VALUE_REGEX  [extern]
```

An applicationData map value must match this regex. Note that even if both the name and value regexes match, it is still necessary to check the sum of the lengths against EDAM_APPLICATIONDATA_ENTRY_LEN_MAX.

6.1.5.13 EDAM_ATTRIBUTE_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_ATTRIBUTE_LEN_MAX  [extern]
```

Maximum length of any string-based attribute, in Unicode chars

6.1.5.14 EDAM_ATTRIBUTE_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_ATTRIBUTE_LEN_MIN  [extern]
```

Minimum length of any string-based attribute, in Unicode chars

6.1.5.15 EDAM_ATTRIBUTE_LIST_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_ATTRIBUTE_LIST_MAX  [extern]
```

The maximum number of values that can be stored in a list-based attribute (e.g. see UserAttributes.recentMailedAddresses)

6.1.5.16 EDAM_ATTRIBUTE_MAP_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_ATTRIBUTE_MAP_MAX  [extern]
```

The maximum number of entries that can be stored in a map-based attribute such as applicationData fields in Resources and Notes.

6.1.5.17 EDAM_ATTRIBUTE_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_ATTRIBUTE_REGEX  [extern]
```

Any string-based attribute must match the provided regular expression. This excludes all Unicode line endings and control characters.

6.1.5.18 EDAM_BUSINESS_MARKETING_CODE_REGEX_PATTERN

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_BUSINESS_MARKETING_CODE_REGEX_PATTERN  [extern]
```

Valid Evernote Business marketing code / affiliate code format.

6.1.5.19 EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_LEN_MAX  [extern]
```

The maximum length, in Unicode characters, of a description for a business notebook.

6.1.5.20 EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_LEN_MIN  [extern]
```

The minimum length, in Unicode characters, of a description for a business notebook.

6.1.5.21 EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_REGEX  [extern]
```

All business notebook descriptions must match this pattern. This excludes control chars or line/paragraph separators. The string may not begin or end with whitespace.

6.1.5.22 EDAM_BUSINESS_NOTEBOOKS_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_BUSINESS_NOTEBOOKS_MAX  [extern]
```

Maximum number of Notebooks in a business account

6.1.5.23 EDAM_BUSINESS_NOTES_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_BUSINESS_NOTES_MAX  [extern]
```

Maximum number of Notes per business account

6.1.5.24 EDAM_BUSINESS_PHONE_NUMBER_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_BUSINESS_PHONE_NUMBER_LEN_MAX  [extern]
```

The maximum length of a business phone number.

6.1.5.25 EDAM_BUSINESS_TAGS_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_BUSINESS_TAGS_MAX  [extern]
```

Maximum number of Tags per business account.

6.1.5.26 EDAM_BUSINESS_URI_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_BUSINESS_URI_LEN_MAX  [extern]
```

The maximum length of an Evernote Business URI

6.1.5.27 EDAM_BUSINESS_WORKSPACES_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_BUSINESS_WORKSPACES_MAX  [extern]
```

Maximum number of Workspaces in a business account

6.1.5.28 EDAM_CONNECTED_IDENTITY_REQUEST_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_CONNECTED_IDENTITY_REQUEST_MAX  [extern]
```

The maximum number of connected identities a client can request.

6.1.5.29 EDAM_CONTENT_CLASS_FOOD_MEAL

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_CONTENT_CLASS_FOOD_MEAL  [extern]
```

The content class prefix used for structured notes created by Evernote Food that captures the experience of a particular meal. When performing a wildcard search via filtered sync chunks or search strings, the ∗ character must be appended to this constant.

6.1.5.30 EDAM_CONTENT_CLASS_HELLO_ENCOUNTER

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_CONTENT_CLASS_HELLO_ENCOUNTER  [extern]
```

The content class prefix used for structured notes created by Evernote Hello that represents an encounter with a person. When performing a wildcard search via filtered sync chunks or search strings, the ∗ character must be appended to this constant.

6.1.5.31 EDAM_CONTENT_CLASS_HELLO_PROFILE

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_CONTENT_CLASS_HELLO_PROFILE  [extern]
```

The content class prefix used for structured notes created by Evernote Hello that represents the user's profile. When performing a wildcard search via filtered sync chunks or search strings, the ∗ character must be appended to this constant.

6.1.5.32 EDAM_CONTENT_CLASS_PENULTIMATE_NOTEBOOK

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_CONTENT_CLASS_PENULTIMATE_NOTEBOOK  [extern]
```

The content class value used for structured notes created by Evernote Penultimate that represents a Penultimate notebook.

6.1.5.33 EDAM_CONTENT_CLASS_PENULTIMATE_PREFIX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_CONTENT_CLASS_PENULTIMATE_PREFIX  [extern]
```

The content class prefix used for structured notes created by Evernote Penultimate. When performing a wildcard search via filtered sync chunks or search strings, the ∗ character must be appended to this constant.

6.1.5.34 EDAM_CONTENT_CLASS_SKITCH

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_CONTENT_CLASS_SKITCH  [extern]
```

The content class value used for structured image notes created by Evernote Skitch.

6.1.5.35 EDAM_CONTENT_CLASS_SKITCH_PDF

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_CONTENT_CLASS_SKITCH_PDF  [extern]
```

The content class value used for structured PDF notes created by Evernote Skitch.

6.1.5.36 EDAM_CONTENT_CLASS_SKITCH_PREFIX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_CONTENT_CLASS_SKITCH_PREFIX  [extern]
```

The content class prefix used for structured notes created by Evernote Skitch. When performing a wildcard search via filtered sync chunks or search strings, the * character must be appended to this constant.

6.1.5.37 EDAM_DEVICE_DESCRIPTION_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_DEVICE_DESCRIPTION_LEN_MAX  [extern]
```

Maximum length of the device description string associated with long sessions.

6.1.5.38 EDAM_DEVICE_DESCRIPTION_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_DEVICE_DESCRIPTION_REGEX  [extern]
```

Regular expression for device description strings associated with long sessions.

6.1.5.39 EDAM_DEVICE_ID_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_DEVICE_ID_LEN_MAX  [extern]
```

Maximum length of the device identifier string associated with long sessions.

6.1.5.40 EDAM_DEVICE_ID_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_DEVICE_ID_REGEX  [extern]
```

Regular expression for device identifier strings associated with long sessions.

6.1.5.41 EDAM_EMAIL_DOMAIN_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_EMAIL_DOMAIN_REGEX  [extern]
```

A regular expression that matches the part of an email address after the '@' symbol.

6.1.5.42 EDAM_EMAIL_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_EMAIL_LEN_MAX  [extern]
```

The maximum length of any email address

6.1.5.43 EDAM_EMAIL_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_EMAIL_LEN_MIN  [extern]
```

The minimum length of any email address

6.1.5.44 EDAM_EMAIL_LOCAL_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_EMAIL_LOCAL_REGEX  [extern]
```

A regular expression that matches the part of an email address before the '@' symbol.

6.1.5.45 EDAM_EMAIL_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_EMAIL_REGEX  [extern]
```

A regular expression that must match any email address given to Evernote. Email addresses must comply with RFC 2821 and 2822.

6.1.5.46 EDAM_FIND_CONTACT_DEFAULT_MAX_RESULTS

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_FIND_CONTACT_DEFAULT_MAX_RESULTS  [extern]
```

Default maximum number of results the service will return for findContact

6.1.5.47 EDAM_FIND_CONTACT_MAX_RESULTS

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_FIND_CONTACT_MAX_RESULTS  [extern]
```

Absolute maximum number of results the service will return for findContact

6.1.5.48 EDAM_FOOD_APP_CONTENT_CLASS_PREFIX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_FOOD_APP_CONTENT_CLASS_PREFIX  [extern]
```

The content class prefix used for all notes created by Evernote Food. This prefix can be used to assemble individual content class strings, or can be used to create a wildcard search to get all notes created by Food. When performing a wildcard search via filtered sync chunks or search strings, the $*$ character must be appended to this constant.

6.1.5.49 EDAM_GET_ORDERS_MAX_RESULTS

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_GET_ORDERS_MAX_RESULTS  [extern]
```

Absolute maximum number of results the servce will return for PersistentInternalMarket.getOrders()

6.1.5.50 EDAM_GUID_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_GUID_LEN_MAX  [extern]
```

The maximum length of a GUID generated by the Evernote service

6.1.5.51 EDAM_GUID_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_GUID_LEN_MIN  [extern]
```

The minimum length of a GUID generated by the Evernote service

6.1.5.52 EDAM_GUID_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_GUID_REGEX  [extern]
```

GUIDs generated by the Evernote service will match the provided pattern

6.1.5.53 EDAM_HASH_LEN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_HASH_LEN  [extern]
```

The exact length of a MD5 hash checksum, in binary bytes. This is the exact length that must be matched for any binary hash value.

6.1.5.54 EDAM_HELLO_APP_CONTENT_CLASS_PREFIX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_HELLO_APP_CONTENT_CLASS_PREFIX  [extern]
```

The content class prefix used for all notes created by Evernote Hello. This prefix can be used to assemble individual content class strings, or can be used to create a wildcard search to get all notes created by Hello. When performing a wildcard search via filtered sync chunks or search strings, the $*$ character must be appended to this constant.

6.1.5.55 EDAM_INDEXABLE_PLAINTEXT_MIME_TYPES

```
QEVERCLOUD_EXPORT const QSet<QString> qevercloud::EDAM_INDEXABLE_PLAINTEXT_MIME_TYPES  [extern]
```

The set of plain text MIME types that Evernote will parse and index for searching. The MIME types which start with "text/" will be handled separately by each client (i.e. hard-coded in each client).

6.1.5.56 EDAM_INDEXABLE_RESOURCE_MIME_TYPES

```
QEVERCLOUD_EXPORT const QSet<QString> qevercloud::EDAM_INDEXABLE_RESOURCE_MIME_TYPES  [extern]
```

The set of MIME types that Evernote will parse and index for searching. With exception of images, PDFs and plain text files, which are handled in a different way.

6.1.5.57 EDAM_MAX_PREFERENCES

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MAX_PREFERENCES  [extern]
```

Maximum number of name/value pairs allowed

6.1.5.58 EDAM_MAX_VALUES_PER_PREFERENCE

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MAX_VALUES_PER_PREFERENCE  [extern]
```

Maximum number of values per preference name when using values of size no greater than EDAM_↩ PREFERENCE_VALUE_LEN_MAX.

6.1.5.59 EDAM_MESSAGE_ATTACHMENT_SNIPPET_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MESSAGE_ATTACHMENT_SNIPPET_LEN_MAX  [extern]
```

The maximum length of a message attachment snippet in unicode characters.

6.1.5.60 EDAM_MESSAGE_ATTACHMENT_SNIPPET_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MESSAGE_ATTACHMENT_SNIPPET_REGEX  [extern]
```

The regex to validate message attachment snippets against

6.1.5.61 EDAM_MESSAGE_ATTACHMENT_TITLE_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MESSAGE_ATTACHMENT_TITLE_LEN_MAX  [extern]
```

The maximum length of a message attachment title in unicode characters.

6.1.5.62 EDAM_MESSAGE_ATTACHMENT_TITLE_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MESSAGE_ATTACHMENT_TITLE_REGEX  [extern]
```

The regex to validate message attachment titles against

6.1.5.63 EDAM_MESSAGE_ATTACHMENTS_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MESSAGE_ATTACHMENTS_MAX  [extern]
```

The maximum number of attachments a Message can have.

6.1.5.64 EDAM_MESSAGE_BODY_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MESSAGE_BODY_LEN_MAX  [extern]
```

The maximum length of a message body in unicode characters.

6.1.5.65 EDAM_MESSAGE_BODY_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MESSAGE_BODY_REGEX  [extern]
```

The regex to validate message.body against

6.1.5.66 EDAM_MESSAGE_RECIPIENTS_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MESSAGE_RECIPIENTS_MAX  [extern]
```

The maximum number of recipients on a MessageThread.

6.1.5.67 EDAM_MIME_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MIME_LEN_MAX  [extern]
```

The maximum length of any MIME type string given to Evernote

6.1.5.68 EDAM_MIME_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MIME_LEN_MIN  [extern]
```

The minimum length of any MIME type string given to Evernote

6.1.5.69 EDAM_MIME_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_REGEX  [extern]
```

Any MIME type string given to Evernote must match the provided pattern. E.g.: image/gif

6.1.5.70 EDAM_MIME_TYPE_AAC

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_AAC  [extern]
```

Canonical MIME type string for AAC audio resources

6.1.5.71 EDAM_MIME_TYPE_AMR

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_AMR  [extern]
```

Canonical MIME type string for AMR audio resources

6.1.5.72 EDAM_MIME_TYPE_BMP

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_BMP  [extern]
```

Canonical MIME type string for BMP image resources

6.1.5.73 EDAM_MIME_TYPE_DEFAULT

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_DEFAULT  [extern]
```

MIME type used for attachments of an unspecified type

6.1.5.74 EDAM_MIME_TYPE_GIF

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_GIF  [extern]
```

Canonical MIME type string for GIF image resources

6.1.5.75 EDAM_MIME_TYPE_INK

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_INK  [extern]
```

Canonical MIME type string for Evernote Ink resources

6.1.5.76 EDAM_MIME_TYPE_JPEG

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_JPEG  [extern]
```

Canonical MIME type string for JPEG image resources

6.1.5.77 EDAM_MIME_TYPE_M4A

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_M4A  [extern]
```

Canonical MIME type string for MP4 audio resources

6.1.5.78 EDAM_MIME_TYPE_MP3

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_MP3  [extern]
```

Canonical MIME type string for MP3 audio resources

6.1.5.79 EDAM_MIME_TYPE_MP4_VIDEO

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_MP4_VIDEO  [extern]
```

Canonical MIME type string for MP4 video resources

6.1.5.80 EDAM_MIME_TYPE_PDF

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_PDF  [extern]
```

Canonical MIME type string for PDF resources

6.1.5.81 EDAM_MIME_TYPE_PNG

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_PNG  [extern]
```

Canonical MIME type string for PNG image resources

6.1.5.82 EDAM_MIME_TYPE_TIFF

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_TIFF  [extern]
```

Canonical MIME type string for TIFF image resources

6.1.5.83 EDAM_MIME_TYPE_WAV

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_WAV  [extern]
```

Canonical MIME type string for WAV audio resources

6.1.5.84 EDAM_MIME_TYPES

```
QEVERCLOUD_EXPORT const QSet<QString> qevercloud::EDAM_MIME_TYPES  [extern]
```

The set of resource MIME types that are expected to be handled correctly by all of the major Evernote client applications.

6.1.5.85 EDAM_NOTE_BUSINESS_SHARED_NOTE_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_BUSINESS_SHARED_NOTE_MAX  [extern]
```

Maximum number of SharedNote records per business note

6.1.5.86 EDAM_NOTE_CONTENT_CLASS_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_CONTENT_CLASS_LEN_MAX  [extern]
```

The maximum length of the content class attribute of a note.

6.1.5.87 EDAM_NOTE_CONTENT_CLASS_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_CONTENT_CLASS_LEN_MIN  [extern]
```

The minimum length of the content class attribute of a note.

6.1.5.88 EDAM_NOTE_CONTENT_CLASS_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_NOTE_CONTENT_CLASS_REGEX  [extern]
```

The regular expression that the content class of a note must match to be valid.

6.1.5.89 EDAM_NOTE_CONTENT_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_CONTENT_LEN_MAX  [extern]
```

Maximum length of a Note.content field Note.content fields must comply with the ENML DTD.

6.1.5.90 EDAM_NOTE_CONTENT_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_CONTENT_LEN_MIN  [extern]
```

Minimum length of a Note.content field. Note.content fields must comply with the ENML DTD.

6.1.5.91 EDAM_NOTE_LOCK_VIEWERS_NOTES_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_LOCK_VIEWERS_NOTES_MAX  [extern]
```

The maximum number of separate notes that may be queried in a single call to NoteStore.getViewersForNotes.

6.1.5.92 EDAM_NOTE_PERSONAL_SHARED_NOTE_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_PERSONAL_SHARED_NOTE_MAX  [extern]
```

Maximum number of SharedNote records per personal note

6.1.5.93 EDAM_NOTE_RESOURCES_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_RESOURCES_MAX  [extern]
```

The maximum number of Resources per Note

6.1.5.94 EDAM_NOTE_SIZE_MAX_FREE

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_SIZE_MAX_FREE  [extern]
```

Maximum total size of a Note that can be added to a Free account. The size of a note is calculated as: ENML content length (in Unicode characters) plus the sum of all resource sizes (in bytes).

6.1.5.95 EDAM_NOTE_SIZE_MAX_PREMIUM

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_SIZE_MAX_PREMIUM  [extern]
```

Maximum total size of a Note that can be added to a Premium account. The size of a note is calculated as: ENML content length (in Unicode characters) plus the sum of all resource sizes (in bytes).

6.1.5.96 EDAM_NOTE_SOURCE_MAIL_CLIP

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_NOTE_SOURCE_MAIL_CLIP  [extern]
```

Standardized value for the 'source' NoteAttribute for notes that were clipped from an email message.

6.1.5.97 EDAM_NOTE_SOURCE_MAIL_SMTP_GATEWAY

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_NOTE_SOURCE_MAIL_SMTP_GATEWAY  [extern]
```

Standardized value for the 'source' NoteAttribute for notes that were created via email sent to Evernote's email interface.

6.1.5.98 EDAM_NOTE_SOURCE_WEB_CLIP

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_NOTE_SOURCE_WEB_CLIP  [extern]
```

Standardized value for the 'source' NoteAttribute for notes that were clipped from the web in some manner.

6.1.5.99 EDAM_NOTE_SOURCE_WEB_CLIP_SIMPLIFIED

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_NOTE_SOURCE_WEB_CLIP_SIMPLIFIED  [extern]
```

Standardized value for the 'source' NoteAttribute for notes that were clipped using the "simplified article" function of the clipper.

6.1.5.100 EDAM_NOTE_TAGS_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_TAGS_MAX  [extern]
```

The maximum number of Tags per Note

6.1.5.101 EDAM_NOTE_TITLE_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_TITLE_LEN_MAX  [extern]
```

The maximum length of a Note.title, in Unicode characters

6.1.5.102 EDAM_NOTE_TITLE_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_TITLE_LEN_MIN  [extern]
```

The minimum length of a Note.title, in Unicode characters

6.1.5.103 EDAM_NOTE_TITLE_QUALITY_HIGH

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_TITLE_QUALITY_HIGH  [extern]
```

A NoteAttributes.noteTitleQuality value indicating that the quality of an automatically generated note title is high. Examples of high quality titles include those based on a scanned business card, such as "John Doe - Scanned Business Card".

6.1.5.104 EDAM_NOTE_TITLE_QUALITY_LOW

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_TITLE_QUALITY_LOW  [extern]
```

A NoteAttributes.noteTitleQuality value indicating that the quality of an automatically generated note title is low. Examples of low quality titles include those based on a note's type and location, such as "Snapshot from 123 Sesame Street in New York".

6.1.5.105 EDAM_NOTE_TITLE_QUALITY_MEDIUM

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_TITLE_QUALITY_MEDIUM  [extern]
```

A NoteAttributes.noteTitleQuality value indicating that the quality of an automatically generated note title is medium. Examples of medium quality titles include those based on a calendar entry, such as "Note from Weekly Staff Meeting".

6.1.5.106 EDAM_NOTE_TITLE_QUALITY_UNTITLED

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_TITLE_QUALITY_UNTITLED  [extern]
```

A NoteAttributes.noteTitleQuality value indicating that a note has no meaningful title, only a placeholder value such as "Untitled Note".

6.1.5.107 EDAM_NOTE_TITLE_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_NOTE_TITLE_REGEX  [extern]
```

All Note.title fields must match this pattern. This excludes control chars or line/paragraph separators. The string may not begin or end with whitespace.

6.1.5.108 EDAM_NOTEBOOK_BUSINESS_SHARED_NOTEBOOK_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTEBOOK_BUSINESS_SHARED_NOTEBOOK_MAX  [extern]
```

Maximum number of shared notebooks per business notebook

6.1.5.109 EDAM_NOTEBOOK_NAME_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTEBOOK_NAME_LEN_MAX  [extern]
```

The maximum length of a Notebook.name, in Unicode characters

6.1.5.110 EDAM_NOTEBOOK_NAME_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTEBOOK_NAME_LEN_MIN  [extern]
```

The minimum length of a Notebook.name, in Unicode characters

6.1.5.111 EDAM_NOTEBOOK_NAME_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_NOTEBOOK_NAME_REGEX  [extern]
```

All Notebook.name fields must match this pattern. This excludes control chars or line/paragraph separators. The string may not begin or end with whitespace.

6.1.5.112 EDAM_NOTEBOOK_PERSONAL_SHARED_NOTEBOOK_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTEBOOK_PERSONAL_SHARED_NOTEBOOK_MAX  [extern]
```

Maximum number of shared notebooks per personal notebook

6.1.5.113 EDAM_NOTEBOOK_STACK_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTEBOOK_STACK_LEN_MAX  [extern]
```

The maximum length of a Notebook.stack, in Unicode characters

6.1.5.114 EDAM_NOTEBOOK_STACK_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTEBOOK_STACK_LEN_MIN  [extern]
```

The minimum length of a Notebook.stack, in Unicode characters

6.1.5.115 EDAM_NOTEBOOK_STACK_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_NOTEBOOK_STACK_REGEX  [extern]
```

All Notebook.stack fields must match this pattern. This excludes control chars or line/paragraph separators. The string may not begin or end with whitespace.

6.1.5.116 EDAM_OPEN_ID_ACCESS_TOKEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_OPEN_ID_ACCESS_TOKEN_MAX  [extern]
```

Maximum length for OpenID token. There is no official enforced limit. The length of the Token ID depends on the provider. 1000 seems to be the safest value at this time.

6.1.5.117 EDAM_PREFERENCE_BUSINESS_DEFAULT_NOTEBOOK

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PREFERENCE_BUSINESS_DEFAULT_NOTEBOOK  [extern]
```

The name of the preferences entry that contains the notebook GUID (not the linked notebook) of the default business notebook. It must be in the format EDAM_GUID_REGEX. If a default business notebook is not set and the user is a business user the user should be prompted to set the default business notebook. The default business notebook must be a read/write notebook. Whenever the default business notebook guid is used, it must be revalidiated as a writable notebook. If it is not valid, the user should be re-prompted to set the value. This value is used by clients only.

6.1.5.118 EDAM_PREFERENCE_BUSINESS_QUICKNOTE

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PREFERENCE_BUSINESS_QUICKNOTE  [extern]
```

The name of the preferences entry that contains a boolean indicating that default quicknotes should go into a business notebook. The EDAM_PREFERENCE_BUSINESS_DEFAULT_NOTEBOOK must be set correctly for this preference to be honored. The quicknote preferences should only be set to "true", if quicknote should use a business notebook. Any value other than "true" (or the omission of a value) should be treated as "false". In this case, quicknotes should be created in in the user's personal default notebook. The interface should not allow users to set quicknote to a business notebook without a valid default business notebook selected, however, clients should handle the edge case of an invalid business notebook guid. If a user stops being a business user or does not have write access to any business notebooks the quicknote preference should be ignored.

6.1.5.119 EDAM_PREFERENCE_NAME_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PREFERENCE_NAME_LEN_MAX  [extern]
```

Maximum length of a preference name

6.1.5.120 EDAM_PREFERENCE_NAME_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PREFERENCE_NAME_LEN_MIN  [extern]
```

Minimum length of a preference name

6.1.5.121 EDAM_PREFERENCE_NAME_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PREFERENCE_NAME_REGEX  [extern]
```

A preference name must match this regex.

6.1.5.122 EDAM_PREFERENCE_ONLY_ONE_VALUE_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PREFERENCE_ONLY_ONE_VALUE_LEN_MAX  [extern]
```

The maximum length of a preference value if you only use one value per preference rather than up to EDAM_↩ MAX_VALUES_PER_PREFERENCE. This option is useful if you want a single string that is larger than EDAM↩ _PREFERENCE_VALUE_LEN_MAX and would otherwise need to split the string into multiple pieces to store it.

6.1.5.123 EDAM_PREFERENCE_ONLY_ONE_VALUE_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PREFERENCE_ONLY_ONE_VALUE_REGEX  [extern]
```

A preference value must match this regex if you are using a single value for a preference.

6.1.5.124 EDAM_PREFERENCE_SHORTCUTS

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PREFERENCE_SHORTCUTS  [extern]
```

The name of the preferences entry that contains shortcuts.

6.1.5.125 EDAM_PREFERENCE_SHORTCUTS_MAX_VALUES

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PREFERENCE_SHORTCUTS_MAX_VALUES  [extern]
```

The maximum number of shortcuts that a user may have.

6.1.5.126 EDAM_PREFERENCE_VALUE_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PREFERENCE_VALUE_LEN_MAX  [extern]
```

Maximum length of a preference value

6.1.5.127 EDAM_PREFERENCE_VALUE_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PREFERENCE_VALUE_LEN_MIN  [extern]
```

Minimum length of a preference value

6.1.5.128 EDAM_PREFERENCE_VALUE_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PREFERENCE_VALUE_REGEX  [extern]
```

A preference value must match this regex if you are using more than a single value for a preference.

6.1.5.129 EDAM_PROMOTION_ID_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PROMOTION_ID_LEN_MAX  [extern]
```

The maximum length of a promotion ID in unicode characters.

6.1.5.130 EDAM_PROMOTION_ID_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PROMOTION_ID_REGEX  [extern]
```

The regex to validate promotion IDs against.

6.1.5.131 EDAM_PUBLISHING_DESCRIPTION_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PUBLISHING_DESCRIPTION_LEN_MAX  [extern]
```

The maximum length of a Publishing.publicDescription field.

6.1.5.132 EDAM_PUBLISHING_DESCRIPTION_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PUBLISHING_DESCRIPTION_LEN_MIN  [extern]
```

The minimum length of a Publishing.publicDescription field.

6.1.5.133 EDAM_PUBLISHING_DESCRIPTION_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PUBLISHING_DESCRIPTION_REGEX  [extern]
```

Any public notebook's Publishing.publicDescription field must match this pattern. No control chars or line/paragraph separators, and can't start or end with whitespace.

6.1.5.134 EDAM_PUBLISHING_URI_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PUBLISHING_URI_LEN_MAX  [extern]
```

The maximum length of a public notebook URI component

6.1.5.135 EDAM_PUBLISHING_URI_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PUBLISHING_URI_LEN_MIN  [extern]
```

The minimum length of a public notebook URI component

6.1.5.136 EDAM_PUBLISHING_URI_PROHIBITED

```
QEVERCLOUD_EXPORT const QSet<QString> qevercloud::EDAM_PUBLISHING_URI_PROHIBITED  [extern]
```

The set of strings that may not be used as a publishing URI

6.1.5.137 EDAM_PUBLISHING_URI_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PUBLISHING_URI_REGEX  [extern]
```

A public notebook URI component must match the provided pattern

6.1.5.138 EDAM_RELATED_MAX_EXPERTS

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RELATED_MAX_EXPERTS  [extern]
```

The maximum number of experts that will be returned from a findRelated() query

6.1.5.139 EDAM_RELATED_MAX_NOTEBOOKS

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RELATED_MAX_NOTEBOOKS  [extern]
```

The maximum number of notebooks that will be returned from a findRelated() query.

6.1.5.140 EDAM_RELATED_MAX_NOTES

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RELATED_MAX_NOTES  [extern]
```

The maximum number of notes that will be returned from a findRelated() query.

6.1.5.141 EDAM_RELATED_MAX_RELATED_CONTENT

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RELATED_MAX_RELATED_CONTENT  [extern]
```

The maximum number of related content snippets that will be returned from a findRelated() query.

6.1.5.142 EDAM_RELATED_MAX_TAGS

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RELATED_MAX_TAGS  [extern]
```

The maximum number of tags that will be returned from a findRelated() query.

6.1.5.143 EDAM_RELATED_PLAINTEXT_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RELATED_PLAINTEXT_LEN_MAX  [extern]
```

The maximum length of the plain text in a findRelated query, assuming that plaintext is being provided.

6.1.5.144 EDAM_RELATED_PLAINTEXT_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RELATED_PLAINTEXT_LEN_MIN  [extern]
```

The minimum length of the plain text in a findRelated query, assuming that plaintext is being provided.

6.1.5.145 EDAM_RESOURCE_SIZE_MAX_FREE

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RESOURCE_SIZE_MAX_FREE  [extern]
```

Maximum size of a resource, in bytes, for Free accounts

6.1.5.146 EDAM_RESOURCE_SIZE_MAX_PREMIUM

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RESOURCE_SIZE_MAX_PREMIUM  [extern]
```

Maximum size of a resource, in bytes, for Premium accounts

6.1.5.147 EDAM_SAVED_SEARCH_NAME_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_SAVED_SEARCH_NAME_LEN_MAX  [extern]
```

The maximum length of a SavedSearch.name field

6.1.5.148 EDAM_SAVED_SEARCH_NAME_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_SAVED_SEARCH_NAME_LEN_MIN  [extern]
```

The minimum length of a SavedSearch.name field

6.1.5.149 EDAM_SAVED_SEARCH_NAME_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SAVED_SEARCH_NAME_REGEX  [extern]
```

SavedSearch.name fields must match this pattern. No control chars or line/paragraph separators, and can't start or end with whitespace.

6.1.5.150 EDAM_SEARCH_QUERY_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_SEARCH_QUERY_LEN_MAX  [extern]
```

The maximum length of a user search query string in Unicode chars

6.1.5.151 EDAM_SEARCH_QUERY_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_SEARCH_QUERY_LEN_MIN  [extern]
```

The minimum length of a user search query string in Unicode chars

6.1.5.152 EDAM_SEARCH_QUERY_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SEARCH_QUERY_REGEX  [extern]
```

Search queries must match the provided pattern. This is used for both ad-hoc queries and SavedSearch.query fields. This excludes all control characters and line/paragraph separators.

6.1.5.153 EDAM_SEARCH_SUGGESTIONS_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_SEARCH_SUGGESTIONS_MAX  [extern]
```

Maximum number of search suggestions that can be returned

6.1.5.154 EDAM_SEARCH_SUGGESTIONS_PREFIX_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_SEARCH_SUGGESTIONS_PREFIX_LEN_MAX  [extern]
```

Maximum length of the search suggestion prefix

6.1.5.155 EDAM_SEARCH_SUGGESTIONS_PREFIX_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_SEARCH_SUGGESTIONS_PREFIX_LEN_MIN  [extern]
```

Minimum length of the search suggestion prefix

6.1.5.156 EDAM_SNIPPETS_NOTES_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_SNIPPETS_NOTES_MAX  [extern]
```

The maximium number of note snippets you can retrieve in a single request

6.1.5.157 EDAM_SOURCE_APPLICATION_ANDROID_SHARE_EXTENSION

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SOURCE_APPLICATION_ANDROID_SHARE_EXTENSION
[extern]
```

The NoteAttributes.sourceApplication value used for notes captured with the Android share extension.

6.1.5.158 EDAM_SOURCE_APPLICATION_EN_SCANSNAP

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SOURCE_APPLICATION_EN_SCANSNAP  [extern]
```

The NoteAttributes.sourceApplication value used for notes captured by PFU ScanSnap Evernote Edition.

6.1.5.159 EDAM_SOURCE_APPLICATION_EWC

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SOURCE_APPLICATION_EWC  [extern]
```

The NoteAttributes.sourceApplication value used for notes captured with the Embedded Web Clipper.

6.1.5.160 EDAM_SOURCE_APPLICATION_IOS_SHARE_EXTENSION

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SOURCE_APPLICATION_IOS_SHARE_EXTENSION  [extern]
```

The NoteAttributes.sourceApplication value used for notes captured with the iOS share extension.

6.1.5.161 EDAM_SOURCE_APPLICATION_MOLESKINE

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SOURCE_APPLICATION_MOLESKINE  [extern]
```

The NoteAttributes.sourceApplication value used for notes captured by the Moleskine page camera.

6.1.5.162 EDAM_SOURCE_APPLICATION_POSTIT

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SOURCE_APPLICATION_POSTIT  [extern]
```

The NoteAttributes.sourceApplication value used for notes captured by the Post-it camera.

6.1.5.163 EDAM_SOURCE_APPLICATION_WEB_CLIPPER

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SOURCE_APPLICATION_WEB_CLIPPER  [extern]
```

The NoteAttributes.sourceApplication value used for notes captured with the Evernote Web Clipper.

6.1.5.164 EDAM_SOURCE_OUTLOOK_CLIPPER

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SOURCE_OUTLOOK_CLIPPER  [extern]
```

The NoteAttributes.source value used for notes captured by the Microsoft Outlook clipper.

6.1.5.165 EDAM_TAG_NAME_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_TAG_NAME_LEN_MAX  [extern]
```

The maximum length of a Tag.name, in Unicode characters

6.1.5.166 EDAM_TAG_NAME_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_TAG_NAME_LEN_MIN  [extern]
```

The minimum length of a Tag.name, in Unicode characters

6.1.5.167 EDAM_TAG_NAME_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_TAG_NAME_REGEX  [extern]
```

All Tag.name fields must match this pattern. This excludes control chars, commas or line/paragraph separators. The string may not begin or end with whitespace.

6.1.5.168 EDAM_TIMEZONE_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_TIMEZONE_LEN_MAX  [extern]
```

The maximum length of a timezone specification string

6.1.5.169 EDAM_TIMEZONE_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_TIMEZONE_LEN_MIN  [extern]
```

The minimum length of a timezone specification string

6.1.5.170 EDAM_TIMEZONE_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_TIMEZONE_REGEX  [extern]
```

Any timezone string given to Evernote must match the provided pattern. This permits either a locale-based standard timezone or a GMT offset. E.g.:

- America/Los_Angeles
- GMT+08:00

6.1.5.171 EDAM_USER_LINKED_NOTEBOOK_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_LINKED_NOTEBOOK_MAX  [extern]
```

Maximum number of linked notebooks per account, for a free account.

6.1.5.172 EDAM_USER_LINKED_NOTEBOOK_MAX_PREMIUM

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_LINKED_NOTEBOOK_MAX_PREMIUM  [extern]
```

Maximum number of linked notebooks per account, for a premium account. Users who are part of an active business are also covered under "premium".

6.1.5.173 EDAM_USER_MAIL_LIMIT_DAILY_FREE

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_MAIL_LIMIT_DAILY_FREE  [extern]
```

The number of emails of any type that can be sent by a user with a Free account from the service per day. If an email is sent to two different recipients, this counts as two emails.

6.1.5.174 EDAM_USER_MAIL_LIMIT_DAILY_PREMIUM

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_MAIL_LIMIT_DAILY_PREMIUM  [extern]
```

The number of emails of any type that can be sent by a user with a Premium account from the service per day. If an email is sent to two different recipients, this counts as two emails.

6.1.5.175 EDAM_USER_NAME_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_NAME_LEN_MAX  [extern]
```

Maximum length of the User.name field

6.1.5.176 EDAM_USER_NAME_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_NAME_LEN_MIN  [extern]
```

Minimum length of the User.name field

6.1.5.177 EDAM_USER_NAME_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_USER_NAME_REGEX  [extern]
```

The User.name field must match this pattern, which excludes line endings and control characters.

6.1.5.178 EDAM_USER_NOTEBOOKS_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_NOTEBOOKS_MAX  [extern]
```

Maximum number of Notebooks per user

6.1.5.179 EDAM_USER_NOTES_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_NOTES_MAX  [extern]
```

Maximum number of Notes per user

6.1.5.180 EDAM_USER_PASSWORD_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_PASSWORD_LEN_MAX  [extern]
```

The maximum length of an Evernote user password

6.1.5.181 EDAM_USER_PASSWORD_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_PASSWORD_LEN_MIN  [extern]
```

The minimum length of an Evernote user password

6.1.5.182 EDAM_USER_PASSWORD_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_USER_PASSWORD_REGEX  [extern]
```

Evernote user passwords must match this regular expression

6.1.5.183 EDAM_USER_PROFILE_PHOTO_MAX_BYTES

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_PROFILE_PHOTO_MAX_BYTES  [extern]
```

Maximum user profile photo size, in bytes, that clients may send to the service. Photos may be resized before being stored on the service.

6.1.5.184 EDAM_USER_RECENT_MAILED_ADDRESSES_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_RECENT_MAILED_ADDRESSES_MAX  [extern]
```

Maximum number of recent email addresses that are maintained (see UserAttributes.recentMailedAddresses)

6.1.5.185 EDAM_USER_SAVED_SEARCHES_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_SAVED_SEARCHES_MAX  [extern]
```

Maximum number of SavedSearches per account

6.1.5.186 EDAM_USER_TAGS_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_TAGS_MAX  [extern]
```

Maximum number of Tags per account

6.1.5.187 EDAM_USER_UPLOAD_LIMIT_BUSINESS

```
QEVERCLOUD_EXPORT const qint64 qevercloud::EDAM_USER_UPLOAD_LIMIT_BUSINESS  [extern]
```

The number of bytes of new data that may be uploaded to a Business user's personal account each month. Note that content uploaded into the Business notebooks by the user does not count against this limit.

6.1.5.188 EDAM_USER_UPLOAD_LIMIT_BUSINESS_FIRST_MONTH

```
QEVERCLOUD_EXPORT const qint64 qevercloud::EDAM_USER_UPLOAD_LIMIT_BUSINESS_FIRST_MONTH  [extern]
```

The number of bytes of new data that may be uploaded to new business account during the first month. 50GB.

6.1.5.189 EDAM_USER_UPLOAD_LIMIT_BUSINESS_NEXT_MONTH

```
QEVERCLOUD_EXPORT const qint64 qevercloud::EDAM_USER_UPLOAD_LIMIT_BUSINESS_NEXT_MONTH  [extern]
```

The number of bytes of new data that may be uploaded to a business account for the next month. 20GB.

6.1.5.190 EDAM_USER_UPLOAD_LIMIT_BUSINESS_PER_USER

```
QEVERCLOUD_EXPORT const qint64 qevercloud::EDAM_USER_UPLOAD_LIMIT_BUSINESS_PER_USER  [extern]
```

The number of bytes of new data that may be uploaded to a Business for each member of the business per month. The total bytes available can be determined by multiplying this with the number of business users.

6.1.5.191 EDAM_USER_UPLOAD_LIMIT_FREE

```
QEVERCLOUD_EXPORT const qint64 qevercloud::EDAM_USER_UPLOAD_LIMIT_FREE  [extern]
```

The number of bytes of new data that may be uploaded to a Free user's account each month.

6.1.5.192 EDAM_USER_UPLOAD_LIMIT_PLUS

```
QEVERCLOUD_EXPORT const qint64 qevercloud::EDAM_USER_UPLOAD_LIMIT_PLUS  [extern]
```

The number of bytes of new data that may be uploaded each month to an account at a Plus service level.

6.1.5.193 EDAM_USER_UPLOAD_LIMIT_PREMIUM

```
QEVERCLOUD_EXPORT const qint64 qevercloud::EDAM_USER_UPLOAD_LIMIT_PREMIUM  [extern]
```

The number of bytes of new data that may be uploaded to a Premium user's account each month.

6.1.5.194 EDAM_USER_UPLOAD_SURVEY_THRESHOLD

```
QEVERCLOUD_EXPORT const qint64 qevercloud::EDAM_USER_UPLOAD_SURVEY_THRESHOLD [extern]
```

The number of bytes of new data uploaded in a monthly quota cycle at which point users should be prompted with a survey to gather information on how they are using Evernote.

6.1.5.195 EDAM_USER_USERNAME_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_USERNAME_LEN_MAX [extern]
```

The maximum length of an Evernote username

6.1.5.196 EDAM_USER_USERNAME_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_USERNAME_LEN_MIN [extern]
```

The minimum length of an Evernote username

6.1.5.197 EDAM_USER_USERNAME_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_USER_USERNAME_REGEX [extern]
```

Any Evernote User.username field must match this pattern. This restricts usernames to a format that could permit use as a domain name component. E.g. "username.whatever.evernote.com"

6.1.5.198 EDAM_USER_WORKSPACES_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_WORKSPACES_MAX [extern]
```

Maximum number of Workspaces per user

6.1.5.199 EDAM_VAT_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_VAT_REGEX [extern]
```

A regular expression that must match any VAT ID given to Evernote. ref http://en.wikipedia.↩
org/wiki/VAT_identification_number ref http://my.safaribooksonline.com/book/programming/re

6.1.5.200 EDAM_VERSION_MAJOR

```
QEVERCLOUD_EXPORT const qint16 qevercloud::EDAM_VERSION_MAJOR [extern]
```

The major version number for the current revision of the EDAM protocol. Clients pass this to the service using UserStore.checkVersion at the beginning of a session to confirm that they are not out of date.

6.1.5.201 EDAM_VERSION_MINOR

```
QEVERCLOUD_EXPORT const qint16 qevercloud::EDAM_VERSION_MINOR  [extern]
```

The minor version number for the current revision of the EDAM protocol. Clients pass this to the service using UserStore.checkVersion at the beginning of a session to confirm that they are not out of date.

6.1.5.202 EDAM_WORKSPACE_DESCRIPTION_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_WORKSPACE_DESCRIPTION_LEN_MAX  [extern]
```

The maximum length of a Workspace.description, in Unicode characters

6.1.5.203 EDAM_WORKSPACE_NAME_LEN_MAX

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_WORKSPACE_NAME_LEN_MAX  [extern]
```

The maximum length of a Workspace.name, in Unicode characters

6.1.5.204 EDAM_WORKSPACE_NAME_LEN_MIN

```
QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_WORKSPACE_NAME_LEN_MIN  [extern]
```

The minimum length of a Workspace.name, in Unicode characters

6.1.5.205 EDAM_WORKSPACE_NAME_REGEX

```
QEVERCLOUD_EXPORT const QString qevercloud::EDAM_WORKSPACE_NAME_REGEX  [extern]
```

All Workspace.name fields must match this pattern. This excludes control chars or line/paragraph separators. The string may not begin or end with whitespace.

6.1.5.206 EverCloudExceptionData

```
class QEVERCLOUD_EXPORT qevercloud::EverCloudExceptionData
```

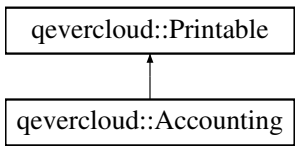
Chapter 7

Class Documentation

7.1 qevercloud::Accounting Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::Accounting:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const Accounting &other) const
- bool operator!= (const Accounting &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< Timestamp > uploadLimitEnd
- Optional< qint64 > uploadLimitNextMonth
- Optional< PremiumOrderStatus > premiumServiceStatus
- Optional< QString > premiumOrderNumber
- Optional< QString > premiumCommerceService
- Optional< Timestamp > premiumServiceStart
- Optional< QString > premiumServiceSKU

- Optional< Timestamp > lastSuccessfulCharge
- Optional< Timestamp > lastFailedCharge
- Optional< QString > lastFailedChargeReason
- Optional< Timestamp > nextPaymentDue
- Optional< Timestamp > premiumLockUntil
- Optional< Timestamp > updated
- Optional< QString > premiumSubscriptionNumber
- Optional< Timestamp > lastRequestedCharge
- Optional< QString > currency
- Optional< qint32 > unitPrice
- Optional< qint32 > businessId
- Optional< QString > businessName
- Optional< BusinessUserRole > businessRole
- Optional< qint32 > unitDiscount
- Optional< Timestamp > nextChargeDate
- Optional< qint32 > availablePoints

7.1.1 Detailed Description

This represents the bookkeeping information for the user's subscription.

7.1.2 Member Function Documentation

7.1.2.1 operator"!=()

```
bool qevercloud::Accounting::operator!= (
            const Accounting & other ) const  [inline]
```

7.1.2.2 operator==()

```
bool qevercloud::Accounting::operator== (
            const Accounting & other ) const  [inline]
```

7.1.2.3 print()

```
virtual void qevercloud::Accounting::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.1.3 Member Data Documentation

7.1.3.1 availablePoints

```
Optional< qint32 > qevercloud::Accounting::availablePoints
```

NOT DOCUMENTED

7.1.3.2 businessId

`Optional< qint32 > qevercloud::Accounting::businessId`

*DEPRECATED:*See BusinessUserInfo.

7.1.3.3 businessName

`Optional< QString > qevercloud::Accounting::businessName`

*DEPRECATED:*See BusinessUserInfo.

7.1.3.4 businessRole

`Optional< BusinessUserRole > qevercloud::Accounting::businessRole`

*DEPRECATED:*See BusinessUserInfo.

7.1.3.5 currency

`Optional< QString > qevercloud::Accounting::currency`

ISO 4217 currency code

7.1.3.6 lastFailedCharge

`Optional< Timestamp > qevercloud::Accounting::lastFailedCharge`

Date the last time a charge was attempted and failed.

7.1.3.7 lastFailedChargeReason

`Optional< QString > qevercloud::Accounting::lastFailedChargeReason`

Reason provided for the charge failure

7.1.3.8 lastRequestedCharge

`Optional< Timestamp > qevercloud::Accounting::lastRequestedCharge`

Date charge last attempted

7.1.3.9 lastSuccessfulCharge

`Optional< Timestamp > qevercloud::Accounting::lastSuccessfulCharge`

Date the last time the user was charged. Null if never charged.

7.1.3.10 localData

```
EverCloudLocalData qevercloud::Accounting::localData
```

See the declaration of EverCloudLocalData for details

7.1.3.11 nextChargeDate

```
Optional< Timestamp > qevercloud::Accounting::nextChargeDate
```

The next time the user will be charged, may or may not be the same as nextPaymentDue

7.1.3.12 nextPaymentDue

```
Optional< Timestamp > qevercloud::Accounting::nextPaymentDue
```

The end of the billing cycle. This could be in the past if there are failed charges.

7.1.3.13 premiumCommerceService

```
Optional< QString > qevercloud::Accounting::premiumCommerceService
```

The commerce system used (paypal, Google checkout, etc)

7.1.3.14 premiumLockUntil

```
Optional< Timestamp > qevercloud::Accounting::premiumLockUntil
```

An internal variable to manage locking operations on the commerce variables.

7.1.3.15 premiumOrderNumber

```
Optional< QString > qevercloud::Accounting::premiumOrderNumber
```

The order number used by the commerce system to process recurring payments

7.1.3.16 premiumServiceSKU

```
Optional< QString > qevercloud::Accounting::premiumServiceSKU
```

The code associated with the purchase eg. monthly or annual purchase. Clients should interpret this value and localize it.

7.1.3.17 premiumServiceStart

```
Optional< Timestamp > qevercloud::Accounting::premiumServiceStart
```

The start date when this premium promotion began (this number will get overwritten if a premium service is canceled and then re-activated).

7.1.3.18 premiumServiceStatus

```
Optional< PremiumOrderStatus > qevercloud::Accounting::premiumServiceStatus
```

Indicates the phases of a premium account during the billing process.

7.1.3.19 premiumSubscriptionNumber

```
Optional< QString > qevercloud::Accounting::premiumSubscriptionNumber
```

The number number identifying the recurring subscription used to make the recurring charges.

7.1.3.20 unitDiscount

```
Optional< qint32 > qevercloud::Accounting::unitDiscount
```

discount per seat in negative amount and smallest unit of the currency (e.g. cents for USD)

7.1.3.21 unitPrice

```
Optional< qint32 > qevercloud::Accounting::unitPrice
```

charge in the smallest unit of the currency (e.g. cents for USD)

7.1.3.22 updated

```
Optional< Timestamp > qevercloud::Accounting::updated
```

The date any modification where made to this record.

7.1.3.23 uploadLimitEnd

```
Optional< Timestamp > qevercloud::Accounting::uploadLimitEnd
```

The date and time when the current upload limit expires. At this time, the monthly upload count reverts to 0 and a new limit is imposed. This date and time is exclusive, so this is effectively the start of the new month.

7.1.3.24 uploadLimitNextMonth

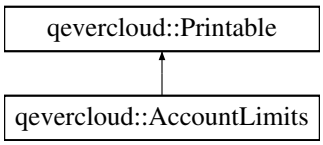
`Optional< qint64 > qevercloud::Accounting::uploadLimitNextMonth`

When uploadLimitEnd is reached, the service will change uploadLimit to uploadLimitNextMonth. If a premium account is canceled, this mechanism will reset the quota appropriately.

7.2 qevercloud::AccountLimits Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::AccountLimits:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const AccountLimits &other) const
- bool operator!= (const AccountLimits &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< qint32 > userMailLimitDaily
- Optional< qint64 > noteSizeMax
- Optional< qint64 > resourceSizeMax
- Optional< qint32 > userLinkedNotebookMax
- Optional< qint64 > uploadLimit
- Optional< qint32 > userNoteCountMax
- Optional< qint32 > userNotebookCountMax
- Optional< qint32 > userTagCountMax
- Optional< qint32 > noteTagCountMax
- Optional< qint32 > userSavedSearchesMax
- Optional< qint32 > noteResourceCountMax

7.2.1 Detailed Description

This structure is used to provide account limits that are in effect for this user.

7.2.2 Member Function Documentation

7.2.2.1 operator"!=()

```
bool qevercloud::AccountLimits::operator!= (
            const AccountLimits & other ) const  [inline]
```

7.2.2.2 operator==()

```
bool qevercloud::AccountLimits::operator== (
            const AccountLimits & other ) const  [inline]
```

7.2.2.3 print()

```
virtual void qevercloud::AccountLimits::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.2.3 Member Data Documentation

7.2.3.1 localData

```
EverCloudLocalData qevercloud::AccountLimits::localData
```

See the declaration of EverCloudLocalData for details

7.2.3.2 noteResourceCountMax

```
Optional< qint32 > qevercloud::AccountLimits::noteResourceCountMax
```

The maximum number of Resources per Note

7.2.3.3 noteSizeMax

```
Optional< qint64 > qevercloud::AccountLimits::noteSizeMax
```

Maximum total size of a Note that can be added. The size of a note is calculated as: ENML content length (in Unicode characters) plus the sum of all resource sizes (in bytes).

7.2.3.4 noteTagCountMax

```
Optional< qint32 > qevercloud::AccountLimits::noteTagCountMax
```

Maximum number of Tags per Note

7.2.3.5 resourceSizeMax

```
Optional< qint64 > qevercloud::AccountLimits::resourceSizeMax
```

Maximum size of a resource, in bytes

7.2.3.6 uploadLimit

```
Optional< qint64 > qevercloud::AccountLimits::uploadLimit
```

The number of bytes that can be uploaded to the account in the current month. For new notes that are created, this is the length of the note content (in Unicode characters) plus the size of each resource (in bytes). For edited notes, this is the the difference between the old length and the new length (if this is greater than 0) plus the size of each new resource.

7.2.3.7 userLinkedNotebookMax

```
Optional< qint32 > qevercloud::AccountLimits::userLinkedNotebookMax
```

Maximum number of linked notebooks per account.

7.2.3.8 userMailLimitDaily

```
Optional< qint32 > qevercloud::AccountLimits::userMailLimitDaily
```

The number of emails of any type that can be sent by a user from the service per day. If an email is sent to two different recipients, this counts as two emails.

7.2.3.9 userNotebookCountMax

```
Optional< qint32 > qevercloud::AccountLimits::userNotebookCountMax
```

Maximum number of Notebooks per user

7.2.3.10 userNoteCountMax

```
Optional< qint32 > qevercloud::AccountLimits::userNoteCountMax
```

Maximum number of Notes per user

7.2.3.11 userSavedSearchesMax

```
Optional< qint32 > qevercloud::AccountLimits::userSavedSearchesMax
```

Maximum number of SavedSearches per account

7.2.3.12 userTagCountMax

```
Optional< qint32 > qevercloud::AccountLimits::userTagCountMax
```

Maximum number of Tags per account

7.3 qevercloud::IDurableService::AsyncRequest Struct Reference

```
#include <DurableService.h>
```

Public Member Functions

- AsyncRequest (const char *name, QString description, AsyncServiceCall &&call)

Public Attributes

- const char * m_name
- QString m_description
- AsyncServiceCall m_call

7.3.1 Constructor & Destructor Documentation

7.3.1.1 AsyncRequest()

```
qevercloud::IDurableService::AsyncRequest::AsyncRequest (
            const char * name,
            QString description,
            AsyncServiceCall && call )  [inline]
```

7.3.2 Member Data Documentation

7.3.2.1 m_call

```
AsyncServiceCall qevercloud::IDurableService::AsyncRequest::m_call
```

7.3.2.2 m_description

```
QString qevercloud::IDurableService::AsyncRequest::m_description
```

7.3.2.3 m_name

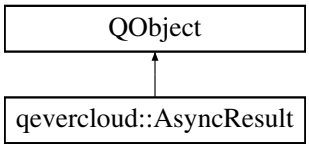
```
const char* qevercloud::IDurableService::AsyncRequest::m_name
```

7.4 qevercloud::AsyncResult Class Reference

Returned by asynchonous versions of functions.

`#include <AsyncResult.h>`

Inheritance diagram for qevercloud::AsyncResult:



Public Types

- typedef QVariant(∗ ReadFunctionType) (QByteArray replyData)

Signals

- void finished (QVariant result, EverCloudExceptionDataPtr error, IRequestContextPtr ctx)

  Emitted upon asynchronous call completition.

Public Member Functions

- AsyncResult (QString url, QByteArray postData, IRequestContextPtr ctx, ReadFunctionType readFunction=AsyncResult::asIs, bool autoDelete=true, QObject ∗parent=nullptr)
- AsyncResult (QNetworkRequest request, QByteArray postData, IRequestContextPtr ctx, ReadFunctionType readFunction=AsyncResult::asIs, bool autoDelete=true, QObject ∗parent=nullptr)
- AsyncResult (QVariant result, EverCloudExceptionDataPtr error, IRequestContextPtr ctx, bool autoDelete=true, QObject ∗parent=nullptr)
- ∼AsyncResult ()
- bool waitForFinished (int timeout=-1)

  Wait for asynchronous operation to complete.

Static Public Member Functions

- static QVariant asIs (QByteArray replyData)

Friends

- class DurableService

7.4.1 Detailed Description

Returned by asynchonous versions of functions.

Wait for AsyncResult::finished signal.

Intended usage is something like this:

```
NoteStore* ns;
Note note;
...
QObject::connect(ns->createNoteAsync(note), &AsyncResult::finished,
                 [ns](
                     QVariant result,
                     EverCloudExceptionDataPtr error,
                     IRequestContextPtr ctx)
                 {
                     if (error) {
                         // do something in case of an error
                     }
                     else {
                         Note note = result.value<Note>();
                         // process returned result
                     }
                 });
```

7.4.2 Member Typedef Documentation

7.4.2.1 ReadFunctionType

```
typedef QVariant(* qevercloud::AsyncResult::ReadFunctionType) (QByteArray replyData)
```

7.4.3 Constructor & Destructor Documentation

7.4.3.1 AsyncResult() [1/3]

```
qevercloud::AsyncResult::AsyncResult (
             QString url,
             QByteArray postData,
             IRequestContextPtr ctx,
             ReadFunctionType readFunction = AsyncResult::asIs,
             bool autoDelete = true,
             QObject * parent = nullptr )
```

7.4.3.2 AsyncResult() [2/3]

```
qevercloud::AsyncResult::AsyncResult (
             QNetworkRequest request,
             QByteArray postData,
             IRequestContextPtr ctx,
             ReadFunctionType readFunction = AsyncResult::asIs,
             bool autoDelete = true,
             QObject * parent = nullptr )
```

7.4.3.3 AsyncResult() [3/3]

```
qevercloud::AsyncResult::AsyncResult (
            QVariant result,
            EverCloudExceptionDataPtr error,
            IRequestContextPtr ctx,
            bool autoDelete = true,
            QObject * parent = nullptr )
```

Constructor accepting already prepared value and/or exception, for use in tests

7.4.3.4 ~AsyncResult()

```
qevercloud::AsyncResult::~AsyncResult ( )
```

7.4.4 Member Function Documentation

7.4.4.1 asIs()

```
static QVariant qevercloud::AsyncResult::asIs (
            QByteArray replyData )  [static]
```

7.4.4.2 finished

```
void qevercloud::AsyncResult::finished (
            QVariant result,
            EverCloudExceptionDataPtr error,
            IRequestContextPtr ctx )  [signal]
```

Emitted upon asynchronous call completition.

Parameters

result	Request result
error	Non-nullptr in case of an error. See EverCloudExceptionData for more details
ctx	Request context used to make the request

AsyncResult deletes itself after emitting this signal (if autoDelete parameter passed to its constructor was set to true). You don't have to manage it's lifetime explicitly.

7.4.4.3 waitForFinished()

```
bool qevercloud::AsyncResult::waitForFinished (
            int timeout = -1 )
```

Wait for asynchronous operation to complete.

Parameters

timeout	Maximum time to wait in milliseconds. Forever if < 0.

Returns

true if finished successfully, false in case of the timeout

7.4.5 Friends And Related Symbol Documentation

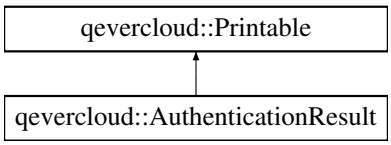
7.4.5.1 DurableService

```
friend class DurableService  [friend]
```

7.5 qevercloud::AuthenticationResult Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::AuthenticationResult:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const AuthenticationResult &other) const
- bool operator!= (const AuthenticationResult &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Timestamp currentTime = 0
- QString authenticationToken
- Timestamp expiration = 0
- Optional< User > user
- Optional< PublicUserInfo > publicUserInfo
- Optional< QString > noteStoreUrl
- Optional< QString > webApiUrlPrefix
- Optional< bool > secondFactorRequired
- Optional< QString > secondFactorDeliveryHint
- Optional< UserUrls > urls

7.5.1 Detailed Description

When an authentication (or re-authentication) is performed, this structure provides the result to the client.

7.5.2 Member Function Documentation

7.5.2.1 operator"!=()

```
bool qevercloud::AuthenticationResult::operator!= (
            const AuthenticationResult & other ) const  [inline]
```

7.5.2.2 operator==()

```
bool qevercloud::AuthenticationResult::operator== (
            const AuthenticationResult & other ) const  [inline]
```

7.5.2.3 print()

```
virtual void qevercloud::AuthenticationResult::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.5.3 Member Data Documentation

7.5.3.1 authenticationToken

```
QString qevercloud::AuthenticationResult::authenticationToken
```

Holds an opaque, ASCII-encoded token that can be used by the client to perform actions on a NoteStore.

7.5.3.2 currentTime

```
Timestamp qevercloud::AuthenticationResult::currentTime = 0
```

The server-side date and time when this result was generated.

7.5.3.3 expiration

```
Timestamp qevercloud::AuthenticationResult::expiration = 0
```

Holds the server-side date and time when the authentication token will expire. This time can be compared to "currentTime" to produce an expiration time that can be reconciled with the client's local clock.

7.5.3.4 localData

```
EverCloudLocalData qevercloud::AuthenticationResult::localData
```

See the declaration of EverCloudLocalData for details

7.5.3.5 noteStoreUrl

```
Optional< QString > qevercloud::AuthenticationResult::noteStoreUrl
```

DEPRECATED - Client applications should use urls.noteStoreUrl.

7.5.3.6 publicUserInfo

```
Optional< PublicUserInfo > qevercloud::AuthenticationResult::publicUserInfo
```

If this authentication result was achieved without full permissions to access the full User structure, this field may be set to give back a more limited public set of data.

7.5.3.7 secondFactorDeliveryHint

```
Optional< QString > qevercloud::AuthenticationResult::secondFactorDeliveryHint
```

When secondFactorRequired is set to true, this field may contain a string describing the second factor delivery method that the user has configured. This will typically be an obfuscated mobile device number, such as "(xxx) xxx-x095". This string can be displayed to the user to remind them how to obtain the required second factor.

7.5.3.8 secondFactorRequired

```
Optional< bool > qevercloud::AuthenticationResult::secondFactorRequired
```

If set to true, this field indicates that the user has enabled two-factor authentication and must enter their second factor in order to complete authentication. In this case the value of authenticationResult will be a short-lived authentication token that may only be used to make a subsequent call to completeTwoFactorAuthentication.

7.5.3.9 urls

```
Optional< UserUrls > qevercloud::AuthenticationResult::urls
```

This structure will contain all of the URLs that clients need to make requests to the Evernote service on behalf of the authenticated User.

7.5.3.10 user

```
Optional< User > qevercloud::AuthenticationResult::user
```

Holds the information about the account which was authenticated if this was a full authentication. May be absent if this particular authentication did not require user information.

7.5.3.11 webApiUrlPrefix

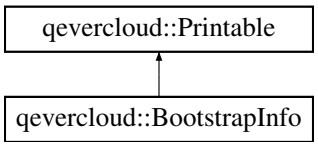
```
Optional< QString > qevercloud::AuthenticationResult::webApiUrlPrefix
```

DEPRECATED - Client applications should use urls.webApiUrlPrefix.

7.6 qevercloud::BootstrapInfo Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::BootstrapInfo:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const BootstrapInfo &other) const
- bool operator!= (const BootstrapInfo &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- QList< BootstrapProfile > profiles

7.6.1 Detailed Description

This structure describes a collection of bootstrap profiles.

7.6.2 Member Function Documentation

7.6.2.1 operator"!=()

```
bool qevercloud::BootstrapInfo::operator!= (
            const BootstrapInfo & other ) const  [inline]
```

7.6.2.2 operator==()

```
bool qevercloud::BootstrapInfo::operator== (
            const BootstrapInfo & other ) const  [inline]
```

7.6.2.3 print()

```
virtual void qevercloud::BootstrapInfo::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.6.3 Member Data Documentation

7.6.3.1 localData

```
EverCloudLocalData qevercloud::BootstrapInfo::localData
```

See the declaration of EverCloudLocalData for details

7.6.3.2 profiles

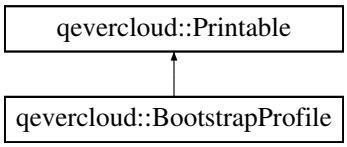
```
QList< BootstrapProfile > qevercloud::BootstrapInfo::profiles
```

List of one or more bootstrap profiles, in descending preference order.

7.7 qevercloud::BootstrapProfile Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::BootstrapProfile:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const BootstrapProfile &other) const
- bool operator!= (const BootstrapProfile &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- QString name
- BootstrapSettings settings

7.7.1 Detailed Description

This structure describes a collection of bootstrap settings.

7.7.2 Member Function Documentation

7.7.2.1 operator"!=()

```
bool qevercloud::BootstrapProfile::operator!= (
            const BootstrapProfile & other ) const  [inline]
```

7.7.2.2 operator==()

```
bool qevercloud::BootstrapProfile::operator== (
            const BootstrapProfile & other ) const  [inline]
```

7.7.2.3 print()

```
virtual void qevercloud::BootstrapProfile::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.7.3 Member Data Documentation

7.7.3.1 localData

```
EverCloudLocalData qevercloud::BootstrapProfile::localData
```

See the declaration of EverCloudLocalData for details

7.7.3.2 name

```
QString qevercloud::BootstrapProfile::name
```

The unique name of the profile, which is guaranteed to remain consistent across calls to getBootstrapInfo.

7.7.3.3 settings

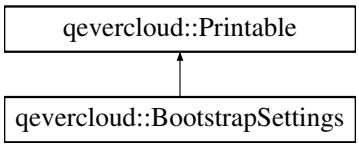
```
BootstrapSettings qevercloud::BootstrapProfile::settings
```

The settings for this profile.

7.8 qevercloud::BootstrapSettings Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::BootstrapSettings:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const BootstrapSettings &other) const
- bool operator!= (const BootstrapSettings &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- QString serviceHost
- QString marketingUrl
- QString supportUrl
- QString accountEmailDomain
- Optional< bool > enableFacebookSharing
- Optional< bool > enableGiftSubscriptions
- Optional< bool > enableSupportTickets
- Optional< bool > enableSharedNotebooks
- Optional< bool > enableSingleNoteSharing
- Optional< bool > enableSponsoredAccounts
- Optional< bool > enableTwitterSharing
- Optional< bool > enableLinkedInSharing
- Optional< bool > enablePublicNotebooks
- Optional< bool > enableGoogle

7.8.1 Detailed Description

This structure describes a collection of bootstrap settings.

7.8.2 Member Function Documentation

7.8.2.1 operator"!=()

```
bool qevercloud::BootstrapSettings::operator!= (
            const BootstrapSettings & other ) const  [inline]
```

7.8.2.2 operator==()

```
bool qevercloud::BootstrapSettings::operator== (
            const BootstrapSettings & other ) const  [inline]
```

7.8.2.3 print()

```
virtual void qevercloud::BootstrapSettings::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.8.3 Member Data Documentation

7.8.3.1 accountEmailDomain

```
QString qevercloud::BootstrapSettings::accountEmailDomain
```

The domain used for an Evernote user's incoming email address, which allows notes to be emailed into an account. E.g. m.evernote.com.

7.8.3.2 enableFacebookSharing

```
Optional< bool > qevercloud::BootstrapSettings::enableFacebookSharing
```

Whether the client application should enable sharing of notes on Facebook.

7.8.3.3 enableGiftSubscriptions

```
Optional< bool > qevercloud::BootstrapSettings::enableGiftSubscriptions
```

Whether the client application should enable gift subscriptions.

7.8.3.4 enableGoogle

```
Optional< bool > qevercloud::BootstrapSettings::enableGoogle
```

Whether the client application should enable authentication with Google, for example to allow integration with a user's Gmail contacts.

7.8.3.5 enableLinkedInSharing

```
Optional< bool > qevercloud::BootstrapSettings::enableLinkedInSharing
```

NOT DOCUMENTED

7.8.3.6 enablePublicNotebooks

```
Optional< bool > qevercloud::BootstrapSettings::enablePublicNotebooks
```

NOT DOCUMENTED

7.8.3.7 enableSharedNotebooks

```
Optional< bool > qevercloud::BootstrapSettings::enableSharedNotebooks
```

Whether the client application should enable shared notebooks.

7.8.3.8 enableSingleNoteSharing

```
Optional< bool > qevercloud::BootstrapSettings::enableSingleNoteSharing
```

Whether the client application should enable single note sharing.

7.8.3.9 enableSponsoredAccounts

```
Optional< bool > qevercloud::BootstrapSettings::enableSponsoredAccounts
```

Whether the client application should enable sponsored accounts.

7.8.3.10 enableSupportTickets

```
Optional< bool > qevercloud::BootstrapSettings::enableSupportTickets
```

Whether the client application should enable in-client creation of support tickets.

7.8.3.11 enableTwitterSharing

```
Optional< bool > qevercloud::BootstrapSettings::enableTwitterSharing
```

Whether the client application should enable sharing of notes on Twitter.

7.8.3.12 localData

```
EverCloudLocalData qevercloud::BootstrapSettings::localData
```

See the declaration of EverCloudLocalData for details

7.8.3.13 marketingUrl

```
QString qevercloud::BootstrapSettings::marketingUrl
```

The URL stem for the Evernote corporate marketing website, e.g. http://www.evernote.com. This stem can be used to compose website URLs. For example, the URL of the Evernote Trunk is composed by appending "/about/trunk/" to the value of marketingUrl.

7.8.3.14 serviceHost

```
QString qevercloud::BootstrapSettings::serviceHost
```

The hostname and optional port for composing Evernote web service URLs. This URL can be used to access the UserStore and related services, but must not be used to compose the NoteStore URL. Client applications must handle serviceHost values that include only the hostname (e.g. www.evernote.com) or both the hostname and port (e.g. www.evernote.com:8080). If no port is specified, or if port 443 is specified, client applications must use the scheme "https" when composing URLs. Otherwise, a client must use the scheme "http".

7.8.3.15 supportUrl

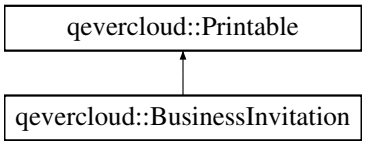
```
QString qevercloud::BootstrapSettings::supportUrl
```

The full URL for the Evernote customer support website, e.g. https://support.evernote.com.

7.9 qevercloud::BusinessInvitation Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::BusinessInvitation:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const BusinessInvitation &other) const
- bool operator!= (const BusinessInvitation &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< qint32 > businessId
- Optional< QString > email
- Optional< BusinessUserRole > role
- Optional< BusinessInvitationStatus > status
- Optional< UserID > requesterId
- Optional< bool > fromWorkChat
- Optional< Timestamp > created
- Optional< Timestamp > mostRecentReminder

7.9.1 Detailed Description

A structure describing an invitation to join a business account.

7.9.2 Member Function Documentation

7.9.2.1 operator"!=()

```
bool qevercloud::BusinessInvitation::operator!= (
            const BusinessInvitation & other ) const  [inline]
```

7.9.2.2 operator==()

```
bool qevercloud::BusinessInvitation::operator== (
            const BusinessInvitation & other ) const  [inline]
```

7.9.2.3 print()

```
virtual void qevercloud::BusinessInvitation::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.9.3 Member Data Documentation

7.9.3.1 businessId

```
Optional< qint32 > qevercloud::BusinessInvitation::businessId
```

The ID of the business to which the invitation grants access.

7.9.3.2 created

```
Optional< Timestamp > qevercloud::BusinessInvitation::created
```

The timestamp at which this invitation was created.

7.9.3.3 email

```
Optional< QString > qevercloud::BusinessInvitation::email
```

The email address that was invited to join the business.

7.9.3.4 fromWorkChat

```
Optional< bool > qevercloud::BusinessInvitation::fromWorkChat
```

If this invitation was created implicitly via a WorkChat, this field will be true.

7.9.3.5 localData

```
EverCloudLocalData qevercloud::BusinessInvitation::localData
```

See the declaration of EverCloudLocalData for details

7.9.3.6 mostRecentReminder

```
Optional< Timestamp > qevercloud::BusinessInvitation::mostRecentReminder
```

The timestamp at which the most recent reminder was sent.

7.9.3.7 requesterId

```
Optional< UserID > qevercloud::BusinessInvitation::requesterId
```

For invitations that were initially requested by a non-admin member of the business, this field specifies the user ID of the requestor. For all other invitations, this field will be unset.

7.9.3.8 role

`Optional< BusinessUserRole > qevercloud::BusinessInvitation::role`

The role to grant the user after the invitation is accepted.

7.9.3.9 status

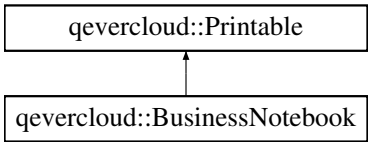
`Optional< BusinessInvitationStatus > qevercloud::BusinessInvitation::status`

The status of the invitation.

7.10 qevercloud::BusinessNotebook Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::BusinessNotebook:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const BusinessNotebook &other) const
- bool operator!= (const BusinessNotebook &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > notebookDescription
- Optional< SharedNotebookPrivilegeLevel > privilege
- Optional< bool > recommended

7.10.1 Detailed Description

If a Notebook contained in an Evernote Business account has been published the to business library, the Notebook will have a reference to one of these structures, which specifies how the Notebook will be represented in the library.

7.10.2 Member Function Documentation

7.10.2.1 operator"!=()

```
bool qevercloud::BusinessNotebook::operator!= (
            const BusinessNotebook & other ) const  [inline]
```

7.10.2.2 operator==()

```
bool qevercloud::BusinessNotebook::operator== (
            const BusinessNotebook & other ) const  [inline]
```

7.10.2.3 print()

```
virtual void qevercloud::BusinessNotebook::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.10.3 Member Data Documentation

7.10.3.1 localData

```
EverCloudLocalData qevercloud::BusinessNotebook::localData
```

See the declaration of EverCloudLocalData for details

7.10.3.2 notebookDescription

```
Optional< QString > qevercloud::BusinessNotebook::notebookDescription
```

A short description of the notebook's content that will be displayed in the business library user interface. The description may not begin or end with whitespace.
Length: EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_LEN_MIN - EDAM_BUSINESS_NOTEBOOK_↩ DESCRIPTION_LEN_MAX
Regex: EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_REGEX

7.10.3.3 privilege

```
Optional< SharedNotebookPrivilegeLevel > qevercloud::BusinessNotebook::privilege
```

The privileges that will be granted to users who join the notebook through the business library.

7.10.3.4 recommended

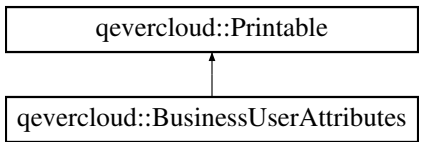
```
Optional< bool > qevercloud::BusinessNotebook::recommended
```

Whether the notebook should be "recommended" when displayed in the business library user interface.

7.11 qevercloud::BusinessUserAttributes Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::BusinessUserAttributes:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const BusinessUserAttributes &other) const
- bool operator!= (const BusinessUserAttributes &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > title
- Optional< QString > location
- Optional< QString > department
- Optional< QString > mobilePhone
- Optional< QString > linkedInProfileUrl
- Optional< QString > workPhone
- Optional< Timestamp > companyStartDate

7.11.1 Detailed Description

A structure holding the optional attributes associated with users in a business.

7.11.2 Member Function Documentation

7.11.2.1 operator"!=()

```
bool qevercloud::BusinessUserAttributes::operator!= (
            const BusinessUserAttributes & other ) const  [inline]
```

7.11.2.2 operator==()

```
bool qevercloud::BusinessUserAttributes::operator== (
            const BusinessUserAttributes & other ) const  [inline]
```

7.11.2.3 print()

```
virtual void qevercloud::BusinessUserAttributes::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.11.3 Member Data Documentation

7.11.3.1 companyStartDate

```
Optional< Timestamp > qevercloud::BusinessUserAttributes::companyStartDate
```

The date on which the user started working at their company.

7.11.3.2 department

```
Optional< QString > qevercloud::BusinessUserAttributes::department
```

Free form text of the department this user belongs to.

7.11.3.3 linkedInProfileUrl

```
Optional< QString > qevercloud::BusinessUserAttributes::linkedInProfileUrl
```

URL to user's public LinkedIn profile page. This should only contain the portion relative to the base LinkedIn URL. For example: "/pub/john-smith/".

7.11.3.4 localData

```
EverCloudLocalData qevercloud::BusinessUserAttributes::localData
```

See the declaration of EverCloudLocalData for details

7.11.3.5 location

```
Optional< QString > qevercloud::BusinessUserAttributes::location
```

City, State (for US) or City / Province for other countries

7.11.3.6 mobilePhone

```
Optional< QString > qevercloud::BusinessUserAttributes::mobilePhone
```

User's mobile phone number. Stored as plain text without any formatting.

7.11.3.7 title

```
Optional< QString > qevercloud::BusinessUserAttributes::title
```

Free form text of this user's title in the business

7.11.3.8 workPhone

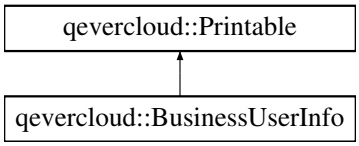
```
Optional< QString > qevercloud::BusinessUserAttributes::workPhone
```

User's work phone number. Stored as plain text without any formatting.

7.12 qevercloud::BusinessUserInfo Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::BusinessUserInfo:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const BusinessUserInfo &other) const
- bool operator!= (const BusinessUserInfo &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< qint32 > businessId
- Optional< QString > businessName
- Optional< BusinessUserRole > role
- Optional< QString > email
- Optional< Timestamp > updated

7.12.1 Detailed Description

This structure is used to provide information about an Evernote Business membership, for members who are part of a business.

7.12.2 Member Function Documentation

7.12.2.1 operator"!=()

```
bool qevercloud::BusinessUserInfo::operator!= (
            const BusinessUserInfo & other ) const  [inline]
```

7.12.2.2 operator==()

```
bool qevercloud::BusinessUserInfo::operator== (
            const BusinessUserInfo & other ) const  [inline]
```

7.12.2.3 print()

```
virtual void qevercloud::BusinessUserInfo::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.12.3 Member Data Documentation

7.12.3.1 businessId

```
Optional< qint32 > qevercloud::BusinessUserInfo::businessId
```

The ID of the Evernote Business account that the user is a member of.

businessName The human-readable name of the Evernote Business account that the user is a member of.

7.12.3.2 businessName

```
Optional< QString > qevercloud::BusinessUserInfo::businessName
```

NOT DOCUMENTED

7.12.3.3 email

```
Optional< QString > qevercloud::BusinessUserInfo::email
```

An e-mail address that will be used by the service in the context of your Evernote Business activities. For example, this e-mail address will be used when you e-mail a business note, when you update notes in the account of your business, etc. The business e-mail cannot be used for identification purposes such as for logging into the service.

7.12.3.4 localData

```
EverCloudLocalData qevercloud::BusinessUserInfo::localData
```

See the declaration of EverCloudLocalData for details

7.12.3.5 role

```
Optional< BusinessUserRole > qevercloud::BusinessUserInfo::role
```

The role of the user within the Evernote Business account that they are a member of.

7.12.3.6 updated

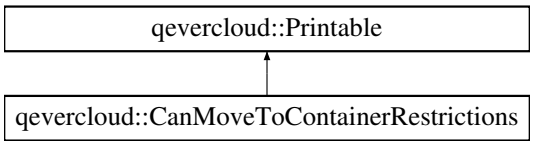
```
Optional< Timestamp > qevercloud::BusinessUserInfo::updated
```

Last time the business user or business user attributes were updated.

7.13 qevercloud::CanMoveToContainerRestrictions Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::CanMoveToContainerRestrictions:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const CanMoveToContainerRestrictions &other) const
- bool operator!= (const CanMoveToContainerRestrictions &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< CanMoveToContainerStatus > canMoveToContainer

7.13.1 Detailed Description

Specifies if the client can move a Notebook to a Workspace.

7.13.2 Member Function Documentation

7.13.2.1 operator"!=()

```
bool qevercloud::CanMoveToContainerRestrictions::operator!= (
            const CanMoveToContainerRestrictions & other ) const  [inline]
```

7.13.2.2 operator==()

```
bool qevercloud::CanMoveToContainerRestrictions::operator== (
            const CanMoveToContainerRestrictions & other ) const  [inline]
```

7.13.2.3 print()

```
virtual void qevercloud::CanMoveToContainerRestrictions::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.13.3 Member Data Documentation

7.13.3.1 canMoveToContainer

```
Optional< CanMoveToContainerStatus > qevercloud::CanMoveToContainerRestrictions::canMoveTo↩
Container
```

NOT DOCUMENTED

7.13.3.2 localData

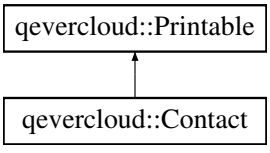
```
EverCloudLocalData qevercloud::CanMoveToContainerRestrictions::localData
```

See the declaration of EverCloudLocalData for details

7.14 qevercloud::Contact Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::Contact:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const Contact &other) const
- bool operator!= (const Contact &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > name
- Optional< QString > id
- Optional< ContactType > type
- Optional< QString > photoUrl
- Optional< Timestamp > photoLastUpdated
- Optional< QByteArray > messagingPermit
- Optional< Timestamp > messagingPermitExpires

7.14.1 Detailed Description

A structure that represents contact information. Note this does not necessarily correspond to an Evernote user.

7.14.2 Member Function Documentation

7.14.2.1 operator"!=()

```
bool qevercloud::Contact::operator!= (
            const Contact & other ) const  [inline]
```

7.14.2.2 operator==()

```
bool qevercloud::Contact::operator== (
            const Contact & other ) const  [inline]
```

7.14.2.3 print()

```
virtual void qevercloud::Contact::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.14.3 Member Data Documentation

7.14.3.1 id

`Optional< QString > qevercloud::Contact::id`

A unique identifier for this ContactType.

7.14.3.2 localData

`EverCloudLocalData qevercloud::Contact::localData`

See the declaration of EverCloudLocalData for details

7.14.3.3 messagingPermit

`Optional< QByteArray > qevercloud::Contact::messagingPermit`

This field will only be filled by the service when it is giving a Contact record to a client, and that client does not normally have enough permission to send a new message to the person represented through this Contact. In that case, this whole Contact record could be used to send a new Message to the Contact, and the service will inspect this permit to confirm that operation was allowed.

7.14.3.4 messagingPermitExpires

`Optional< Timestamp > qevercloud::Contact::messagingPermitExpires`

If this field is set, then this (whole) Contact record may be used in calls to sendMessage until this time. After that time, those calls may be rejected by the service if the caller does not have direct permission to initiate a message with the represented Evernote user.

7.14.3.5 name

`Optional< QString > qevercloud::Contact::name`

The displayable name of this contact. This field is filled in by the service and is read-only to clients.

7.14.3.6 photoLastUpdated

`Optional< Timestamp > qevercloud::Contact::photoLastUpdated`

timestamp when the profile photo at 'photoUrl' was last updated. This field will be null if the user has never set a profile photo. This field is filled in by the service and is read-only to clients.

7.14.3.7 photoUrl

`Optional< QString > qevercloud::Contact::photoUrl`

A URL of a profile photo representing this Contact. This field is filled in by the service and is read-only to clients.

7.14.3.8 type

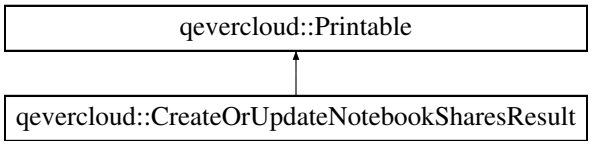
```
Optional< ContactType > qevercloud::Contact::type
```

What service does this contact come from?

7.15 qevercloud::CreateOrUpdateNotebookSharesResult Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::CreateOrUpdateNotebookSharesResult:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const CreateOrUpdateNotebookSharesResult &other) const
- bool operator!= (const CreateOrUpdateNotebookSharesResult &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< qint32 > updateSequenceNum
- Optional< QList< SharedNotebook > > matchingShares

Properties

- OptionalQList< SharedNotebook > matchingShares

7.15.1 Detailed Description

A structure containing the results of a call to createOrUpdateNotebookShares.

7.15.2 Member Function Documentation

7.15.2.1 operator"!=()

```
bool qevercloud::CreateOrUpdateNotebookSharesResult::operator!= (
            const CreateOrUpdateNotebookSharesResult & other ) const  [inline]
```

7.15.2.2 operator==()

```
bool qevercloud::CreateOrUpdateNotebookSharesResult::operator== (
            const CreateOrUpdateNotebookSharesResult & other ) const  [inline]
```

7.15.2.3 print()

```
virtual void qevercloud::CreateOrUpdateNotebookSharesResult::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.15.3 Member Data Documentation

7.15.3.1 localData

```
EverCloudLocalData qevercloud::CreateOrUpdateNotebookSharesResult::localData
```

See the declaration of EverCloudLocalData for details

7.15.3.2 matchingShares

```
Optional<QList<SharedNotebook> > qevercloud::CreateOrUpdateNotebookSharesResult::matching↩
Shares
```

A list of SharedNotebook records that match the desired recipients. These records may have been either created or updated by the call to createOrUpdateNotebookShares, or they may have been at the desired privilege privilege level prior to the call.

7.15.3.3 updateSequenceNum

```
Optional< qint32 > qevercloud::CreateOrUpdateNotebookSharesResult::updateSequenceNum
```

The USN of the notebook after the call.

7.15.4 Property Documentation

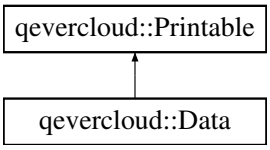
7.15.4.1 matchingShares

```
OptionalQList<SharedNotebook> qevercloud::CreateOrUpdateNotebookSharesResult::matchingShares
```

7.16 qevercloud::Data Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::Data:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const Data &other) const
- bool operator!= (const Data &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QByteArray > bodyHash
- Optional< qint32 > size
- Optional< QByteArray > body

7.16.1 Detailed Description

In several places, EDAM exchanges blocks of bytes of data for a component which may be relatively large. For example: the contents of a clipped HTML note, the bytes of an embedded image, or the recognition XML for a large image. This structure is used in the protocol to represent any of those large blocks of data when they are transmitted or when they are only referenced their metadata.

7.16.2 Member Function Documentation

7.16.2.1 operator"!=()

```
bool qevercloud::Data::operator!= (
            const Data & other ) const  [inline]
```

7.16.2.2 operator==()

```
bool qevercloud::Data::operator== (
            const Data & other ) const  [inline]
```

7.16.2.3 print()

```
virtual void qevercloud::Data::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.16.3 Member Data Documentation

7.16.3.1 body

```
Optional< QByteArray > qevercloud::Data::body
```

This field is set to contain the binary contents of the data whenever the resource is being transferred. If only metadata is being exchanged, this field will be empty. For example, a client could notify the service about the change to an attribute for a resource without transmitting the binary resource contents.

7.16.3.2 bodyHash

```
Optional< QByteArray > qevercloud::Data::bodyHash
```

This field carries a one-way hash of the contents of the data body, in binary form. The hash function is MD5 Length: EDAM_HASH_LEN (exactly)

7.16.3.3 localData

```
EverCloudLocalData qevercloud::Data::localData
```

See the declaration of EverCloudLocalData for details

7.16.3.4 size

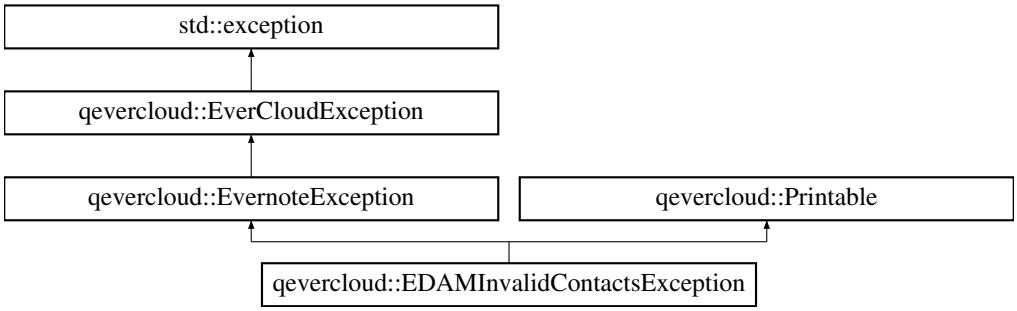
```
Optional< qint32 > qevercloud::Data::size
```

The length, in bytes, of the data body.

7.17 qevercloud::EDAMInvalidContactsException Class Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::EDAMInvalidContactsException:



Public Member Functions

- EDAMInvalidContactsException ()
- virtual ~EDAMInvalidContactsException () noexcept override
- EDAMInvalidContactsException (const EDAMInvalidContactsException &other)
- const char * what () const noexcept override
- virtual EverCloudExceptionDataPtr exceptionData () const override
- virtual void print (QTextStream &strm) const override
- bool operator== (const EDAMInvalidContactsException &other) const
- bool operator!= (const EDAMInvalidContactsException &other) const

Public Member Functions inherited from qevercloud::EvernoteException

- EvernoteException ()
- EvernoteException (QString error)
- EvernoteException (const std::string &error)
- EvernoteException (const char *error)

Public Member Functions inherited from qevercloud::EverCloudException

- EverCloudException ()
- EverCloudException (QString error)
- EverCloudException (const std::string &error)
- EverCloudException (const char *error)
- virtual ~EverCloudException () noexcept override

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- QList< Contact > contacts
- Optional< QString > parameter
- Optional< QList< EDAMInvalidContactReason > > reasons

Properties

- OptionalQList< EDAMInvalidContactReason > reasons

Additional Inherited Members

Protected Attributes inherited from qevercloud::EverCloudException

- QByteArray m_error

7.17.1 Detailed Description

An exception thrown when the provided Contacts fail validation. For instance, email domains could be invalid, phone numbers might not be valid for SMS, etc.

We will not provide individual reasons for each Contact's validation failure. The presence of the Contact in this exception means that the user must figure out how to take appropriate action to fix this Contact.

contacts The list of Contacts that are considered invalid by the service

parameter If the error applied to a particular input parameter, this will indicate which parameter.

reasons If supplied, the list of reasons why the server considered a contact invalid, matching, in order, the list returned in the contacts field.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 EDAMInvalidContactsException() [1/2]

```
qevercloud::EDAMInvalidContactsException::EDAMInvalidContactsException ( )
```

7.17.2.2 ~EDAMInvalidContactsException()

```
virtual qevercloud::EDAMInvalidContactsException::~EDAMInvalidContactsException ( )  [override],
[virtual], [noexcept]
```

7.17.2.3 EDAMInvalidContactsException() [2/2]

```
qevercloud::EDAMInvalidContactsException::EDAMInvalidContactsException (
            const EDAMInvalidContactsException & other )
```

7.17.3 Member Function Documentation

7.17.3.1 exceptionData()

```
virtual EverCloudExceptionDataPtr qevercloud::EDAMInvalidContactsException::exceptionData ( )
const  [override], [virtual]
```

Reimplemented from qevercloud::EvernoteException.

7.17.3.2 operator"!=()

```
bool qevercloud::EDAMInvalidContactsException::operator!= (
            const EDAMInvalidContactsException & other ) const  [inline]
```

7.17.3.3 operator==()

```
bool qevercloud::EDAMInvalidContactsException::operator== (
            const EDAMInvalidContactsException & other ) const  [inline]
```

7.17.3.4 print()

```
virtual void qevercloud::EDAMInvalidContactsException::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.17.3.5 what()

```
const char * qevercloud::EDAMInvalidContactsException::what ( ) const  [override], [virtual],
[noexcept]
```

Reimplemented from qevercloud::EverCloudException.

7.17.4 Member Data Documentation

7.17.4.1 contacts

```
QList< Contact > qevercloud::EDAMInvalidContactsException::contacts
```

7.17.4.2 parameter

```
Optional< QString > qevercloud::EDAMInvalidContactsException::parameter
```

7.17.4.3 reasons

```
Optional<QList<EDAMInvalidContactReason> > qevercloud::EDAMInvalidContactsException::reasons
```

7.17.5 Property Documentation

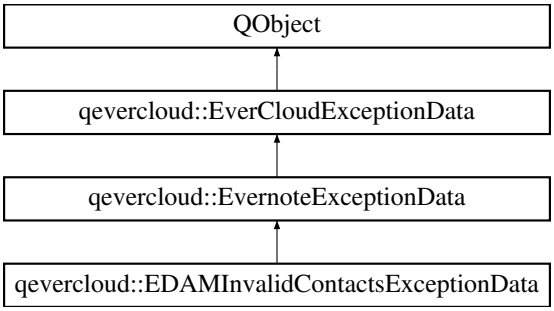
7.17.5.1 reasons

```
OptionalQList<EDAMInvalidContactReason> qevercloud::EDAMInvalidContactsException::reasons
```

7.18 qevercloud::EDAMInvalidContactsExceptionData Class Reference

```
#include <Exceptions.h>
```

Inheritance diagram for qevercloud::EDAMInvalidContactsExceptionData:



Public Member Functions

- EDAMInvalidContactsExceptionData (QList< Contact > contacts, Optional< QString > parameter, Optional< QList< EDAMInvalidContactReason > > reasons)
- virtual void throwException () const override

Public Member Functions inherited from qevercloud::EvernoteExceptionData

- EvernoteExceptionData (QString error)

Public Member Functions inherited from qevercloud::EverCloudExceptionData

- EverCloudExceptionData (QString error)

Protected Attributes

- QList< Contact > m_contacts
- Optional< QString > m_parameter
- Optional< QList< EDAMInvalidContactReason > > m_reasons

Additional Inherited Members

Public Attributes inherited from qevercloud::EverCloudExceptionData

- QString errorMessage

7.18.1 Detailed Description

Asynchronous API counterpart of EDAMInvalidContactsException. See EverCloudExceptionData for more details.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 EDAMInvalidContactsExceptionData()

```
qevercloud::EDAMInvalidContactsExceptionData::EDAMInvalidContactsExceptionData (
            QList< Contact > contacts,
            Optional< QString > parameter,
            Optional< QList< EDAMInvalidContactReason > > reasons )  [explicit]
```

7.18.3 Member Function Documentation

7.18.3.1 throwException()

```
virtual void qevercloud::EDAMInvalidContactsExceptionData::throwException ( ) const  [override],
[virtual]
```

If you want to throw an exception that corresponds to a received EverCloudExceptionData descendant than call this function. Do not use `throw` statement, it's not polymorphic.

Reimplemented from qevercloud::EvernoteExceptionData.

7.18.4 Member Data Documentation

7.18.4.1 m_contacts

```
QList<Contact> qevercloud::EDAMInvalidContactsExceptionData::m_contacts  [protected]
```

7.18.4.2 m_parameter

```
Optional<QString> qevercloud::EDAMInvalidContactsExceptionData::m_parameter  [protected]
```

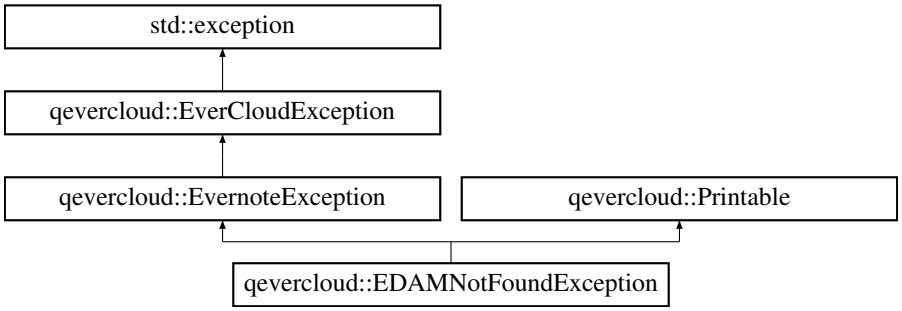
7.18.4.3 m_reasons

```
Optional<QList<EDAMInvalidContactReason> > qevercloud::EDAMInvalidContactsExceptionData::m_↩
reasons  [protected]
```

7.19 qevercloud::EDAMNotFoundException Class Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::EDAMNotFoundException:



Public Member Functions

- EDAMNotFoundException ()
- virtual ~EDAMNotFoundException () noexcept override
- EDAMNotFoundException (const EDAMNotFoundException &other)
- const char * what () const noexcept override
- virtual EverCloudExceptionDataPtr exceptionData () const override
- virtual void print (QTextStream &strm) const override
- bool operator== (const EDAMNotFoundException &other) const
- bool operator!= (const EDAMNotFoundException &other) const

Public Member Functions inherited from qevercloud::EvernoteException

- EvernoteException ()
- EvernoteException (QString error)
- EvernoteException (const std::string &error)
- EvernoteException (const char *error)

Public Member Functions inherited from qevercloud::EverCloudException

- EverCloudException ()
- EverCloudException (QString error)
- EverCloudException (const std::string &error)
- EverCloudException (const char *error)
- virtual ~EverCloudException () noexcept override

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- Optional< QString > identifier
- Optional< QString > key

Additional Inherited Members

Protected Attributes inherited from qevercloud::EverCloudException

- QByteArray m_error

7.19.1 Detailed Description

This exception is thrown by EDAM procedures when a caller asks to perform an operation on an object that does not exist. This may be thrown based on an invalid primary identifier (e.g. a bad GUID), or when the caller refers to an object by another unique identifier (e.g. a User's email address).

identifier: A description of the object that was not found on the server. For example, "Note.notebookGuid" when a caller attempts to create a note in a notebook that does not exist in the user's account.

key: The value passed from the client in the identifier, which was not found. For example, the GUID that was not found.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 EDAMNotFoundException() [1/2]

```
qevercloud::EDAMNotFoundException::EDAMNotFoundException ( )
```

7.19.2.2 ~EDAMNotFoundException()

```
virtual qevercloud::EDAMNotFoundException::~EDAMNotFoundException ( )  [override], [virtual],
[noexcept]
```

7.19.2.3 EDAMNotFoundException() [2/2]

```
qevercloud::EDAMNotFoundException::EDAMNotFoundException (
            const EDAMNotFoundException & other )
```

7.19.3 Member Function Documentation

7.19.3.1 exceptionData()

```
virtual EverCloudExceptionDataPtr qevercloud::EDAMNotFoundException::exceptionData ( ) const
[override], [virtual]
```

Reimplemented from qevercloud::EvernoteException.

7.19.3.2 operator"!=()

```
bool qevercloud::EDAMNotFoundException::operator!= (
            const EDAMNotFoundException & other ) const  [inline]
```

7.19.3.3 operator==()

```
bool qevercloud::EDAMNotFoundException::operator== (
            const EDAMNotFoundException & other ) const  [inline]
```

7.19.3.4 print()

```
virtual void qevercloud::EDAMNotFoundException::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.19.3.5 what()

```
const char * qevercloud::EDAMNotFoundException::what ( ) const  [override], [virtual], [noexcept]
```

Reimplemented from qevercloud::EverCloudException.

7.19.4 Member Data Documentation

7.19.4.1 identifier

```
Optional< QString > qevercloud::EDAMNotFoundException::identifier
```

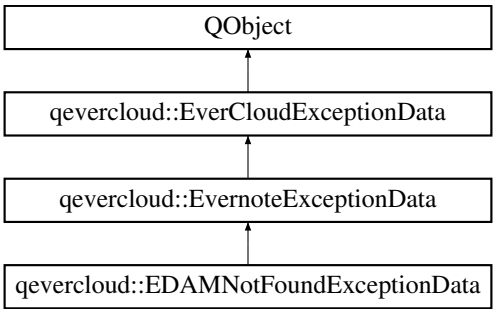
7.19.4.2 key

```
Optional< QString > qevercloud::EDAMNotFoundException::key
```

7.20 qevercloud::EDAMNotFoundExceptionData Class Reference

```
#include <Exceptions.h>
```

Inheritance diagram for qevercloud::EDAMNotFoundExceptionData:



Public Member Functions

- EDAMNotFoundExceptionData (QString error, Optional< QString > identifier, Optional< QString > key)
- virtual void throwException () const override

Public Member Functions inherited from qevercloud::EvernoteExceptionData

- EvernoteExceptionData (QString error)

Public Member Functions inherited from qevercloud::EverCloudExceptionData

- EverCloudExceptionData (QString error)

Protected Attributes

- Optional< QString > m_identifier
- Optional< QString > m_key

Additional Inherited Members

Public Attributes inherited from qevercloud::EverCloudExceptionData

- QString errorMessage

7.20.1 Detailed Description

Asynchronous API counterpart of EDAMNotFoundException. See EverCloudExceptionData for more details.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 EDAMNotFoundExceptionData()

```
qevercloud::EDAMNotFoundExceptionData::EDAMNotFoundExceptionData (
            QString error,
            Optional< QString > identifier,
            Optional< QString > key )  [explicit]
```

7.20.3 Member Function Documentation

7.20.3.1 throwException()

```
virtual void qevercloud::EDAMNotFoundExceptionData::throwException ( ) const  [override],
[virtual]
```

If you want to throw an exception that corresponds to a received EverCloudExceptionData descendant than call this function. Do not use `throw` statement, it's not polymorphic.

Reimplemented from qevercloud::EvernoteExceptionData.

7.20.4 Member Data Documentation

7.20.4.1 m_identifier

```
Optional<QString> qevercloud::EDAMNotFoundExceptionData::m_identifier  [protected]
```

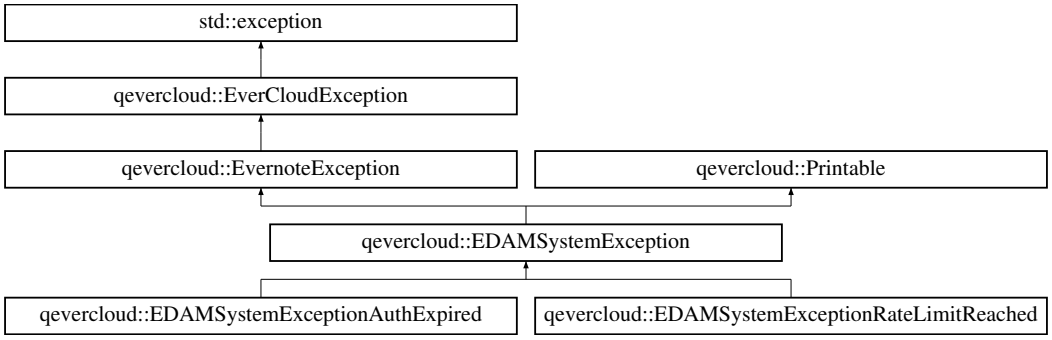
7.20.4.2 m_key

```
Optional<QString> qevercloud::EDAMNotFoundExceptionData::m_key  [protected]
```

7.21 qevercloud::EDAMSystemException Class Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::EDAMSystemException:



Public Member Functions

- EDAMSystemException ()
- virtual ~EDAMSystemException () noexcept override
- EDAMSystemException (const EDAMSystemException &other)
- const char * what () const noexcept override
- virtual EverCloudExceptionDataPtr exceptionData () const override
- virtual void print (QTextStream &strm) const override
- bool operator== (const EDAMSystemException &other) const
- bool operator!= (const EDAMSystemException &other) const

Public Member Functions inherited from qevercloud::EvernoteException

- EvernoteException ()
- EvernoteException (QString error)
- EvernoteException (const std::string &error)
- EvernoteException (const char *error)

Public Member Functions inherited from qevercloud::EverCloudException

- EverCloudException ()
- EverCloudException (QString error)
- EverCloudException (const std::string &error)
- EverCloudException (const char *error)
- virtual ~EverCloudException () noexcept override

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EDAMErrorCode errorCode
- Optional< QString > message
- Optional< qint32 > rateLimitDuration

Additional Inherited Members

Protected Attributes inherited from qevercloud::EverCloudException

- QByteArray m_error

7.21.1 Detailed Description

This exception is thrown by EDAM procedures when a call fails as a result of a problem in the service that could not be changed through caller action.

errorCode: The numeric code indicating the type of error that occurred. must be one of the values of EDAMError↩ Code.

message: This may contain additional information about the error

rateLimitDuration: Indicates the minimum number of seconds that an application should expect subsequent API calls for this user to fail. The application should not retry API requests for the user until at least this many seconds have passed. Present only when errorCode is RATE_LIMIT_REACHED,

7.21.2 Constructor & Destructor Documentation

7.21.2.1 EDAMSystemException() [1/2]

```
qevercloud::EDAMSystemException::EDAMSystemException ( )
```

7.21.2.2 ~EDAMSystemException()

```
virtual qevercloud::EDAMSystemException::~EDAMSystemException ( )  [override], [virtual],
[noexcept]
```

7.21.2.3 EDAMSystemException() [2/2]

```
qevercloud::EDAMSystemException::EDAMSystemException (
            const EDAMSystemException & other )
```

7.21.3 Member Function Documentation

7.21.3.1 exceptionData()

```
virtual EverCloudExceptionDataPtr qevercloud::EDAMSystemException::exceptionData ( ) const
[override], [virtual]
```

Reimplemented from qevercloud::EvernoteException.

Reimplemented in qevercloud::EDAMSystemExceptionRateLimitReached, and qevercloud::EDAMSystemExceptionAuthExpired.

7.21.3.2 operator"!=()

```
bool qevercloud::EDAMSystemException::operator!= (
            const EDAMSystemException & other ) const  [inline]
```

7.21.3.3 operator==()

```
bool qevercloud::EDAMSystemException::operator== (
            const EDAMSystemException & other ) const  [inline]
```

7.21.3.4 print()

```
virtual void qevercloud::EDAMSystemException::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.21.3.5 what()

```
const char * qevercloud::EDAMSystemException::what ( ) const  [override], [virtual], [noexcept]
```

Reimplemented from qevercloud::EverCloudException.

7.21.4 Member Data Documentation

7.21.4.1 errorCode

```
EDAMErrorCode qevercloud::EDAMSystemException::errorCode
```

7.21.4.2 message

```
Optional< QString > qevercloud::EDAMSystemException::message
```

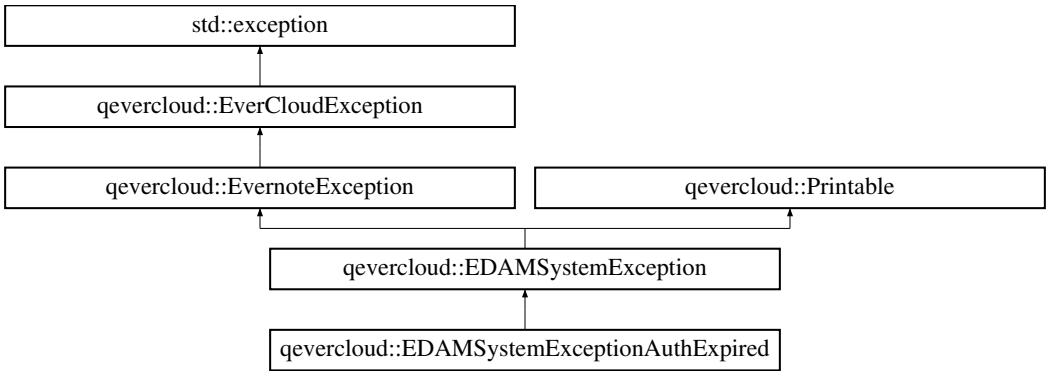
7.21.4.3 rateLimitDuration

```
Optional< qint32 > qevercloud::EDAMSystemException::rateLimitDuration
```

7.22 qevercloud::EDAMSystemExceptionAuthExpired Class Reference

```
#include <Exceptions.h>
```

Inheritance diagram for qevercloud::EDAMSystemExceptionAuthExpired:



Public Member Functions

- virtual EverCloudExceptionDataPtr exceptionData () const override

Public Member Functions inherited from qevercloud::EDAMSystemException

- EDAMSystemException ()
- virtual ~EDAMSystemException () noexcept override
- EDAMSystemException (const EDAMSystemException &other)
- const char * what () const noexcept override
- virtual void print (QTextStream &strm) const override
- bool operator== (const EDAMSystemException &other) const
- bool operator!= (const EDAMSystemException &other) const

Public Member Functions inherited from qevercloud::EvernoteException

- EvernoteException ()
- EvernoteException (QString error)
- EvernoteException (const std::string &error)
- EvernoteException (const char *error)

Public Member Functions inherited from qevercloud::EverCloudException

- EverCloudException ()
- EverCloudException (QString error)
- EverCloudException (const std::string &error)
- EverCloudException (const char *error)
- virtual ~EverCloudException () noexcept override

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Additional Inherited Members

Public Attributes inherited from qevercloud::EDAMSystemException

- EDAMErrorCode errorCode
- Optional< QString > message
- Optional< qint32 > rateLimitDuration

Protected Attributes inherited from qevercloud::EverCloudException

- QByteArray m_error

7.22.1 Detailed Description

EDAMSystemException for `errorCode = AUTH_EXPIRED`

7.22.2 Member Function Documentation

7.22.2.1 exceptionData()

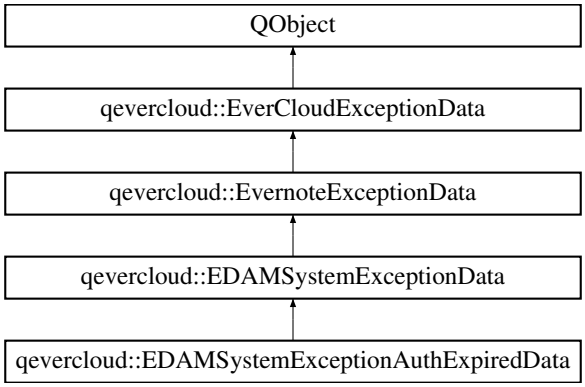
```
virtual EverCloudExceptionDataPtr qevercloud::EDAMSystemExceptionAuthExpired::exceptionData (
) const  [override], [virtual]
```

Reimplemented from qevercloud::EDAMSystemException.

7.23 qevercloud::EDAMSystemExceptionAuthExpiredData Class Reference

`#include <Exceptions.h>`

Inheritance diagram for qevercloud::EDAMSystemExceptionAuthExpiredData:



Public Member Functions

- EDAMSystemExceptionAuthExpiredData (QString error, EDAMErrorCode errorCode, Optional< QString > message, Optional< qint32 > rateLimitDuration)
- virtual void throwException () const override

Public Member Functions inherited from qevercloud::EDAMSystemExceptionData

- EDAMSystemExceptionData (QString err, EDAMErrorCode errorCode, Optional< QString > message, Optional< qint32 > rateLimitDuration)

Public Member Functions inherited from qevercloud::EvernoteExceptionData

- EvernoteExceptionData (QString error)

Public Member Functions inherited from qevercloud::EverCloudExceptionData

- EverCloudExceptionData (QString error)

Additional Inherited Members

Public Attributes inherited from qevercloud::EverCloudExceptionData

- QString errorMessage

Protected Attributes inherited from qevercloud::EDAMSystemExceptionData

- EDAMErrorCode m_errorCode
- Optional< QString > m_message
- Optional< qint32 > m_rateLimitDuration

7.23.1 Detailed Description

Asynchronous API counterpart of EDAMSystemExceptionAuthExpired. See EverCloudExceptionData for more details.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 EDAMSystemExceptionAuthExpiredData()

```
qevercloud::EDAMSystemExceptionAuthExpiredData::EDAMSystemExceptionAuthExpiredData (
            QString error,
            EDAMErrorCode errorCode,
            Optional< QString > message,
            Optional< qint32 > rateLimitDuration )  [explicit]
```

7.23.3 Member Function Documentation

7.23.3.1 throwException()

```
virtual void qevercloud::EDAMSystemExceptionAuthExpiredData::throwException ( ) const  [override],
[virtual]
```

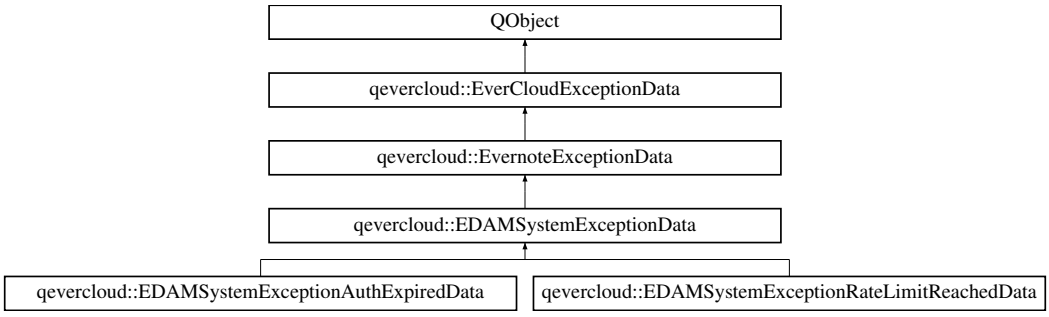
If you want to throw an exception that corresponds to a received EverCloudExceptionData descendant than call this function. Do not use `throw` statement, it's not polymorphic.

Reimplemented from qevercloud::EDAMSystemExceptionData.

7.24 qevercloud::EDAMSystemExceptionData Class Reference

```
#include <Exceptions.h>
```

Inheritance diagram for qevercloud::EDAMSystemExceptionData:



Public Member Functions

- EDAMSystemExceptionData (QString err, EDAMErrorCode errorCode, Optional< QString > message, Optional< qint32 > rateLimitDuration)
- virtual void throwException () const override

Public Member Functions inherited from qevercloud::EvernoteExceptionData

- EvernoteExceptionData (QString error)

Public Member Functions inherited from qevercloud::EverCloudExceptionData

- EverCloudExceptionData (QString error)

Protected Attributes

- EDAMErrorCode m_errorCode
- Optional< QString > m_message
- Optional< qint32 > m_rateLimitDuration

Additional Inherited Members

Public Attributes inherited from qevercloud::EverCloudExceptionData

- QString errorMessage

7.24.1 Detailed Description

Asynchronous API counterpart of EDAMSystemException. See EverCloudExceptionData for more details.

7.24.2 Constructor & Destructor Documentation

7.24.2.1 EDAMSystemExceptionData()

```
qevercloud::EDAMSystemExceptionData::EDAMSystemExceptionData (
            QString err,
            EDAMErrorCode errorCode,
            Optional< QString > message,
            Optional< qint32 > rateLimitDuration )  [explicit]
```

7.24.3 Member Function Documentation

7.24.3.1 throwException()

```
virtual void qevercloud::EDAMSystemExceptionData::throwException ( ) const  [override], [virtual]
```

If you want to throw an exception that corresponds to a received EverCloudExceptionData descendant than call this function. Do not use `throw` statement, it's not polymorphic.

Reimplemented from qevercloud::EvernoteExceptionData.

Reimplemented in qevercloud::EDAMSystemExceptionRateLimitReachedData, and qevercloud::EDAMSystemExceptionAuthExpiredD

7.24.4 Member Data Documentation

7.24.4.1 m_errorCode

```
EDAMErrorCode qevercloud::EDAMSystemExceptionData::m_errorCode  [protected]
```

7.24.4.2 m_message

```
Optional<QString> qevercloud::EDAMSystemExceptionData::m_message  [protected]
```

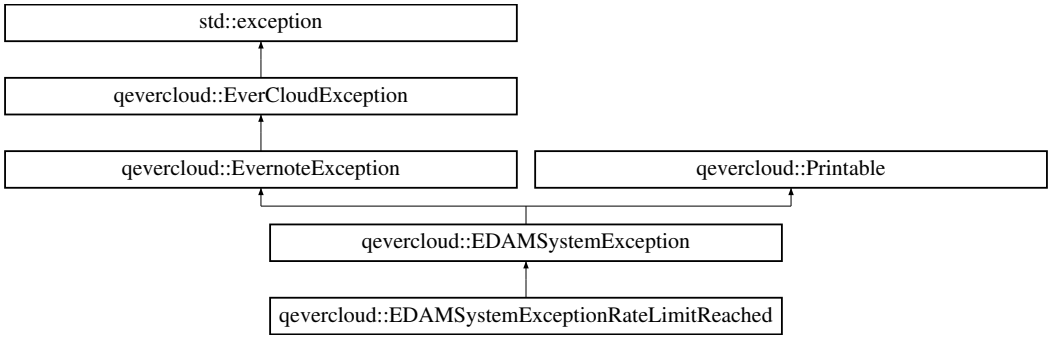
7.24.4.3 m_rateLimitDuration

```
Optional<qint32> qevercloud::EDAMSystemExceptionData::m_rateLimitDuration  [protected]
```

7.25 qevercloud::EDAMSystemExceptionRateLimitReached Class Reference

```
#include <Exceptions.h>
```

Inheritance diagram for qevercloud::EDAMSystemExceptionRateLimitReached:



Public Member Functions

- virtual EverCloudExceptionDataPtr exceptionData () const override

Public Member Functions inherited from qevercloud::EDAMSystemException

- EDAMSystemException ()
- virtual ~EDAMSystemException () noexcept override
- EDAMSystemException (const EDAMSystemException &other)
- const char * what () const noexcept override
- virtual void print (QTextStream &strm) const override
- bool operator== (const EDAMSystemException &other) const
- bool operator!= (const EDAMSystemException &other) const

Public Member Functions inherited from qevercloud::EvernoteException

- EvernoteException ()
- EvernoteException (QString error)
- EvernoteException (const std::string &error)
- EvernoteException (const char *error)

Public Member Functions inherited from qevercloud::EverCloudException

- EverCloudException ()
- EverCloudException (QString error)
- EverCloudException (const std::string &error)
- EverCloudException (const char *error)
- virtual ~EverCloudException () noexcept override

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Additional Inherited Members

Public Attributes inherited from qevercloud::EDAMSystemException

- EDAMErrorCode errorCode
- Optional< QString > message
- Optional< qint32 > rateLimitDuration

Protected Attributes inherited from qevercloud::EverCloudException

- QByteArray m_error

7.25.1 Detailed Description

EDAMSystemException for `errorCode = RATE_LIMIT_REACHED`

7.25.2 Member Function Documentation

7.25.2.1 exceptionData()

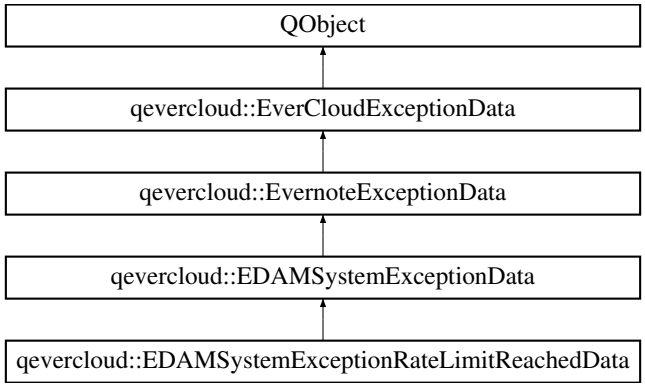
```
virtual EverCloudExceptionDataPtr qevercloud::EDAMSystemExceptionRateLimitReached::exception↩
Data ( ) const  [override], [virtual]
```

Reimplemented from qevercloud::EDAMSystemException.

7.26 qevercloud::EDAMSystemExceptionRateLimitReachedData Class Reference

`#include <Exceptions.h>`

Inheritance diagram for qevercloud::EDAMSystemExceptionRateLimitReachedData:



Public Member Functions

- EDAMSystemExceptionRateLimitReachedData (QString error, EDAMErrorCode errorCode, Optional< QString > message, Optional< qint32 > rateLimitDuration)
- virtual void throwException () const override

Public Member Functions inherited from qevercloud::EDAMSystemExceptionData

- EDAMSystemExceptionData (QString err, EDAMErrorCode errorCode, Optional< QString > message, Optional< qint32 > rateLimitDuration)

Public Member Functions inherited from qevercloud::EvernoteExceptionData

- EvernoteExceptionData (QString error)

Public Member Functions inherited from qevercloud::EverCloudExceptionData

- EverCloudExceptionData (QString error)

Additional Inherited Members

Public Attributes inherited from qevercloud::EverCloudExceptionData

- QString errorMessage

Protected Attributes inherited from qevercloud::EDAMSystemExceptionData

- EDAMErrorCode m_errorCode
- Optional< QString > m_message
- Optional< qint32 > m_rateLimitDuration

7.26.1 Detailed Description

Asynchronous API counterpart of EDAMSystemExceptionRateLimitReached. See EverCloudExceptionData for more details.

7.26.2 Constructor & Destructor Documentation

7.26.2.1 EDAMSystemExceptionRateLimitReachedData()

```
qevercloud::EDAMSystemExceptionRateLimitReachedData::EDAMSystemExceptionRateLimitReachedData (
            QString error,
            EDAMErrorCode errorCode,
            Optional< QString > message,
            Optional< qint32 > rateLimitDuration )  [explicit]
```

7.26.3 Member Function Documentation

7.26.3.1 throwException()

```
virtual void qevercloud::EDAMSystemExceptionRateLimitReachedData::throwException ( ) const
[override], [virtual]
```

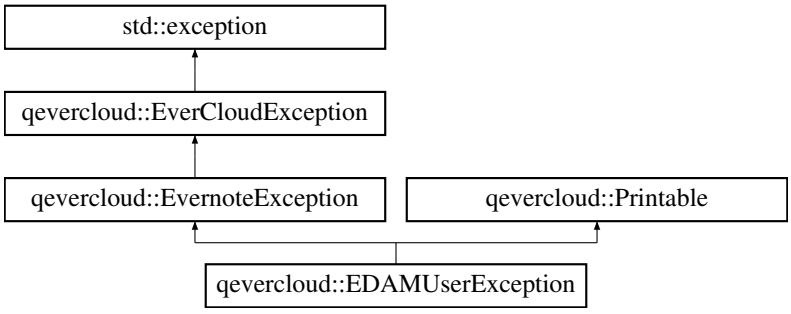
If you want to throw an exception that corresponds to a received EverCloudExceptionData descendant than call this function. Do not use `throw` statement, it's not polymorphic.

Reimplemented from qevercloud::EDAMSystemExceptionData.

7.27 qevercloud::EDAMUserException Class Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::EDAMUserException:



Public Member Functions

- EDAMUserException ()
- virtual ~EDAMUserException () noexcept override
- EDAMUserException (const EDAMUserException &other)
- const char * what () const noexcept override
- virtual EverCloudExceptionDataPtr exceptionData () const override
- virtual void print (QTextStream &strm) const override
- bool operator== (const EDAMUserException &other) const
- bool operator!= (const EDAMUserException &other) const

Public Member Functions inherited from qevercloud::EvernoteException

- EvernoteException ()
- EvernoteException (QString error)
- EvernoteException (const std::string &error)
- EvernoteException (const char *error)

Public Member Functions inherited from qevercloud::EverCloudException

- EverCloudException ()
- EverCloudException (QString error)
- EverCloudException (const std::string &error)
- EverCloudException (const char *error)
- virtual ~EverCloudException () noexcept override

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EDAMErrorCode errorCode
- Optional< QString > parameter

Additional Inherited Members

Protected Attributes inherited from qevercloud::EverCloudException

- QByteArray m_error

7.27.1 Detailed Description

This exception is thrown by EDAM procedures when a call fails as a result of a problem that a caller may be able to resolve. For example, if the user attempts to add a note to their account which would exceed their storage quota, this type of exception may be thrown to indicate the source of the error so that they can choose an alternate action.

This exception would not be used for internal system errors that do not reflect user actions, but rather reflect a problem within the service that the user cannot resolve.

errorCode: The numeric code indicating the type of error that occurred. must be one of the values of EDAMError↩ Code.

parameter: If the error applied to a particular input parameter, this will indicate which parameter. For some errors (USER_NOT_ASSOCIATED, USER_NOT_REGISTERED, SSO_AUTHENTICATION_REQUIRED), this is the user's email.

7.27.2 Constructor & Destructor Documentation

7.27.2.1 EDAMUserException() [1/2]

```
qevercloud::EDAMUserException::EDAMUserException ( )
```

7.27.2.2 ~EDAMUserException()

```
virtual qevercloud::EDAMUserException::~EDAMUserException ( )  [override], [virtual], [noexcept]
```

7.27.2.3 EDAMUserException() [2/2]

```
qevercloud::EDAMUserException::EDAMUserException (
            const EDAMUserException & other )
```

7.27.3 Member Function Documentation

7.27.3.1 exceptionData()

```
virtual EverCloudExceptionDataPtr qevercloud::EDAMUserException::exceptionData ( ) const  [override],
[virtual]
```

Reimplemented from qevercloud::EvernoteException.

7.27.3.2 operator"!=()

```
bool qevercloud::EDAMUserException::operator!= (
            const EDAMUserException & other ) const  [inline]
```

7.27.3.3 operator==()

```
bool qevercloud::EDAMUserException::operator== (
            const EDAMUserException & other ) const  [inline]
```

7.27.3.4 print()

```
virtual void qevercloud::EDAMUserException::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.27.3.5 what()

```
const char * qevercloud::EDAMUserException::what ( ) const  [override], [virtual], [noexcept]
```

Reimplemented from qevercloud::EverCloudException.

7.27.4 Member Data Documentation

7.27.4.1 errorCode

```
EDAMErrorCode qevercloud::EDAMUserException::errorCode
```

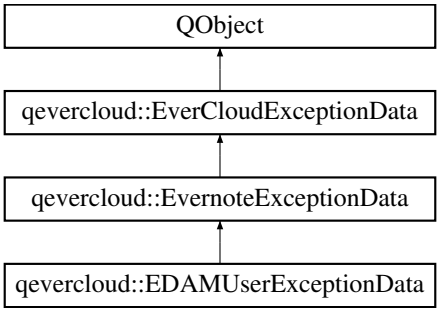
7.27.4.2 parameter

```
Optional< QString > qevercloud::EDAMUserException::parameter
```

7.28 qevercloud::EDAMUserExceptionData Class Reference

```
#include <Exceptions.h>
```

Inheritance diagram for qevercloud::EDAMUserExceptionData:



Public Member Functions

- EDAMUserExceptionData (QString error, EDAMErrorCode errorCode, Optional< QString > parameter)
- virtual void throwException () const override

Public Member Functions inherited from qevercloud::EvernoteExceptionData

- EvernoteExceptionData (QString error)

Public Member Functions inherited from qevercloud::EverCloudExceptionData

- EverCloudExceptionData (QString error)

Protected Attributes

- EDAMErrorCode m_errorCode
- Optional< QString > m_parameter

Additional Inherited Members

Public Attributes inherited from qevercloud::EverCloudExceptionData

- QString errorMessage

7.28.1 Detailed Description

Asynchronous API counterpart of EDAMUserException. See EverCloudExceptionData for more details.

7.28.2 Constructor & Destructor Documentation

7.28.2.1 EDAMUserExceptionData()

```
qevercloud::EDAMUserExceptionData::EDAMUserExceptionData (
            QString error,
            EDAMErrorCode errorCode,
            Optional< QString > parameter )  [explicit]
```

7.28.3 Member Function Documentation

7.28.3.1 throwException()

```
virtual void qevercloud::EDAMUserExceptionData::throwException ( ) const  [override], [virtual]
```

If you want to throw an exception that corresponds to a received EverCloudExceptionData descendant than call this function. Do not use `throw` statement, it's not polymorphic.

Reimplemented from qevercloud::EvernoteExceptionData.

7.28.4 Member Data Documentation

7.28.4.1 m_errorCode

```
EDAMErrorCode qevercloud::EDAMUserExceptionData::m_errorCode  [protected]
```

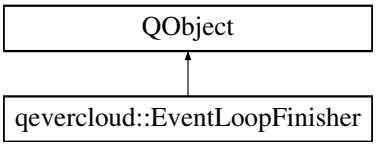
7.28.4.2 m_parameter

```
Optional<QString> qevercloud::EDAMUserExceptionData::m_parameter  [protected]
```

7.29 qevercloud::EventLoopFinisher Class Reference

```
#include <EventLoopFinisher.h>
```

Inheritance diagram for qevercloud::EventLoopFinisher:



Public Slots

- void stopEventLoop ()

Public Member Functions

- EventLoopFinisher (QEventLoop *loop, int exitCode, QObject *parent=Q_NULLPTR)
- ~EventLoopFinisher ()

7.29.1 Constructor & Destructor Documentation

7.29.1.1 EventLoopFinisher()

```
qevercloud::EventLoopFinisher::EventLoopFinisher (
            QEventLoop * loop,
            int exitCode,
            QObject * parent = Q_NULLPTR )  [explicit]
```

7.29.1.2 ~EventLoopFinisher()

```
qevercloud::EventLoopFinisher::~EventLoopFinisher ( )
```

7.29.2 Member Function Documentation

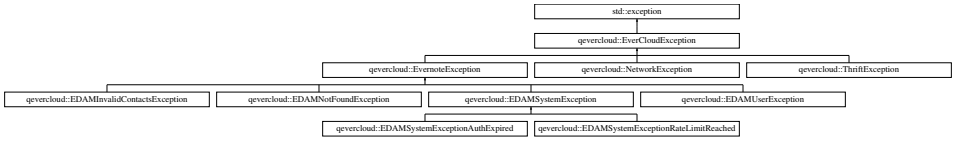
7.29.2.1 stopEventLoop

```
void qevercloud::EventLoopFinisher::stopEventLoop ( )  [slot]
```

7.30 qevercloud::EverCloudException Class Reference

```
#include <EverCloudException.h>
```

Inheritance diagram for qevercloud::EverCloudException:



Public Member Functions

- EverCloudException ()
- EverCloudException (QString error)
- EverCloudException (const std::string &error)
- EverCloudException (const char *error)
- virtual ~EverCloudException () noexcept override
- virtual const char * what () const noexcept override
- virtual std::shared_ptr< EverCloudExceptionData > exceptionData () const

Protected Attributes

- QByteArray m_error

7.30.1 Detailed Description

All exceptions thrown by the library are of this class or its descendants.

7.30.2 Constructor & Destructor Documentation

7.30.2.1 EverCloudException() [1/4]

```
qevercloud::EverCloudException::EverCloudException ( )  [explicit]
```

7.30.2.2 EverCloudException() [2/4]

```
qevercloud::EverCloudException::EverCloudException (
            QString error )  [explicit]
```

7.30.2.3 EverCloudException() [3/4]

```
qevercloud::EverCloudException::EverCloudException (
            const std::string & error )  [explicit]
```

7.30.2.4 EverCloudException() [4/4]

```
qevercloud::EverCloudException::EverCloudException (
            const char * error )  [explicit]
```

7.30.2.5 ~EverCloudException()

```
virtual qevercloud::EverCloudException::~EverCloudException ( )  [override], [virtual], [noexcept]
```

7.30.3 Member Function Documentation

7.30.3.1 exceptionData()

```
virtual std::shared_ptr< EverCloudExceptionData > qevercloud::EverCloudException::exception↩
Data ( ) const  [virtual]
```

Reimplemented in qevercloud::EvernoteException, qevercloud::NetworkException, qevercloud::ThriftException, qevercloud::EDAMSystemExceptionRateLimitReached, qevercloud::EDAMSystemExceptionAuthExpired, qevercloud::EDAMUserExc qevercloud::EDAMSystemException, qevercloud::EDAMNotFoundException, and qevercloud::EDAMInvalidContactsException.

7.30.3.2 what()

```
virtual const char * qevercloud::EverCloudException::what ( ) const  [override], [virtual],
[noexcept]
```

Reimplemented in qevercloud::NetworkException, qevercloud::ThriftException, qevercloud::EDAMUserException, qevercloud::EDAMSystemException, qevercloud::EDAMNotFoundException, and qevercloud::EDAMInvalidContactsException.

7.30.4 Member Data Documentation

7.30.4.1 m_error

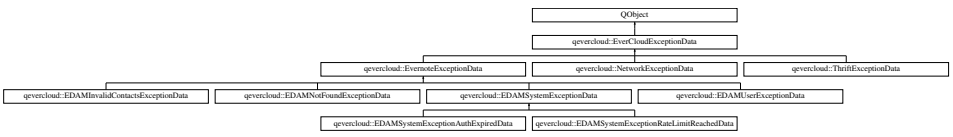
```
QByteArray qevercloud::EverCloudException::m_error  [mutable], [protected]
```

7.31 qevercloud::EverCloudExceptionData Class Reference

EverCloudException counterpart for asynchronous API.

```
#include <EverCloudException.h>
```

Inheritance diagram for qevercloud::EverCloudExceptionData:



Public Member Functions

- EverCloudExceptionData (QString error)
- virtual void throwException () const

Public Attributes

- QString errorMessage

7.31.1 Detailed Description

EverCloudException counterpart for asynchronous API.

Asynchronous functions cannot throw exceptions so descendants of EverCloudExceptionData are retunded instead in case of an error. Every exception class has its own counterpart. The EverCloudExceptionData descendants hierarchy is a copy of the EverCloudException descendants hierarchy.

The main reason not to use exception classes directly is that dynamic_cast does not work across module (exe, dll, etc) boundaries in general, while `qobject_cast` do work as expected. That's why I decided to inherit my error classes from QObject.

In general error checking in asynchronous API look like this:

```
NoteStore* ns;
...
QObject::connect(ns->getNotebook(notebookGuid), &AsyncResult::finished,
                 [](QVariant result, EverCloudExceptionData error)
                 {
                     if (!error.isNull())
                     {
                         auto errorNotFound =
                            std::dynamic_pointer_cast<EDAMNotFoundExceptionData>(
                                error);

                         auto errorUser =
                            std::dynamic_pointer_cast<EDAMUserExceptionData>(
                                error);

                         auto errorSystem =
                            std::dynamic_pointer_cast<EDAMSystemExceptionData>(
                                error);

                         if (!errorNotFound.isNull())
                         {
                             qDebug() << "notebook not found"
                                 << errorNotFound.identifier << errorNotFound.key;
                         }
                         else if (!errorUser.isNull())
                         {
                             qDebug() << errorUser.errorMessage;
                         }
                         else if (!errorSystem.isNull())
                         {
                             if (errorSystem.errorCode ==
                                EDAMErrorCode::RATE_LIMIT_REACHED)
                             {
                                 qDebug() << "Evernote API rate limits are reached";
                             }
                             else if (errorSystem.errorCode ==
                                      EDAMErrorCode::AUTH_EXPIRED)
                             {
                                 qDebug() << "Authorization token is inspired";
                             }
                             else
                             {
                                 // some other Evernote trouble
                                 qDebug() << errorSystem.errorMessage;
                             }
                         }
                         else
                         {
                             // some unexpected error
                             qDebug() << error.errorMessage;
                         }
                     }
                     else
                     {
                         // success
                     }
                 });
```

7.31.2 Constructor & Destructor Documentation

7.31.2.1 EverCloudExceptionData()

```
qevercloud::EverCloudExceptionData::EverCloudExceptionData (
            QString error )  [explicit]
```

7.31.3 Member Function Documentation

7.31.3.1 throwException()

```
virtual void qevercloud::EverCloudExceptionData::throwException ( ) const  [virtual]
```

If you want to throw an exception that corresponds to a received EverCloudExceptionData descendant than call this function. Do not use `throw` statement, it's not polymorphic.

Reimplemented in qevercloud::EvernoteExceptionData, qevercloud::NetworkExceptionData, qevercloud::ThriftExceptionData, qevercloud::EDAMUserExceptionData, qevercloud::EDAMSystemExceptionData, qevercloud::EDAMNotFoundExceptionData, qevercloud::EDAMInvalidContactsExceptionData, qevercloud::EDAMSystemExceptionRateLimitReachedData, and qevercloud::EDAMSystemExceptionAuthExpiredData.

7.31.4 Member Data Documentation

7.31.4.1 errorMessage

```
QString qevercloud::EverCloudExceptionData::errorMessage
```

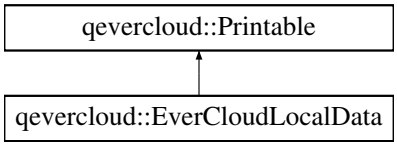
Contains an error message. It's the std::exception::what() counterpart.

7.32 qevercloud::EverCloudLocalData Class Reference

The EverCloudLocalData class contains several data elements which are not synchronized with Evernote service but which are nevertheless useful in applications using QEverCloud to implement feature rich full sync Evernote clients. Values of this class' types are contained within QEverCloud types corresponding to actual Evernote API types.

```
#include <Types.h>
```

Inheritance diagram for qevercloud::EverCloudLocalData:



Public Types

- using Dict = QHash< QString, QVariant >

Public Member Functions

- EverCloudLocalData ()
- virtual ~EverCloudLocalData () noexcept override
- virtual void print (QTextStream &strm) const override
- bool operator== (const EverCloudLocalData &other) const
- bool operator!= (const EverCloudLocalData &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- QString id

  id property can be used as a local unique identifier for any data item before it has been synchronized with Evernote and thus before it can be identified using its guid.
- bool dirty = false

  dirty property can be used to keep track which objects have been modified locally and thus need to be synchronized with Evernote service
- bool local = false

  local property can be used to keep track which data items are meant to be local only and thus never be synchronized with Evernote service
- bool favorited = false

  favorited property can be used to keep track which data items were favorited in the client. Unfortunately, Evernote has never provided a way to synchronize such property between different clients
- QHash< QString, QVariant > dict

  dict can be used for storage of any other auxiliary values associated with objects of QEverCloud types

Properties

- Dict dict

7.32.1 Detailed Description

The EverCloudLocalData class contains several data elements which are not synchronized with Evernote service but which are nevertheless useful in applications using QEverCloud to implement feature rich full sync Evernote clients. Values of this class' types are contained within QEverCloud types corresponding to actual Evernote API types.

7.32.2 Member Typedef Documentation

7.32.2.1 Dict

```
using qevercloud::EverCloudLocalData::Dict = QHash<QString, QVariant>
```

7.32.3 Constructor & Destructor Documentation

7.32.3.1 EverCloudLocalData()

```
qevercloud::EverCloudLocalData::EverCloudLocalData ( )
```

7.32.3.2 ~EverCloudLocalData()

```
virtual qevercloud::EverCloudLocalData::~EverCloudLocalData ( )  [override], [virtual], [noexcept]
```

7.32.4 Member Function Documentation

7.32.4.1 operator"!=()

```
bool qevercloud::EverCloudLocalData::operator!= (
            const EverCloudLocalData & other ) const
```

7.32.4.2 operator==()

```
bool qevercloud::EverCloudLocalData::operator== (
            const EverCloudLocalData & other ) const
```

7.32.4.3 print()

```
virtual void qevercloud::EverCloudLocalData::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.32.5 Member Data Documentation

7.32.5.1 dict

```
QHash<QString, QVariant> qevercloud::EverCloudLocalData::dict
```

dict can be used for storage of any other auxiliary values associated with objects of QEverCloud types

7.32.5.2 dirty

```
bool qevercloud::EverCloudLocalData::dirty = false
```

dirty property can be used to keep track which objects have been modified locally and thus need to be synchronized with Evernote service

7.32.5.3 favorited

```
bool qevercloud::EverCloudLocalData::favorited = false
```

favorited property can be used to keep track which data items were favorited in the client. Unfortunately, Evernote has never provided a way to synchronize such property between different clients

7.32.5.4 id

```
QString qevercloud::EverCloudLocalData::id
```

id property can be used as a local unique identifier for any data item before it has been synchronized with Evernote and thus before it can be identified using its guid.

id property is generated automatically on EverCloudLocalData construction for convenience but can be overridden manually

7.32.5.5 local

```
bool qevercloud::EverCloudLocalData::local = false
```

local property can be used to keep track which data items are meant to be local only and thus never be synchronized with Evernote service

7.32.6 Property Documentation

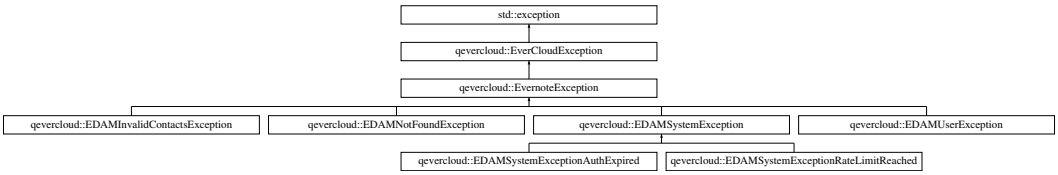
7.32.6.1 dict

```
Dict qevercloud::EverCloudLocalData::dict
```

7.33 qevercloud::EvernoteException Class Reference

```
#include <EverCloudException.h>
```

Inheritance diagram for qevercloud::EvernoteException:



Public Member Functions

- EvernoteException ()
- EvernoteException (QString error)
- EvernoteException (const std::string &error)
- EvernoteException (const char *error)
- virtual EverCloudExceptionDataPtr exceptionData () const override

Public Member Functions inherited from qevercloud::EverCloudException

- EverCloudException ()
- EverCloudException (QString error)
- EverCloudException (const std::string &error)
- EverCloudException (const char *error)
- virtual ~EverCloudException () noexcept override
- virtual const char * what () const noexcept override

Additional Inherited Members

Protected Attributes inherited from qevercloud::EverCloudException

- QByteArray m_error

7.33.1 Detailed Description

All exception sent by Evernote servers (as opposed to other error conditions, for example http errors) are descendants of this class.

7.33.2 Constructor & Destructor Documentation

7.33.2.1 EvernoteException() [1/4]

```
qevercloud::EvernoteException::EvernoteException ( )  [explicit]
```

7.33.2.2 EvernoteException() [2/4]

```
qevercloud::EvernoteException::EvernoteException (
            QString error )  [explicit]
```

7.33.2.3 EvernoteException() [3/4]

```
qevercloud::EvernoteException::EvernoteException (
            const std::string & error )  [explicit]
```

7.33.2.4 EvernoteException() [4/4]

```
qevercloud::EvernoteException::EvernoteException (
            const char * error )  [explicit]
```

7.33.3 Member Function Documentation

7.33.3.1 exceptionData()

```
virtual EverCloudExceptionDataPtr qevercloud::EvernoteException::exceptionData ( ) const  [override],
[virtual]
```

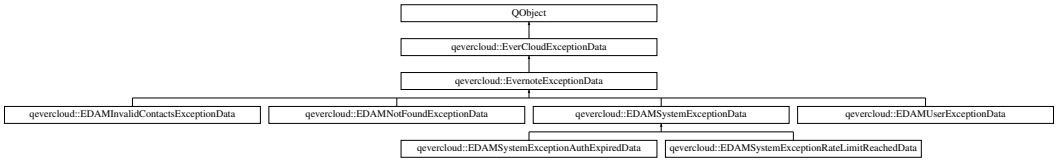
Reimplemented from qevercloud::EverCloudException.

Reimplemented in qevercloud::EDAMSystemExceptionRateLimitReached, qevercloud::EDAMSystemExceptionAuthExpired, qevercloud::EDAMUserException, qevercloud::EDAMSystemException, qevercloud::EDAMNotFoundException, and qevercloud::EDAMInvalidContactsException.

7.34 qevercloud::EvernoteExceptionData Class Reference

```
#include <EverCloudException.h>
```

Inheritance diagram for qevercloud::EvernoteExceptionData:



Public Member Functions

- EvernoteExceptionData (QString error)
- virtual void throwException () const override

Public Member Functions inherited from qevercloud::EverCloudExceptionData

- EverCloudExceptionData (QString error)

Additional Inherited Members

Public Attributes inherited from qevercloud::EverCloudExceptionData

- QString errorMessage

7.34.1 Detailed Description

Asynchronous API conterpart of EvernoteException. See EverCloudExceptionData for more details.

7.34.2 Constructor & Destructor Documentation

7.34.2.1 EvernoteExceptionData()

```
qevercloud::EvernoteExceptionData::EvernoteExceptionData (
            QString error )  [explicit]
```

7.34.3 Member Function Documentation

7.34.3.1 throwException()

```
virtual void qevercloud::EvernoteExceptionData::throwException ( ) const  [override], [virtual]
```

If you want to throw an exception that corresponds to a received EverCloudExceptionData descendant than call this function. Do not use `throw` statement, it's not polymorphic.

Reimplemented from qevercloud::EverCloudExceptionData.

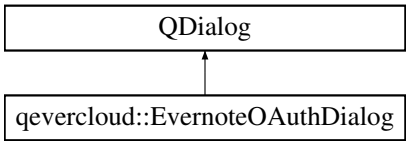
Reimplemented in qevercloud::EDAMUserExceptionData, qevercloud::EDAMSystemExceptionData, qevercloud::EDAMNotFoundExc
qevercloud::EDAMInvalidContactsExceptionData, qevercloud::EDAMSystemExceptionRateLimitReachedData, and
qevercloud::EDAMSystemExceptionAuthExpiredData.

7.35 qevercloud::EvernoteOAuthDialog Class Reference

Authorizes your app with the Evernote service by means of OAuth authentication.

`#include <OAuth.h>`

Inheritance diagram for qevercloud::EvernoteOAuthDialog:



Public Types

- using OAuthResult = EvernoteOAuthWebView::OAuthResult

Public Member Functions

- EvernoteOAuthDialog (QString consumerKey, QString consumerSecret, QString host=QStringLiteral("www.↩
evernote.com"), QWidget *parent=Q_NULLPTR)
- virtual ~EvernoteOAuthDialog () override
- void setWebViewSizeHint (QSize sizeHint)
- bool isSucceeded () const
- QString oauthError () const
- OAuthResult oauthResult () const
- virtual int exec () override
- virtual void open () override

7.35.1 Detailed Description

Authorizes your app with the Evernote service by means of OAuth authentication.

Intended usage:

```
#include <QEverCloudOAuth.h>

EvernoteOAuthDialog d(myConsumerKey, myConsumerSecret);
if(d.exec() == QDialog::Accepted) {
    EvernoteOAuthDialog::OAuthResult res = d.oauthResult();
    // Connect to Evernote
    ...
} else {
    QString errorText = d.oauthError();
    // handle an authentication error
    ...
}
```

Note that you have to include QEverCloudOAuth.h header.

By default EvernoteOAuthDialog uses qrand() for generating nonce so do not forget to call qsrand() in your application. See setNonceGenerator If you want more control over nonce generation.

7.35.2 Member Typedef Documentation

7.35.2.1 OAuthResult

```
using qevercloud::EvernoteOAuthDialog::OAuthResult = EvernoteOAuthWebView::OAuthResult
```

7.35.3 Constructor & Destructor Documentation

7.35.3.1 EvernoteOAuthDialog()

```
qevercloud::EvernoteOAuthDialog::EvernoteOAuthDialog (
            QString consumerKey,
            QString consumerSecret,
            QString host = QStringLiteral("www.evernote.com"),
            QWidget * parent = Q_NULLPTR )
```

Constructs the dialog.

Parameters

host	Evernote host to authorize with. You need one of this: • "www.evernote.com" - the production service. It's the default value. • "sandox.evernote.com" - the developers "sandbox" service
consumerKey	get it from the Evernote
consumerSecret	along with this

7.35.3.2 ~EvernoteOAuthDialog()

```
virtual qevercloud::EvernoteOAuthDialog::~EvernoteOAuthDialog ( )  [override], [virtual]
```

7.35.4 Member Function Documentation

7.35.4.1 exec()

```
virtual int qevercloud::EvernoteOAuthDialog::exec ( )  [override], [virtual]
```

Returns

QDialog::Accepted on a successful authentication.

7.35.4.2 isSucceeded()

```
bool qevercloud::EvernoteOAuthDialog::isSucceeded ( ) const
```

Returns

true in case of a successful authentication. You probably better check exec() return value instead.

7.35.4.3 oauthError()

```
QString qevercloud::EvernoteOAuthDialog::oauthError ( ) const
```

Returns

In case of an authentification error may return some information about the error.

7.35.4.4 oauthResult()

```
OAuthResult qevercloud::EvernoteOAuthDialog::oauthResult ( ) const
```

Returns

the result of a successful authentication.

7.35.4.5 open()

```
virtual void qevercloud::EvernoteOAuthDialog::open ( )  [override], [virtual]
```

Shows the dialog as a window modal dialog, returning immediately.

7.35.4.6 setWebViewSizeHint()

```
void qevercloud::EvernoteOAuthDialog::setWebViewSizeHint (
            QSize sizeHint )
```

The dialog adjusts its initial size automatically based on the contained QWebView preffered size. Use this method to set the size.

Parameters

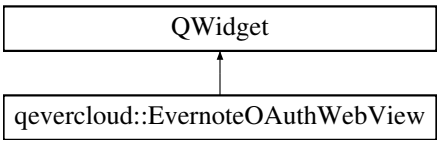
sizeHint	will be used as the preffered size of the contained QWebView.

7.36 qevercloud::EvernoteOAuthWebView Class Reference

The class is tailored specifically for OAuth authorization with Evernote.

`#include <OAuth.h>`

Inheritance diagram for qevercloud::EvernoteOAuthWebView:



Classes

- struct OAuthResult

Signals

- void authenticationFinished (bool success)
- void authenticationSuceeded ()
- void authenticationFailed ()

Public Member Functions

- EvernoteOAuthWebView (QWidget *parent=Q_NULLPTR)
- void authenticate (QString host, QString consumerKey, QString consumerSecret, const qint64 timeoutMsec=30000)
- bool isSucceeded () const
- QString oauthError () const
- OAuthResult oauthResult () const
- void setSizeHint (QSize sizeHint)
- QSize sizeHint () const override

7.36.1 Detailed Description

The class is tailored specifically for OAuth authorization with Evernote.

While it is functional by itself you probably will prefer to use EvernoteOAuthDialog.

Note that you have to include OAuth.h header.

By default EvernoteOAuthWebView uses qrand() for generating nonce so do not forget to call qsrand() in your application. See setNonceGenerator If you want more control over nonce generation.

7.36.2 Constructor & Destructor Documentation

7.36.2.1 EvernoteOAuthWebView()

```
qevercloud::EvernoteOAuthWebView::EvernoteOAuthWebView (
            QWidget * parent = Q_NULLPTR )
```

7.36.3 Member Function Documentation

7.36.3.1 authenticate()

```
void qevercloud::EvernoteOAuthWebView::authenticate (
            QString host,
            QString consumerKey,
            QString consumerSecret,
            const qint64 timeoutMsec = 30000 )
```

This function starts the OAuth sequence. In the end of the sequence will be emitted one of the signals↩ : authenticationSuceeded or authenticationFailed.

Do not call the function while its call is in effect, i.e. one of the signals is not emitted.

Parameters

host	Evernote host to authorize with. You need one of this: • "www.evernote.com" - the production service. It's the default value. • "sandox.evernote.com" - the developers "sandbox" service
consumerKey	get it from the Evernote
consumerSecret	along with this
timeoutMsec	Timeout for network requests in milliseconds

7.36.3.2 authenticationFailed

```
void qevercloud::EvernoteOAuthWebView::authenticationFailed ( )  [signal]
```

Emitted when the OAuth sequence is finished with a failure. Some error info may be available with errorText().

7.36.3.3 authenticationFinished

```
void qevercloud::EvernoteOAuthWebView::authenticationFinished (
            bool success )  [signal]
```

Emitted when the OAuth sequence started with authenticate() call is finished

7.36.3.4 authenticationSuceeded

```
void qevercloud::EvernoteOAuthWebView::authenticationSuceeded ( )  [signal]
```

Emitted when the OAuth sequence is successfully finished. Call oauthResult() to get the data.

7.36.3.5 isSucceeded()

```
bool qevercloud::EvernoteOAuthWebView::isSucceeded ( ) const
```

Returns

true if the last call to authenticate resulted in a successful authentication.

7.36.3.6 oauthError()

```
QString qevercloud::EvernoteOAuthWebView::oauthError ( ) const
```

Returns

error message resulted from the last call to authenticate

7.36.3.7 oauthResult()

```
OAuthResult qevercloud::EvernoteOAuthWebView::oauthResult ( ) const
```

Returns

the result of the last authentication, i.e. authenticate() call.

7.36.3.8 setSizeHint()

```
void qevercloud::EvernoteOAuthWebView::setSizeHint (
            QSize sizeHint )
```

The method is useful to specify default size for a EverOAuthWebView.

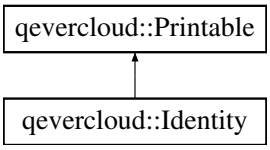
7.36.3.9 sizeHint()

```
QSize qevercloud::EvernoteOAuthWebView::sizeHint ( ) const  [override]
```

7.37 qevercloud::Identity Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::Identity:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const Identity &other) const
- bool operator!= (const Identity &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- IdentityID id = 0
- Optional< Contact > contact
- Optional< UserID > userId
- Optional< bool > deactivated
- Optional< bool > sameBusiness
- Optional< bool > blocked
- Optional< bool > userConnected
- Optional< MessageEventID > eventId

7.37.1 Detailed Description

An object that represents the relationship between a Contact that possibly belongs to an Evernote User.

7.37.2 Member Function Documentation

7.37.2.1 operator"!=()

```
bool qevercloud::Identity::operator!= (
            const Identity & other ) const  [inline]
```

7.37.2.2 operator==()

```
bool qevercloud::Identity::operator== (
            const Identity & other ) const  [inline]
```

7.37.2.3 print()

```
virtual void qevercloud::Identity::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.37.3 Member Data Documentation

7.37.3.1 blocked

```
Optional< bool > qevercloud::Identity::blocked
```

Has the caller blocked the Evernote user this Identity represents?

7.37.3.2 contact

```
Optional< Contact > qevercloud::Identity::contact
```

NOT DOCUMENTED

7.37.3.3 deactivated

```
Optional< bool > qevercloud::Identity::deactivated
```

Indicates that the contact for this identity is no longer active and should not be used when creating new threads using Destination.recipients, unless you know of another Identity instance with the same contact information that is active. If you are connected to the user (see userConnected), you can still create threads using their Evernote-type contact.

7.37.3.4 eventId

```
Optional< MessageEventID > qevercloud::Identity::eventId
```

A server-assigned sequence number for the events in the messages subsystem.

7.37.3.5 id

```
IdentityID qevercloud::Identity::id = 0
```

The unique identifier for this mapping.

7.37.3.6 localData

```
EverCloudLocalData qevercloud::Identity::localData
```

See the declaration of EverCloudLocalData for details

7.37.3.7 sameBusiness

```
Optional< bool > qevercloud::Identity::sameBusiness
```

Does this Identity belong to someone who is in the same business as the caller?

7.37.3.8 userConnected

```
Optional< bool > qevercloud::Identity::userConnected
```

Indicates that the caller is "connected" to the user of this identity via this identity. When you have a connection via an identity, you should always create new threads using the Evernote-type contact (see ContactType) using the userId field from a connected Identity. On the Evernote service, the Evernote-type contact is the most durable. Phone numbers and e-mail addresses can get re-assigned but your Evernote account user ID will remain the same. A connection exists when both of you are in the same business or the user has replied to a thread that you are on. When connected, you will also get to see more information about the user who has claimed the identity. Note that you are never connected to yourself since you won't be sending messages to yourself, but you will obviously see your own profile information.

7.37.3.9 userId

```
Optional< UserID > qevercloud::Identity::userId
```

The Evernote User id that is connected to the Contact. May be unset if this identity has not yet been claimed, or the caller is not connected to this identity.

7.38 qevercloud::IDurableService Class Reference

```
#include <DurableService.h>
```

Classes

- struct AsyncRequest
- struct SyncRequest

Public Types

- using SyncResult = std::pair< QVariant, EverCloudExceptionDataPtr >
- using SyncServiceCall = std::function< SyncResult(IRequestContextPtr)>
- using AsyncServiceCall = std::function< AsyncResult *(IRequestContextPtr)>

Public Member Functions

- virtual SyncResult executeSyncRequest (SyncRequest &&syncRequest, IRequestContextPtr ctx)=0
- virtual AsyncResult * executeAsyncRequest (AsyncRequest &&asyncRequest, IRequestContextPtr ctx)=0

7.38.1 Member Typedef Documentation

7.38.1.1 AsyncServiceCall

```
using qevercloud::IDurableService::AsyncServiceCall = std::function<AsyncResult*(IRequestContextPtr)>
```

7.38.1.2 SyncResult

```
using qevercloud::IDurableService::SyncResult = std::pair<QVariant,EverCloudExceptionDataPtr>
```

7.38.1.3 SyncServiceCall

```
using qevercloud::IDurableService::SyncServiceCall = std::function<SyncResult(IRequestContextPtr)>
```

7.38.2 Member Function Documentation

7.38.2.1 executeAsyncRequest()

```
virtual AsyncResult * qevercloud::IDurableService::executeAsyncRequest (
            AsyncRequest && asyncRequest,
            IRequestContextPtr ctx )  [pure virtual]
```

7.38.2.2 executeSyncRequest()

```
virtual SyncResult qevercloud::IDurableService::executeSyncRequest (
            SyncRequest && syncRequest,
            IRequestContextPtr ctx )  [pure virtual]
```

7.39 qevercloud::ILogger Class Reference

```
#include <Log.h>
```

Public Member Functions

- virtual bool shouldLog (const LogLevel level, const char *component) const =0
- virtual void log (const LogLevel level, const char *component, const char *fileName, const quint32 lineNumber, const qint64 timestamp, const QString &message)=0
- virtual void setLevel (const LogLevel level)=0
- virtual LogLevel level () const =0

7.39.1 Member Function Documentation

7.39.1.1 level()

```
virtual LogLevel qevercloud::ILogger::level ( ) const  [pure virtual]
```

7.39.1.2 log()

```
virtual void qevercloud::ILogger::log (
            const LogLevel level,
            const char * component,
            const char * fileName,
            const quint32 lineNumber,
            const qint64 timestamp,
            const QString & message )  [pure virtual]
```

7.39.1.3 setLevel()

```
virtual void qevercloud::ILogger::setLevel (
            const LogLevel level )  [pure virtual]
```

7.39.1.4 shouldLog()

```
virtual bool qevercloud::ILogger::shouldLog (
            const LogLevel level,
            const char * component ) const  [pure virtual]
```

7.40 qevercloud::InkNoteImageDownloader Class Reference

the InkNoteImageDownloader class is for downloading the images of ink notes which can be created with the official Evernote client on Windows (only with it, at least at the time of this writing).

```
#include <InkNoteImageDownloader.h>
```

Public Member Functions

- InkNoteImageDownloader ()

  Default constructor.
- InkNoteImageDownloader (QString host, QString shardId, QString authenticationToken, int width, int height)

  Constructs InkNoteImageDownloader.
- virtual ~InkNoteImageDownloader ()
- InkNoteImageDownloader & setHost (QString host)
- InkNoteImageDownloader & setShardId (QString shardId)
- InkNoteImageDownloader & setAuthenticationToken (QString authenticationToken)
- InkNoteImageDownloader & setWidth (int width)
- InkNoteImageDownloader & setHeight (int height)
- QByteArray download (Guid guid, const bool isPublic=false, const qint64 timeoutMsec=30000)

  Downloads the image for the ink note.

7.40.1 Detailed Description

the InkNoteImageDownloader class is for downloading the images of ink notes which can be created with the official Evernote client on Windows (only with it, at least at the time of this writing).

On all other platforms the most one can get instead of the actual ink note is its non-editable image. This class retrieves just these, exclusively in PNG format.

NOTE: almost the entirety of this class' content represents an ad-hoc solution to a completely undocumented feature of Evernote service. A very small glimpse of information was once available on Evernote forums but it's deleted now... I collected an even smaller glimpse of information in this SO question: https://stackoverflow.com/q/39179012/1217285. For all practical purposes it is the only piece of information on this feature in existence now.

7.40.2 Constructor & Destructor Documentation

7.40.2.1 InkNoteImageDownloader() [1/2]

```
qevercloud::InkNoteImageDownloader::InkNoteImageDownloader ( )
```

Default constructor.

host, shardId, authenticationToken, width, height have to be specified before calling download or createPostRequest

7.40.2.2 InkNoteImageDownloader() [2/2]

```
qevercloud::InkNoteImageDownloader::InkNoteImageDownloader (
            QString host,
            QString shardId,
            QString authenticationToken,
            int width,
            int height )
```

Constructs InkNoteImageDownloader.

Parameters

host	www.evernote.com or sandbox.evernote.com
shardId	You can get the value from UserStore service or as a result of an authentication.
authenticationToken	For working private ink notes you must supply a valid authentication token. For public resources the value specified is not used.
width	Width of the ink note's resource
height	Height of the ink note's resource

7.40.2.3 ~InkNoteImageDownloader()

```
virtual qevercloud::InkNoteImageDownloader::~InkNoteImageDownloader ( )  [virtual]
```

7.40.3 Member Function Documentation

7.40.3.1 download()

```
QByteArray qevercloud::InkNoteImageDownloader::download (
            Guid guid,
            const bool isPublic = false,
            const qint64 timeoutMsec = 30000 )
```

Downloads the image for the ink note.

Unlike other pieces of QEverCloud API, downloading of ink note images is currently synchronous only. The reason for that is that AsyncResult is bounded to a single QNetworkRequest object but downloading of the ink note image might take multiple requests for several ink note image's vertical stripes which are then merged together to form a single image. Downloading the entire ink note's image via a single request works sometimes but sometimes Evernote replies to such request with messed up data which cannot be loaded into a QImage. The reason for that behaviour is unknown at the moment, and, given the state of official documentation

- missing - it is likely to stay so. if someone has an idea how to make it more reliable, please let me know.

Parameters

guid	The guid of the ink note's resource
isPublic	Specify true for public ink notes. In this case authentication token is not sent to with the request as it should be according to the docs.
timeoutMsec	Timeout for download request in milliseconds

Returns

downloaded data.

7.40.3.2 setAuthenticationToken()

```
InkNoteImageDownloader & qevercloud::InkNoteImageDownloader::setAuthenticationToken (
            QString authenticationToken )
```

Parameters

authenticationToken	For working private ink notes you must supply a valid authentication token. For public resources the value specified is not used.

7.40.3.3 setHeight()

```
InkNoteImageDownloader & qevercloud::InkNoteImageDownloader::setHeight (
            int height )
```

Parameters

height	Height of the ink note's resource

7.40.3.4 setHost()

```
InkNoteImageDownloader & qevercloud::InkNoteImageDownloader::setHost (
            QString host )
```

Parameters

host	www.evernote.com or sandbox.evernote.com

7.40.3.5 setShardId()

```
InkNoteImageDownloader & qevercloud::InkNoteImageDownloader::setShardId (
            QString shardId )
```

Parameters

shardId	You can get the value from UserStore service or as a result of an authentication.

7.40.3.6 setWidth()

```
InkNoteImageDownloader & qevercloud::InkNoteImageDownloader::setWidth (
            int width )
```

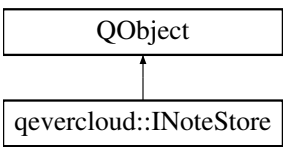
Parameters

width	Width of the ink note's resource

7.41 qevercloud::INoteStore Class Reference

```
#include <Services.h>
```

Inheritance diagram for qevercloud::INoteStore:



Public Member Functions

- virtual QString noteStoreUrl () const =0
- virtual void setNoteStoreUrl (QString url)=0
- virtual SyncState getSyncState (IRequestContextPtr ctx={})=0
- virtual AsyncResult * getSyncStateAsync (IRequestContextPtr ctx={})=0
- virtual SyncChunk getFilteredSyncChunk (qint32 afterUSN, qint32 maxEntries, const SyncChunkFilter &filter, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getFilteredSyncChunkAsync (qint32 afterUSN, qint32 maxEntries, const SyncChunkFilter &filter, IRequestContextPtr ctx={})=0
- virtual SyncState getLinkedNotebookSyncState (const LinkedNotebook &linkedNotebook, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getLinkedNotebookSyncStateAsync (const LinkedNotebook &linkedNotebook, IRequestContextPtr ctx={})=0
- virtual SyncChunk getLinkedNotebookSyncChunk (const LinkedNotebook &linkedNotebook, qint32 afterUSN, qint32 maxEntries, bool fullSyncOnly, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getLinkedNotebookSyncChunkAsync (const LinkedNotebook &linkedNotebook, qint32 afterUSN, qint32 maxEntries, bool fullSyncOnly, IRequestContextPtr ctx={})=0
- virtual QList< Notebook > listNotebooks (IRequestContextPtr ctx={})=0
- virtual AsyncResult * listNotebooksAsync (IRequestContextPtr ctx={})=0
- virtual QList< Notebook > listAccessibleBusinessNotebooks (IRequestContextPtr ctx={})=0
- virtual AsyncResult * listAccessibleBusinessNotebooksAsync (IRequestContextPtr ctx={})=0
- virtual Notebook getNotebook (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getNotebookAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual Notebook getDefaultNotebook (IRequestContextPtr ctx={})=0
- virtual AsyncResult * getDefaultNotebookAsync (IRequestContextPtr ctx={})=0
- virtual Notebook createNotebook (const Notebook &notebook, IRequestContextPtr ctx={})=0
- virtual AsyncResult * createNotebookAsync (const Notebook &notebook, IRequestContextPtr ctx={})=0
- virtual qint32 updateNotebook (const Notebook &notebook, IRequestContextPtr ctx={})=0
- virtual AsyncResult * updateNotebookAsync (const Notebook &notebook, IRequestContextPtr ctx={})=0
- virtual qint32 expungeNotebook (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * expungeNotebookAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual QList< Tag > listTags (IRequestContextPtr ctx={})=0
- virtual AsyncResult * listTagsAsync (IRequestContextPtr ctx={})=0
- virtual QList< Tag > listTagsByNotebook (Guid notebookGuid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * listTagsByNotebookAsync (Guid notebookGuid, IRequestContextPtr ctx={})=0
- virtual Tag getTag (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getTagAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual Tag createTag (const Tag &tag, IRequestContextPtr ctx={})=0
- virtual AsyncResult * createTagAsync (const Tag &tag, IRequestContextPtr ctx={})=0
- virtual qint32 updateTag (const Tag &tag, IRequestContextPtr ctx={})=0
- virtual AsyncResult * updateTagAsync (const Tag &tag, IRequestContextPtr ctx={})=0
- virtual void untagAll (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * untagAllAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual qint32 expungeTag (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * expungeTagAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual QList< SavedSearch > listSearches (IRequestContextPtr ctx={})=0
- virtual AsyncResult * listSearchesAsync (IRequestContextPtr ctx={})=0
- virtual SavedSearch getSearch (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getSearchAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual SavedSearch createSearch (const SavedSearch &search, IRequestContextPtr ctx={})=0
- virtual AsyncResult * createSearchAsync (const SavedSearch &search, IRequestContextPtr ctx={})=0
- virtual qint32 updateSearch (const SavedSearch &search, IRequestContextPtr ctx={})=0
- virtual AsyncResult * updateSearchAsync (const SavedSearch &search, IRequestContextPtr ctx={})=0
- virtual qint32 expungeSearch (Guid guid, IRequestContextPtr ctx={})=0

- virtual AsyncResult * expungeSearchAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual qint32 findNoteOffset (const NoteFilter &filter, Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * findNoteOffsetAsync (const NoteFilter &filter, Guid guid, IRequestContextPtr ctx={})=0
- virtual NotesMetadataList findNotesMetadata (const NoteFilter &filter, qint32 offset, qint32 maxNotes, const NotesMetadataResultSpec &resultSpec, IRequestContextPtr ctx={})=0
- virtual AsyncResult * findNotesMetadataAsync (const NoteFilter &filter, qint32 offset, qint32 maxNotes, const NotesMetadataResultSpec &resultSpec, IRequestContextPtr ctx={})=0
- virtual NoteCollectionCounts findNoteCounts (const NoteFilter &filter, bool withTrash, IRequestContextPtr ctx={})=0
- virtual AsyncResult * findNoteCountsAsync (const NoteFilter &filter, bool withTrash, IRequestContextPtr ctx={})=0
- virtual Note getNoteWithResultSpec (Guid guid, const NoteResultSpec &resultSpec, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getNoteWithResultSpecAsync (Guid guid, const NoteResultSpec &resultSpec, IRequestContextPtr ctx={})=0
- virtual Note getNote (Guid guid, bool withContent, bool withResourcesData, bool withResourcesRecognition, bool withResourcesAlternateData, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getNoteAsync (Guid guid, bool withContent, bool withResourcesData, bool withResourcesRecognition, bool withResourcesAlternateData, IRequestContextPtr ctx={})=0
- virtual LazyMap getNoteApplicationData (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getNoteApplicationDataAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual QString getNoteApplicationDataEntry (Guid guid, QString key, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getNoteApplicationDataEntryAsync (Guid guid, QString key, IRequestContextPtr ctx={})=0
- virtual qint32 setNoteApplicationDataEntry (Guid guid, QString key, QString value, IRequestContextPtr ctx={})=0
- virtual AsyncResult * setNoteApplicationDataEntryAsync (Guid guid, QString key, QString value, IRequestContextPtr ctx={})=0
- virtual qint32 unsetNoteApplicationDataEntry (Guid guid, QString key, IRequestContextPtr ctx={})=0
- virtual AsyncResult * unsetNoteApplicationDataEntryAsync (Guid guid, QString key, IRequestContextPtr ctx={})=0
- virtual QString getNoteContent (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getNoteContentAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual QString getNoteSearchText (Guid guid, bool noteOnly, bool tokenizeForIndexing, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getNoteSearchTextAsync (Guid guid, bool noteOnly, bool tokenizeForIndexing, IRequestContextPtr ctx={})=0
- virtual QString getResourceSearchText (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getResourceSearchTextAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual QStringList getNoteTagNames (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getNoteTagNamesAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual Note createNote (const Note &note, IRequestContextPtr ctx={})=0
- virtual AsyncResult * createNoteAsync (const Note &note, IRequestContextPtr ctx={})=0
- virtual Note updateNote (const Note &note, IRequestContextPtr ctx={})=0
- virtual AsyncResult * updateNoteAsync (const Note &note, IRequestContextPtr ctx={})=0
- virtual qint32 deleteNote (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * deleteNoteAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual qint32 expungeNote (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * expungeNoteAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual Note copyNote (Guid noteGuid, Guid toNotebookGuid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * copyNoteAsync (Guid noteGuid, Guid toNotebookGuid, IRequestContextPtr ctx={})=0
- virtual QList< NoteVersionId > listNoteVersions (Guid noteGuid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * listNoteVersionsAsync (Guid noteGuid, IRequestContextPtr ctx={})=0
- virtual Note getNoteVersion (Guid noteGuid, qint32 updateSequenceNum, bool withResourcesData, bool withResourcesRecognition, bool withResourcesAlternateData, IRequestContextPtr ctx={})=0

- virtual AsyncResult * getNoteVersionAsync (Guid noteGuid, qint32 updateSequenceNum, bool withResourcesData, bool withResourcesRecognition, bool withResourcesAlternateData, IRequestContextPtr ctx={})=0
- virtual Resource getResource (Guid guid, bool withData, bool withRecognition, bool withAttributes, bool withAlternateData, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getResourceAsync (Guid guid, bool withData, bool withRecognition, bool withAttributes, bool withAlternateData, IRequestContextPtr ctx={})=0
- virtual LazyMap getResourceApplicationData (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getResourceApplicationDataAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual QString getResourceApplicationDataEntry (Guid guid, QString key, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getResourceApplicationDataEntryAsync (Guid guid, QString key, IRequestContextPtr ctx={})=0
- virtual qint32 setResourceApplicationDataEntry (Guid guid, QString key, QString value, IRequestContextPtr ctx={})=0
- virtual AsyncResult * setResourceApplicationDataEntryAsync (Guid guid, QString key, QString value, IRequestContextPtr ctx={})=0
- virtual qint32 unsetResourceApplicationDataEntry (Guid guid, QString key, IRequestContextPtr ctx={})=0
- virtual AsyncResult * unsetResourceApplicationDataEntryAsync (Guid guid, QString key, IRequestContextPtr ctx={})=0
- virtual qint32 updateResource (const Resource &resource, IRequestContextPtr ctx={})=0
- virtual AsyncResult * updateResourceAsync (const Resource &resource, IRequestContextPtr ctx={})=0
- virtual QByteArray getResourceData (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getResourceDataAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual Resource getResourceByHash (Guid noteGuid, QByteArray contentHash, bool withData, bool withRecognition, bool withAlternateData, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getResourceByHashAsync (Guid noteGuid, QByteArray contentHash, bool withData, bool withRecognition, bool withAlternateData, IRequestContextPtr ctx={})=0
- virtual QByteArray getResourceRecognition (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getResourceRecognitionAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual QByteArray getResourceAlternateData (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getResourceAlternateDataAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual ResourceAttributes getResourceAttributes (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getResourceAttributesAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual Notebook getPublicNotebook (UserID userId, QString publicUri, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getPublicNotebookAsync (UserID userId, QString publicUri, IRequestContextPtr ctx={})=0
- virtual SharedNotebook shareNotebook (const SharedNotebook &sharedNotebook, QString message, IRequestContextPtr ctx={})=0
- virtual AsyncResult * shareNotebookAsync (const SharedNotebook &sharedNotebook, QString message, IRequestContextPtr ctx={})=0
- virtual CreateOrUpdateNotebookSharesResult createOrUpdateNotebookShares (const NotebookShareTemplate &shareTemplate, IRequestContextPtr ctx={})=0
- virtual AsyncResult * createOrUpdateNotebookSharesAsync (const NotebookShareTemplate &shareTemplate, IRequestContextPtr ctx={})=0
- virtual qint32 updateSharedNotebook (const SharedNotebook &sharedNotebook, IRequestContextPtr ctx={})=0
- virtual AsyncResult * updateSharedNotebookAsync (const SharedNotebook &sharedNotebook, IRequestContextPtr ctx={})=0
- virtual Notebook setNotebookRecipientSettings (QString notebookGuid, const NotebookRecipientSettings &recipientSettings, IRequestContextPtr ctx={})=0
- virtual AsyncResult * setNotebookRecipientSettingsAsync (QString notebookGuid, const NotebookRecipientSettings &recipientSettings, IRequestContextPtr ctx={})=0
- virtual QList< SharedNotebook > listSharedNotebooks (IRequestContextPtr ctx={})=0
- virtual AsyncResult * listSharedNotebooksAsync (IRequestContextPtr ctx={})=0
- virtual LinkedNotebook createLinkedNotebook (const LinkedNotebook &linkedNotebook, IRequestContextPtr ctx={})=0

- virtual AsyncResult * createLinkedNotebookAsync (const LinkedNotebook &linkedNotebook, IRequestContextPtr ctx={})=0
- virtual qint32 updateLinkedNotebook (const LinkedNotebook &linkedNotebook, IRequestContextPtr ctx={})=0
- virtual AsyncResult * updateLinkedNotebookAsync (const LinkedNotebook &linkedNotebook, IRequestContextPtr ctx={})=0
- virtual QList< LinkedNotebook > listLinkedNotebooks (IRequestContextPtr ctx={})=0
- virtual AsyncResult * listLinkedNotebooksAsync (IRequestContextPtr ctx={})=0
- virtual qint32 expungeLinkedNotebook (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * expungeLinkedNotebookAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual AuthenticationResult authenticateToSharedNotebook (QString shareKeyOrGlobalId, IRequestContextPtr ctx={})=0
- virtual AsyncResult * authenticateToSharedNotebookAsync (QString shareKeyOrGlobalId, IRequestContextPtr ctx={})=0
- virtual SharedNotebook getSharedNotebookByAuth (IRequestContextPtr ctx={})=0
- virtual AsyncResult * getSharedNotebookByAuthAsync (IRequestContextPtr ctx={})=0
- virtual void emailNote (const NoteEmailParameters &parameters, IRequestContextPtr ctx={})=0
- virtual AsyncResult * emailNoteAsync (const NoteEmailParameters &parameters, IRequestContextPtr ctx={})=0
- virtual QString shareNote (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * shareNoteAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual void stopSharingNote (Guid guid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * stopSharingNoteAsync (Guid guid, IRequestContextPtr ctx={})=0
- virtual AuthenticationResult authenticateToSharedNote (QString guid, QString noteKey, IRequestContextPtr ctx={})=0
- virtual AsyncResult * authenticateToSharedNoteAsync (QString guid, QString noteKey, IRequestContextPtr ctx={})=0
- virtual RelatedResult findRelated (const RelatedQuery &query, const RelatedResultSpec &resultSpec, IRequestContextPtr ctx={})=0
- virtual AsyncResult * findRelatedAsync (const RelatedQuery &query, const RelatedResultSpec &resultSpec, IRequestContextPtr ctx={})=0
- virtual UpdateNoteIfUsnMatchesResult updateNoteIfUsnMatches (const Note &note, IRequestContextPtr ctx={})=0
- virtual AsyncResult * updateNoteIfUsnMatchesAsync (const Note &note, IRequestContextPtr ctx={})=0
- virtual ManageNotebookSharesResult manageNotebookShares (const ManageNotebookSharesParameters &parameters, IRequestContextPtr ctx={})=0
- virtual AsyncResult * manageNotebookSharesAsync (const ManageNotebookSharesParameters &parameters, IRequestContextPtr ctx={})=0
- virtual ShareRelationships getNotebookShares (QString notebookGuid, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getNotebookSharesAsync (QString notebookGuid, IRequestContextPtr ctx={})=0

Protected Member Functions

- INoteStore (QObject *parent)

7.41.1 Detailed Description

Service: NoteStore

The NoteStore service is used by EDAM clients to exchange information about the collection of notes in an account. This is primarily used for synchronization, but could also be used by a "thin" client without a full local cache.

Most functions take an "authenticationToken" parameter, which is the value returned by the UserStore which permits access to the account.

Calls which require an authenticationToken may throw an EDAMUserException for the following reasons:

- DATA_REQUIRED "authenticationToken" - token is empty
- BAD_DATA_FORMAT "authenticationToken" - token is malformed
- INVALID_AUTH "authenticationToken" - token signature is invalid
- AUTH_EXPIRED "authenticationToken" - token has expired or been revoked
- PERMISSION_DENIED "authenticationToken" - token does not grant permission to perform the requested action
- BUSINESS_SECURITY_LOGIN_REQUIRED "sso" - the user is a member of a business that requires single sign-on, and must complete SSO before accessing business content.

7.41.2 Constructor & Destructor Documentation

7.41.2.1 INoteStore()

```
qevercloud::INoteStore::INoteStore (
            QObject * parent )  [inline], [protected]
```

7.41.3 Member Function Documentation

7.41.3.1 authenticateToSharedNote()

```
virtual AuthenticationResult qevercloud::INoteStore::authenticateToSharedNote (
            QString guid,
            QString noteKey,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asks the service to produce an authentication token that can be used to access the contents of a single Note which was individually shared from someone's account. This authenticationToken can be used with the various other NoteStore calls to find and retrieve the Note and its directly-referenced children.

Parameters

guid	The GUID identifying this Note on this shard.
noteKey	The 'noteKey' identifier from the Note that was originally created via a call to shareNote() and then given to a recipient to access.
authenticationToken	An optional authenticationToken that identifies the user accessing the shared note. This parameter may be required to access some shared notes.

Exceptions

EDAMUserException	• PERMISSION_DENIED "Note" - the Note with that GUID is either not shared, or the noteKey doesn't match the current key for this note • PERMISSION_DENIED "authenticationToken" - an authentication token is required to access this Note, but either no authentication token or a "non-owner" authentication token was provided.

Exceptions

EDAMNotFoundException	• "guid" - the note with that GUID is not found
EDAMSystemException	• TAKEN_DOWN "Note" - The specified shared note is taken down (for all requesters). • TAKEN_DOWN "Country" - The specified shared note is taken down for the requester because of an IP-based country lookup.

7.41.3.2 authenticateToSharedNoteAsync()

```
virtual AsyncResult * qevercloud::INoteStore::authenticateToSharedNoteAsync (
            QString guid,
            QString noteKey,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of authenticateToSharedNote

7.41.3.3 authenticateToSharedNotebook()

```
virtual AuthenticationResult qevercloud::INoteStore::authenticateToSharedNotebook (
            QString shareKeyOrGlobalId,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asks the service to produce an authentication token that can be used to access the contents of a shared notebook from someone else's account. This authenticationToken can be used with the various other NoteStore calls to find and retrieve notes, and if the permissions in the shared notebook are sufficient, to make changes to the contents of the notebook.

Parameters

shareKeyOr↩ GlobalId	May be one of the following: • A share key for a shared notebook that was granted to some recipient Must be used if you are joining a notebook unless it was shared via createOrUpdateNotebookShares. Share keys are delivered out-of-band and are generally not available to clients. For security reasons, share keys may be invalidated at the discretion of the service. • The shared notebook global identifier. May be used to access a notebook that is already joined. • The Notebook GUID. May be used to access a notebook that was already joined, or to access a notebook that was shared with the recipient via createOrUpdateNotebookShares.
authenticationToken	If a non-empty string is provided, this is the full user-based authentication token that identifies the user who is currently logged in and trying to access the shared notebook. If this string is empty, the service will attempt to authenticate to the shared notebook without any logged in user.

Exceptions

EDAMSystemException	• BAD_DATA_FORMAT "shareKey" - invalid shareKey string • INVALID_AUTH "shareKey" - bad signature on shareKey string
EDAMNotFoundException	• "SharedNotebook.id" - the shared notebook no longer exists
EDAMUserException	• DATA_REQUIRED "authenticationToken" - the share requires login, and no valid authentication token was provided. • PERMISSION_DENIED "SharedNotebook.username" - share requires login, and another username has already been bound to this notebook.

7.41.3.4 authenticateToSharedNotebookAsync()

```
virtual AsyncResult * qevercloud::INoteStore::authenticateToSharedNotebookAsync (
            QString shareKeyOrGlobalId,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of authenticateToSharedNotebook

7.41.3.5 copyNote()

```
virtual Note qevercloud::INoteStore::copyNote (
            Guid noteGuid,
            Guid toNotebookGuid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Performs a deep copy of the Note with the provided GUID 'noteGuid' into the Notebook with the provided GUID 'toNotebookGuid'. The caller must be the owner of both the Note and the Notebook. This creates a new Note in the destination Notebook with new content and Resources that match all of the content and Resources from the original Note, but with new GUID identifiers. The original Note is not modified by this operation. The copied note is considered as an "upload" for the purpose of upload transfer limit calculation, so its size is added to the upload count for the owner.

If the original note has been shared and has SharedNote records, the shares are NOT copied.

Parameters

noteGuid	The GUID of the Note to copy.
toNotebookGuid	The GUID of the Notebook that should receive the new Note.

Returns

The metadata for the new Note that was created. This will include the new GUID for this Note (and any copied Resources), but will not include the content body or the binary bodies of any Resources.

Exceptions

EDAMUserException	• LIMIT_REACHED "Note" - at max number per account • PERMISSION_DENIED "Notebook.guid" - destination not owned by user • PERMISSION_DENIED "Note" - user doesn't own • QUOTA_REACHED "Accounting.uploadLimit" - note exceeds upload quota
EDAMNotFoundException	• "Notebook.guid" - not found, by GUID

7.41.3.6 copyNoteAsync()

```
virtual AsyncResult * qevercloud::INoteStore::copyNoteAsync (
            Guid noteGuid,
            Guid toNotebookGuid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of copyNote

7.41.3.7 createLinkedNotebook()

```
virtual LinkedNotebook qevercloud::INoteStore::createLinkedNotebook (
            const LinkedNotebook & linkedNotebook,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asks the service to make a linked notebook with the provided name, username of the owner and identifiers provided. A linked notebook can be either a link to a public notebook or to a private shared notebook.

Parameters

linkedNotebook	The desired fields for the linked notebook must be provided on this object. The name of the linked notebook must be set. Either a username uri or a shard id and share key must be provided otherwise a EDAMUserException is thrown.

Returns

The newly created LinkedNotebook. The server-side id will be saved in this object's 'id' field.

Exceptions

EDAMUserException	• DATA_REQUIRED "LinkedNotebook.shareName" - missing shareName • BAD_DATA_FORMAT "LinkedNotebook.name" - invalid shareName length or pattern • BAD_DATA_FORMAT "LinkedNotebook.username" - bad username format • BAD_DATA_FORMAT "LinkedNotebook.uri" - if public notebook set but bad uri • DATA_REQUIRED "LinkedNotebook.shardId" - if private notebook but shard id not provided • BAD_DATA_FORMAT "LinkedNotebook.stack" - invalid stack name length or pattern
EDAMSystemException	• BAD_DATA_FORMAT "LinkedNotebook.sharedNotebookGlobalId" - if a bad global identifer was set on a private notebook

7.41.3.8 createLinkedNotebookAsync()

```
virtual AsyncResult * qevercloud::INoteStore::createLinkedNotebookAsync (
            const LinkedNotebook & linkedNotebook,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of createLinkedNotebook

7.41.3.9 createNote()

```
virtual Note qevercloud::INoteStore::createNote (
            const Note & note,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asks the service to make a note with the provided set of information.

Parameters

note	A Note object containing the desired fields to be populated on the service.

Returns

The newly created Note from the service. The server-side GUIDs for the Note and any Resources will be saved in this object. The service will include the meta-data for each resource in the note, but the binary contents of the resources and their recognition data will be omitted (except Recognition Resource body, for which the behavior is unspecified).

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Note.title" - invalid length or pattern • BAD_DATA_FORMAT "Note.content" - invalid length for ENML content • BAD_DATA_FORMAT "Resource.mime" - invalid resource MIME type • BAD_DATA_FORMAT "NoteAttributes.∗" - bad resource string • BAD_DATA_FORMAT "ResourceAttributes.∗" - bad resource string • DATA_CONFLICT "Note.deleted" - deleted time set on active note • DATA_REQUIRED "Resource.data" - resource data body missing • ENML_VALIDATION "∗" - note content doesn't validate against DTD • LIMIT_REACHED "Note" - at max number per account • LIMIT_REACHED "Note.size" - total note size too large • LIMIT_REACHED "Note.resources" - too many resources on Note • LIMIT_REACHED "Note.tagGuids" - too many Tags on Note • LIMIT_REACHED "Resource.data.size" - resource too large • LIMIT_REACHED "NoteAttribute.∗" - attribute string too long • LIMIT_REACHED "ResourceAttribute.∗" - attribute string too long • PERMISSION_DENIED "Note.notebookGuid" - NB not owned by user • QUOTA_REACHED "Accounting.uploadLimit" - note exceeds upload quota • BAD_DATA_FORMAT "Tag.name" - Note.tagNames was provided, and one of the specified tags had an invalid length or pattern • LIMIT_REACHED "Tag" - Note.tagNames was provided, and the required new tags would exceed the maximum number per account
EDAMNotFoundException	• "Note.notebookGuid" - not found, by GUID

7.41.3.10 createNoteAsync()

```
virtual AsyncResult * qevercloud::INoteStore::createNoteAsync (
            const Note & note,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of createNote

7.41.3.11 createNotebook()

```
virtual Notebook qevercloud::INoteStore::createNotebook (
            const Notebook & notebook,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asks the service to make a notebook with the provided name.

Parameters

notebook	The desired fields for the notebook must be provided on this object. The name of the notebook must be set, and either the 'active' or 'defaultNotebook' fields may be set by the client at creation. If a notebook exists in the account with the same name (via case-insensitive compare), this will throw an EDAMUserException.

Returns

The newly created Notebook. The server-side GUID will be saved in this object's 'guid' field.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Notebook.name" - invalid length or pattern • BAD_DATA_FORMAT "Notebook.stack" - invalid length or pattern • BAD_DATA_FORMAT "Publishing.uri" - if publishing set but bad uri • BAD_DATA_FORMAT "Publishing.publicDescription" - if too long • DATA_CONFLICT "Notebook.name" - name already in use • DATA_CONFLICT "Publishing.uri" - if URI already in use • DATA_REQUIRED "Publishing.uri" - if publishing set but uri missing • DATA_REQUIRED "Notebook" - notebook parameter was null • PERMISSION_DENIED "Notebook.defaultNotebook" - if the 'defaultNotebook' field is set to 'true' for a Notebook that is not owned by the user identified by the passed authenticationToken. • LIMIT_REACHED "Notebook" - at max number of notebooks
EDAMNotFoundException	• "Workspace.guid" - if workspaceGuid set and no Workspace exists for the GUID

7.41.3.12 createNotebookAsync()

```
virtual AsyncResult * qevercloud::INoteStore::createNotebookAsync (
            const Notebook & notebook,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of createNotebook

7.41.3.13 createOrUpdateNotebookShares()

```
virtual CreateOrUpdateNotebookSharesResult qevercloud::INoteStore::createOrUpdateNotebook↩
Shares (
```

```
const NotebookShareTemplate & shareTemplate,
IRequestContextPtr ctx = {} )  [pure virtual]
```

Share a notebook by a messaging thread ID or a list of contacts. This function is intended to be used in conjunction with Evernote messaging, and as such does not notify the recipient that a notebook has been shared with them.

Sharing with a subset of participants on a thread is accomplished by specifying both a thread ID and a list of contacts. This ensures that even if those contacts are on the thread under a deactivated identity, the correct user (the one who has the given contact on the thread) receives the share.

Parameters

authenticationToken	An authentication token that grants the caller permission to share the notebook. This should be an owner token if the notebook is owned by the caller. If the notebook is a business notebook to which the caller has full access, this should be their business authentication token. If the notebook is a shared (non-business) notebook to which the caller has full access, this should be the shared notebook authentication token returned by NoteStore.authenticateToNotebook.
shareTemplate	Specifies the GUID of the notebook to be shared, the privilege at which the notebook should be shared, and the recipient information.

Returns

A structure containing the USN of the Notebook after the change and a list of created or updated SharedNotebooks.

Exceptions

EDAMUserException	• DATA_REQUIRED "Notebook.guid" - if no notebook GUID was specified • BAD_DATA_FORMAT "Notebook.guid" - if shareTemplate.notebookGuid is not a valid GUID • DATA_REQUIRED "shareTemplate" - if the shareTemplate parameter was missing • DATA_REQUIRED "NotebookShareTemplate.privilege" - if no privilege was specified • DATA_CONFLICT "NotebookShareTemplate.privilege" - if the specified privilege is not allowed. • DATA_REQUIRED "NotebookShareTemplate.recipients" - if no recipients were specified, either by thread ID or as a list of contacts • LIMIT_REACHED "SharedNotebook" - if the notebook has reached its maximum number of shares
EDAMInvalidContactsException	• "NotebookShareTemplate.recipients" - if one or more of the recipients specified in shareTemplate.recipients was not syntactically valid, or if attempting to share a notebook with an Evernote identity that the sharer does not have a connection to. The exception will specify which recipients were invalid.

Exceptions

EDAMNotFoundException	• "Notebook.guid" - if no notebook with the specified GUID was found • "NotebookShareTemplate.recipientThreadId" - if the recipient thread ID was specified, but no thread with that ID exists

7.41.3.14 createOrUpdateNotebookSharesAsync()

```
virtual AsyncResult * qevercloud::INoteStore::createOrUpdateNotebookSharesAsync (
            const NotebookShareTemplate & shareTemplate,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of createOrUpdateNotebookShares

7.41.3.15 createSearch()

```
virtual SavedSearch qevercloud::INoteStore::createSearch (
            const SavedSearch & search,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asks the service to make a saved search with a set of information.

Parameters

search	The desired list of fields for the search are specified in this object. The caller must specify the name and query for the search, and may optionally specify a search scope. The SavedSearch.format field is ignored by the service.

Returns

The newly created SavedSearch. The server-side GUID will be saved in this object.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "SavedSearch.name" - invalid length or pattern • BAD_DATA_FORMAT "SavedSearch.query" - invalid length • DATA_CONFLICT "SavedSearch.name" - name already in use • LIMIT_REACHED "SavedSearch" - at max number of searches

7.41.3.16 createSearchAsync()

```
virtual AsyncResult * qevercloud::INoteStore::createSearchAsync (
```

```
            const SavedSearch & search,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of createSearch

7.41.3.17 createTag()

```
virtual Tag qevercloud::INoteStore::createTag (
            const Tag & tag,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asks the service to make a tag with a set of information.

Parameters

tag	The desired list of fields for the tag are specified in this object. The caller must specify the tag name, and may provide the parentGUID.

Returns

The newly created Tag. The server-side GUID will be saved in this object.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Tag.name" - invalid length or pattern • BAD_DATA_FORMAT "Tag.parentGuid" - malformed GUID • DATA_CONFLICT "Tag.name" - name already in use • LIMIT_REACHED "Tag" - at max number of tags
EDAMNotFoundException	• "Tag.parentGuid" - not found, by GUID

7.41.3.18 createTagAsync()

```
virtual AsyncResult * qevercloud::INoteStore::createTagAsync (
            const Tag & tag,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of createTag

7.41.3.19 deleteNote()

```
virtual qint32 qevercloud::INoteStore::deleteNote (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Moves the note into the trash. The note may still be undeleted, unless it is expunged. This is equivalent to calling updateNote() after setting Note.active = false

Parameters

guid	The GUID of the note to delete.

Returns

The Update Sequence Number for this change within the account.

Exceptions

EDAMUserException	• PERMISSION_DENIED "Note" - user doesn't have permission to update the note.
EDAMUserException	• DATA_CONFLICT "Note.guid" - the note is already deleted
EDAMNotFoundException	• "Note.guid" - not found, by GUID

7.41.3.20 deleteNoteAsync()

```
virtual AsyncResult * qevercloud::INoteStore::deleteNoteAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of deleteNote

7.41.3.21 emailNote()

```
virtual void qevercloud::INoteStore::emailNote (
            const NoteEmailParameters & parameters,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Attempts to send a single note to one or more email recipients.

NOTE: This function is generally not available to third party applications. Calls will result in an EDAMUserException with the error code PERMISSION_DENIED.

Parameters

authenticationToken	The note will be sent as the user logged in via this token, using that user's registered email address. If the authenticated user doesn't have permission to read that note, the emailing will fail.
parameters	The note must be specified either by GUID (in which case it will be sent using the existing data in the service), or else the full Note must be passed to this call. This also specifies the additional email fields that will be used in the email.

Exceptions

EDAMUserException	• LIMIT_REACHED "NoteEmailParameters.toAddresses" - The email can't be sent because this would exceed the user's daily email limit. • BAD_DATA_FORMAT "(email address)" - email address malformed • DATA_REQUIRED "NoteEmailParameters.toAddresses" - if there are no To: or Cc: addresses provided. • DATA_REQUIRED "Note.title" - if the caller provides a Note parameter with no title • DATA_REQUIRED "Note.content" - if the caller provides a Note parameter with no content • ENML_VALIDATION "∗" - note content doesn't validate against DTD • DATA_REQUIRED "NoteEmailParameters.note" - if no guid or note provided • PERMISSION_DENIED "Note" - private note, user doesn't own
EDAMNotFoundException	• "Note.guid" - not found, by GUID

7.41.3.22 emailNoteAsync()

```
virtual AsyncResult * qevercloud::INoteStore::emailNoteAsync (
            const NoteEmailParameters & parameters,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of emailNote

7.41.3.23 expungeLinkedNotebook()

```
virtual qint32 qevercloud::INoteStore::expungeLinkedNotebook (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Permanently expunges the linked notebook from the account.

NOTE: This function is generally not available to third party applications. Calls will result in an EDAMUserException with the error code PERMISSION_DENIED.

Parameters

guid	The LinkedNotebook.guid field of the LinkedNotebook to permanently remove from the account.

7.41.3.24 expungeLinkedNotebookAsync()

```
virtual AsyncResult * qevercloud::INoteStore::expungeLinkedNotebookAsync (
```

```
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of expungeLinkedNotebook

7.41.3.25 expungeNote()

```
virtual qint32 qevercloud::INoteStore::expungeNote (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Permanently removes a Note, and all of its Resources, from the service.

NOTE: This function is not available to third party applications. Calls will result in an EDAMUserException with the error code PERMISSION_DENIED.

Parameters

guid	The GUID of the note to delete.

Returns

The Update Sequence Number for this change within the account.

Exceptions

EDAMUserException	• PERMISSION_DENIED "Note" - user doesn't own
EDAMNotFoundException	• "Note.guid" - not found, by GUID

7.41.3.26 expungeNoteAsync()

```
virtual AsyncResult * qevercloud::INoteStore::expungeNoteAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of expungeNote

7.41.3.27 expungeNotebook()

```
virtual qint32 qevercloud::INoteStore::expungeNotebook (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Permanently removes the notebook from the user's account. After this action, the notebook is no longer available for undeletion, etc. If the notebook contains any Notes, they will be moved to the current default notebook and moved into the trash (i.e. Note.active=false).

NOTE: This function is generally not available to third party applications. Calls will result in an EDAMUserException with the error code PERMISSION_DENIED.

Parameters

guid	The GUID of the notebook to delete.

Returns

The Update Sequence Number for this change within the account.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Notebook.guid" - if the parameter is missing • LIMIT_REACHED "Notebook" - trying to expunge the last Notebook • PERMISSION_DENIED "Notebook" - private notebook, user doesn't own

7.41.3.28 expungeNotebookAsync()

```
virtual AsyncResult * qevercloud::INoteStore::expungeNotebookAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of expungeNotebook

7.41.3.29 expungeSearch()

```
virtual qint32 qevercloud::INoteStore::expungeSearch (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Permanently deletes the saved search with the provided GUID, if present.

NOTE: This function is generally not available to third party applications. Calls will result in an EDAMUserException with the error code PERMISSION_DENIED.

Parameters

guid	The GUID of the search to delete.

Returns

The Update Sequence Number for this change within the account.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "SavedSearch.guid" - if the guid parameter is empty • PERMISSION_DENIED "SavedSearch" - user doesn't own
EDAMNotFoundException	• "SavedSearch.guid" - not found, by GUID

7.41.3.30 expungeSearchAsync()

```
virtual AsyncResult * qevercloud::INoteStore::expungeSearchAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of expungeSearch

7.41.3.31 expungeTag()

```
virtual qint32 qevercloud::INoteStore::expungeTag (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Permanently deletes the tag with the provided GUID, if present.

NOTE: This function is not generally available to third party applications. Calls will result in an EDAMUserException with the error code PERMISSION_DENIED.

Parameters

guid	The GUID of the tag to delete.

Returns

The Update Sequence Number for this change within the account.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Tag.guid" - if the guid parameter is missing • PERMISSION_DENIED "Tag" - user doesn't own tag
EDAMNotFoundException	• "Tag.guid" - tag not found, by GUID

7.41.3.32 expungeTagAsync()

```
virtual AsyncResult * qevercloud::INoteStore::expungeTagAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of expungeTag

7.41.3.33 findNoteCounts()

```
virtual NoteCollectionCounts qevercloud::INoteStore::findNoteCounts (
            const NoteFilter & filter,
            bool withTrash,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

This function is used to determine how many notes are found for each notebook and tag in the user's account, given a current set of filter parameters that determine the current selection. This function will return a structure that gives the note count for each notebook and tag that has at least one note under the requested filter. Any notebook or tag that has zero notes in the filtered set will not be listed in the reply to this function (so they can be assumed to be 0).

Parameters

authenticationToken	Must be a valid token for the user's account unless the NoteFilter 'notebookGuid' is the GUID of a public notebook.
filter	The note selection filter that is currently being applied. The note counts are to be calculated with this filter applied to the total set of notes in the user's account.
withTrash	If true, then the NoteCollectionCounts.trashCount will be calculated and supplied in the reply. Otherwise, the trash value will be omitted.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "NoteFilter.notebookGuid" - if malformed • BAD_DATA_FORMAT "NoteFilter.notebookGuids" - if any are malformed • BAD_DATA_FORMAT "NoteFilter.words" - if search string too long
EDAMNotFoundException	• "Notebook.guid" - not found, by GUID

7.41.3.34 findNoteCountsAsync()

```
virtual AsyncResult * qevercloud::INoteStore::findNoteCountsAsync (
            const NoteFilter & filter,
            bool withTrash,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of findNoteCounts

7.41.3.35 findNoteOffset()

```
virtual qint32 qevercloud::INoteStore::findNoteOffset (
            const NoteFilter & filter,
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Finds the position of a note within a sorted subset of all of the user's notes. This may be useful for thin clients that are displaying a paginated listing of a large account, which need to know where a particular note sits in the list without retrieving all notes first.

Parameters

authenticationToken	Must be a valid token for the user's account unless the NoteFilter 'notebookGuid' is the GUID of a public notebook.
filter	The list of criteria that will constrain the notes to be returned.
guid	The GUID of the note to be retrieved.

Returns

If the note with the provided GUID is found within the matching note list, this will return the offset of that note within that list (where the first offset is 0). If the note is not found within the set of notes, this will return -1.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "offset" - not between 0 and EDAM_USER_NOTES_MAX • BAD_DATA_FORMAT "maxNotes" - not between 0 and EDAM_USER_NOTES_MAX • BAD_DATA_FORMAT "NoteFilter.notebookGuid" - if malformed • BAD_DATA_FORMAT "NoteFilter.tagGuids" - if any are malformed • BAD_DATA_FORMAT "NoteFilter.words" - if search string too long
EDAMNotFoundException	• "Notebook.guid" - not found, by GUID • "Note.guid" - not found, by GUID

7.41.3.36 findNoteOffsetAsync()

```
virtual AsyncResult * qevercloud::INoteStore::findNoteOffsetAsync (
            const NoteFilter & filter,
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of findNoteOffset

7.41.3.37 findNotesMetadata()

```
virtual NotesMetadataList qevercloud::INoteStore::findNotesMetadata (
            const NoteFilter & filter,
            qint32 offset,
            qint32 maxNotes,
            const NotesMetadataResultSpec & resultSpec,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Used to find the high-level information about a set of the notes from a user's account based on various criteria specified via a NoteFilter object.

Web applications that wish to periodically check for new content in a user's Evernote account should consider using webhooks instead of polling this API. See http://dev.evernote.com/documentation/cloud/chapters/polling_notification.php for more information.

Parameters

authenticationToken	Must be a valid token for the user's account unless the NoteFilter 'notebookGuid' is the GUID of a public notebook.
filter	The list of criteria that will constrain the notes to be returned.
offset	The numeric index of the first note to show within the sorted results. The numbering scheme starts with "0". This can be used for pagination.
maxNotes	The maximum notes to return in this query. The service will return a set of notes that is no larger than this number, but may return fewer notes if needed. The NoteList.totalNotes field in the return value will indicate whether there are more values available after the returned set. Currently, the service will not return more than 250 notes in a single request, but this number may change in the future.
resultSpec	This specifies which information should be returned for each matching Note. The fields on this structure can be used to eliminate data that the client doesn't need, which will reduce the time and bandwidth to receive and process the reply.

Returns

The list of notes that match the criteria. The Notes.sharedNotes field will not be set.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "offset" - not between 0 and EDAM_USER_NOTES_MAX • BAD_DATA_FORMAT "maxNotes" - not between 0 and EDAM_USER_NOTES_MAX • BAD_DATA_FORMAT "NoteFilter.notebookGuid" - if malformed • BAD_DATA_FORMAT "NoteFilter.tagGuids" - if any are malformed • BAD_DATA_FORMAT "NoteFilter.words" - if search string too long
EDAMNotFoundException	• "Notebook.guid" - not found, by GUID

7.41.3.38 findNotesMetadataAsync()

```
virtual AsyncResult * qevercloud::INoteStore::findNotesMetadataAsync (
            const NoteFilter & filter,
            qint32 offset,
            qint32 maxNotes,
            const NotesMetadataResultSpec & resultSpec,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of findNotesMetadata

7.41.3.39 findRelated()

```
virtual RelatedResult qevercloud::INoteStore::findRelated (
            const RelatedQuery & query,
            const RelatedResultSpec & resultSpec,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Identify related entities on the service, such as notes, notebooks, tags and users in a business related to notes or content.

Parameters

query	The information about which we are finding related entities.
resultSpec	Allows the client to indicate the type and quantity of information to be returned, allowing a saving of time and bandwidth.

Returns

The result of the query, with information considered to likely be relevantly related to the information described by the query.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "RelatedQuery.plainText" - If you provided a a zero-length plain text value. • BAD_DATA_FORMAT "RelatedQuery.noteGuid" - If you provided an invalid Note GUID, that is, one that does not match the constraints defined by EDAM_GUID_LEN_MIN, EDAM_GUID_LEN_MAX, EDAM_GUID_REGEX. • BAD_DATA_FORMAT "NoteFilter.notebookGuid" - if malformed • BAD_DATA_FORMAT "NoteFilter.tagGuids" - if any are malformed • BAD_DATA_FORMAT "NoteFilter.words" - if search string too long • PERMISSION_DENIED "Note" - If the caller does not have access to the note identified by RelatedQuery.noteGuid. • PERMISSION_DENIED "authenticationToken" - If the caller has requested to findExperts in the context of a non business user (i.e. The authenticationToken is not a business auth token). • DATA_REQUIRED "RelatedResultSpec" - If you did not not set any values in the result spec.
EDAMNotFoundException	• "RelatedQuery.noteGuid" - the note with that GUID is not found, if that field has been set in the query.

7.41.3.40 findRelatedAsync()

```
virtual AsyncResult * qevercloud::INoteStore::findRelatedAsync (
            const RelatedQuery & query,
            const RelatedResultSpec & resultSpec,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of findRelated

7.41.3.41 getDefaultNotebook()

```
virtual Notebook qevercloud::INoteStore::getDefaultNotebook (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns the notebook that should be used to store new notes in the user's account when no other notebooks are specified.

7.41.3.42 getDefaultNotebookAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getDefaultNotebookAsync (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getDefaultNotebook

7.41.3.43 getFilteredSyncChunk()

```
virtual SyncChunk qevercloud::INoteStore::getFilteredSyncChunk (
            qint32 afterUSN,
            qint32 maxEntries,
            const SyncChunkFilter & filter,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asks the NoteStore to provide the state of the account in order of last modification. This request retrieves one block of the server's state so that a client can make several small requests against a large account rather than getting the entire state in one big message. This call gives fine-grained control of the data that will be received by a client by omitting data elements that a client doesn't need. This may reduce network traffic and sync times.

Parameters

afterUSN	The client can pass this value to ask only for objects that have been updated after a certain point. This allows the client to receive updates after its last checkpoint rather than doing a full synchronization on every pass. The default value of "0" indicates that the client wants to get objects from the start of the account.
maxEntries	The maximum number of modified objects that should be returned in the result SyncChunk. This can be used to limit the size of each individual message to be friendly for network transfer.
filter	The caller must set some of the flags in this structure to specify which data types should be returned during the synchronization. See the SyncChunkFilter structure for information on each flag.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "afterUSN" - if negative • BAD_DATA_FORMAT "maxEntries" - if less than 1

7.41.3.44 getFilteredSyncChunkAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getFilteredSyncChunkAsync (
            qint32 afterUSN,
            qint32 maxEntries,
            const SyncChunkFilter & filter,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getFilteredSyncChunk

7.41.3.45 getLinkedNotebookSyncChunk()

```
virtual SyncChunk qevercloud::INoteStore::getLinkedNotebookSyncChunk (
            const LinkedNotebook & linkedNotebook,
            qint32 afterUSN,
            qint32 maxEntries,
            bool fullSyncOnly,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asks the NoteStore to provide information about the contents of a linked notebook that has been shared with the caller, or that is public to the world. This will return a result that is similar to getSyncChunk, but will only contain entries that are visible to the caller. I.e. only that particular Notebook will be visible, along with its Notes, and Tags on those Notes.

This function must be called on the shard that owns the referenced notebook. (I.e. the shardId in /shard/shard↩ Id/edam/note must be the same as LinkedNotebook.shardId.)

Parameters

authenticationToken	This should be an authenticationToken for the guest who has received the invitation to the share. (I.e. this should not be the result of NoteStore.authenticateToSharedNotebook)
linkedNotebook	This structure should contain identifying information and permissions to access the notebook in question. This must contain the valid fields for either a shared notebook (e.g. shareKey) or a public notebook (e.g. username, uri)
afterUSN	The client can pass this value to ask only for objects that have been updated after a certain point. This allows the client to receive updates after its last checkpoint rather than doing a full synchronization on every pass. The default value of "0" indicates that the client wants to get objects from the start of the account.
maxEntries	The maximum number of modified objects that should be returned in the result SyncChunk. This can be used to limit the size of each individual message to be friendly for network transfer. Applications should not request more than 256 objects at a time, and must handle the case where the service returns less than the requested number of objects in a given request even though more objects are available on the service.
fullSyncOnly	If true, then the client only wants initial data for a full sync. In this case, the service will not return any expunged objects, and will not return any Resources, since these are also provided in their corresponding Notes.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "afterUSN" - if negative • BAD_DATA_FORMAT "maxEntries" - if less than 1
EDAMNotFoundException	• "LinkedNotebook" - if the provided information doesn't match any valid notebook • "LinkedNotebook.uri" - if the provided public URI doesn't match any valid notebook • "SharedNotebook.id" - if the provided information indicates a shared notebook that no longer exists

7.41.3.46 getLinkedNotebookSyncChunkAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getLinkedNotebookSyncChunkAsync (
            const LinkedNotebook & linkedNotebook,
            qint32 afterUSN,
            qint32 maxEntries,
```

```
            bool fullSyncOnly,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getLinkedNotebookSyncChunk

7.41.3.47 getLinkedNotebookSyncState()

```
virtual SyncState qevercloud::INoteStore::getLinkedNotebookSyncState (
            const LinkedNotebook & linkedNotebook,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asks the NoteStore to provide information about the status of a linked notebook that has been shared with the caller, or that is public to the world. This will return a result that is similar to getSyncState, but may omit SyncState.uploaded if the caller doesn't have permission to write to the linked notebook.

This function must be called on the shard that owns the referenced notebook. (I.e. the shardId in /shard/shard↩ Id/edam/note must be the same as LinkedNotebook.shardId.)

Parameters

authenticationToken	This should be an authenticationToken for the guest who has received the invitation to the share. (I.e. this should not be the result of NoteStore.authenticateToSharedNotebook)
linkedNotebook	This structure should contain identifying information and permissions to access the notebook in question.

Exceptions

EDAMUserException	• DATA_REQUIRED "LinkedNotebook.username" - The username field must be populated with the current username of the owner of the notebook for which you are obtaining sync state.
EDAMNotFoundException	• "LinkedNotebook.username" - If the LinkedNotebook.username field does not correspond to a current user on the service.
SystemException	• SHARD_UNAVAILABLE - If the provided LinkedNotebook.username corresponds to a user whose account is on a shard other than that on which this method was invoked.

7.41.3.48 getLinkedNotebookSyncStateAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getLinkedNotebookSyncStateAsync (
            const LinkedNotebook & linkedNotebook,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getLinkedNotebookSyncState

7.41.3.49 getNote()

```
virtual Note qevercloud::INoteStore::getNote (
            Guid guid,
            bool withContent,
            bool withResourcesData,
            bool withResourcesRecognition,
            bool withResourcesAlternateData,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

DEPRECATED. See getNoteWithResultSpec.

This function is equivalent to getNoteWithResultSpec, with each of the boolean parameters mapping to the equivalent field of a NoteResultSpec. The Note.sharedNotes field is never populated on the returned note. To get a note with its shares, use getNoteWithResultSpec.

7.41.3.50 getNoteApplicationData()

```
virtual LazyMap qevercloud::INoteStore::getNoteApplicationData (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Get all of the application data for the note identified by GUID, with values returned within the LazyMap fullMap field. If there are no applicationData entries, then a LazyMap with an empty fullMap will be returned. If your application only needs to fetch its own applicationData entry, use getNoteApplicationDataEntry instead.

7.41.3.51 getNoteApplicationDataAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getNoteApplicationDataAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getNoteApplicationData

7.41.3.52 getNoteApplicationDataEntry()

```
virtual QString qevercloud::INoteStore::getNoteApplicationDataEntry (
            Guid guid,
            QString key,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Get the value of a single entry in the applicationData map for the note identified by GUID.

Exceptions

EDAMNotFoundException	• "Note.guid" - note not found, by GUID • "NoteAttributes.applicationData.key" - note not found, by key

7.41.3.53 getNoteApplicationDataEntryAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getNoteApplicationDataEntryAsync (
            Guid guid,
            QString key,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getNoteApplicationDataEntry

7.41.3.54 getNoteAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getNoteAsync (
            Guid guid,
            bool withContent,
            bool withResourcesData,
            bool withResourcesRecognition,
            bool withResourcesAlternateData,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getNote

7.41.3.55 getNotebook()

```
virtual Notebook qevercloud::INoteStore::getNotebook (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns the current state of the notebook with the provided GUID. The notebook may be active or deleted (but not expunged).

Parameters

guid	The GUID of the notebook to be retrieved.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Notebook.guid" - if the parameter is missing • PERMISSION_DENIED "Notebook" - private notebook, user doesn't own
EDAMNotFoundException	• "Notebook.guid" - tag not found, by GUID

7.41.3.56 getNotebookAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getNotebookAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getNotebook

7.41.3.57 getNotebookShares()

```
virtual ShareRelationships qevercloud::INoteStore::getNotebookShares (
            QString notebookGuid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Return the share relationships for the given notebook, including both the invitations and the memberships.

Note: Beta method! This method is currently intended for limited use by Evernote clients that have discussed using this routine with the platform team.

7.41.3.58 getNotebookSharesAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getNotebookSharesAsync (
            QString notebookGuid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getNotebookShares

7.41.3.59 getNoteContent()

```
virtual QString qevercloud::INoteStore::getNoteContent (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns XHTML contents of the note with the provided GUID. If the Note is found in a public notebook, the authenticationToken will be ignored (so it could be an empty string).

Parameters

guid	The GUID of the note to be retrieved.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Note.guid" - if the parameter is missing • PERMISSION_DENIED "Note" - private note, user doesn't own
EDAMNotFoundException	• "Note.guid" - not found, by GUID

7.41.3.60 getNoteContentAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getNoteContentAsync (
```

```
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getNoteContent

7.41.3.61 getNoteSearchText()

```
virtual QString qevercloud::INoteStore::getNoteSearchText (
            Guid guid,
            bool noteOnly,
            bool tokenizeForIndexing,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns a block of the extracted plain text contents of the note with the provided GUID. This text can be indexed for search purposes by a light client that doesn't have capabilities to extract all of the searchable text content from the note and its resources.

If the Note is found in a public notebook, the authenticationToken will be ignored (so it could be an empty string).

Parameters

guid	The GUID of the note to be retrieved.
noteOnly	If true, this will only return the text extracted from the ENML contents of the note itself. If false, this will also include the extracted text from any text-bearing resources (PDF, recognized images)
tokenizeForIndexing	If true, this will break the text into cleanly separated and sanitized tokens. If false, this will return the more raw text extraction, with its original punctuation, capitalization, spacing, etc.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Note.guid" - if the parameter is missing • PERMISSION_DENIED "Note" - private note, user doesn't own
EDAMNotFoundException	• "Note.guid" - not found, by GUID

7.41.3.62 getNoteSearchTextAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getNoteSearchTextAsync (
            Guid guid,
            bool noteOnly,
            bool tokenizeForIndexing,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getNoteSearchText

7.41.3.63 getNoteTagNames()

```
virtual QStringList qevercloud::INoteStore::getNoteTagNames (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns a list of the names of the tags for the note with the provided guid. This can be used with authentication to get the tags for a user's own note, or can be used without valid authentication to retrieve the names of the tags for a note in a public notebook.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Note.guid" - if the parameter is missing • PERMISSION_DENIED "Note" - private note, user doesn't own
EDAMNotFoundException	• "Note.guid" - not found, by GUID

7.41.3.64 getNoteTagNamesAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getNoteTagNamesAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getNoteTagNames

7.41.3.65 getNoteVersion()

```
virtual Note qevercloud::INoteStore::getNoteVersion (
            Guid noteGuid,
            qint32 updateSequenceNum,
            bool withResourcesData,
            bool withResourcesRecognition,
            bool withResourcesAlternateData,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

This can be used to retrieve a previous version of a Note after it has been updated within the service. The caller must identify the note (via its guid) and the version (via the updateSequenceNumber of that version). to find a listing of the stored version USNs for a note, call listNoteVersions. This call is only available for notes in Premium accounts. (I.e. access to past versions of Notes is a Premium-only feature.)

Parameters

noteGuid	The GUID of the note to be retrieved.
updateSequenceNum	The USN of the version of the note that is being retrieved
withResourcesData	If true, any Resource elements in this Note will include the binary contents of their 'data' field's body.
withResourcesRecognition	If true, any Resource elements will include the binary contents of the 'recognition' field's body if recognition data is present.
withResourcesAlternateData	If true, any Resource elements in this Note will include the binary contents of their 'alternateData' fields' body, if an alternate form is present.

Exceptions

EDAMUserException	• DATA_REQUIRED "Note.guid" - if GUID is null or empty string. • BAD_DATA_FORMAT "Note.guid" - if GUID is not of correct length.
EDAMNotFoundException	• "Note.guid" - not found, by GUID. • "Note.updateSequenceNumber" - the Note doesn't have a version with the corresponding USN.

7.41.3.66 getNoteVersionAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getNoteVersionAsync (
            Guid noteGuid,
            qint32 updateSequenceNum,
            bool withResourcesData,
            bool withResourcesRecognition,
            bool withResourcesAlternateData,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getNoteVersion

7.41.3.67 getNoteWithResultSpec()

```
virtual Note qevercloud::INoteStore::getNoteWithResultSpec (
            Guid guid,
            const NoteResultSpec & resultSpec,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns the current state of the note in the service with the provided GUID. The ENML contents of the note will only be provided if the 'withContent' parameter is true. The service will include the meta-data for each resource in the note, but the binary content depends on whether it is explicitly requested in resultSpec parameter. If the Note is found in a public notebook, the authenticationToken will be ignored (so it could be an empty string). The applicationData fields are returned as keysOnly.

Parameters

authenticationToken	An authentication token that grants the caller access to the requested note.
guid	The GUID of the note to be retrieved.
resultSpec	A structure specifying the fields of the note that the caller would like to get.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Note.guid" - if the parameter is missing • PERMISSION_DENIED "Note" - private note, user doesn't own

Exceptions

EDAMNotFoundException	• "Note.guid" - not found, by GUID

7.41.3.68 getNoteWithResultSpecAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getNoteWithResultSpecAsync (
            Guid guid,
            const NoteResultSpec & resultSpec,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getNoteWithResultSpec

7.41.3.69 getPublicNotebook()

```
virtual Notebook qevercloud::INoteStore::getPublicNotebook (
            UserID userId,
            QString publicUri,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Looks for a user account with the provided userId on this NoteStore shard and determines whether that account contains a public notebook with the given URI. If the account is not found, or no public notebook exists with this URI, this will throw an EDAMNotFoundException, otherwise this will return the information for that Notebook.

If a notebook is visible on the web with a full URL like `http://www.evernote.com/pub/sethdemo/api` Then 'sethdemo' is the username that can be used to look up the userId, and 'api' is the publicUri.

Parameters

userId	The numeric identifier for the user who owns the public notebook. To find this value based on a username string, you can invoke UserStore.getPublicUserInfo
publicUri	The uri string for the public notebook, from Notebook.publishing.uri.

Exceptions

EDAMNotFoundException	• "Publishing.uri" - not found, by URI
EDAMSystemException	• TAKEN_DOWN "PublicNotebook" - The specified public notebook is taken down (for all requesters). • TAKEN_DOWN "Country" - The specified public notebook is taken down for the requester because of an IP-based country lookup.

7.41.3.70 getPublicNotebookAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getPublicNotebookAsync (
            UserID userId,
            QString publicUri,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getPublicNotebook

7.41.3.71 getResource()

```
virtual Resource qevercloud::INoteStore::getResource (
            Guid guid,
            bool withData,
            bool withRecognition,
            bool withAttributes,
            bool withAlternateData,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns the current state of the resource in the service with the provided GUID. If the Resource is found in a public notebook, the authenticationToken will be ignored (so it could be an empty string). Only the keys for the applicationData will be returned.

Parameters

guid	The GUID of the resource to be retrieved.
withData	If true, the Resource will include the binary contents of the 'data' field's body.
withRecognition	If true, the Resource will include the binary contents of the 'recognition' field's body if recognition data is present.
withAttributes	If true, the Resource will include the attributes
withAlternateData	If true, the Resource will include the binary contents of the 'alternateData' field's body, if an alternate form is present.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Resource.guid" - if the parameter is missing • PERMISSION_DENIED "Resource" - private resource, user doesn't own
EDAMNotFoundException	• "Resource.guid" - not found, by GUID

7.41.3.72 getResourceAlternateData()

```
virtual QByteArray qevercloud::INoteStore::getResourceAlternateData (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

If the Resource with the provided GUID has an alternate data representation (indicated via the Resource.alternateData field), then this request can be used to retrieve the binary contents of that alternate data file. If the caller asks about a resource that has no alternate data form, this will throw EDAMNotFoundException.

Parameters

guid	The GUID of the resource whose recognition data should be retrieved.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Resource.guid" - if the parameter is missing • PERMISSION_DENIED "Resource" - private resource, user doesn't own
EDAMNotFoundException	• "Resource.guid" - not found, by GUID • "Resource.alternateData" - resource has no recognition

7.41.3.73 getResourceAlternateDataAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getResourceAlternateDataAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getResourceAlternateData

7.41.3.74 getResourceApplicationData()

```
virtual LazyMap qevercloud::INoteStore::getResourceApplicationData (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Get all of the application data for the Resource identified by GUID, with values returned within the LazyMap fullMap field. If there are no applicationData entries, then a LazyMap with an empty fullMap will be returned. If your application only needs to fetch its own applicationData entry, use getResourceApplicationDataEntry instead.

7.41.3.75 getResourceApplicationDataAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getResourceApplicationDataAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getResourceApplicationData

7.41.3.76 getResourceApplicationDataEntry()

```
virtual QString qevercloud::INoteStore::getResourceApplicationDataEntry (
            Guid guid,
            QString key,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Get the value of a single entry in the applicationData map for the Resource identified by GUID.

Exceptions

EDAMNotFoundException	• "Resource.guid" - Resource not found, by GUID • "ResourceAttributes.applicationData.key" - Resource not found, by key

7.41.3.77 getResourceApplicationDataEntryAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getResourceApplicationDataEntryAsync (
            Guid guid,
            QString key,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getResourceApplicationDataEntry

7.41.3.78 getResourceAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getResourceAsync (
            Guid guid,
            bool withData,
            bool withRecognition,
            bool withAttributes,
            bool withAlternateData,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getResource

7.41.3.79 getResourceAttributes()

```
virtual ResourceAttributes qevercloud::INoteStore::getResourceAttributes (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns the set of attributes for the Resource with the provided GUID. If the Resource is found in a public notebook, the authenticationToken will be ignored (so it could be an empty string).

Parameters

guid	The GUID of the resource whose attributes should be retrieved.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Resource.guid" - if the parameter is missing • PERMISSION_DENIED "Resource" - private resource, user doesn't own

Exceptions

EDAMNotFoundException	• "Resource.guid" - not found, by GUID

7.41.3.80 getResourceAttributesAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getResourceAttributesAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getResourceAttributes

7.41.3.81 getResourceByHash()

```
virtual Resource qevercloud::INoteStore::getResourceByHash (
            Guid noteGuid,
            QByteArray contentHash,
            bool withData,
            bool withRecognition,
            bool withAlternateData,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns the current state of a resource, referenced by containing note GUID and resource content hash.

Parameters

noteGuid	The GUID of the note that holds the resource to be retrieved.
contentHash	The MD5 checksum of the resource within that note. Note that this is the binary checksum, for example from Resource.data.bodyHash, and not the hex-encoded checksum that is used within an en-media tag in a note body.
withData	If true, the Resource will include the binary contents of the 'data' field's body.
withRecognition	If true, the Resource will include the binary contents of the 'recognition' field's body.
withAlternateData	If true, the Resource will include the binary contents of the 'alternateData' field's body, if an alternate form is present.

Exceptions

EDAMUserException	• DATA_REQUIRED "Note.guid" - noteGuid param missing • DATA_REQUIRED "Note.contentHash" - contentHash param missing • PERMISSION_DENIED "Resource" - private resource, user doesn't own
EDAMNotFoundException	• "Note" - not found, by guid • "Resource" - not found, by hash

7.41.3.82 getResourceByHashAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getResourceByHashAsync (
            Guid noteGuid,
            QByteArray contentHash,
            bool withData,
            bool withRecognition,
            bool withAlternateData,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getResourceByHash

7.41.3.83 getResourceData()

```
virtual QByteArray qevercloud::INoteStore::getResourceData (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns binary data of the resource with the provided GUID. For example, if this were an image resource, this would contain the raw bits of the image. If the Resource is found in a public notebook, the authenticationToken will be ignored (so it could be an empty string).

Parameters

guid	The GUID of the resource to be retrieved.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Resource.guid" - if the parameter is missing • PERMISSION_DENIED "Resource" - private resource, user doesn't own
EDAMNotFoundException	• "Resource.guid" - not found, by GUID

7.41.3.84 getResourceDataAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getResourceDataAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getResourceData

7.41.3.85 getResourceRecognition()

```
virtual QByteArray qevercloud::INoteStore::getResourceRecognition (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns the binary contents of the recognition index for the resource with the provided GUID. If the caller asks about a resource that has no recognition data, this will throw EDAMNotFoundException. If the Resource is found in a public notebook, the authenticationToken will be ignored (so it could be an empty string).

Parameters

guid	The GUID of the resource whose recognition data should be retrieved.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Resource.guid" - if the parameter is missing • PERMISSION_DENIED "Resource" - private resource, user doesn't own
EDAMNotFoundException	• "Resource.guid" - not found, by GUID • "Resource.recognition" - resource has no recognition

7.41.3.86 getResourceRecognitionAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getResourceRecognitionAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getResourceRecognition

7.41.3.87 getResourceSearchText()

```
virtual QString qevercloud::INoteStore::getResourceSearchText (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns a block of the extracted plain text contents of the resource with the provided GUID. This text can be indexed for search purposes by a light client that doesn't have capability to extract all of the searchable text content from a resource.

If the Resource is found in a public notebook, the authenticationToken will be ignored (so it could be an empty string).

Parameters

guid	The GUID of the resource to be retrieved.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Resource.guid" - if the parameter is missing • PERMISSION_DENIED "Resource" - private resource, user doesn't own
EDAMNotFoundException	• "Resource.guid" - not found, by GUID

7.41.3.88 getResourceSearchTextAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getResourceSearchTextAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getResourceSearchText

7.41.3.89 getSearch()

```
virtual SavedSearch qevercloud::INoteStore::getSearch (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns the current state of the search with the provided GUID.

Parameters

guid	The GUID of the search to be retrieved.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "SavedSearch.guid" - if the parameter is missing • PERMISSION_DENIED "SavedSearch" - private Tag, user doesn't own
EDAMNotFoundException	• "SavedSearch.guid" - not found, by GUID

7.41.3.90 getSearchAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getSearchAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getSearch

7.41.3.91 getSharedNotebookByAuth()

```
virtual SharedNotebook qevercloud::INoteStore::getSharedNotebookByAuth (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

This function is used to retrieve extended information about a shared notebook by a guest who has already authenticated to access that notebook. This requires an 'authenticationToken' parameter which should be the resut of a call to authenticateToSharedNotebook(...). I.e. this is the token that gives access to the particular shared notebook in someone else's account – it's not the authenticationToken for the owner of the notebook itself.

Parameters

authenticationToken	Should be the authentication token retrieved from the reply of authenticateToSharedNotebook(), proving access to a particular shared notebook.

Exceptions

EDAMUserException	• PERMISSION_DENIED "authenticationToken" - authentication token doesn't correspond to a valid shared notebook
EDAMNotFoundException	• "SharedNotebook.id" - the shared notebook no longer exists

7.41.3.92 getSharedNotebookByAuthAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getSharedNotebookByAuthAsync (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getSharedNotebookByAuth

7.41.3.93 getSyncState()

```
virtual SyncState qevercloud::INoteStore::getSyncState (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asks the NoteStore to provide information about the status of the user account corresponding to the provided authentication token.

7.41.3.94 getSyncStateAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getSyncStateAsync (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getSyncState

7.41.3.95 getTag()

```
virtual Tag qevercloud::INoteStore::getTag (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns the current state of the Tag with the provided GUID.

Parameters

guid	The GUID of the tag to be retrieved.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Tag.guid" - if the parameter is missing • PERMISSION_DENIED "Tag" - private Tag, user doesn't own
EDAMNotFoundException	• "Tag.guid" - tag not found, by GUID

7.41.3.96 getTagAsync()

```
virtual AsyncResult * qevercloud::INoteStore::getTagAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getTag

7.41.3.97 listAccessibleBusinessNotebooks()

```
virtual QList< Notebook > qevercloud::INoteStore::listAccessibleBusinessNotebooks (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns a list of all the notebooks in a business that the user has permission to access, regardless of whether the user has joined them. This includes notebooks that have been shared with the entire business as well as notebooks that have been shared directly with the user.

Parameters

authenticationToken	A business authentication token obtained by calling UserStore.authenticateToBusiness.

Exceptions

EDAMUserException	• INVALID_AUTH "authenticationToken" - if the authentication token is not a business auth token.

7.41.3.98 listAccessibleBusinessNotebooksAsync()

```
virtual AsyncResult * qevercloud::INoteStore::listAccessibleBusinessNotebooksAsync (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of listAccessibleBusinessNotebooks

7.41.3.99 listLinkedNotebooks()

```
virtual QList< LinkedNotebook > qevercloud::INoteStore::listLinkedNotebooks (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns a list of linked notebooks

7.41.3.100 listLinkedNotebooksAsync()

```
virtual AsyncResult * qevercloud::INoteStore::listLinkedNotebooksAsync (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of listLinkedNotebooks

7.41.3.101 listNotebooks()

```
virtual QList< Notebook > qevercloud::INoteStore::listNotebooks (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns a list of all of the notebooks in the account.

7.41.3.102 listNotebooksAsync()

```
virtual AsyncResult * qevercloud::INoteStore::listNotebooksAsync (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of listNotebooks

7.41.3.103 listNoteVersions()

```
virtual QList< NoteVersionId > qevercloud::INoteStore::listNoteVersions (
            Guid noteGuid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns a list of the prior versions of a particular note that are saved within the service. These prior versions are stored to provide a recovery from unintentional removal of content from a note. The identifiers that are returned by this call can be used with getNoteVersion to retrieve the previous note. The identifiers will be listed from the most recent versions to the oldest. This call is only available for notes in Premium accounts. (I.e. access to past versions of Notes is a Premium-only feature.)

Exceptions

EDAMUserException	• DATA_REQUIRED "Note.guid" - if GUID is null or empty string. • BAD_DATA_FORMAT "Note.guid" - if GUID is not of correct length.
EDAMNotFoundException	• "Note.guid" - not found, by GUID.

7.41.3.104 listNoteVersionsAsync()

```
virtual AsyncResult * qevercloud::INoteStore::listNoteVersionsAsync (
            Guid noteGuid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of listNoteVersions

7.41.3.105 listSearches()

```
virtual QList< SavedSearch > qevercloud::INoteStore::listSearches (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns a list of the searches in the account. Evernote does not support the undeletion of searches, so this will only include active searches.

7.41.3.106 listSearchesAsync()

```
virtual AsyncResult * qevercloud::INoteStore::listSearchesAsync (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of listSearches

7.41.3.107 listSharedNotebooks()

```
virtual QList< SharedNotebook > qevercloud::INoteStore::listSharedNotebooks (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Lists the collection of shared notebooks for all notebooks in the users account.

Returns

The list of all SharedNotebooks for the user

7.41.3.108 listSharedNotebooksAsync()

```
virtual AsyncResult * qevercloud::INoteStore::listSharedNotebooksAsync (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of listSharedNotebooks

7.41.3.109 listTags()

```
virtual QList< Tag > qevercloud::INoteStore::listTags (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns a list of the tags in the account. Evernote does not support the undeletion of tags, so this will only include active tags.

7.41.3.110 listTagsAsync()

```
virtual AsyncResult * qevercloud::INoteStore::listTagsAsync (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of listTags

7.41.3.111 listTagsByNotebook()

```
virtual QList< Tag > qevercloud::INoteStore::listTagsByNotebook (
            Guid notebookGuid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns a list of the tags that are applied to at least one note within the provided notebook. If the notebook is public, the authenticationToken may be ignored.

Parameters

notebookGuid	the GUID of the notebook to use to find tags

Exceptions

EDAMNotFoundException	• "Notebook.guid" - notebook not found by GUID

7.41.3.112 listTagsByNotebookAsync()

```
virtual AsyncResult * qevercloud::INoteStore::listTagsByNotebookAsync (
            Guid notebookGuid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of listTagsByNotebook

7.41.3.113 manageNotebookShares()

```
virtual ManageNotebookSharesResult qevercloud::INoteStore::manageNotebookShares (
            const ManageNotebookSharesParameters & parameters,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Manage invitations and memberships associated with a given notebook.

Note: Beta method! This method is currently intended for limited use by Evernote clients that have discussed using this routine with the platform team.

Parameters

parameters	A structure containing all parameters for the updates. See the structure documentation for details.

Exceptions

EDAMUserException	• EDAMErrorCode.LIMIT_REACHED "SharedNotebook" - Trying to share a notebook while the notebook already has EDAM_NOTEBOOK_SHARED_NOTEBOOK_MAX shares.

7.41.3.114 manageNotebookSharesAsync()

```
virtual AsyncResult * qevercloud::INoteStore::manageNotebookSharesAsync (
            const ManageNotebookSharesParameters & parameters,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of manageNotebookShares

7.41.3.115 noteStoreUrl()

```
virtual QString qevercloud::INoteStore::noteStoreUrl ( ) const  [pure virtual]
```

7.41.3.116 setNoteApplicationDataEntry()

```
virtual qint32 qevercloud::INoteStore::setNoteApplicationDataEntry (
            Guid guid,
            QString key,
            QString value,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Update, or create, an entry in the applicationData map for the note identified by guid.

7.41.3.117 setNoteApplicationDataEntryAsync()

```
virtual AsyncResult * qevercloud::INoteStore::setNoteApplicationDataEntryAsync (
            Guid guid,
            QString key,
            QString value,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of setNoteApplicationDataEntry

7.41.3.118 setNotebookRecipientSettings()

```
virtual Notebook qevercloud::INoteStore::setNotebookRecipientSettings (
            QString notebookGuid,
            const NotebookRecipientSettings & recipientSettings,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Set values for the recipient settings associated with a notebook share. Only the recipient of the share can update their recipient settings.

If you do *not* wish to, or cannot, change one of the recipient settings fields, you must leave that field unset in recipientSettings. This method will skip that field for updates and attempt to leave the existing value as it is.

If recipientSettings.inMyList is false, both reminderNotifyInApp and reminderNotifyEmail will be either left as null or converted to false (if currently true).

To unset a notebook's stack, pass in the empty string for the stack field.

Parameters

authenticationToken	The owner authentication token for the recipient of the share.

Returns

The updated Notebook with the new recipient settings. Note that some of the recipient settings may differ from what was requested. Clients should update their state based on this return value.

Exceptions

EDAMNotFoundException	• Notebook.guid - Thrown if the service does not have a notebook record with the notebookGuid on the given shard. • Publishing.publishState - Thrown if the business notebook is not shared with the user and is also not published to their business.
EDAMUserException	• PEMISSION_DENIED "authenticationToken" - If the owner of the given token is not allowed to set recipient settings on the specified notebook. • DATA_CONFLICT "recipientSettings.reminderNotifyEmail" - Setting reminderNotifyEmail is allowed only for notebooks which belong to the same business as the user. • DATA_CONFLICT "recipientSettings.inMyList" - If the request is setting inMyList to false and any of reminder∗ settings to true.

7.41.3.119 setNotebookRecipientSettingsAsync()

```
virtual AsyncResult * qevercloud::INoteStore::setNotebookRecipientSettingsAsync (
            QString notebookGuid,
            const NotebookRecipientSettings & recipientSettings,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of setNotebookRecipientSettings

7.41.3.120 setNoteStoreUrl()

```
virtual void qevercloud::INoteStore::setNoteStoreUrl (
            QString url )  [pure virtual]
```

7.41.3.121 setResourceApplicationDataEntry()

```
virtual qint32 qevercloud::INoteStore::setResourceApplicationDataEntry (
            Guid guid,
            QString key,
            QString value,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Update, or create, an entry in the applicationData map for the Resource identified by guid.

7.41.3.122 setResourceApplicationDataEntryAsync()

```
virtual AsyncResult * qevercloud::INoteStore::setResourceApplicationDataEntryAsync (
            Guid guid,
            QString key,
            QString value,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of setResourceApplicationDataEntry

7.41.3.123 shareNote()

```
virtual QString qevercloud::INoteStore::shareNote (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

If this note is not already shared publicly (via its own direct URL), then this will start sharing that note. This will return the secret "Note Key" for this note that can currently be used in conjunction with the Note's GUID to gain direct read-only access to the Note. If the note is already shared, then this won't make any changes to the note, and the existing "Note Key" will be returned. The only way to change the Note Key for an existing note is to stopSharingNote first, and then call this function.

Parameters

guid	The GUID of the note to be shared.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Note.guid" - if the parameter is missing • PERMISSION_DENIED "Note" - private note, user doesn't own
EDAMNotFoundException	• "Note.guid" - not found, by GUID

7.41.3.124 shareNoteAsync()

```
virtual AsyncResult * qevercloud::INoteStore::shareNoteAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of shareNote

7.41.3.125 shareNotebook()

```
virtual SharedNotebook qevercloud::INoteStore::shareNotebook (
            const SharedNotebook & sharedNotebook,
```

```
        QString message,
        IRequestContextPtr ctx = {} )  [pure virtual]
```

@Deprecated for first-party clients. See createOrUpdateNotebookShares.

Share a notebook with an email address, and optionally to a specific recipient. If an existing SharedNotebook associated with sharedNotebook.notebookGuid is found by recipientUsername or email, then the values of shared↩ Notebook will be used to update the existing record, else a new record will be created.

If recipientUsername is set and there is already a SharedNotebook for that Notebook with that recipientUsername and the privileges on the existing notebook are lower, than on this one, this will update the privileges and sharer↩ UserId. If there isn't an existing SharedNotebook for recipientUsername, this will create and return a shared notebook for that email and recipientUsername. If recipientUsername is not set and there already is a SharedNotebook for a Notebook for that email address and the privileges on the existing SharedNotebook are lower than on this one, this will update the privileges and sharerUserId, and return the updated SharedNotebook. Otherwise, this will create and return a SharedNotebook for the email address.

If the authenticationToken is a Business auth token, recipientUsername is set and the recipient is in the same business as the business auth token, this method will also auto-join the business user to the SharedNotebook - that is it will set serviceJoined on the SharedNotebook and create a LinkedNotebook on the recipient's account pointing to the SharedNotebook. The LinkedNotebook creation happens out-of-band, so there will be a delay on the order of half a minute between the SharedNotebook and LinkedNotebook creation.

Also handles sending an email to the email addresses: if a SharedNotebook is being created, this will send the shared notebook invite email, and if a SharedNotebook already exists, it will send the shared notebook reminder email. Both these emails contain a link to join the notebook. If the notebook is being auto-joined, it sends an email with that information to the recipient.

Parameters

authenticationToken	Must be an authentication token from the owner or a shared notebook authentication token or business authentication token with sufficient permissions to change invitations for a notebook.
sharedNotebook	A shared notebook object populated with the email address of the share recipient, the notebook guid and the access permissions. All other attributes of the shared object are ignored. The SharedNotebook.allowPreview field must be explicitly set with either a true or false value.
message	The sharer-defined message to put in the email sent out.

Returns

The fully populated SharedNotebook object including the server assigned globalId which can both be used to uniquely identify the SharedNotebook.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "SharedNotebook.email" - if the email was not valid • DATA_REQUIRED "SharedNotebook.privilege" - if the SharedNotebook.privilegeLevel was not set. • BAD_DATA_FORMAT "SharedNotebook.requireLogin" - if requireLogin was set. requireLogin is deprecated. • BAD_DATA_FORMAT "SharedNotebook.privilegeLevel" - if the SharedNotebook.privilegeLevel field was unset or set to GROUP. • PERMISSION_DENIED "user" - if the email address on the authenticationToken's owner's account is not confirmed. • PERMISSION_DENIED "SharedNotebook.recipientSettings" - if recipientSettings is set in the sharedNotebook. Only the recipient can set these values via the setSharedNotebookRecipientSettings method. • EDAMErrorCode.LIMIT_REACHED "SharedNotebook" - The notebook already has EDAM_NOTEBOOK_SHARED_NOTEBOOK_MAX shares.
EDAMNotFoundException	• Notebook.guid - if the notebookGuid is not a valid GUID for the user.

7.41.3.126 shareNotebookAsync()

```
virtual AsyncResult * qevercloud::INoteStore::shareNotebookAsync (
            const SharedNotebook & sharedNotebook,
            QString message,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of shareNotebook

7.41.3.127 stopSharingNote()

```
virtual void qevercloud::INoteStore::stopSharingNote (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

If this note is shared publicly then this will stop sharing that note and invalidate its "Note Key", so any existing URLs to access that Note will stop working.

If the Note is not shared, then this function will do nothing.

This function does not remove individual shares for the note. To remove individual shares, see stopSharingNote↩ WithRecipients.

Parameters

guid	The GUID of the note to be un-shared.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Note.guid" - if the parameter is missing • PERMISSION_DENIED "Note" - private note, user doesn't own
EDAMNotFoundException	• "Note.guid" - not found, by GUID

7.41.3.128 stopSharingNoteAsync()

```
virtual AsyncResult * qevercloud::INoteStore::stopSharingNoteAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of stopSharingNote

7.41.3.129 unsetNoteApplicationDataEntry()

```
virtual qint32 qevercloud::INoteStore::unsetNoteApplicationDataEntry (
            Guid guid,
            QString key,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Remove an entry identified by 'key' from the applicationData map for the note identified by 'guid'. Silently ignores an unset of a non-existing key.

7.41.3.130 unsetNoteApplicationDataEntryAsync()

```
virtual AsyncResult * qevercloud::INoteStore::unsetNoteApplicationDataEntryAsync (
            Guid guid,
            QString key,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of unsetNoteApplicationDataEntry

7.41.3.131 unsetResourceApplicationDataEntry()

```
virtual qint32 qevercloud::INoteStore::unsetResourceApplicationDataEntry (
            Guid guid,
            QString key,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Remove an entry identified by 'key' from the applicationData map for the Resource identified by 'guid'.

7.41.3.132 unsetResourceApplicationDataEntryAsync()

```
virtual AsyncResult * qevercloud::INoteStore::unsetResourceApplicationDataEntryAsync (
            Guid guid,
            QString key,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of unsetResourceApplicationDataEntry

7.41.3.133 untagAll()

```
virtual void qevercloud::INoteStore::untagAll (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Removes the provided tag from every note that is currently tagged with this tag. If this operation is successful, the tag will still be in the account, but it will not be tagged on any notes.

This function is not indended for use by full synchronizing clients, since it does not provide enough result information to the client to reconcile the local state without performing a follow-up sync from the service. This is intended for "thin clients" that need to efficiently support this as a UI operation.

Parameters

guid	The GUID of the tag to remove from all notes.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Tag.guid" - if the guid parameter is missing • PERMISSION_DENIED "Tag" - user doesn't own tag
EDAMNotFoundException	• "Tag.guid" - tag not found, by GUID

7.41.3.134 untagAllAsync()

```
virtual AsyncResult * qevercloud::INoteStore::untagAllAsync (
            Guid guid,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of untagAll

7.41.3.135 updateLinkedNotebook()

```
virtual qint32 qevercloud::INoteStore::updateLinkedNotebook (
            const LinkedNotebook & linkedNotebook,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Parameters

linkedNotebook	Updates the name of a linked notebook.

Returns

The Update Sequence Number for this change within the account.

Exceptions

EDAMUserException	• DATA_REQUIRED "LinkedNotebook.shareName" - missing shareName • BAD_DATA_FORMAT "LinkedNotebook.shareName" - invalid shareName length or pattern • BAD_DATA_FORMAT "LinkedNotebook.stack" - invalid stack name length or pattern

7.41.3.136 updateLinkedNotebookAsync()

```
virtual AsyncResult * qevercloud::INoteStore::updateLinkedNotebookAsync (
            const LinkedNotebook & linkedNotebook,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of updateLinkedNotebook

7.41.3.137 updateNote()

```
virtual Note qevercloud::INoteStore::updateNote (
            const Note & note,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Submit a set of changes to a note to the service. The provided data must include the note's guid field for identification. The note's title must also be set.

Parameters

note	A Note object containing the desired fields to be populated on the service. With the exception of the note's title and guid, fields that are not being changed do not need to be set. If the content is not being modified, note.content should be left unset. If the list of resources is not being modified, note.resources should be left unset.

Returns

The Note.sharedNotes field will not be set. The service will include the meta-data for each resource in the note, but the binary contents of the resources and their recognition data will be omitted.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Note.title" - invalid length or pattern • BAD_DATA_FORMAT "Note.content" - invalid length for ENML body • BAD_DATA_FORMAT "NoteAttributes.∗" - bad resource string • BAD_DATA_FORMAT "ResourceAttributes.∗" - bad resource string • BAD_DATA_FORMAT "Resource.mime" - invalid resource MIME type • DATA_CONFLICT "Note.deleted" - deleted time set on active note • DATA_REQUIRED "Resource.data" - resource data body missing • ENML_VALIDATION "∗" - note content doesn't validate against DTD • LIMIT_REACHED "Note.tagGuids" - too many Tags on Note • LIMIT_REACHED "Note.resources" - too many resources on Note • LIMIT_REACHED "Note.size" - total note size too large • LIMIT_REACHED "Resource.data.size" - resource too large • LIMIT_REACHED "NoteAttribute.∗" - attribute string too long • LIMIT_REACHED "ResourceAttribute.∗" - attribute string too long • PERMISSION_DENIED "Note.notebookGuid" - user doesn't own destination • PERMISSION_DENIED "Note.tags" - user doesn't have permission to modify the note's tags. note.tags must be unset. • PERMISSION_DENIED "Note.attributes" - user doesn't have permission to modify the note's attributes. note.attributes must be unset. • QUOTA_REACHED "Accounting.uploadLimit" - note exceeds upload quota • BAD_DATA_FORMAT "Tag.name" - Note.tagNames was provided, and one of the specified tags had an invalid length or pattern • LIMIT_REACHED "Tag" - Note.tagNames was provided, and the required new tags would exceed the maximum number per account
EDAMNotFoundException	• "Note.guid" - note not found, by GUID • "Note.notebookGuid" - if notebookGuid provided, but not found

7.41.3.138 updateNoteAsync()

```
virtual AsyncResult * qevercloud::INoteStore::updateNoteAsync (
            const Note & note,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of updateNote

7.41.3.139 updateNotebook()

```
virtual qint32 qevercloud::INoteStore::updateNotebook (
            const Notebook & notebook,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Submits notebook changes to the service. The provided data must include the notebook's guid field for identification.

The Notebook will be moved to the specified Workspace, if a non empty Notebook.workspaceGuid is provided. If an empty Notebook.workspaceGuid is set and the Notebook is in a Workspace, then it will be removed from the Workspace and a full access SharedNotebook record will be ensured for the caller. If the caller does not already have a full access share, either the privilege of an existing share will be upgraded or a new share will be created. It is illegal to set a Notebook.workspaceGuid on a Workspace backing Notebook.

Parameters

notebook	The notebook object containing the requested changes.

Returns

The Update Sequence Number for this change within the account.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Notebook.name" - invalid length or pattern • BAD_DATA_FORMAT "Notebook.stack" - invalid length or pattern • BAD_DATA_FORMAT "Publishing.uri" - if publishing set but bad uri • BAD_DATA_FORMAT "Publishing.publicDescription" - if too long • DATA_CONFLICT "Notebook.name" - name already in use • DATA_CONFLICT "Publishing.uri" - if URI already in use • DATA_REQUIRED "Publishing.uri" - if publishing set but uri missing • DATA_REQUIRED "Notebook" - notebook parameter was null • PERMISSION_DENIED "Notebook.defaultNotebook" - if the 'defaultNotebook' field is set to 'true' for a Notebook that is not owned by the user identified by the passed authenticationToken.
EDAMNotFoundException	• "Notebook.guid" - not found, by GUID • "Workspace.guid" - if a non empty workspaceGuid set and no Workspace exists for the GUID

7.41.3.140 updateNotebookAsync()

```
virtual AsyncResult * qevercloud::INoteStore::updateNotebookAsync (
```

```
            const Notebook & notebook,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of updateNotebook

7.41.3.141 updateNoteIfUsnMatches()

```
virtual UpdateNoteIfUsnMatchesResult qevercloud::INoteStore::updateNoteIfUsnMatches (
            const Note & note,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Perform the same operation as updateNote() would provided that the update sequence number on the parameter Note object matches the current update sequence number that the service has for the note. If they do *not* match, then *no* update is performed and the return value will have the current server state in the note field and updated will be false. If the update sequence numbers between the client and server do match, then the note will be updated and the note field of the return value will be returned as it would be for the updateNote method. This method allows you to check for an update to the note on the service, by another client instance, from when you obtained the note state as a baseline for your edits and the time when you wish to save your edits. If your client can merge the conflict, you can avoid overwriting changes that were saved to the service by the other client.

See the updateNote method for information on the exceptions and parameters for this method. The only difference is that you must have an update sequence number defined on the note parameter (equal to the USN of the note as synched to the client), and the following additional exceptions might be thrown.

Exceptions

EDAMUserException	• DATA_REQUIRED "Note.updateSequenceNum" - If the update sequence number was not provided. This includes a value that is set as 0. • BAD_DATA_FORMAT "Note.updateSequenceNum" - If the note has an update sequence number that is larger than the current server value, which should not happen if your client is working correctly.

7.41.3.142 updateNoteIfUsnMatchesAsync()

```
virtual AsyncResult * qevercloud::INoteStore::updateNoteIfUsnMatchesAsync (
            const Note & note,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of updateNoteIfUsnMatches

7.41.3.143 updateResource()

```
virtual qint32 qevercloud::INoteStore::updateResource (
            const Resource & resource,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Submit a set of changes to a resource to the service. This can be used to update the meta-data about the resource, but cannot be used to change the binary contents of the resource (including the length and hash). These cannot be changed directly without creating a new resource and removing the old one via updateNote.

Parameters

resource	A Resource object containing the desired fields to be populated on the service. The service will attempt to update the resource with the following fields from the client: • guid: must be provided to identify the resource • mime • width • height • duration • attributes: optional. if present, the set of attributes will be replaced.

Returns

The Update Sequence Number of the resource after the changes have been applied.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Resource.guid" - if the parameter is missing • BAD_DATA_FORMAT "Resource.mime" - invalid resource MIME type • BAD_DATA_FORMAT "ResourceAttributes.∗" - bad resource string • LIMIT_REACHED "ResourceAttribute.∗" - attribute string too long • PERMISSION_DENIED "Resource" - private resource, user doesn't own
EDAMNotFoundException	• "Resource.guid" - not found, by GUID

7.41.3.144 updateResourceAsync()

```
virtual AsyncResult * qevercloud::INoteStore::updateResourceAsync (
            const Resource & resource,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of updateResource

7.41.3.145 updateSearch()

```
virtual qint32 qevercloud::INoteStore::updateSearch (
            const SavedSearch & search,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Submits search changes to the service. The provided data must include the search's guid field for identification. The service will apply updates to the following search fields: name, query, and scope.

Parameters

search	The search object containing the requested changes.

Returns

The Update Sequence Number for this change within the account.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "SavedSearch.name" - invalid length or pattern • BAD_DATA_FORMAT "SavedSearch.query" - invalid length • DATA_CONFLICT "SavedSearch.name" - name already in use • PERMISSION_DENIED "SavedSearch" - user doesn't own tag
EDAMNotFoundException	• "SavedSearch.guid" - not found, by GUID

7.41.3.146 updateSearchAsync()

```
virtual AsyncResult * qevercloud::INoteStore::updateSearchAsync (
            const SavedSearch & search,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of updateSearch

7.41.3.147 updateSharedNotebook()

```
virtual qint32 qevercloud::INoteStore::updateSharedNotebook (
            const SharedNotebook & sharedNotebook,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

@Deprecated See createOrUpdateNotebookShares and manageNotebookShares.

7.41.3.148 updateSharedNotebookAsync()

```
virtual AsyncResult * qevercloud::INoteStore::updateSharedNotebookAsync (
            const SharedNotebook & sharedNotebook,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of updateSharedNotebook

7.41.3.149 updateTag()

```
virtual qint32 qevercloud::INoteStore::updateTag (
            const Tag & tag,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Submits tag changes to the service. The provided data must include the tag's guid field for identification. The service will apply updates to the following tag fields: name, parentGuid

Parameters

tag	The tag object containing the requested changes.

Returns

The Update Sequence Number for this change within the account.

Exceptions

EDAMUserException	• BAD_DATA_FORMAT "Tag.name" - invalid length or pattern • BAD_DATA_FORMAT "Tag.parentGuid" - malformed GUID • DATA_CONFLICT "Tag.name" - name already in use • DATA_CONFLICT "Tag.parentGuid" - can't set parent: circular • PERMISSION_DENIED "Tag" - user doesn't own tag
EDAMNotFoundException	• "Tag.guid" - tag not found, by GUID • "Tag.parentGuid" - parent not found, by GUID

7.41.3.150 updateTagAsync()

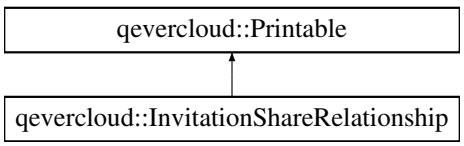
```
virtual AsyncResult * qevercloud::INoteStore::updateTagAsync (
            const Tag & tag,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of updateTag

7.42 qevercloud::InvitationShareRelationship Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::InvitationShareRelationship:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const InvitationShareRelationship &other) const
- bool operator!= (const InvitationShareRelationship &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > displayName
- Optional< UserIdentity > recipientUserIdentity
- Optional< ShareRelationshipPrivilegeLevel > privilege
- Optional< UserID > sharerUserId

7.42.1 Detailed Description

Describes an invitation to a person to use their Evernote credentials to become a member of a notebook.

7.42.2 Member Function Documentation

7.42.2.1 operator"!=()

```
bool qevercloud::InvitationShareRelationship::operator!= (
            const InvitationShareRelationship & other ) const  [inline]
```

7.42.2.2 operator==()

```
bool qevercloud::InvitationShareRelationship::operator== (
            const InvitationShareRelationship & other ) const  [inline]
```

7.42.2.3 print()

```
virtual void qevercloud::InvitationShareRelationship::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.42.3 Member Data Documentation

7.42.3.1 displayName

```
Optional< QString > qevercloud::InvitationShareRelationship::displayName
```

The string that clients should show to users to represent this invitation.

7.42.3.2 localData

```
EverCloudLocalData qevercloud::InvitationShareRelationship::localData
```

See the declaration of EverCloudLocalData for details

7.42.3.3 privilege

```
Optional< ShareRelationshipPrivilegeLevel > qevercloud::InvitationShareRelationship::privilege
```

The privilege level at which the member will be joined, if it turns out that the member is not already joined at a higher level. Note that the `identity` field may not uniquely identify an Evernote User ID, and so we won't know until the invitation is redeemed whether or not the recipient already has privilege.

7.42.3.4 recipientUserIdentity

```
Optional< UserIdentity > qevercloud::InvitationShareRelationship::recipientUserIdentity
```

Identifies the recipient of the invitation. The user identity type can be either EMAIL, EVERNOTE or IDENTITYID. If the invitation was created using the classic notebook sharing APIs it will be EMAIL. If it was created using the new identity-based notebook sharing APIs it will either be EVERNOTE or IDENTITYID, depending on whether we can map the identity to an Evernote user at the time of creation.

7.42.3.5 sharerUserId

```
Optional< UserID > qevercloud::InvitationShareRelationship::sharerUserId
```

The user id of the user who most recently shared this notebook to this identity. This field is used by the service to convey information to the user, so clients should treat it as read-only.

7.43 qevercloud::IRequestContext Class Reference

```
#include <RequestContext.h>
```

Public Member Functions

- virtual QUuid requestId () const =0
- virtual QString authenticationToken () const =0
- virtual qint64 requestTimeout () const =0
- virtual bool increaseRequestTimeoutExponentially () const =0
- virtual qint64 maxRequestTimeout () const =0
- virtual quint32 maxRequestRetryCount () const =0
- virtual QList< QNetworkCookie > cookies () const =0
- virtual IRequestContext * clone () const =0
- virtual ~IRequestContext ()=default

Friends

- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &strm, const IRequestContext &ctx)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &dbg, const IRequestContext &ctx)

7.43.1 Detailed Description

IRequestContext carries several request scoped values defining the way request is handled by QEverCloud

7.43.2 Constructor & Destructor Documentation

7.43.2.1 ~IRequestContext()

```
virtual qevercloud::IRequestContext::~IRequestContext ( )  [virtual], [default]
```

7.43.3 Member Function Documentation

7.43.3.1 authenticationToken()

```
virtual QString qevercloud::IRequestContext::authenticationToken ( ) const  [pure virtual]
```

Authentication token to use along with the request

7.43.3.2 clone()

```
virtual IRequestContext * qevercloud::IRequestContext::clone ( ) const  [pure virtual]
```

Create a new instance of IRequestContext with all the same parameters as in the source but a distinct id

7.43.3.3 cookies()

```
virtual QList< QNetworkCookie > qevercloud::IRequestContext::cookies ( ) const  [pure virtual]
```

Cookies to set to QNetworkRequest corresponding to Evernote API call

7.43.3.4 increaseRequestTimeoutExponentially()

```
virtual bool qevercloud::IRequestContext::increaseRequestTimeoutExponentially ( ) const  [pure
virtual]
```

Should request timeout be exponentially increased on retries or not

7.43.3.5 maxRequestRetryCount()

```
virtual quint32 qevercloud::IRequestContext::maxRequestRetryCount ( ) const  [pure virtual]
```

Max number of attempts to retry a request

7.43.3.6 maxRequestTimeout()

```
virtual qint64 qevercloud::IRequestContext::maxRequestTimeout ( ) const  [pure virtual]
```

Max request timeout in milliseconds (upper boundary for exponentially increasing timeouts on retries)

7.43.3.7 requestId()

```
virtual QUuid qevercloud::IRequestContext::requestId ( ) const  [pure virtual]
```

Automatically generated unique identifier for each request

7.43.3.8 requestTimeout()

```
virtual qint64 qevercloud::IRequestContext::requestTimeout ( ) const  [pure virtual]
```

Request timeout in milliseconds

7.43.4 Friends And Related Symbol Documentation

7.43.4.1 operator<< [1/2]

```
QEVERCLOUD_EXPORT QDebug & operator<< (
            QDebug & dbg,
            const IRequestContext & ctx )  [friend]
```

7.43.4.2 operator<< [2/2]

```
QEVERCLOUD_EXPORT QTextStream & operator<< (
            QTextStream & strm,
            const IRequestContext & ctx )  [friend]
```

7.44 qevercloud::IRetryPolicy Struct Reference

```
#include <DurableService.h>
```

Public Member Functions

- virtual bool shouldRetry (const EverCloudExceptionDataPtr &exceptionData)=0

7.44.1 Member Function Documentation

7.44.1.1 shouldRetry()

```
virtual bool qevercloud::IRetryPolicy::shouldRetry (
            const EverCloudExceptionDataPtr & exceptionData )  [pure virtual]
```

7.45 qevercloud::QAssociativeContainerConstReferenceWrapper< Container >::iterator Struct Reference

```
#include <Helpers.h>
```

Public Member Functions

- iterator (const typename Container::const_iterator it)
- Container::const_iterator operator* ()
- iterator & operator++ ()
- bool operator!= (const iterator &other) const

Public Attributes

- Container::const_iterator m_iterator

7.45.1 Constructor & Destructor Documentation

7.45.1.1 iterator()

```
template<typename Container >
qevercloud::QAssociativeContainerConstReferenceWrapper< Container >::iterator::iterator (
            const typename Container::const_iterator it )  [inline]
```

7.45.2 Member Function Documentation

7.45.2.1 operator"!=()

```
template<typename Container >
bool qevercloud::QAssociativeContainerConstReferenceWrapper< Container >::iterator::operator!=
(
            const iterator & other ) const  [inline]
```

7.45.2.2 operator*()

```
template<typename Container >
Container::const_iterator qevercloud::QAssociativeContainerConstReferenceWrapper< Container
>::iterator::operator* ( )  [inline]
```

7.45.2.3 operator++()

```
template<typename Container >
iterator & qevercloud::QAssociativeContainerConstReferenceWrapper< Container >::iterator↩
::operator++ ( )  [inline]
```

7.45.3 Member Data Documentation

7.45.3.1 m_iterator

```
template<typename Container >
Container::const_iterator qevercloud::QAssociativeContainerConstReferenceWrapper< Container
>::iterator::m_iterator
```

7.46 qevercloud::QAssociativeContainerReferenceWrapper< Container >::iterator Struct Reference

```
#include <Helpers.h>
```

Public Member Functions

- iterator (const typename Container::iterator it)
- Container::iterator operator* ()
- iterator & operator++ ()
- bool operator!= (const iterator &other) const

Public Attributes

- Container::iterator m_iterator

7.46.1 Constructor & Destructor Documentation

7.46.1.1 iterator()

```
template<typename Container >
qevercloud::QAssociativeContainerReferenceWrapper< Container >::iterator::iterator (
            const typename Container::iterator it )  [inline]
```

7.46.2 Member Function Documentation

7.46.2.1 operator"!=()

```
template<typename Container >
bool qevercloud::QAssociativeContainerReferenceWrapper< Container >::iterator::operator!= (
            const iterator & other ) const  [inline]
```

7.46.2.2 operator*()

```
template<typename Container >
Container::iterator qevercloud::QAssociativeContainerReferenceWrapper< Container >::iterator↩
::operator* ( )  [inline]
```

7.46.2.3 operator++()

```
template<typename Container >
iterator & qevercloud::QAssociativeContainerReferenceWrapper< Container >::iterator::operator++
( )  [inline]
```

7.46.3 Member Data Documentation

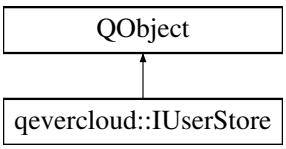
7.46.3.1 m_iterator

```
template<typename Container >
Container::iterator qevercloud::QAssociativeContainerReferenceWrapper< Container >::iterator↩
::m_iterator
```

7.47 qevercloud::IUserStore Class Reference

`#include <Services.h>`

Inheritance diagram for qevercloud::IUserStore:



Public Member Functions

- virtual QString userStoreUrl () const =0
- virtual void setUserStoreUrl (QString url)=0
- virtual bool checkVersion (QString clientName, qint16 edamVersionMajor=EDAM_VERSION_MAJOR, qint16 edamVersionMinor=EDAM_VERSION_MINOR, IRequestContextPtr ctx={})=0
- virtual AsyncResult * checkVersionAsync (QString clientName, qint16 edamVersionMajor=EDAM_VERSION_MAJOR, qint16 edamVersionMinor=EDAM_VERSION_MINOR, IRequestContextPtr ctx={})=0
- virtual BootstrapInfo getBootstrapInfo (QString locale, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getBootstrapInfoAsync (QString locale, IRequestContextPtr ctx={})=0
- virtual AuthenticationResult authenticateLongSession (QString username, QString password, QString consumerKey, QString consumerSecret, QString deviceIdentifier, QString deviceDescription, bool supportsTwoFactor, IRequestContextPtr ctx={})=0
- virtual AsyncResult * authenticateLongSessionAsync (QString username, QString password, QString consumerKey, QString consumerSecret, QString deviceIdentifier, QString deviceDescription, bool supportsTwoFactor, IRequestContextPtr ctx={})=0

- virtual AuthenticationResult completeTwoFactorAuthentication (QString oneTimeCode, QString deviceIdentifier, QString deviceDescription, IRequestContextPtr ctx={})=0
- virtual AsyncResult * completeTwoFactorAuthenticationAsync (QString oneTimeCode, QString deviceIdentifier, QString deviceDescription, IRequestContextPtr ctx={})=0
- virtual void revokeLongSession (IRequestContextPtr ctx={})=0
- virtual AsyncResult * revokeLongSessionAsync (IRequestContextPtr ctx={})=0
- virtual AuthenticationResult authenticateToBusiness (IRequestContextPtr ctx={})=0
- virtual AsyncResult * authenticateToBusinessAsync (IRequestContextPtr ctx={})=0
- virtual User getUser (IRequestContextPtr ctx={})=0
- virtual AsyncResult * getUserAsync (IRequestContextPtr ctx={})=0
- virtual PublicUserInfo getPublicUserInfo (QString username, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getPublicUserInfoAsync (QString username, IRequestContextPtr ctx={})=0
- virtual UserUrls getUserUrls (IRequestContextPtr ctx={})=0
- virtual AsyncResult * getUserUrlsAsync (IRequestContextPtr ctx={})=0
- virtual void inviteToBusiness (QString emailAddress, IRequestContextPtr ctx={})=0
- virtual AsyncResult * inviteToBusinessAsync (QString emailAddress, IRequestContextPtr ctx={})=0
- virtual void removeFromBusiness (QString emailAddress, IRequestContextPtr ctx={})=0
- virtual AsyncResult * removeFromBusinessAsync (QString emailAddress, IRequestContextPtr ctx={})=0
- virtual void updateBusinessUserIdentifier (QString oldEmailAddress, QString newEmailAddress, IRequestContextPtr ctx={})=0
- virtual AsyncResult * updateBusinessUserIdentifierAsync (QString oldEmailAddress, QString newEmailAddress, IRequestContextPtr ctx={})=0
- virtual QList< UserProfile > listBusinessUsers (IRequestContextPtr ctx={})=0
- virtual AsyncResult * listBusinessUsersAsync (IRequestContextPtr ctx={})=0
- virtual QList< BusinessInvitation > listBusinessInvitations (bool includeRequestedInvitations, IRequestContextPtr ctx={})=0
- virtual AsyncResult * listBusinessInvitationsAsync (bool includeRequestedInvitations, IRequestContextPtr ctx={})=0
- virtual AccountLimits getAccountLimits (ServiceLevel serviceLevel, IRequestContextPtr ctx={})=0
- virtual AsyncResult * getAccountLimitsAsync (ServiceLevel serviceLevel, IRequestContextPtr ctx={})=0

Protected Member Functions

- IUserStore (QObject *parent)

7.47.1 Detailed Description

Service: UserStore

The UserStore service is primarily used by EDAM clients to establish authentication via username and password over a trusted connection (e.g. SSL). A client's first call to this interface should be checkVersion() to ensure that the client's software is up to date.

All calls which require an authenticationToken may throw an EDAMUserException for the following reasons:

- AUTH_EXPIRED "authenticationToken" - token has expired
- BAD_DATA_FORMAT "authenticationToken" - token is malformed
- DATA_REQUIRED "authenticationToken" - token is empty
- INVALID_AUTH "authenticationToken" - token signature is invalid
- PERMISSION_DENIED "authenticationToken" - token does not convey sufficient privileges

7.47.2 Constructor & Destructor Documentation

7.47.2.1 IUserStore()

```
qevercloud::IUserStore::IUserStore (
            QObject * parent )  [inline], [protected]
```

7.47.3 Member Function Documentation

7.47.3.1 authenticateLongSession()

```
virtual AuthenticationResult qevercloud::IUserStore::authenticateLongSession (
            QString username,
            QString password,
            QString consumerKey,
            QString consumerSecret,
            QString deviceIdentifier,
            QString deviceDescription,
            bool supportsTwoFactor,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

This is used to check a username and password in order to create a long-lived authentication token that can be used for further actions.

This function is not available to most third party applications, which typically authenticate using OAuth as described at dev.evernote.com. If you believe that your application requires permission to authenticate using username and password instead of OAuth, please contact Evernote developer support by visiting dev.evernote.com.

Parameters

username	The username or registered email address of the account to authenticate against.
password	The plaintext password to check against the account. Since this is not protected by the EDAM protocol, this information must be provided over a protected transport (i.e. SSL).
consumerKey	The "consumer key" portion of the API key issued to the client application by Evernote.
consumerSecret	The "consumer secret" portion of the API key issued to the client application by Evernote.
deviceIdentifier	An optional string that uniquely identifies the device from which the authentication is being performed. This string allows the service to return the same authentication token when a given application requests authentication repeatedly from the same device. This may happen when the user logs out of an application and then logs back in, or when the application is uninstalled and later reinstalled. If no reliable device identifier can be created, this value should be omitted. If set, the device identifier must be between 1 and EDAM_DEVICE_ID_LEN_MAX characters long and must match the regular expression EDAM_DEVICE_ID_REGEX.
deviceDescription	A description of the device from which the authentication is being performed. This field is displayed to the user in a list of authorized applications to allow them to distinguish between multiple tokens issued to the same client application on different devices. For example, the Evernote iOS client on a user's iPhone and iPad might pass the iOS device names "Bob's iPhone" and "Bob's iPad". The device description must be between 1 and EDAM_DEVICE_DESCRIPTION_LEN_MAX characters long and must match the regular expression EDAM_DEVICE_DESCRIPTION_REGEX.
supportsTwoFactor	Whether the calling application supports two-factor authentication. If this parameter is false, this method will fail with the error code INVALID_AUTH and the parameter "password" when called for a user who has enabled two-factor authentication.

Returns

The result of the authentication. The level of detail provided in the returned AuthenticationResult.User structure depends on the access level granted by calling application's API key.

If the user has two-factor authentication enabled, AuthenticationResult.secondFactorRequired will be set and AuthenticationResult.authenticationToken will contain a short-lived token that may only be used to complete the two-factor authentication process by calling UserStore.completeTwoFactorAuthentication.

Exceptions

EDAMUserException	• DATA_REQUIRED "username" - username is empty • DATA_REQUIRED "password" - password is empty • DATA_REQUIRED "consumerKey" - consumerKey is empty • DATA_REQUIRED "consumerSecret" - consumerSecret is empty • DATA_REQUIRED "deviceDescription" - deviceDescription is empty • BAD_DATA_FORMAT "deviceDescription" - deviceDescription is not valid. • BAD_DATA_FORMAT "deviceIdentifier" - deviceIdentifier is not valid. • INVALID_AUTH "username" - username not found • INVALID_AUTH "password" - password did not match • INVALID_AUTH "consumerKey" - consumerKey is not authorized • INVALID_AUTH "consumerSecret" - consumerSecret is incorrect • INVALID_AUTH "businessOnly" - the user is a business-only account • PERMISSION_DENIED "User.active" - user account is closed • PERMISSION_DENIED "User.tooManyFailuresTryAgainLater" - user has failed authentication too often • AUTH_EXPIRED "password" - user password is expired

7.47.3.2 authenticateLongSessionAsync()

```
virtual AsyncResult * qevercloud::IUserStore::authenticateLongSessionAsync (
            QString username,
            QString password,
            QString consumerKey,
            QString consumerSecret,
            QString deviceIdentifier,
            QString deviceDescription,
            bool supportsTwoFactor,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of authenticateLongSession

7.47.3.3 authenticateToBusiness()

```
virtual AuthenticationResult qevercloud::IUserStore::authenticateToBusiness (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

This is used to take an existing authentication token that grants access to an individual user account (returned from 'authenticate', 'authenticateLongSession' or an OAuth authorization) and obtain an additional authentication token that may be used to access business notebooks if the user is a member of an Evernote Business account.

The resulting authentication token may be used to make NoteStore API calls against the business using the NoteStore URL returned in the result.

Parameters

authenticationToken	The authentication token for the user. This may not be a shared authentication token (returned by NoteStore.authenticateToSharedNotebook or NoteStore.authenticateToSharedNote) or a business authentication token.

Returns

The result of the authentication, with the token granting access to the business in the result's 'authenticationToken' field. The URL that must be used to access the business account NoteStore will be returned in the result's 'noteStoreUrl' field. The 'User' field will not be set in the result.

Exceptions

EDAMUserException	• PERMISSION_DENIED "authenticationToken" - the provided authentication token is a shared or business authentication token. • PERMISSION_DENIED "Business" - the user identified by the provided authentication token is not currently a member of a business. • PERMISSION_DENIED "Business.status" - the business that the user is a member of is not currently in an active status. • BUSINESS_SECURITY_LOGIN_REQUIRED "sso" - the user must complete single sign-on before authenticating to the business.

7.47.3.4 authenticateToBusinessAsync()

```
virtual AsyncResult * qevercloud::IUserStore::authenticateToBusinessAsync (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of authenticateToBusiness

7.47.3.5 checkVersion()

```
virtual bool qevercloud::IUserStore::checkVersion (
            QString clientName,
            qint16 edamVersionMajor = EDAM_VERSION_MAJOR,
```

```
            qint16 edamVersionMinor = EDAM_VERSION_MINOR,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

This should be the first call made by a client to the EDAM service. It tells the service what protocol version is used by the client. The service will then return true if the client is capable of talking to the service, and false if the client's protocol version is incompatible with the service, so the client must upgrade. If a client receives a false value, it should report the incompatibility to the user and not continue with any more EDAM requests (UserStore or NoteStore).

Parameters

clientName	This string provides some information about the client for tracking/logging on the service. It should provide information about the client's software and platform. The structure should be: application/version; platform/version; [device/version] E.g. "Evernote Windows/3.0.1; Windows/XP SP3".
edamVersionMajor	This should be the major protocol version that was compiled by the client. This should be the current value of the EDAM_VERSION_MAJOR constant for the client.
edamVersionMinor	This should be the major protocol version that was compiled by the client. This should be the current value of the EDAM_VERSION_MINOR constant for the client.

7.47.3.6 checkVersionAsync()

```
virtual AsyncResult * qevercloud::IUserStore::checkVersionAsync (
            QString clientName,
            qint16 edamVersionMajor = EDAM_VERSION_MAJOR,
            qint16 edamVersionMinor = EDAM_VERSION_MINOR,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of checkVersion

7.47.3.7 completeTwoFactorAuthentication()

```
virtual AuthenticationResult qevercloud::IUserStore::completeTwoFactorAuthentication (
            QString oneTimeCode,
            QString deviceIdentifier,
            QString deviceDescription,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Complete the authentication process when a second factor is required. This call is made after a successful call to authenticate or authenticateLongSession when the authenticating user has enabled two-factor authentication.

Parameters

authenticationToken	An authentication token returned by a previous call to UserStore.authenticate or UserStore.authenticateLongSession that could not be completed in a single call because a second factor was required.
oneTimeCode	The one time code entered by the user. This value is delivered out-of-band, typically via SMS or an authenticator application.
deviceIdentifier	See the corresponding parameter in authenticateLongSession.
deviceDescription	See the corresponding parameter in authenticateLongSession.

Returns

The result of the authentication. The level of detail provided in the returned AuthenticationResult.User structure depends on the access level granted by the calling application's API key. If the initial authentication call was made to authenticateLongSession, the AuthenticationResult will contain a long-lived authentication token.

Exceptions

EDAMUserException	• DATA_REQUIRED "authenticationToken" - authenticationToken is empty • DATA_REQUIRED "oneTimeCode" - oneTimeCode is empty • BAD_DATA_FORMAT "deviceIdentifier" - deviceIdentifier is not valid • BAD_DATA_FORMAT "authenticationToken" - authenticationToken is not well formed • INVALID_AUTH "oneTimeCode" - oneTimeCode did not match • AUTH_EXPIRED "authenticationToken" - authenticationToken has expired • PERMISSION_DENIED "authenticationToken" - authenticationToken is not valid • PERMISSION_DENIED "User.active" - user account is closed • PERMISSION_DENIED "User.tooManyFailuresTryAgainLater" - user has failed authentication too often • DATA_CONFLICT "User.twoFactorAuthentication" - The user has not enabled two-factor authentication.

7.47.3.8 completeTwoFactorAuthenticationAsync()

```
virtual AsyncResult * qevercloud::IUserStore::completeTwoFactorAuthenticationAsync (
            QString oneTimeCode,
            QString deviceIdentifier,
            QString deviceDescription,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of completeTwoFactorAuthentication

7.47.3.9 getAccountLimits()

```
virtual AccountLimits qevercloud::IUserStore::getAccountLimits (
            ServiceLevel serviceLevel,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Retrieve the standard account limits for a given service level. This should only be called when necessary, e.g. to determine if a higher level is available should the user upgrade, and should be cached for long periods (e.g. 30 days) as the values are not expected to fluctuate frequently.

Exceptions

EDAMUserException	• DATA_REQUIRED "serviceLevel" - serviceLevel is null

7.47.3.10 getAccountLimitsAsync()

```
virtual AsyncResult * qevercloud::IUserStore::getAccountLimitsAsync (
            ServiceLevel serviceLevel,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getAccountLimits

7.47.3.11 getBootstrapInfo()

```
virtual BootstrapInfo qevercloud::IUserStore::getBootstrapInfo (
            QString locale,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

This provides bootstrap information to the client. Various bootstrap profiles and settings may be used by the client to configure itself.

Parameters

locale	The client's current locale, expressed in language[_country] format. E.g., "en_US". See ISO-639 and ISO-3166 for valid language and country codes.

Returns

The bootstrap information suitable for this client.

7.47.3.12 getBootstrapInfoAsync()

```
virtual AsyncResult * qevercloud::IUserStore::getBootstrapInfoAsync (
            QString locale,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getBootstrapInfo

7.47.3.13 getPublicUserInfo()

```
virtual PublicUserInfo qevercloud::IUserStore::getPublicUserInfo (
            QString username,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asks the UserStore about the publicly available location information for a particular username.

Exceptions

EDAMUserException	• DATA_REQUIRED "username" - username is empty

7.47.3.14 getPublicUserInfoAsync()

```
virtual AsyncResult * qevercloud::IUserStore::getPublicUserInfoAsync (
            QString username,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getPublicUserInfo

7.47.3.15 getUser()

```
virtual User qevercloud::IUserStore::getUser (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns the User corresponding to the provided authentication token, or throws an exception if this token is not valid. The level of detail provided in the returned User structure depends on the access level granted by the token, so a web service client may receive fewer fields than an integrated desktop client.

7.47.3.16 getUserAsync()

```
virtual AsyncResult * qevercloud::IUserStore::getUserAsync (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getUser

7.47.3.17 getUserUrls()

```
virtual UserUrls qevercloud::IUserStore::getUserUrls (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns the URLs that should be used when sending requests to the service on behalf of the account represented by the provided authenticationToken.

This method isn't needed by most clients, who can retrieve the correct set of UserUrls from the AuthenticationResult returned from UserStore::authenticateLongSession(). This method is typically only needed to look up the correct URLs for an existing long-lived authentication token.

7.47.3.18 getUserUrlsAsync()

```
virtual AsyncResult * qevercloud::IUserStore::getUserUrlsAsync (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of getUserUrls

7.47.3.19 inviteToBusiness()

```
virtual void qevercloud::IUserStore::inviteToBusiness (
            QString emailAddress,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Invite a user to join an Evernote Business account.

Behavior will depend on the auth token.

1. auth token with privileges to manage Evernote Business membership. "External Provisioning" - The user will receive an email inviting them to join the business. They do not need to have an existing Evernote account. If the user has already been invited, a new invitation email will be sent.
2. business auth token issued to an admin user. Only for first-party clients: "Approve Invitation" - If there has been a request to invite the email, approve it. Invited user will receive email with a link to join business. "Invite User" - If no invitation for the email exists, create an approved invitation for the email. An email will be sent to the emailAddress with a link to join the caller's business. business auth token: "Request Invitation" - If no invitation exists, create a request to invite the user to the business. These requests do not count towards a business' max active user limit.

Parameters

authenticationToken	the authentication token with sufficient privileges to manage Evernote Business membership or a business auth token.
emailAddress	the email address of the user to invite to join the Evernote Business account.

Exceptions

EDAMUserException	• DATA_REQUIRED "email" - if no email address was provided • BAD_DATA_FORMAT "email" - if the email address is not well formed • DATA_CONFLICT "BusinessUser.email" - if there is already a user in the business whose business email address matches the specified email address. • LIMIT_REACHED "Business.maxActiveUsers" - if the business has reached its user limit.

7.47.3.20 inviteToBusinessAsync()

```
virtual AsyncResult * qevercloud::IUserStore::inviteToBusinessAsync (
            QString emailAddress,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of inviteToBusiness

7.47.3.21 listBusinessInvitations()

```
virtual QList< BusinessInvitation > qevercloud::IUserStore::listBusinessInvitations (
            bool includeRequestedInvitations,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns a list of outstanding invitations to join an Evernote Business account.

Only outstanding invitations are returned by this function. Users who have accepted an invitation and joined a business are listed using listBusinessUsers.

Parameters

authenticationToken	An authentication token with sufficient privileges to manage Evernote Business membership.
includeRequestedInvitations	If true, invitations with a status of BusinessInvitationStatus.REQUESTED will be included in the returned list. If false, only invitations with a status of BusinessInvitationStatus.APPROVED will be included.

7.47.3.22 listBusinessInvitationsAsync()

```
virtual AsyncResult * qevercloud::IUserStore::listBusinessInvitationsAsync (
            bool includeRequestedInvitations,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of listBusinessInvitations

7.47.3.23 listBusinessUsers()

```
virtual QList< UserProfile > qevercloud::IUserStore::listBusinessUsers (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Returns a list of active business users in a given business.

Clients are required to cache this information and re-fetch no more than once per day or when they encountered a user ID or username that was not known to them.

To avoid excessive look ups, clients should also track user IDs and usernames that belong to users who are not in the business, since they will not be included in the result.

I.e., when a client encounters a previously unknown user ID as a note's creator, it may query listBusinessUsers to find information about this user. If the user is not in the resulting list, the client should track that fact and not re-query the service the next time that it sees this user on a note.

Parameters

authenticationToken	A business authentication token returned by authenticateToBusiness or with sufficient privileges to manage Evernote Business membership.

7.47.3.24 listBusinessUsersAsync()

```
virtual AsyncResult * qevercloud::IUserStore::listBusinessUsersAsync (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of listBusinessUsers

7.47.3.25 removeFromBusiness()

```
virtual void qevercloud::IUserStore::removeFromBusiness (
            QString emailAddress,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Remove a user from an Evernote Business account. Once removed, the user will no longer be able to access content within the Evernote Business account.

The email address of the user to remove from the business must match the email address used to invite a user to join the business via UserStore.inviteToBusiness. This function will only remove users who were invited by external provisioning

Parameters

authenticationToken	An authentication token with sufficient privileges to manage Evernote Business membership.
emailAddress	The email address of the user to remove from the Evernote Business account.

Exceptions

EDAMUserException	• DATA_REQUIRED "email" - if no email address was provided • BAD_DATA_FORMAT "email" - The email address is not well formed
EDAMNotFoundException	• "email" - If there is no user with the specified email address in the business or that user was not invited via external provisioning.

7.47.3.26 removeFromBusinessAsync()

```
virtual AsyncResult * qevercloud::IUserStore::removeFromBusinessAsync (
            QString emailAddress,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of removeFromBusiness

7.47.3.27 revokeLongSession()

```
virtual void qevercloud::IUserStore::revokeLongSession (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Revoke an existing long lived authentication token. This can be used to revoke OAuth tokens or tokens created by calling authenticateLongSession, and allows a user to effectively log out of Evernote from the perspective of the application that holds the token. The authentication token that is passed is immediately revoked and may not be used to call any authenticated EDAM function.

Parameters

authenticationToken	the authentication token to revoke.

Exceptions

EDAMUserException	• DATA_REQUIRED "authenticationToken" - no authentication token provided • BAD_DATA_FORMAT "authenticationToken" - the authentication token is not well formed • INVALID_AUTH "authenticationToken" - the authentication token is invalid • AUTH_EXPIRED "authenticationToken" - the authentication token is expired or is already revoked.

7.47.3.28 revokeLongSessionAsync()

```
virtual AsyncResult * qevercloud::IUserStore::revokeLongSessionAsync (
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of revokeLongSession

7.47.3.29 setUserStoreUrl()

```
virtual void qevercloud::IUserStore::setUserStoreUrl (
            QString url )  [pure virtual]
```

7.47.3.30 updateBusinessUserIdentifier()

```
virtual void qevercloud::IUserStore::updateBusinessUserIdentifier (
            QString oldEmailAddress,
            QString newEmailAddress,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Update the email address used to uniquely identify an Evernote Business user.

This will update the identifier for a user who was previously invited using inviteToBusiness, ensuring that caller and the Evernote service maintain an agreed-upon identifier for a specific user.

For example, the following sequence of calls would invite a user to join a business, update their email address, and then remove the user from the business using the updated email address.

inviteToBusiness("foo@bar.com") updateBusinessUserIdentifier("foo@bar.com", "baz@bar.com") removeFrom↩ Business("baz@bar.com")

Parameters

authenticationToken	An authentication token with sufficient privileges to manage Evernote Business membership.
oldEmailAddress	The existing email address used to uniquely identify the user.
newEmailAddress	The new email address used to uniquely identify the user.

Exceptions

EDAMUserException	• DATA_REQUIRED "oldEmailAddress" - No old email address was provided • DATA_REQUIRED "newEmailAddress" - No new email address was provided • BAD_DATA_FORMAT "oldEmailAddress" - The old email address is not well formed • BAD_DATA_FORMAT "newEmailAddress" - The new email address is not well formed • DATA_CONFLICT "oldEmailAddress" - The old and new email addresses were the same • DATA_CONFLICT "newEmailAddress" - There is already an invitation or registered user with the provided new email address. • DATA_CONFLICT "invitation.externallyProvisioned" - The user identified by oldEmailAddress was not added via UserStore.inviteToBusiness and therefore cannot be updated.
EDAMNotFoundException	• "oldEmailAddress" - If there is no user or invitation with the specified oldEmailAddress in the business.

7.47.3.31 updateBusinessUserIdentifierAsync()

```
virtual AsyncResult * qevercloud::IUserStore::updateBusinessUserIdentifierAsync (
            QString oldEmailAddress,
            QString newEmailAddress,
            IRequestContextPtr ctx = {} )  [pure virtual]
```

Asynchronous version of updateBusinessUserIdentifier

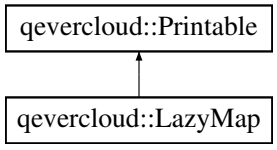
7.47.3.32 userStoreUrl()

```
virtual QString qevercloud::IUserStore::userStoreUrl ( ) const  [pure virtual]
```

7.48 qevercloud::LazyMap Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::LazyMap:



Public Types

- using FullMap = QMap< QString, QString >

Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const LazyMap &other) const
- bool operator!= (const LazyMap &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QSet< QString > > keysOnly
- Optional< QMap< QString, QString > > fullMap

Properties

- OptionalQSet< QString > keysOnly
- Optional< FullMap > fullMap

7.48.1 Detailed Description

A structure that wraps a map of name/value pairs whose values are not always present in the structure in order to reduce space when obtaining batches of entities that contain the map.

When the server provides the client with a LazyMap, it will fill in either the keysOnly field or the fullMap field, but never both, based on the API and parameters.

When a client provides a LazyMap to the server as part of an update to an object, the server will only update the LazyMap if the fullMap field is set. If the fullMap field is not set, the server will not make any changes to the map.

Check the API documentation of the individual calls involving the LazyMap for full details including the constraints of the names and values of the map.

7.48.2 Member Typedef Documentation

7.48.2.1 FullMap

```
using qevercloud::LazyMap::FullMap = QMap<QString, QString>
```

7.48.3 Member Function Documentation

7.48.3.1 operator"!=()

```
bool qevercloud::LazyMap::operator!= (
            const LazyMap & other ) const  [inline]
```

7.48.3.2 operator==()

```
bool qevercloud::LazyMap::operator== (
            const LazyMap & other ) const  [inline]
```

7.48.3.3 print()

```
virtual void qevercloud::LazyMap::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.48.4 Member Data Documentation

7.48.4.1 fullMap

```
Optional<QMap<QString, QString> > qevercloud::LazyMap::fullMap
```

The complete map, including all keys and values.

7.48.4.2 keysOnly

```
Optional<QSet<QString> > qevercloud::LazyMap::keysOnly
```

The set of keys for the map. This field is ignored by the server when set.

7.48.4.3 localData

```
EverCloudLocalData qevercloud::LazyMap::localData
```

See the declaration of EverCloudLocalData for details

7.48.5 Property Documentation

7.48.5.1 fullMap

```
Optional<FullMap> qevercloud::LazyMap::fullMap
```

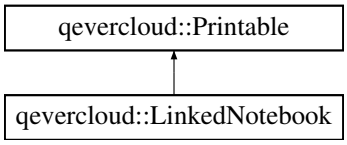
7.48.5.2 keysOnly

```
OptionalQSet<QString> qevercloud::LazyMap::keysOnly
```

7.49 qevercloud::LinkedNotebook Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::LinkedNotebook:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const LinkedNotebook &other) const
- bool operator!= (const LinkedNotebook &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > shareName
- Optional< QString > username
- Optional< QString > shardId
- Optional< QString > sharedNotebookGlobalId
- Optional< QString > uri
- Optional< Guid > guid
- Optional< qint32 > updateSequenceNum
- Optional< QString > noteStoreUrl
- Optional< QString > webApiUrlPrefix
- Optional< QString > stack
- Optional< qint32 > businessId

7.49.1 Detailed Description

A link in a user's account that refers them to a public or individual shared notebook in another user's account.

7.49.2 Member Function Documentation

7.49.2.1 operator"!=()

```
bool qevercloud::LinkedNotebook::operator!= (
            const LinkedNotebook & other ) const  [inline]
```

7.49.2.2 operator==()

```
bool qevercloud::LinkedNotebook::operator== (
            const LinkedNotebook & other ) const  [inline]
```

7.49.2.3 print()

```
virtual void qevercloud::LinkedNotebook::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.49.3 Member Data Documentation

7.49.3.1 businessId

```
Optional< qint32 > qevercloud::LinkedNotebook::businessId
```

If set, this will be the unique identifier for the business that owns the notebook to which the linked notebook refers.

7.49.3.2 guid

```
Optional< Guid > qevercloud::LinkedNotebook::guid
```

The unique identifier of this linked notebook. Will be set whenever a linked notebook is retrieved from the service, but may be null when a client is creating a linked notebook.
Length: EDAM_GUID_LEN_MIN - EDAM_GUID_LEN_MAX
Regex: EDAM_GUID_REGEX

7.49.3.3 localData

```
EverCloudLocalData qevercloud::LinkedNotebook::localData
```

See the declaration of EverCloudLocalData for details

7.49.3.4 noteStoreUrl

```
Optional< QString > qevercloud::LinkedNotebook::noteStoreUrl
```

This field will contain the full URL that clients should use to make NoteStore requests to the server shard that contains that notebook's data. I.e. this is the URL that should be used to create the Thrift HTTP client transport to send messages to the NoteStore service for the account.

7.49.3.5 shardId

```
Optional< QString > qevercloud::LinkedNotebook::shardId
```

The shard ID of the notebook if the notebook is not public.

uri The identifier of the public notebook.

7.49.3.6 sharedNotebookGlobalId

```
Optional< QString > qevercloud::LinkedNotebook::sharedNotebookGlobalId
```

The globally unique identifier (globalId) of the shared notebook that corresponds to the share key, or the GUID of the Notebook that the linked notebook refers to. This field must be filled in with the SharedNotebook.globalId or Notebook.GUID value when creating new LinkedNotebooks. This field replaces the deprecated "shareKey" field.

7.49.3.7 shareName

```
Optional< QString > qevercloud::LinkedNotebook::shareName
```

The display name of the shared notebook. The link owner can change this.

7.49.3.8 stack

```
Optional< QString > qevercloud::LinkedNotebook::stack
```

If this is set, then the notebook is visually contained within a stack of notebooks with this name. All notebooks in the same account with the same 'stack' field are considered to be in the same stack. Notebooks with no stack set are "top level" and not contained within a stack. The link owner can change this and this field is for the benefit of the link owner.

7.49.3.9 updateSequenceNum

```
Optional< qint32 > qevercloud::LinkedNotebook::updateSequenceNum
```

A number identifying the last transaction to modify the state of this object. The USN values are sequential within an account, and can be used to compare the order of modifications within the service.

7.49.3.10 uri

```
Optional< QString > qevercloud::LinkedNotebook::uri
```

NOT DOCUMENTED

7.49.3.11 username

```
Optional< QString > qevercloud::LinkedNotebook::username
```

The username of the user who owns the shared or public notebook.

7.49.3.12 webApiUrlPrefix

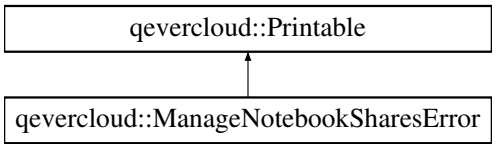
```
Optional< QString > qevercloud::LinkedNotebook::webApiUrlPrefix
```

This field will contain the initial part of the URLs that should be used to make requests to Evernote's thin client "web API", which provide optimized operations for clients that aren't capable of manipulating the full contents of accounts via the full Thrift data model. Clients should concatenate the relative path for the various servlets onto the end of this string to construct the full URL, as documented on our developer web site.

7.50 qevercloud::ManageNotebookSharesError Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::ManageNotebookSharesError:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const ManageNotebookSharesError &other) const
- bool operator!= (const ManageNotebookSharesError &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< UserIdentity > userIdentity
- Optional< EDAMUserException > userException
- Optional< EDAMNotFoundException > notFoundException

7.50.1 Detailed Description

A structure to capture certain errors that occurred during a call to manageNotebookShares. That method can be run best-effort, meaning that some change requests can be applied while others fail. Note that some errors such as system errors will still fail the entire transaction regardless of running best effort. When some change requests do not succeed, the error conditions are captured in instances of this class, captured by the identity of the share relationship and one of the exception fields.

7.50.2 Member Function Documentation

7.50.2.1 operator"!=()

```
bool qevercloud::ManageNotebookSharesError::operator!= (
            const ManageNotebookSharesError & other ) const  [inline]
```

7.50.2.2 operator==()

```
bool qevercloud::ManageNotebookSharesError::operator== (
            const ManageNotebookSharesError & other ) const  [inline]
```

7.50.2.3 print()

```
virtual void qevercloud::ManageNotebookSharesError::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.50.3 Member Data Documentation

7.50.3.1 localData

```
EverCloudLocalData qevercloud::ManageNotebookSharesError::localData
```

See the declaration of EverCloudLocalData for details

7.50.3.2 notFoundException

```
Optional< EDAMNotFoundException > qevercloud::ManageNotebookSharesError::notFoundException
```

If the error is represented as an EDAMNotFoundException that would have otherwise been thrown without best-effort execution. Only one exception field will be set.

7.50.3.3 userException

```
Optional< EDAMUserException > qevercloud::ManageNotebookSharesError::userException
```

If the error is represented as an EDAMUserException that would have otherwise been thrown without best-effort execution. Only one exception field will be set.

7.50.3.4 userIdentity

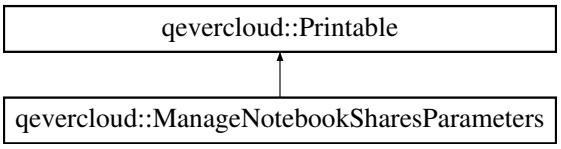
```
Optional< UserIdentity > qevercloud::ManageNotebookSharesError::userIdentity
```

The identity of the share relationship whose update encountered an error.

7.51 qevercloud::ManageNotebookSharesParameters Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::ManageNotebookSharesParameters:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const ManageNotebookSharesParameters &other) const
- bool operator!= (const ManageNotebookSharesParameters &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > notebookGuid
- Optional< QString > inviteMessage
- Optional< QList< MemberShareRelationship > > membershipsToUpdate
- Optional< QList< InvitationShareRelationship > > invitationsToCreateOrUpdate
- Optional< QList< UserIdentity > > unshares

Properties

- OptionalQList< MemberShareRelationship > membershipsToUpdate
- OptionalQList< InvitationShareRelationship > invitationsToCreateOrUpdate
- OptionalQList< UserIdentity > unshares

7.51.1 Detailed Description

A structure that captures parameters used by clients to manage the shares for a given notebook via the manage↩ NotebookShares method.

7.51.2 Member Function Documentation

7.51.2.1 operator"!=()

```
bool qevercloud::ManageNotebookSharesParameters::operator!= (
            const ManageNotebookSharesParameters & other ) const  [inline]
```

7.51.2.2 operator==()

```
bool qevercloud::ManageNotebookSharesParameters::operator== (
            const ManageNotebookSharesParameters & other ) const  [inline]
```

7.51.2.3 print()

```
virtual void qevercloud::ManageNotebookSharesParameters::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.51.3 Member Data Documentation

7.51.3.1 invitationsToCreateOrUpdate

```
Optional<QList<InvitationShareRelationship> > qevercloud::ManageNotebookSharesParameters↩
::invitationsToCreateOrUpdate
```

The list of invitations to update, as matched by the identity field of the InvitationShareRelationship instances, or to create if an existing invitation does not exist. This field is not intended to be the full set of invitations on the notebook and should only include those invitations that you wish to create or update. Note that your invitation could convert into a membership via a service-supported auto-join operation. This happens, for example, when you use an invitation with an Evernote UserID type for a recipient who is a member of the business to which the notebook belongs. Note that to discover the user IDs for business members, the sharer must also be part of the business.

7.51.3.2 inviteMessage

```
Optional< QString > qevercloud::ManageNotebookSharesParameters::inviteMessage
```

If the service sends a message to invitees, this parameter will be used to form the actual message that is sent.

7.51.3.3 localData

```
EverCloudLocalData qevercloud::ManageNotebookSharesParameters::localData
```

See the declaration of EverCloudLocalData for details

7.51.3.4 membershipsToUpdate

```
Optional<QList<MemberShareRelationship> > qevercloud::ManageNotebookSharesParameters::memberships↩
ToUpdate
```

The list of existing memberships to update. This field is not intended to be the full set of memberships for the notebook and should only include those already-existing memberships that you actually want to change. If you want to remove shares, see the unshares fields. If you want to create a membership, i.e. auto-join a business user, you can do this via the invitationsToCreateOrUpdate field using an Evernote UserID of a fellow business member (the created invitation is automatically joined by the service, so the client is creating an invitation, not a membership).

7.51.3.5 notebookGuid

```
Optional< QString > qevercloud::ManageNotebookSharesParameters::notebookGuid
```

The GUID of the notebook whose shares are being managed.

7.51.3.6 unshares

```
Optional<QList<UserIdentity> > qevercloud::ManageNotebookSharesParameters::unshares
```

The list of share relationships to expunge from the service. If the user identity is for an Evernote UserID, then matching invitations or memberships will be removed. If it's an e-mail, then e-mail based shared notebook invitations will be removed. If it's for an Identity ID, then any invitations that match the identity (by identity ID or user ID or e-mail for legacy invitations) will be removed.

7.51.4 Property Documentation

7.51.4.1 invitationsToCreateOrUpdate

```
OptionalQList<InvitationShareRelationship> qevercloud::ManageNotebookSharesParameters::invitations↩
ToCreateOrUpdate
```

7.51.4.2 membershipsToUpdate

```
OptionalQList<MemberShareRelationship> qevercloud::ManageNotebookSharesParameters::memberships↩
ToUpdate
```

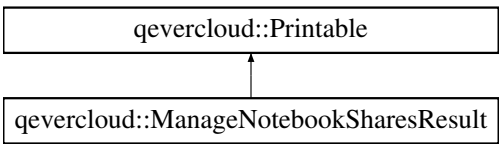
7.51.4.3 unshares

```
OptionalQList<UserIdentity> qevercloud::ManageNotebookSharesParameters::unshares
```

7.52 qevercloud::ManageNotebookSharesResult Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::ManageNotebookSharesResult:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const ManageNotebookSharesResult &other) const
- bool operator!= (const ManageNotebookSharesResult &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QList< ManageNotebookSharesError > > errors

Properties

- OptionalQList< ManageNotebookSharesError > errors

7.52.1 Detailed Description

The return value of a call to the manageNotebookShares method.

7.52.2 Member Function Documentation

7.52.2.1 operator"!=()

```
bool qevercloud::ManageNotebookSharesResult::operator!= (
            const ManageNotebookSharesResult & other ) const  [inline]
```

7.52.2.2 operator==()

```
bool qevercloud::ManageNotebookSharesResult::operator== (
            const ManageNotebookSharesResult & other ) const  [inline]
```

7.52.2.3 print()

```
virtual void qevercloud::ManageNotebookSharesResult::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.52.3 Member Data Documentation

7.52.3.1 errors

```
Optional<QList<ManageNotebookSharesError> > qevercloud::ManageNotebookSharesResult::errors
```

If the method completed without throwing exceptions, some errors might still have occurred, and in that case, this field will contain the list of those errors the occurred.

7.52.3.2 localData

```
EverCloudLocalData qevercloud::ManageNotebookSharesResult::localData
```

See the declaration of EverCloudLocalData for details

7.52.4 Property Documentation

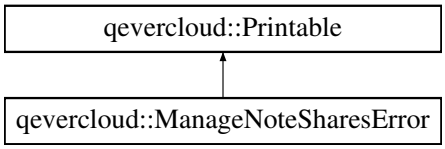
7.52.4.1 errors

```
OptionalQList<ManageNotebookSharesError> qevercloud::ManageNotebookSharesResult::errors
```

7.53 qevercloud::ManageNoteSharesError Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::ManageNoteSharesError:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const ManageNoteSharesError &other) const
- bool operator!= (const ManageNoteSharesError &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< IdentityID > identityID
- Optional< UserID > userID
- Optional< EDAMUserException > userException
- Optional< EDAMNotFoundException > notFoundException

7.53.1 Detailed Description

Captures errors that occur during a call to manageNoteShares. That function can be run best-effort, meaning that some change requests can be applied while others fail. Note that some errors such as system exceptions may still cause the entire call to fail.

Only one of the two ID fields will be set on a given error.

Only one of the two exception fields will be set on a given error.

7.53.2 Member Function Documentation

7.53.2.1 operator"!=()

```
bool qevercloud::ManageNoteSharesError::operator!= (
            const ManageNoteSharesError & other ) const  [inline]
```

7.53.2.2 operator==()

```
bool qevercloud::ManageNoteSharesError::operator== (
            const ManageNoteSharesError & other ) const  [inline]
```

7.53.2.3 print()

```
virtual void qevercloud::ManageNoteSharesError::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.53.3 Member Data Documentation

7.53.3.1 identityID

```
Optional< IdentityID > qevercloud::ManageNoteSharesError::identityID
```

The identity ID of an outstanding invitation that was not updated due to the error.

7.53.3.2 localData

```
EverCloudLocalData qevercloud::ManageNoteSharesError::localData
```

See the declaration of EverCloudLocalData for details

7.53.3.3 notFoundException

```
Optional< EDAMNotFoundException > qevercloud::ManageNoteSharesError::notFoundException
```

If the error is represented as an EDAMNotFoundException that would have otherwise been thrown without best-effort execution. The identifier field of the exception will be either "Identity.id" or "User.id", indicating that no existing share could be found for the specified recipient.

7.53.3.4 userException

```
Optional< EDAMUserException > qevercloud::ManageNoteSharesError::userException
```

If the error is represented as an EDAMUserException that would have otherwise been thrown without best-effort execution.

7.53.3.5 userID

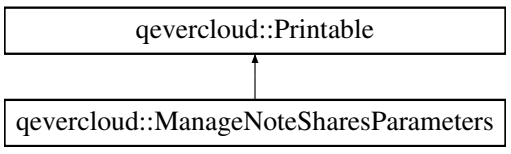
```
Optional< UserID > qevercloud::ManageNoteSharesError::userID
```

The user ID of an existing membership that was not updated due to the error.

7.54 qevercloud::ManageNoteSharesParameters Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::ManageNoteSharesParameters:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const ManageNoteSharesParameters &other) const
- bool operator!= (const ManageNoteSharesParameters &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > noteGuid
- Optional< QList< NoteMemberShareRelationship > > membershipsToUpdate
- Optional< QList< NoteInvitationShareRelationship > > invitationsToUpdate
- Optional< QList< UserID > > membershipsToUnshare
- Optional< QList< IdentityID > > invitationsToUnshare

Properties

- OptionalQList< NoteMemberShareRelationship > membershipsToUpdate
- OptionalQList< NoteInvitationShareRelationship > invitationsToUpdate
- OptionalQList< UserID > membershipsToUnshare
- OptionalQList< IdentityID > invitationsToUnshare

7.54.1 Detailed Description

Captures parameters used by clients to manage the shares for a given note via the manageNoteShares function. This is used only to manage the existing memberships and invitations for a note. To invite a new recipient, use NoteStore.createOrUpdateSharedNotes.

The only field of an existing membership or invitation that can be updated by this function is the share privilege.

7.54.2 Member Function Documentation

7.54.2.1 operator"!=()

```
bool qevercloud::ManageNoteSharesParameters::operator!= (
            const ManageNoteSharesParameters & other ) const  [inline]
```

7.54.2.2 operator==()

```
bool qevercloud::ManageNoteSharesParameters::operator== (
            const ManageNoteSharesParameters & other ) const  [inline]
```

7.54.2.3 print()

```
virtual void qevercloud::ManageNoteSharesParameters::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.54.3 Member Data Documentation

7.54.3.1 invitationsToUnshare

```
Optional<QList<IdentityID> > qevercloud::ManageNoteSharesParameters::invitationsToUnshare
```

A list of outstanding invitations to expunge from the service.

7.54.3.2 invitationsToUpdate

```
Optional<QList<NoteInvitationShareRelationship> > qevercloud::ManageNoteSharesParameters↩
::invitationsToUpdate
```

The list of outstanding invitations to update, as matched by the identity field of the NoteInvitationShareRelatioship instances. This field is not meant to be the full set of invitations for the note. Clients should only include those existing invitations that they wish to modify.

7.54.3.3 localData

```
EverCloudLocalData qevercloud::ManageNoteSharesParameters::localData
```

See the declaration of EverCloudLocalData for details

7.54.3.4 membershipsToUnshare

```
Optional<QList<UserID> > qevercloud::ManageNoteSharesParameters::membershipsToUnshare
```

A list of existing memberships to expunge from the service.

7.54.3.5 membershipsToUpdate

```
Optional<QList<NoteMemberShareRelationship> > qevercloud::ManageNoteSharesParameters::memberships↩
ToUpdate
```

A list of existing memberships to update. This field is not meant to be the full set of memberships for the note. Clients should only include those existing memberships that they wish to modify. To remove an existing membership, see the unshares field.

7.54.3.6 noteGuid

```
Optional< QString > qevercloud::ManageNoteSharesParameters::noteGuid
```

The GUID of the note whose shares are being managed.

7.54.4 Property Documentation

7.54.4.1 invitationsToUnshare

```
OptionalQList<IdentityID> qevercloud::ManageNoteSharesParameters::invitationsToUnshare
```

7.54.4.2 invitationsToUpdate

```
OptionalQList<NoteInvitationShareRelationship> qevercloud::ManageNoteSharesParameters::invitations↩
ToUpdate
```

7.54.4.3 membershipsToUnshare

```
OptionalQList<UserID> qevercloud::ManageNoteSharesParameters::membershipsToUnshare
```

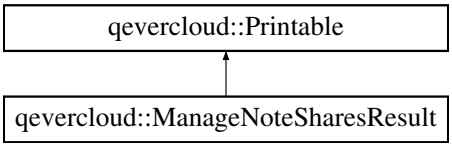
7.54.4.4 membershipsToUpdate

```
OptionalQList<NoteMemberShareRelationship> qevercloud::ManageNoteSharesParameters::memberships↩
ToUpdate
```

7.55 qevercloud::ManageNoteSharesResult Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::ManageNoteSharesResult:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const ManageNoteSharesResult &other) const
- bool operator!= (const ManageNoteSharesResult &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QList< ManageNoteSharesError > > errors

Properties

- OptionalQList< ManageNoteSharesError > errors

7.55.1 Detailed Description

The return value of a call to the manageNoteShares function.

7.55.2 Member Function Documentation

7.55.2.1 operator"!=()

```
bool qevercloud::ManageNoteSharesResult::operator!= (
            const ManageNoteSharesResult & other ) const  [inline]
```

7.55.2.2 operator==()

```
bool qevercloud::ManageNoteSharesResult::operator== (
            const ManageNoteSharesResult & other ) const  [inline]
```

7.55.2.3 print()

```
virtual void qevercloud::ManageNoteSharesResult::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.55.3 Member Data Documentation

7.55.3.1 errors

```
Optional<QList<ManageNoteSharesError> > qevercloud::ManageNoteSharesResult::errors
```

If the call succeeded without throwing an exception, some errors might still have occurred. In that case, this field will contain the list of errors.

7.55.3.2 localData

```
EverCloudLocalData qevercloud::ManageNoteSharesResult::localData
```

See the declaration of EverCloudLocalData for details

7.55.4 Property Documentation

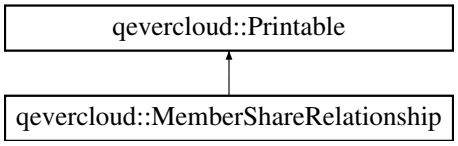
7.55.4.1 errors

```
OptionalQList<ManageNoteSharesError> qevercloud::ManageNoteSharesResult::errors
```

7.56 qevercloud::MemberShareRelationship Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::MemberShareRelationship:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const MemberShareRelationship &other) const
- bool operator!= (const MemberShareRelationship &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > displayName
- Optional< UserID > recipientUserId
- Optional< ShareRelationshipPrivilegeLevel > bestPrivilege
- Optional< ShareRelationshipPrivilegeLevel > individualPrivilege
- Optional< ShareRelationshipRestrictions > restrictions
- Optional< UserID > sharerUserId

7.56.1 Detailed Description

Describes the association between a Notebook and an Evernote User who is a member of that notebook.

7.56.2 Member Function Documentation

7.56.2.1 operator"!=()

```
bool qevercloud::MemberShareRelationship::operator!= (
            const MemberShareRelationship & other ) const  [inline]
```

7.56.2.2 operator==()

```
bool qevercloud::MemberShareRelationship::operator== (
            const MemberShareRelationship & other ) const  [inline]
```

7.56.2.3 print()

```
virtual void qevercloud::MemberShareRelationship::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.56.3 Member Data Documentation

7.56.3.1 bestPrivilege

```
Optional< ShareRelationshipPrivilegeLevel > qevercloud::MemberShareRelationship::bestPrivilege
```

The privilege at which the member can access the notebook, which is the best privilege granted either individually or to a group to which a member belongs, such as a business. This field is used by the service to convey information to the user, so clients should treat it as read-only.

7.56.3.2 displayName

```
Optional< QString > qevercloud::MemberShareRelationship::displayName
```

The string that clients should show to users to represent this member.

7.56.3.3 individualPrivilege

```
Optional< ShareRelationshipPrivilegeLevel > qevercloud::MemberShareRelationship::individual↩
Privilege
```

The individually granted privilege for the member, which does not take GROUP privileges into account. This value may be unset if only a group-assigned privilege has been granted to the member. This value can be managed by others with sufficient rights using the manageNotebookShares method. The valid values that clients should present to users for selection are given via the the 'restrictions' field.

7.56.3.4 localData

```
EverCloudLocalData qevercloud::MemberShareRelationship::localData
```

See the declaration of EverCloudLocalData for details

7.56.3.5 recipientUserId

```
Optional< UserID > qevercloud::MemberShareRelationship::recipientUserId
```

The Evernote User ID of the recipient of this notebook share.

7.56.3.6 restrictions

```
Optional< ShareRelationshipRestrictions > qevercloud::MemberShareRelationship::restrictions
```

The restrictions on which privileges may be individually assigned to the recipient of this share relationship.

7.56.3.7 sharerUserId

```
Optional< UserID > qevercloud::MemberShareRelationship::sharerUserId
```

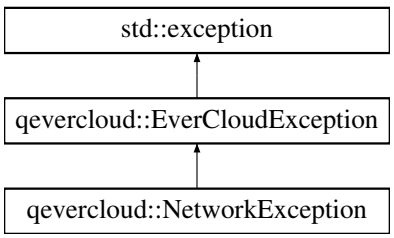
The user id of the user who most recently shared the notebook to this user. This field is currently unset for a MemberShareRelationship created by joining a notebook that has been published to the business (MemberShare↩ Relationships where the individual privilege is unset). This field is used by the service to convey information to the user, so clients should treat it as read-only.

7.57 qevercloud::NetworkException Class Reference

The NetworkException class represents QNetworkReply level errors.

```
#include <Exceptions.h>
```

Inheritance diagram for qevercloud::NetworkException:



Public Member Functions

- NetworkException ()
- NetworkException (QNetworkReply::NetworkError error)
- NetworkException (QNetworkReply::NetworkError error, QString message)
- virtual ~NetworkException () noexcept override
- bool operator== (const NetworkException &other) const
- bool operator!= (const NetworkException &other) const
- QNetworkReply::NetworkError type () const
- const char * what () const noexcept override
- virtual EverCloudExceptionDataPtr exceptionData () const override

Public Member Functions inherited from qevercloud::EverCloudException

- EverCloudException ()
- EverCloudException (QString error)
- EverCloudException (const std::string &error)
- EverCloudException (const char *error)
- virtual ~EverCloudException () noexcept override

Protected Attributes

- QNetworkReply::NetworkError m_type

Protected Attributes inherited from qevercloud::EverCloudException

- QByteArray m_error

7.57.1 Detailed Description

The NetworkException class represents QNetworkReply level errors.

7.57.2 Constructor & Destructor Documentation

7.57.2.1 NetworkException() [1/3]

```
qevercloud::NetworkException::NetworkException ( )
```

7.57.2.2 NetworkException() [2/3]

```
qevercloud::NetworkException::NetworkException (
            QNetworkReply::NetworkError error )
```

7.57.2.3 NetworkException() [3/3]

```
qevercloud::NetworkException::NetworkException (
            QNetworkReply::NetworkError error,
            QString message )
```

7.57.2.4 ~NetworkException()

```
virtual qevercloud::NetworkException::~NetworkException ( )  [override], [virtual], [noexcept]
```

7.57.3 Member Function Documentation

7.57.3.1 exceptionData()

```
virtual EverCloudExceptionDataPtr qevercloud::NetworkException::exceptionData ( ) const  [override],
[virtual]
```

Reimplemented from qevercloud::EverCloudException.

7.57.3.2 operator"!=()

```
bool qevercloud::NetworkException::operator!= (
            const NetworkException & other ) const
```

7.57.3.3 operator==()

```
bool qevercloud::NetworkException::operator== (
            const NetworkException & other ) const
```

7.57.3.4 type()

```
QNetworkReply::NetworkError qevercloud::NetworkException::type ( ) const
```

7.57.3.5 what()

```
const char * qevercloud::NetworkException::what ( ) const  [override], [virtual], [noexcept]
```

Reimplemented from qevercloud::EverCloudException.

7.57.4 Member Data Documentation

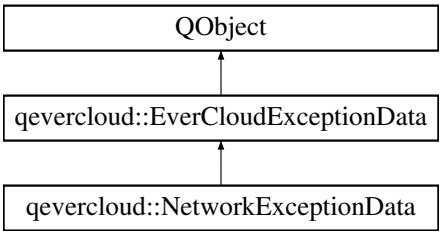
7.57.4.1 m_type

```
QNetworkReply::NetworkError qevercloud::NetworkException::m_type  [protected]
```

7.58 qevercloud::NetworkExceptionData Class Reference

`#include <Exceptions.h>`

Inheritance diagram for qevercloud::NetworkExceptionData:



Public Member Functions

- NetworkExceptionData (QString error, QNetworkReply::NetworkError type)
- virtual void throwException () const override

Public Member Functions inherited from qevercloud::EverCloudExceptionData

- EverCloudExceptionData (QString error)

Protected Attributes

- QNetworkReply::NetworkError m_type

Additional Inherited Members

Public Attributes inherited from qevercloud::EverCloudExceptionData

- QString errorMessage

7.58.1 Detailed Description

Asynchronous API counterpart of NetworkException. See EverCloudExceptionData for more details.

7.58.2 Constructor & Destructor Documentation

7.58.2.1 NetworkExceptionData()

```
qevercloud::NetworkExceptionData::NetworkExceptionData (
            QString error,
            QNetworkReply::NetworkError type )  [explicit]
```

7.58.3 Member Function Documentation

7.58.3.1 throwException()

```
virtual void qevercloud::NetworkExceptionData::throwException ( ) const  [override], [virtual]
```

If you want to throw an exception that corresponds to a received EverCloudExceptionData descendant than call this function. Do not use `throw` statement, it's not polymorphic.

Reimplemented from qevercloud::EverCloudExceptionData.

7.58.4 Member Data Documentation

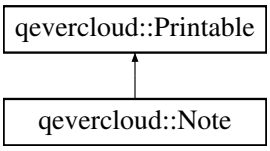
7.58.4.1 m_type

```
QNetworkReply::NetworkError qevercloud::NetworkExceptionData::m_type  [protected]
```

7.59 qevercloud::Note Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::Note:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const Note &other) const
- bool operator!= (const Note &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< Guid > guid
- Optional< QString > title
- Optional< QString > content
- Optional< QByteArray > contentHash
- Optional< qint32 > contentLength
- Optional< Timestamp > created
- Optional< Timestamp > updated
- Optional< Timestamp > deleted
- Optional< bool > active
- Optional< qint32 > updateSequenceNum
- Optional< QString > notebookGuid
- Optional< QList< Guid > > tagGuids
- Optional< QList< Resource > > resources
- Optional< NoteAttributes > attributes
- Optional< QStringList > tagNames
- Optional< QList< SharedNote > > sharedNotes
- Optional< NoteRestrictions > restrictions
- Optional< NoteLimits > limits

Properties

- OptionalQList< Guid > tagGuids
- OptionalQList< Resource > resources
- OptionalQList< SharedNote > sharedNotes

7.59.1 Detailed Description

Represents a single note in the user's account.

7.59.2 Member Function Documentation

7.59.2.1 operator"!=()

```
bool qevercloud::Note::operator!= (
            const Note & other ) const  [inline]
```

7.59.2.2 operator==()

```
bool qevercloud::Note::operator== (
            const Note & other ) const  [inline]
```

7.59.2.3 print()

```
virtual void qevercloud::Note::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.59.3 Member Data Documentation

7.59.3.1 active

`Optional< bool > qevercloud::Note::active`

If the note is available for normal actions and viewing, this flag will be set to true.

7.59.3.2 attributes

`Optional< NoteAttributes > qevercloud::Note::attributes`

A list of the attributes for this note. If the list of attributes are omitted on a call to updateNote(), then the server will assume that no changes have been made to the resources.

7.59.3.3 content

`Optional< QString > qevercloud::Note::content`

The XHTML block that makes up the note. This is the canonical form of the note's contents, so will include abstract Evernote tags for internal resource references. A client may create a separate transformed version of this content for internal presentation, but the same canonical bytes should be used for transmission and comparison unless the user chooses to modify their content.
Length: EDAM_NOTE_CONTENT_LEN_MIN - EDAM_NOTE_CONTENT_LEN_MAX

7.59.3.4 contentHash

`Optional< QByteArray > qevercloud::Note::contentHash`

The binary MD5 checksum of the UTF-8 encoded content body. This will always be set by the server, but clients may choose to omit this when they submit a note with content.
Length: EDAM_HASH_LEN (exactly)

7.59.3.5 contentLength

`Optional< qint32 > qevercloud::Note::contentLength`

The number of Unicode characters in the content of the note. This will always be set by the service, but clients may choose to omit this value when they submit a Note.

7.59.3.6 created

`Optional< Timestamp > qevercloud::Note::created`

The date and time when the note was created in one of the clients. In most cases, this will match the user's sense of when the note was created, and ordering between notes will be based on ordering of this field. However, this is not a "reliable" timestamp if a client has an incorrect clock, so it cannot provide a true absolute ordering between notes. Notes created directly through the service (e.g. via the web GUI) will have an absolutely ordered "created" value.

7.59.3.7 deleted

```
Optional< Timestamp > qevercloud::Note::deleted
```

If present, the note is considered "deleted", and this stores the date and time when the note was deleted by one of the clients. In most cases, this will match the user's sense of when the note was deleted, but this field may be unreliable due to the possibility of client clock errors.

7.59.3.8 guid

```
Optional< Guid > qevercloud::Note::guid
```

The unique identifier of this note. Will be set by the server, but will be omitted by clients calling NoteStore.create↩ Note()
Length: EDAM_GUID_LEN_MIN - EDAM_GUID_LEN_MAX
Regex: EDAM_GUID_REGEX

7.59.3.9 limits

```
Optional< NoteLimits > qevercloud::Note::limits
```

NOT DOCUMENTED

7.59.3.10 localData

```
EverCloudLocalData qevercloud::Note::localData
```

See the declaration of EverCloudLocalData for details

7.59.3.11 notebookGuid

```
Optional< QString > qevercloud::Note::notebookGuid
```

The unique identifier of the notebook that contains this note. If no notebookGuid is provided on a call to create↩ Note(), the default notebook will be used instead.
Length: EDAM_GUID_LEN_MIN - EDAM_GUID_LEN_MAX
Regex: EDAM_GUID_REGEX

7.59.3.12 resources

```
Optional<QList<Resource> > qevercloud::Note::resources
```

The list of resources that are embedded within this note. If the list of resources are omitted on a call to update↩ Note(), then the server will assume that no changes have been made to the resources. The binary contents of the resources must be provided when the resource is first sent to the service, but it will be omitted by the service when the Note is returned in the future. Maximum: EDAM_NOTE_RESOURCES_MAX resources per note

7.59.3.13 restrictions

```
Optional< NoteRestrictions > qevercloud::Note::restrictions
```

If this field is set, the user has note-level permissions that may differ from their notebook-level permissions. In this case, the restrictions structure specifies a set of restrictions limiting the actions that a user may take on the note based on their note-level permissions. If this field is unset, then there are no note-specific restrictions. However, a client may still be limited based on the user's notebook permissions.

7.59.3.14 sharedNotes

```
Optional<QList<SharedNote> > qevercloud::Note::sharedNotes
```

The list of recipients with whom this note has been shared. This field will be unset if the caller has access to the note via the containing notebook, but does not have activity feed permission for that notebook. This field is read-only. Clients may not make changes to a note's sharing state via this field.

7.59.3.15 tagGuids

```
Optional<QList<Guid> > qevercloud::Note::tagGuids
```

A list of the GUID identifiers for tags that are applied to this note. This may be provided in a call to createNote() to unambiguously declare the tags that should be assigned to the new note. Alternately, clients may pass the names of desired tags via the 'tagNames' field during note creation. If the list of tags are omitted on a call to createNote(), then the server will assume that no changes have been made to the resources. Maximum: EDAM_NOTE_TAGS_MAX tags per note

7.59.3.16 tagNames

```
Optional< QStringList > qevercloud::Note::tagNames
```

May be provided by clients during calls to createNote() as an alternative to providing the tagGuids of existing tags. If any tagNames are provided during createNote(), these will be found, or created if they don't already exist. Created tags will have no parent (they will be at the top level of the tag panel).

7.59.3.17 title

```
Optional< QString > qevercloud::Note::title
```

The subject of the note. Can't begin or end with a space.
Length: EDAM_NOTE_TITLE_LEN_MIN - EDAM_NOTE_TITLE_LEN_MAX
Regex: EDAM_NOTE_TITLE_REGEX

7.59.3.18 updated

```
Optional< Timestamp > qevercloud::Note::updated
```

The date and time when the note was last modified in one of the clients. In most cases, this will match the user's sense of when the note was modified, but this field may not be absolutely reliable due to the possibility of client clock errors.

7.59.3.19 updateSequenceNum

```
Optional< qint32 > qevercloud::Note::updateSequenceNum
```

A number identifying the last transaction to modify the state of this note (including changes to the note's attributes or resources). The USN values are sequential within an account, and can be used to compare the order of modifications within the service.

7.59.4 Property Documentation

7.59.4.1 resources

```
OptionalQList<Resource> qevercloud::Note::resources
```

7.59.4.2 sharedNotes

```
OptionalQList<SharedNote> qevercloud::Note::sharedNotes
```

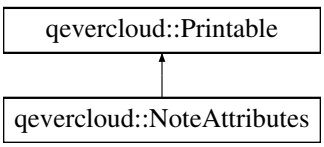
7.59.4.3 tagGuids

```
OptionalQList<Guid> qevercloud::Note::tagGuids
```

7.60 qevercloud::NoteAttributes Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NoteAttributes:



Public Types

- using Classifications = QMap< QString, QString >

Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NoteAttributes &other) const
- bool operator!= (const NoteAttributes &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< Timestamp > subjectDate
- Optional< double > latitude
- Optional< double > longitude
- Optional< double > altitude
- Optional< QString > author
- Optional< QString > source
- Optional< QString > sourceURL
- Optional< QString > sourceApplication
- Optional< Timestamp > shareDate
- Optional< qint64 > reminderOrder
- Optional< Timestamp > reminderDoneTime
- Optional< Timestamp > reminderTime
- Optional< QString > placeName
- Optional< QString > contentClass
- Optional< LazyMap > applicationData
- Optional< QString > lastEditedBy
- Optional< QMap< QString, QString > > classifications
- Optional< UserID > creatorId
- Optional< UserID > lastEditorId
- Optional< bool > sharedWithBusiness
- Optional< Guid > conflictSourceNoteGuid
- Optional< qint32 > noteTitleQuality

Properties

- Optional< Classifications > classifications

7.60.1 Detailed Description

The list of optional attributes that can be stored on a note.

7.60.2 Member Typedef Documentation

7.60.2.1 Classifications

```
using qevercloud::NoteAttributes::Classifications = QMap<QString, QString>
```

7.60.3 Member Function Documentation

7.60.3.1 operator"!=()

```
bool qevercloud::NoteAttributes::operator!= (
            const NoteAttributes & other ) const  [inline]
```

7.60.3.2 operator==()

```
bool qevercloud::NoteAttributes::operator== (
            const NoteAttributes & other ) const  [inline]
```

7.60.3.3 print()

```
virtual void qevercloud::NoteAttributes::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.60.4 Member Data Documentation

7.60.4.1 altitude

```
Optional< double > qevercloud::NoteAttributes::altitude
```

the altitude where the note was taken

7.60.4.2 applicationData

```
Optional< LazyMap > qevercloud::NoteAttributes::applicationData
```

Provides a location for applications to store a relatively small (4kb) blob of data that is not meant to be visible to the user and that is opaque to the Evernote service. A single application may use at most one entry in this map, using its API consumer key as the map key. See the documentation for LazyMap for a description of when the actual map values are returned by the service.

To safely add or modify your application's entry in the map, use NoteStore.setNoteApplicationDataEntry. To safely remove your application's entry from the map, use NoteStore.unsetNoteApplicationDataEntry.

Minimum length of a name (key): EDAM_APPLICATIONDATA_NAME_LEN_MIN
Sum max size of key and value: EDAM_APPLICATIONDATA_ENTRY_LEN_MAX
Syntax regex for name (key): EDAM_APPLICATIONDATA_NAME_REGEX

7.60.4.3 author

```
Optional< QString > qevercloud::NoteAttributes::author
```

the author of the content of the note
Length: EDAM_ATTRIBUTE_LEN_MIN - EDAM_ATTRIBUTE_LEN_MAX

7.60.4.4 classifications

```
Optional<QMap<QString, QString> > qevercloud::NoteAttributes::classifications
```

A map of classifications applied to the note by clients or by the Evernote service. The key is the string name of the classification type, and the value is a constant that begins with CLASSIFICATION_.

7.60.4.5 conflictSourceNoteGuid

```
Optional< Guid > qevercloud::NoteAttributes::conflictSourceNoteGuid
```

If set, this specifies the GUID of a note that caused a sync conflict resulting in the creation of a duplicate note. The duplicated note contains the user's changes that could not be applied as a result of the sync conflict, and uses the conflictSourceNoteGuid field to specify the note that caused the conflict. This allows clients to provide a customized user experience for note conflicts.

7.60.4.6 contentClass

```
Optional< QString > qevercloud::NoteAttributes::contentClass
```

The class (or type) of note. This field is used to indicate to clients that special structured information is represented within the note such that special rules apply when making modifications. If contentClass is set and the client application does not specifically support the specified class, the client MUST treat the note as read-only. In this case, the client MAY modify the note's notebook and tags via the Note.notebookGuid and Note.tagGuids fields. The client MAY also modify the reminderOrder field as well as the reminderTime and reminderDoneTime fields.

Applications should set contentClass only when they are creating notes that contain structured information that needs to be maintained in order for the user to be able to use the note within that application. Setting content↩ Class makes a note read-only in other applications, so there is a trade-off when an application chooses to use contentClass. Applications that set contentClass when creating notes must use a contentClass string of the form *CompanyName.ApplicationName* to ensure uniqueness.

Length restrictions: EDAM_NOTE_CONTENT_CLASS_LEN_MIN, EDAM_NOTE_CONTENT_CLASS_LEN_MAX
Regex: EDAM_NOTE_CONTENT_CLASS_REGEX

7.60.4.7 creatorId

```
Optional< UserID > qevercloud::NoteAttributes::creatorId
```

The numeric user ID of the user who originally created the note.

7.60.4.8 lastEditedBy

```
Optional< QString > qevercloud::NoteAttributes::lastEditedBy
```

An indication of who made the last change to the note. If you are accessing the note via a shared notebook to which you have modification rights, or if you are the owner of the notebook to which the note belongs, then you have access to the value. In this case, the value will be unset if the owner of the notebook containing the note was the last to make the modification, else it will be a string describing the guest who made the last edit. If you do not have access to this value, it will be left unset. This field is read-only by clients. The server will ignore all values set by clients into this field.

7.60.4.9 lastEditorId

`Optional< UserID > qevercloud::NoteAttributes::lastEditorId`

The numeric user ID of the user described in lastEditedBy.

7.60.4.10 latitude

`Optional< double > qevercloud::NoteAttributes::latitude`

the latitude where the note was taken

7.60.4.11 localData

`EverCloudLocalData qevercloud::NoteAttributes::localData`

See the declaration of EverCloudLocalData for details

7.60.4.12 longitude

`Optional< double > qevercloud::NoteAttributes::longitude`

the longitude where the note was taken

7.60.4.13 noteTitleQuality

`Optional< qint32 > qevercloud::NoteAttributes::noteTitleQuality`

If set, this specifies that the note's title was automatically generated and indicates the likelihood that the generated title is useful for display to the user. If not set, the note's title was manually entered by the user.

Clients MUST set this attribute to one of the following values when the corresponding note's title was not manually entered by the user: EDAM_NOTE_TITLE_QUALITY_UNTITLED, EDAM_NOTE_TITLE_QUALITY_LOW, EDAM↩ _NOTE_TITLE_QUALITY_MEDIUM or EDAM_NOTE_TITLE_QUALITY_HIGH.

When a user edits a note's title, clients MUST unset this value.

7.60.4.14 placeName

`Optional< QString > qevercloud::NoteAttributes::placeName`

Allows the user to assign a human-readable location name associated with a note. Users may assign values like 'Home' and 'Work'. Place names may also be populated with values from geonames database (e.g., a restaurant name). Applications are encouraged to normalize values so that grouping values by place name provides a useful result. Applications MUST NOT automatically add place name values based on geolocation without confirmation from the user; that is, the value in this field should be more useful than a simple automated lookup based on the note's latitude and longitude.

7.60.4.15 reminderDoneTime

```
Optional< Timestamp > qevercloud::NoteAttributes::reminderDoneTime
```

The date and time when a user dismissed/"marked done" the reminder on the note. Users typically do not manually set this value directly as it is set to the time when the user dismissed/"marked done" the reminder.

7.60.4.16 reminderOrder

```
Optional< qint64 > qevercloud::NoteAttributes::reminderOrder
```

The set of notes with this parameter set are considered "reminders" and are to be treated specially by clients to give them higher UI prominence within a notebook. The value is used to sort the reminder notes within the notebook with higher values representing greater prominence. Outside of the context of a notebook, the value of this parameter is undefined. The value is not intended to be compared to the values of reminder notes in other notebooks. In order to allow clients to place a note at a higher precedence than other notes, you should never set a value greater than the current time (as defined for a Timetstamp). To place a note at higher precedence than existing notes, set the value to the current time as defined for a timestamp (milliseconds since the epoch). Synchronizing clients must remember the time when the update was performed, using the local clock on the client, and use that value when they later upload the note to the service. Clients must not set the reminderOrder to the reminderTime as the reminderTime could be in the future. Those two fields are never intended to be related. The correct value for reminderOrder field for new notes is the "current" time when the user indicated that the note is a reminder. Clients may implement a separate "sort by date" feature to show notes ordered by reminderTime. Whenever a reminder↩ DoneTime or reminderTime is set but a reminderOrder is not set, the server will fill in the current server time for the reminderOrder field.

7.60.4.17 reminderTime

```
Optional< Timestamp > qevercloud::NoteAttributes::reminderTime
```

The date and time a user has selected to be reminded of the note. A note with this value set is known as a "reminder" and the user can be reminded, via e-mail or client-specific notifications, of the note when the time is reached or about to be reached. When a user sets a reminder time on a note that has a reminder done time, and that reminder time is in the future, then the reminder done time should be cleared. This should happen regardless of any existing reminder time that may have previously existed on the note.

7.60.4.18 shareDate

```
Optional< Timestamp > qevercloud::NoteAttributes::shareDate
```

The date and time when this note was directly shared via its own URL. This is only set on notes that were individually shared - it is independent of any notebook-level sharing of the containing notebook. This field is treated as "read-only" for clients; the server will ignore changes to this field from an external client.

7.60.4.19 sharedWithBusiness

```
Optional< bool > qevercloud::NoteAttributes::sharedWithBusiness
```

When this flag is set on a business note, any user in that business may view the note if they request it by GUID. This field is read-only by clients. The server will ignore all values set by clients into this field.

To share a note with the business, use NoteStore.shareNoteWithBusiness and to stop sharing a note with the business, use NoteStore.stopSharingNoteWithBusiness.

7.60.4.20 source

`Optional< QString > qevercloud::NoteAttributes::source`

the method that the note was added to the account, if the note wasn't directly authored in an Evernote desktop client.
Length: EDAM_ATTRIBUTE_LEN_MIN - EDAM_ATTRIBUTE_LEN_MAX

7.60.4.21 sourceApplication

`Optional< QString > qevercloud::NoteAttributes::sourceApplication`

an identifying string for the application that created this note. This string does not have a guaranteed syntax or structure – it is intended for human inspection and tracking.
Length: EDAM_ATTRIBUTE_LEN_MIN - EDAM_ATTRIBUTE_LEN_MAX

7.60.4.22 sourceURL

`Optional< QString > qevercloud::NoteAttributes::sourceURL`

the original location where the resource was hosted. For web clips, this will be the URL of the page that was clipped.
Length: EDAM_ATTRIBUTE_LEN_MIN - EDAM_ATTRIBUTE_LEN_MAX

7.60.4.23 subjectDate

`Optional< Timestamp > qevercloud::NoteAttributes::subjectDate`

time that the note refers to

7.60.5 Property Documentation

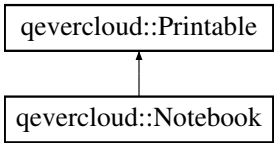
7.60.5.1 classifications

`Optional<Classifications> qevercloud::NoteAttributes::classifications`

7.61 qevercloud::Notebook Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::Notebook:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const Notebook &other) const
- bool operator!= (const Notebook &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< Guid > guid
- Optional< QString > name
- Optional< qint32 > updateSequenceNum
- Optional< bool > defaultNotebook
- Optional< Timestamp > serviceCreated
- Optional< Timestamp > serviceUpdated
- Optional< Publishing > publishing
- Optional< bool > published
- Optional< QString > stack
- Optional< QList< qint64 > > sharedNotebookIds
- Optional< QList< SharedNotebook > > sharedNotebooks
- Optional< BusinessNotebook > businessNotebook
- Optional< User > contact
- Optional< NotebookRestrictions > restrictions
- Optional< NotebookRecipientSettings > recipientSettings

Properties

- OptionalQList< qint64 > sharedNotebookIds
- OptionalQList< SharedNotebook > sharedNotebooks

7.61.1 Detailed Description

A unique container for a set of notes.

7.61.2 Member Function Documentation

7.61.2.1 operator"!=()

```
bool qevercloud::Notebook::operator!= (
            const Notebook & other ) const  [inline]
```

7.61.2.2 operator==()

```
bool qevercloud::Notebook::operator== (
            const Notebook & other ) const  [inline]
```

7.61.2.3 print()

```
virtual void qevercloud::Notebook::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.61.3 Member Data Documentation

7.61.3.1 businessNotebook

```
Optional< BusinessNotebook > qevercloud::Notebook::businessNotebook
```

If the notebook is part of a business account and has been shared with the entire business, this will contain sharing information. The presence or absence of this field is not a reliable test of whether a given notebook is in fact a business notebook - the field is only used when a notebook is or has been shared with the entire business.

7.61.3.2 contact

```
Optional< User > qevercloud::Notebook::contact
```

Intended for use with Business accounts, this field identifies the user who has been designated as the "contact". For notebooks created in business accounts, the server will automatically set this value to the user who created the notebook unless Notebook.contact.username has been set, in which that value will be used. When updating a notebook, it is common to leave Notebook.contact field unset, indicating that no change to the value is being requested and that the existing value, if any, should be preserved.

7.61.3.3 defaultNotebook

```
Optional< bool > qevercloud::Notebook::defaultNotebook
```

If true, this notebook should be used for new notes whenever the user has not (or cannot) specify a desired target notebook. For example, if a note is submitted via SMTP email. The service will maintain at most one default↩ Notebook per account. If a second notebook is created or updated with defaultNotebook set to true, the service will automatically update the prior notebook's defaultNotebook field to false. If the default notebook is deleted (i.e. "active" set to false), the "defaultNotebook" field will be set to false by the service. If the account has no default notebook set, the service will use the most recent notebook as the default.

7.61.3.4 guid

```
Optional< Guid > qevercloud::Notebook::guid
```

The unique identifier of this notebook.
Length: EDAM_GUID_LEN_MIN - EDAM_GUID_LEN_MAX
Regex: EDAM_GUID_REGEX

7.61.3.5 localData

```
EverCloudLocalData qevercloud::Notebook::localData
```

See the declaration of EverCloudLocalData for details

7.61.3.6 name

```
Optional< QString > qevercloud::Notebook::name
```

A sequence of characters representing the name of the notebook. May be changed by clients, but the account may not contain two notebooks with names that are equal via a case-insensitive comparison. Can't begin or end with a space.
Length: EDAM_NOTEBOOK_NAME_LEN_MIN - EDAM_NOTEBOOK_NAME_LEN_MAX
Regex: EDAM_NOTEBOOK_NAME_REGEX

7.61.3.7 published

```
Optional< bool > qevercloud::Notebook::published
```

If this is set to true, then the Notebook will be accessible either to the public, or for business users to their business, via the 'publishing' or 'businessNotebook' specifications, which must also be set. If this is set to false, the Notebook will not be available to the public (or business). Clients that do not wish to change the publishing behavior of a Notebook should not set this value when calling NoteStore.updateNotebook().

7.61.3.8 publishing

```
Optional< Publishing > qevercloud::Notebook::publishing
```

If the Notebook has been opened for public access, then this will point to the set of publishing information for the Notebook (URI, description, etc.). A Notebook cannot be published without providing this information, but it will persist for later use if publishing is ever disabled on the Notebook. Clients that do not wish to change the publishing behavior of a Notebook should not set this value when calling NoteStore.updateNotebook(). Note that this structure is never populated for business notebooks, see the businessNotebook field.

7.61.3.9 recipientSettings

```
Optional< NotebookRecipientSettings > qevercloud::Notebook::recipientSettings
```

This represents the preferences/settings that a recipient has set for this notebook. These are intended to be changed only by the recipient, and each recipient has their own recipient settings.

7.61.3.10 restrictions

```
Optional< NotebookRestrictions > qevercloud::Notebook::restrictions
```

NOT DOCUMENTED

7.61.3.11 serviceCreated

```
Optional< Timestamp > qevercloud::Notebook::serviceCreated
```

The time when this notebook was created on the service. This will be set on the service during creation, and the service will provide this value when it returns a Notebook to a client. The service will ignore this value if it is sent by clients.

7.61.3.12 serviceUpdated

```
Optional< Timestamp > qevercloud::Notebook::serviceUpdated
```

The time when this notebook was last modified on the service. This will be set on the service during creation, and the service will provide this value when it returns a Notebook to a client. The service will ignore this value if it is sent by clients.

7.61.3.13 sharedNotebookIds

```
Optional<QList<qint64> > qevercloud::Notebook::sharedNotebookIds
```

DEPRECATED - replaced by sharedNotebooks.

7.61.3.14 sharedNotebooks

```
Optional<QList<SharedNotebook> > qevercloud::Notebook::sharedNotebooks
```

The list of recipients to whom this notebook has been shared (one SharedNotebook object per recipient email address). This field will be unset if you do not have permission to access this data. If you are accessing the notebook as the owner or via a shared notebook that is modifiable, then you have access to this data and the value will be set. This field is read-only. Clients may not make changes to shared notebooks via this field.

7.61.3.15 stack

```
Optional< QString > qevercloud::Notebook::stack
```

If this is set, then the notebook is visually contained within a stack of notebooks with this name. All notebooks in the same account with the same 'stack' field are considered to be in the same stack. Notebooks with no stack set are "top level" and not contained within a stack.

7.61.3.16 updateSequenceNum

```
Optional< qint32 > qevercloud::Notebook::updateSequenceNum
```

A number identifying the last transaction to modify the state of this object. The USN values are sequential within an account, and can be used to compare the order of modifications within the service.

7.61.4 Property Documentation

7.61.4.1 sharedNotebookIds

```
OptionalQList<qint64> qevercloud::Notebook::sharedNotebookIds
```

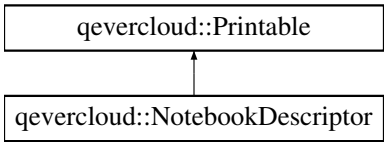
7.61.4.2 sharedNotebooks

```
OptionalQList<SharedNotebook> qevercloud::Notebook::sharedNotebooks
```

7.62 qevercloud::NotebookDescriptor Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NotebookDescriptor:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NotebookDescriptor &other) const
- bool operator!= (const NotebookDescriptor &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< Guid > guid
- Optional< QString > notebookDisplayName
- Optional< QString > contactName
- Optional< bool > hasSharedNotebook
- Optional< qint32 > joinedUserCount

7.62.1 Detailed Description

A structure that describes a notebook or a user's relationship with a notebook. NotebookDescriptor is expected to remain a lighter-weight structure when compared to Notebook.

7.62.2 Member Function Documentation

7.62.2.1 operator"!=()

```
bool qevercloud::NotebookDescriptor::operator!= (
            const NotebookDescriptor & other ) const  [inline]
```

7.62.2.2 operator==()

```
bool qevercloud::NotebookDescriptor::operator== (
            const NotebookDescriptor & other ) const  [inline]
```

7.62.2.3 print()

```
virtual void qevercloud::NotebookDescriptor::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.62.3 Member Data Documentation

7.62.3.1 contactName

```
Optional< QString > qevercloud::NotebookDescriptor::contactName
```

The User.name value of the notebook's "contact".

7.62.3.2 guid

```
Optional< Guid > qevercloud::NotebookDescriptor::guid
```

The unique identifier of the notebook.

7.62.3.3 hasSharedNotebook

```
Optional< bool > qevercloud::NotebookDescriptor::hasSharedNotebook
```

Whether a SharedNotebook record exists between the calling user and this notebook.

7.62.3.4 joinedUserCount

```
Optional< qint32 > qevercloud::NotebookDescriptor::joinedUserCount
```

The number of users who have joined this notebook.

7.62.3.5 localData

```
EverCloudLocalData qevercloud::NotebookDescriptor::localData
```

See the declaration of EverCloudLocalData for details

7.62.3.6 notebookDisplayName

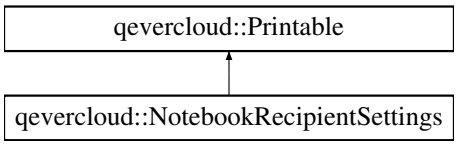
```
Optional< QString > qevercloud::NotebookDescriptor::notebookDisplayName
```

A sequence of characters representing the name of the notebook.

7.63 qevercloud::NotebookRecipientSettings Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NotebookRecipientSettings:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NotebookRecipientSettings &other) const
- bool operator!= (const NotebookRecipientSettings &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< bool > reminderNotifyEmail
- Optional< bool > reminderNotifyInApp
- Optional< bool > inMyList
- Optional< QString > stack
- Optional< RecipientStatus > recipientStatus

7.63.1 Detailed Description

Settings meant for the recipient of a notebook share.

Some of these fields have a 3-state read value but a 2-state write value. On read, it is possible to observe "unset", true, or false. The initial state is "unset". When you choose to set a value, you may set it to either true or false, but you cannot unset the value. Once one of these members has a true/false value, it will always have a true/false value.

7.63.2 Member Function Documentation

7.63.2.1 operator"!=()

```
bool qevercloud::NotebookRecipientSettings::operator!= (
            const NotebookRecipientSettings & other ) const  [inline]
```

7.63.2.2 operator==()

```
bool qevercloud::NotebookRecipientSettings::operator== (
            const NotebookRecipientSettings & other ) const  [inline]
```

7.63.2.3 print()

```
virtual void qevercloud::NotebookRecipientSettings::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.63.3 Member Data Documentation

7.63.3.1 inMyList

```
Optional< bool > qevercloud::NotebookRecipientSettings::inMyList
```

DEPRECATED: Use recipientStatus instead. The notebook is on the recipient's notebook list (formerly, we would say that the recipient has "joined" the notebook)

7.63.3.2 localData

```
EverCloudLocalData qevercloud::NotebookRecipientSettings::localData
```

See the declaration of EverCloudLocalData for details

7.63.3.3 recipientStatus

```
Optional< RecipientStatus > qevercloud::NotebookRecipientSettings::recipientStatus
```

The notebook is on/off the recipient's notebook list (formerly, we would say that the recipient has "joined" the notebook) and perhaps also their default notebook

7.63.3.4 reminderNotifyEmail

```
Optional< bool > qevercloud::NotebookRecipientSettings::reminderNotifyEmail
```

Indicates that the user wishes to receive daily e-mail notifications for reminders associated with the notebook. This may be true only for business notebooks that belong to the business of which the user is a member. You may only set this value on a notebook in your business. This value will initially be unset.

7.63.3.5 reminderNotifyInApp

```
Optional< bool > qevercloud::NotebookRecipientSettings::reminderNotifyInApp
```

Indicates that the user wishes to receive notifications for reminders by applications that support providing such notifications. The exact nature of the notification is defined by the individual applications. This value will initially be unset.

7.63.3.6 stack

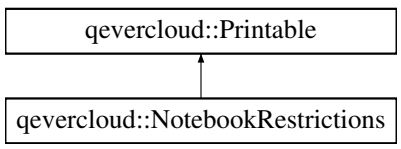
```
Optional< QString > qevercloud::NotebookRecipientSettings::stack
```

The stack the recipient has put this notebook into. See Notebook.stack for a definition. Every recipient can have their own stack value for the same notebook.

7.64 qevercloud::NotebookRestrictions Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NotebookRestrictions:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NotebookRestrictions &other) const
- bool operator!= (const NotebookRestrictions &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< bool > noReadNotes
- Optional< bool > noCreateNotes
- Optional< bool > noUpdateNotes
- Optional< bool > noExpungeNotes
- Optional< bool > noShareNotes
- Optional< bool > noEmailNotes
- Optional< bool > noSendMessageToRecipients
- Optional< bool > noUpdateNotebook
- Optional< bool > noExpungeNotebook
- Optional< bool > noSetDefaultNotebook
- Optional< bool > noSetNotebookStack
- Optional< bool > noPublishToPublic
- Optional< bool > noPublishToBusinessLibrary
- Optional< bool > noCreateTags
- Optional< bool > noUpdateTags
- Optional< bool > noExpungeTags
- Optional< bool > noSetParentTag
- Optional< bool > noCreateSharedNotebooks
- Optional< SharedNotebookInstanceRestrictions > updateWhichSharedNotebookRestrictions
- Optional< SharedNotebookInstanceRestrictions > expungeWhichSharedNotebookRestrictions
- Optional< bool > noShareNotesWithBusiness
- Optional< bool > noRenameNotebook
- Optional< bool > noSetInMyList
- Optional< bool > noChangeContact
- Optional< CanMoveToContainerRestrictions > canMoveToContainerRestrictions
- Optional< bool > noSetReminderNotifyEmail
- Optional< bool > noSetReminderNotifyInApp
- Optional< bool > noSetRecipientSettingsStack
- Optional< bool > noCanMoveNote

7.64.1 Detailed Description

This structure captures information about the types of operations that cannot be performed on a given notebook with a type of authenticated access and credentials. The values filled into this structure are based on then-current values in the server database for shared notebooks and notebook publishing records, as well as information related to the authentication token. Information from the authentication token includes the application that is accessing the server, as defined by the permissions granted by consumer (api) key, and the method used to obtain the token, for example via authenticateToSharedNotebook, authenticateToBusiness, etc. Note that changes to values in this structure that are the result of shared notebook or publishing record changes are communicated to the client via a change in the notebook USN during sync. It is important to use the same access method, parameters, and consumer key in order obtain correct results from the sync engine.

The server has the final say on what is allowed as values may change between calls to obtain NotebookRestrictions instances and to operate on data on the service.

If the following are set and true, then the given restriction is in effect, as accessed by the same authentication token from which the values were obtained.

7.64.2 Member Function Documentation

7.64.2.1 operator"!=()

```
bool qevercloud::NotebookRestrictions::operator!= (
            const NotebookRestrictions & other ) const  [inline]
```

7.64.2.2 operator==()

```
bool qevercloud::NotebookRestrictions::operator== (
            const NotebookRestrictions & other ) const  [inline]
```

7.64.2.3 print()

```
virtual void qevercloud::NotebookRestrictions::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.64.3 Member Data Documentation

7.64.3.1 canMoveToContainerRestrictions

```
Optional< CanMoveToContainerRestrictions > qevercloud::NotebookRestrictions::canMoveToContainer↩
Restrictions
```

Specifies if the client can move this notebook to a container and if not, the reason why.

7.64.3.2 expungeWhichSharedNotebookRestrictions

```
Optional< SharedNotebookInstanceRestrictions > qevercloud::NotebookRestrictions::expunge↩
WhichSharedNotebookRestrictions
```

Restrictions on which shared notebook instances can be expunged. If the value is not set or null, then the client can expunge any of the shared notebooks associated with the notebook on which the NotebookRestrictions are defined. See the enumeration for further details.

7.64.3.3 localData

```
EverCloudLocalData qevercloud::NotebookRestrictions::localData
```

See the declaration of EverCloudLocalData for details

7.64.3.4 noCanMoveNote

```
Optional< bool > qevercloud::NotebookRestrictions::noCanMoveNote
```

If set, the client cannot move a Note into or out of the Notebook.

7.64.3.5 noChangeContact

Optional< bool > qevercloud::NotebookRestrictions::noChangeContact

NOT DOCUMENTED

7.64.3.6 noCreateNotes

Optional< bool > qevercloud::NotebookRestrictions::noCreateNotes

The client may not create new notes in the notebook.

7.64.3.7 noCreateSharedNotebooks

Optional< bool > qevercloud::NotebookRestrictions::noCreateSharedNotebooks

The client is unable to create shared notebooks for the notebook.

7.64.3.8 noCreateTags

Optional< bool > qevercloud::NotebookRestrictions::noCreateTags

The client may not complete an operation that results in a new tag being created in the owner's account.

7.64.3.9 noEmailNotes

Optional< bool > qevercloud::NotebookRestrictions::noEmailNotes

The client may not e-mail notes by guid via the Evernote service by using the emailNote method. Email notes by value by populating the note parameter instead.

7.64.3.10 noExpungeNotebook

Optional< bool > qevercloud::NotebookRestrictions::noExpungeNotebook

The client may not expunge the Notebook object itself, for example, via the expungeNotebook method.

7.64.3.11 noExpungeNotes

Optional< bool > qevercloud::NotebookRestrictions::noExpungeNotes

The client may not expunge notes currently in the notebook.

7.64.3.12 noExpungeTags

`Optional< bool > qevercloud::NotebookRestrictions::noExpungeTags`

The client may not expunge tags in the owner's account.

7.64.3.13 noPublishToBusinessLibrary

`Optional< bool > qevercloud::NotebookRestrictions::noPublishToBusinessLibrary`

The client may not publish the notebook to the business library.

7.64.3.14 noPublishToPublic

`Optional< bool > qevercloud::NotebookRestrictions::noPublishToPublic`

The client may not publish the notebook to the public. For example, business notebooks may not be shared publicly.

7.64.3.15 noReadNotes

`Optional< bool > qevercloud::NotebookRestrictions::noReadNotes`

The client is not able to read notes from the service and the notebook is write-only.

7.64.3.16 noRenameNotebook

`Optional< bool > qevercloud::NotebookRestrictions::noRenameNotebook`

The client may not rename this notebook.

7.64.3.17 noSendMessageToRecipients

`Optional< bool > qevercloud::NotebookRestrictions::noSendMessageToRecipients`

The client may not send messages to the share recipients of the notebook.

7.64.3.18 noSetDefaultNotebook

`Optional< bool > qevercloud::NotebookRestrictions::noSetDefaultNotebook`

The client may not set this notebook to be the default notebook. The caller should leave Notebook.defaultNotebook unset.

7.64.3.19 noSetInMyList

```
Optional< bool > qevercloud::NotebookRestrictions::noSetInMyList
```

clients may not change the NotebookRecipientSettings.inMyList settings for this notebook.

7.64.3.20 noSetNotebookStack

```
Optional< bool > qevercloud::NotebookRestrictions::noSetNotebookStack
```

If the client is able to update the Notebook, the Notebook.stack value may not be set.

7.64.3.21 noSetParentTag

```
Optional< bool > qevercloud::NotebookRestrictions::noSetParentTag
```

If the client is able to create or update tags in the owner's account, then they will not be able to set the parent tag. Leave the value unset.

7.64.3.22 noSetRecipientSettingsStack

```
Optional< bool > qevercloud::NotebookRestrictions::noSetRecipientSettingsStack
```

NOT DOCUMENTED

7.64.3.23 noSetReminderNotifyEmail

```
Optional< bool > qevercloud::NotebookRestrictions::noSetReminderNotifyEmail
```

NOT DOCUMENTED

7.64.3.24 noSetReminderNotifyInApp

```
Optional< bool > qevercloud::NotebookRestrictions::noSetReminderNotifyInApp
```

NOT DOCUMENTED

7.64.3.25 noShareNotes

```
Optional< bool > qevercloud::NotebookRestrictions::noShareNotes
```

The client may not share notes in the notebook via the shareNote or createOrUpdateSharedNotes methods.

7.64.3.26 noShareNotesWithBusiness

```
Optional< bool > qevercloud::NotebookRestrictions::noShareNotesWithBusiness
```

The client may not share notes in the notebook via the shareNoteWithBusiness method.

7.64.3.27 noUpdateNotebook

```
Optional< bool > qevercloud::NotebookRestrictions::noUpdateNotebook
```

The client may not update the Notebook object itself, for example, via the updateNotebook method.

7.64.3.28 noUpdateNotes

```
Optional< bool > qevercloud::NotebookRestrictions::noUpdateNotes
```

The client may not update notes currently in the notebook.

7.64.3.29 noUpdateTags

```
Optional< bool > qevercloud::NotebookRestrictions::noUpdateTags
```

The client may not update tags in the owner's account.

7.64.3.30 updateWhichSharedNotebookRestrictions

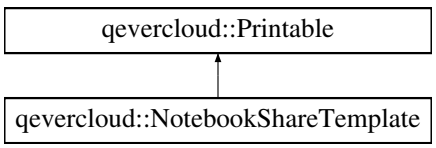
```
Optional< SharedNotebookInstanceRestrictions > qevercloud::NotebookRestrictions::updateWhich↩
SharedNotebookRestrictions
```

Restrictions on which shared notebook instances can be updated. If the value is not set or null, then the client can update any of the shared notebooks associated with the notebook on which the NotebookRestrictions are defined. See the enumeration for further details.

7.65 qevercloud::NotebookShareTemplate Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NotebookShareTemplate:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NotebookShareTemplate &other) const
- bool operator!= (const NotebookShareTemplate &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< Guid > notebookGuid
- Optional< MessageThreadID > recipientThreadId
- Optional< QList< Contact > > recipientContacts
- Optional< SharedNotebookPrivilegeLevel > privilege

Properties

- OptionalQList< Contact > recipientContacts

7.65.1 Detailed Description

A structure used to share a notebook with one or more recipients at a given privilege.

7.65.2 Member Function Documentation

7.65.2.1 operator"!=()

```
bool qevercloud::NotebookShareTemplate::operator!= (
            const NotebookShareTemplate & other ) const  [inline]
```

7.65.2.2 operator==()

```
bool qevercloud::NotebookShareTemplate::operator== (
            const NotebookShareTemplate & other ) const  [inline]
```

7.65.2.3 print()

```
virtual void qevercloud::NotebookShareTemplate::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.65.3 Member Data Documentation

7.65.3.1 localData

```
EverCloudLocalData qevercloud::NotebookShareTemplate::localData
```

See the declaration of EverCloudLocalData for details

7.65.3.2 notebookGuid

```
Optional< Guid > qevercloud::NotebookShareTemplate::notebookGuid
```

The GUID of the notebook.

7.65.3.3 privilege

```
Optional< SharedNotebookPrivilegeLevel > qevercloud::NotebookShareTemplate::privilege
```

The privilege level to be granted.

7.65.3.4 recipientContacts

```
Optional<QList<Contact> > qevercloud::NotebookShareTemplate::recipientContacts
```

The recipients of the notebook share specified as a list of contacts. This should only be set if the sharing takes place before the thread is created. Use recipientThreadId instead when sharing with an existing thread. Either this field or recipientThreadId must be set.

7.65.3.5 recipientThreadId

```
Optional< MessageThreadID > qevercloud::NotebookShareTemplate::recipientThreadId
```

The recipients of the notebook share specified as a messaging thread ID. If you have an existing messaging thread to share the note with, specify its ID here instead of recipientContacts in order to properly support defunct identities. The sharer must be a participant of the thread. Either this field or recipientContacts must be set.

7.65.4 Property Documentation

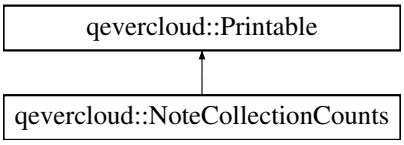
7.65.4.1 recipientContacts

```
OptionalQList<Contact> qevercloud::NotebookShareTemplate::recipientContacts
```

7.66 qevercloud::NoteCollectionCounts Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::NoteCollectionCounts:



Public Types

- using TagCounts = QMap< Guid, qint32 >

Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NoteCollectionCounts &other) const
- bool operator!= (const NoteCollectionCounts &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QMap< Guid, qint32 > > notebookCounts
- Optional< QMap< Guid, qint32 > > tagCounts
- Optional< qint32 > trashCount

Properties

- Optional< TagCounts > notebookCounts
- Optional< TagCounts > tagCounts

7.66.1 Detailed Description

A data structure representing the number of notes for each notebook and tag with a non-zero set of applicable notes.

7.66.2 Member Typedef Documentation

7.66.2.1 TagCounts

```
using qevercloud::NoteCollectionCounts::TagCounts = QMap<Guid, qint32>
```

7.66.3 Member Function Documentation

7.66.3.1 operator"!=()

```
bool qevercloud::NoteCollectionCounts::operator!= (
            const NoteCollectionCounts & other ) const  [inline]
```

7.66.3.2 operator==()

```
bool qevercloud::NoteCollectionCounts::operator== (
            const NoteCollectionCounts & other ) const  [inline]
```

7.66.3.3 print()

```
virtual void qevercloud::NoteCollectionCounts::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.66.4 Member Data Documentation

7.66.4.1 localData

```
EverCloudLocalData qevercloud::NoteCollectionCounts::localData
```

See the declaration of EverCloudLocalData for details

7.66.4.2 notebookCounts

```
Optional<QMap<Guid, qint32> > qevercloud::NoteCollectionCounts::notebookCounts
```

A mapping from the Notebook GUID to the number of notes (from some selection) that are in the corresponding notebook.

7.66.4.3 tagCounts

```
Optional<QMap<Guid, qint32> > qevercloud::NoteCollectionCounts::tagCounts
```

A mapping from the Tag GUID to the number of notes (from some selection) that have the corresponding tag.

7.66.4.4 trashCount

```
Optional< qint32 > qevercloud::NoteCollectionCounts::trashCount
```

If this is set, then this is the number of notes that are in the trash. If this is not set, then the number of notes in the trash hasn't been reported. (I.e. if there are no notes in the trash, this will be set to 0.)

7.66.5 Property Documentation

7.66.5.1 notebookCounts

```
Optional<TagCounts> qevercloud::NoteCollectionCounts::notebookCounts
```

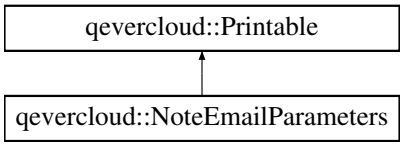
7.66.5.2 tagCounts

```
Optional<TagCounts> qevercloud::NoteCollectionCounts::tagCounts
```

7.67 qevercloud::NoteEmailParameters Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NoteEmailParameters:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NoteEmailParameters &other) const
- bool operator!= (const NoteEmailParameters &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > guid
- Optional< Note > note
- Optional< QStringList > toAddresses
- Optional< QStringList > ccAddresses
- Optional< QString > subject
- Optional< QString > message

7.67.1 Detailed Description

Parameters that must be given to the NoteStore emailNote call. These allow the caller to specify the note to send, the recipient addresses, etc.

7.67.2 Member Function Documentation

7.67.2.1 operator"!=()

```
bool qevercloud::NoteEmailParameters::operator!= (
            const NoteEmailParameters & other ) const  [inline]
```

7.67.2.2 operator==()

```
bool qevercloud::NoteEmailParameters::operator== (
            const NoteEmailParameters & other ) const  [inline]
```

7.67.2.3 print()

```
virtual void qevercloud::NoteEmailParameters::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.67.3 Member Data Documentation

7.67.3.1 ccAddresses

```
Optional< QStringList > qevercloud::NoteEmailParameters::ccAddresses
```

If provided, this should contain a list of the SMTP email addresses that should be included in the "Cc:" line of the email. Callers must specify at least one "to" or "cc" email address.

7.67.3.2 guid

```
Optional< QString > qevercloud::NoteEmailParameters::guid
```

If set, this must be the GUID of a note within the user's account that should be retrieved from the service and sent as email. If not set, the 'note' field must be provided instead.

7.67.3.3 localData

```
EverCloudLocalData qevercloud::NoteEmailParameters::localData
```

See the declaration of EverCloudLocalData for details

7.67.3.4 message

```
Optional< QString > qevercloud::NoteEmailParameters::message
```

If provided, this is additional personal text that should be included into the email as a message from the owner to the recipient(s).

7.67.3.5 note

```
Optional< Note > qevercloud::NoteEmailParameters::note
```

If the 'guid' field is not set, this field must be provided, including the full contents of the note note (and all of its Resources) to send. This can be used for a Note that as not been created in the service, for example by a local client with local notes.

7.67.3.6 subject

```
Optional< QString > qevercloud::NoteEmailParameters::subject
```

If provided, this should contain the subject line of the email that will be sent. If not provided, the title of the note will be used as the subject of the email.

7.67.3.7 toAddresses

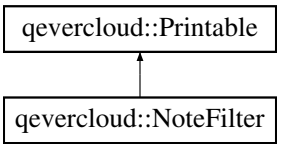
```
Optional< QStringList > qevercloud::NoteEmailParameters::toAddresses
```

If provided, this should contain a list of the SMTP email addresses that should be included in the "To:" line of the email. Callers must specify at least one "to" or "cc" email address.

7.68 qevercloud::NoteFilter Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::NoteFilter:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NoteFilter &other) const
- bool operator!= (const NoteFilter &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< qint32 > order
- Optional< bool > ascending
- Optional< QString > words
- Optional< Guid > notebookGuid
- Optional< QList< Guid > > tagGuids
- Optional< QString > timeZone
- Optional< bool > inactive
- Optional< QString > emphasized
- Optional< bool > includeAllReadableNotebooks
- Optional< bool > includeAllReadableWorkspaces
- Optional< QString > context
- Optional< QString > rawWords
- Optional< QByteArray > searchContextBytes

Properties

- OptionalQList< Guid > tagGuids

7.68.1 Detailed Description

A list of criteria that are used to indicate which notes are desired from the account. This is used in queries to the NoteStore to determine which notes should be retrieved.

7.68.2 Member Function Documentation

7.68.2.1 operator"!=()

```
bool qevercloud::NoteFilter::operator!= (
            const NoteFilter & other ) const  [inline]
```

7.68.2.2 operator==()

```
bool qevercloud::NoteFilter::operator== (
            const NoteFilter & other ) const  [inline]
```

7.68.2.3 print()

```
virtual void qevercloud::NoteFilter::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.68.3 Member Data Documentation

7.68.3.1 ascending

```
Optional< bool > qevercloud::NoteFilter::ascending
```

If true, the results will be ascending in the requested sort order. If false, the results will be descending.

7.68.3.2 context

```
Optional< QString > qevercloud::NoteFilter::context
```

Specifies the context to consider when determining result ranking. Clients must leave this value unset unless they wish to explicitly specify a known non-default context.

7.68.3.3 emphasized

```
Optional< QString > qevercloud::NoteFilter::emphasized
```

If present, a search query string that may or may not influence the notes to be returned, both in terms of coverage as well as of order. Think of it as a wish list, not a requirement. Accepts the full search grammar documented in the Evernote API Overview.

7.68.3.4 inactive

```
Optional< bool > qevercloud::NoteFilter::inactive
```

If true, then only notes that are not active (i.e. notes in the Trash) will be returned. Otherwise, only active notes will be returned. There is no way to find both active and inactive notes in a single query.

7.68.3.5 includeAllReadableNotebooks

```
Optional< bool > qevercloud::NoteFilter::includeAllReadableNotebooks
```

If true, then the search will include all business notebooks that are readable by the user. A business authentication token must be supplied for this option to take effect when calling search APIs.

7.68.3.6 includeAllReadableWorkspaces

```
Optional< bool > qevercloud::NoteFilter::includeAllReadableWorkspaces
```

If true, then the search will include all workspaces that are readable by the user. A business authentication token must be supplied for this option to take effect when calling search APIs.

7.68.3.7 localData

```
EverCloudLocalData qevercloud::NoteFilter::localData
```

See the declaration of EverCloudLocalData for details

7.68.3.8 notebookGuid

```
Optional< Guid > qevercloud::NoteFilter::notebookGuid
```

If present, the Guid of the notebook that must contain the notes.

7.68.3.9 order

```
Optional< qint32 > qevercloud::NoteFilter::order
```

The NoteSortOrder value indicating what criterion should be used to sort the results of the filter.

7.68.3.10 rawWords

```
Optional< QString > qevercloud::NoteFilter::rawWords
```

If present, the raw user query input. Accepts the full search grammar documented in the Evernote API Overview.

7.68.3.11 searchContextBytes

```
Optional< QByteArray > qevercloud::NoteFilter::searchContextBytes
```

Specifies the correlating information about the current search session, in byte array. If this request is not for the first page of search results, the client should populate this field with the value of searchContextBytes from the NotesMetadataList of the original search response.

7.68.3.12 tagGuids

```
Optional<QList<Guid> > qevercloud::NoteFilter::tagGuids
```

If present, the list of tags (by GUID) that must be present on the notes.

7.68.3.13 timeZone

```
Optional< QString > qevercloud::NoteFilter::timeZone
```

The zone ID for the user, which will be used to interpret any dates or times in the queries that do not include their desired zone information. For example, if a query requests notes created "yesterday", this will be evaluated from the provided time zone, if provided. The format must be encoded as a standard zone ID such as "America/Los_↩ Angeles".

7.68.3.14 words

```
Optional< QString > qevercloud::NoteFilter::words
```

If present, a search query string that will filter the set of notes to be returned. Accepts the full search grammar documented in the Evernote API Overview.

7.68.4 Property Documentation

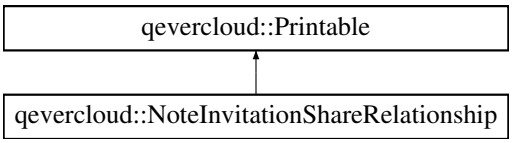
7.68.4.1 tagGuids

```
OptionalQList<Guid> qevercloud::NoteFilter::tagGuids
```

7.69 qevercloud::NoteInvitationShareRelationship Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NoteInvitationShareRelationship:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NoteInvitationShareRelationship &other) const
- bool operator!= (const NoteInvitationShareRelationship &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > displayName
- Optional< IdentityID > recipientIdentityId
- Optional< SharedNotePrivilegeLevel > privilege
- Optional< UserID > sharerUserId

7.69.1 Detailed Description

Describes an invitation to a person to use their Evernote credentials to gain access to a note belonging to another user.

7.69.2 Member Function Documentation

7.69.2.1 operator"!=()

```
bool qevercloud::NoteInvitationShareRelationship::operator!= (
            const NoteInvitationShareRelationship & other ) const  [inline]
```

7.69.2.2 operator==()

```
bool qevercloud::NoteInvitationShareRelationship::operator== (
            const NoteInvitationShareRelationship & other ) const  [inline]
```

7.69.2.3 print()

```
virtual void qevercloud::NoteInvitationShareRelationship::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.69.3 Member Data Documentation

7.69.3.1 displayName

```
Optional< QString > qevercloud::NoteInvitationShareRelationship::displayName
```

The string that clients should show to users to represent this invitation.

7.69.3.2 localData

```
EverCloudLocalData qevercloud::NoteInvitationShareRelationship::localData
```

See the declaration of EverCloudLocalData for details

7.69.3.3 privilege

```
Optional< SharedNotePrivilegeLevel > qevercloud::NoteInvitationShareRelationship::privilege
```

The privilege level that the recipient will be granted when they accept this invitation. If the user already has a higher privilege to access this note then this will not affect the recipient's privileges.

7.69.3.4 recipientIdentityId

```
Optional< IdentityID > qevercloud::NoteInvitationShareRelationship::recipientIdentityId
```

Identifies the identity of the invitation recipient. Once the identity has been claimed by an Evernote user and they have accessed the note at least once, the invitation will be used up and will no longer be returned by the service to clients. Instead, that recipient will be included in the list of NoteMemberShareRelationships.

7.69.3.5 sharerUserId

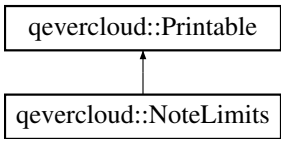
```
Optional< UserID > qevercloud::NoteInvitationShareRelationship::sharerUserId
```

The user id of the user who most recently shared this note to this recipient. This field is used by the service to convey information to the user, so clients should treat it as read-only.

7.70 qevercloud::NoteLimits Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NoteLimits:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NoteLimits &other) const
- bool operator!= (const NoteLimits &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< qint32 > noteResourceCountMax
- Optional< qint64 > uploadLimit
- Optional< qint64 > resourceSizeMax
- Optional< qint64 > noteSizeMax
- Optional< qint64 > uploaded

7.70.1 Detailed Description

Represents the owner's account related limits on a Note. The field uploaded represents the total number of bytes that have been uploaded to this account and is taken from the SyncState struct. All other fields represent account related limits and are taken from the AccountLimits struct.

See SyncState and AccountLimits struct field definitions for more details.

7.70.2 Member Function Documentation

7.70.2.1 operator"!=()

```
bool qevercloud::NoteLimits::operator!= (
            const NoteLimits & other ) const  [inline]
```

7.70.2.2 operator==()

```
bool qevercloud::NoteLimits::operator== (
            const NoteLimits & other ) const  [inline]
```

7.70.2.3 print()

```
virtual void qevercloud::NoteLimits::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.70.3 Member Data Documentation

7.70.3.1 localData

```
EverCloudLocalData qevercloud::NoteLimits::localData
```

See the declaration of EverCloudLocalData for details

7.70.3.2 noteResourceCountMax

```
Optional< qint32 > qevercloud::NoteLimits::noteResourceCountMax
```

NOT DOCUMENTED

7.70.3.3 noteSizeMax

```
Optional< qint64 > qevercloud::NoteLimits::noteSizeMax
```

NOT DOCUMENTED

7.70.3.4 resourceSizeMax

```
Optional< qint64 > qevercloud::NoteLimits::resourceSizeMax
```

NOT DOCUMENTED

7.70.3.5 uploaded

```
Optional< qint64 > qevercloud::NoteLimits::uploaded
```

NOT DOCUMENTED

7.70.3.6 uploadLimit

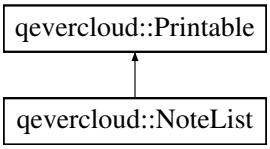
```
Optional< qint64 > qevercloud::NoteLimits::uploadLimit
```

NOT DOCUMENTED

7.71 qevercloud::NoteList Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NoteList:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NoteList &other) const
- bool operator!= (const NoteList &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- qint32 startIndex = 0
- qint32 totalNotes = 0
- QList< Note > notes
- Optional< QStringList > stoppedWords
- Optional< QStringList > searchedWords
- Optional< qint32 > updateCount
- Optional< QByteArray > searchContextBytes
- Optional< QString > debugInfo

7.71.1 Detailed Description

A small structure for returning a list of notes out of a larger set.

7.71.2 Member Function Documentation

7.71.2.1 operator"!=()

```
bool qevercloud::NoteList::operator!= (
            const NoteList & other ) const  [inline]
```

7.71.2.2 operator==()

```
bool qevercloud::NoteList::operator== (
            const NoteList & other ) const  [inline]
```

7.71.2.3 print()

```
virtual void qevercloud::NoteList::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.71.3 Member Data Documentation

7.71.3.1 debugInfo

```
Optional< QString > qevercloud::NoteList::debugInfo
```

Depends on the value of `context` in NoteFilter, this field may contain debug information if the service decides to do so.

7.71.3.2 localData

```
EverCloudLocalData qevercloud::NoteList::localData
```

See the declaration of EverCloudLocalData for details

7.71.3.3 notes

```
QList< Note > qevercloud::NoteList::notes
```

The list of notes from this range. The Notes will include all metadata (attributes, resources, etc.), but will not include the ENML content of the note or the binary contents of any resources.

7.71.3.4 searchContextBytes

```
Optional< QByteArray > qevercloud::NoteList::searchContextBytes
```

Specifies the correlating information about the current search session, in byte array.

7.71.3.5 searchedWords

```
Optional< QStringList > qevercloud::NoteList::searchedWords
```

If the NoteList was produced using a text based search query that included viable search words or quoted expressions, this will include a list of those words. Any stopped words will not be included in this list.

7.71.3.6 startIndex

```
qint32 qevercloud::NoteList::startIndex = 0
```

The starting index within the overall set of notes. This is also the number of notes that are "before" this list in the set.

7.71.3.7 stoppedWords

```
Optional< QStringList > qevercloud::NoteList::stoppedWords
```

If the NoteList was produced using a text based search query that included words that are not indexed or searched by the service, this will include a list of those ignored words.

7.71.3.8 totalNotes

```
qint32 qevercloud::NoteList::totalNotes = 0
```

The number of notes in the larger set. This can be used to calculate how many notes are "after" this note in the set. (I.e. remaining = totalNotes - (startIndex + notes.length))

7.71.3.9 updateCount

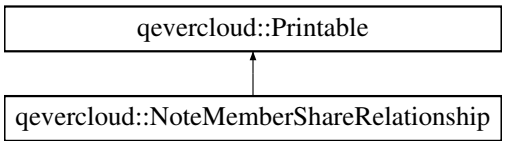
```
Optional< qint32 > qevercloud::NoteList::updateCount
```

Indicates the total number of transactions that have been committed within the account. This reflects (for example) the number of discrete additions or modifications that have been made to the data in this account (tags, notes, resources, etc.). This number is the "high water mark" for Update Sequence Numbers (USN) within the account.

7.72 qevercloud::NoteMemberShareRelationship Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NoteMemberShareRelationship:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NoteMemberShareRelationship &other) const
- bool operator!= (const NoteMemberShareRelationship &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > displayName
- Optional< UserID > recipientUserId
- Optional< SharedNotePrivilegeLevel > privilege
- Optional< NoteShareRelationshipRestrictions > restrictions
- Optional< UserID > sharerUserId

7.72.1 Detailed Description

Describes the association between a Note and an Evernote User who is a member of that note.

7.72.2 Member Function Documentation

7.72.2.1 operator"!=()

```
bool qevercloud::NoteMemberShareRelationship::operator!= (
            const NoteMemberShareRelationship & other ) const  [inline]
```

7.72.2.2 operator==()

```
bool qevercloud::NoteMemberShareRelationship::operator== (
            const NoteMemberShareRelationship & other ) const  [inline]
```

7.72.2.3 print()

```
virtual void qevercloud::NoteMemberShareRelationship::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.72.3 Member Data Documentation

7.72.3.1 displayName

```
Optional< QString > qevercloud::NoteMemberShareRelationship::displayName
```

The string that clients should show to users to represent this member.

7.72.3.2 localData

```
EverCloudLocalData qevercloud::NoteMemberShareRelationship::localData
```

See the declaration of EverCloudLocalData for details

7.72.3.3 privilege

```
Optional< SharedNotePrivilegeLevel > qevercloud::NoteMemberShareRelationship::privilege
```

The privilege at which the member can access the note, which is the best privilege granted to the user across all of their individual shares for this note. This field is used by the service to convey information to the user, so clients should treat it as read-only.

7.72.3.4 recipientUserId

```
Optional< UserID > qevercloud::NoteMemberShareRelationship::recipientUserId
```

The Evernote UserID of the user who is a member to the note.

7.72.3.5 restrictions

```
Optional< NoteShareRelationshipRestrictions > qevercloud::NoteMemberShareRelationship::restrictions
```

The restrictions on which privileges may be individually assigned to the recipient of this share relationship. This field is used by the service to convey information to the user, so clients should treat it as read-only.

7.72.3.6 sharerUserId

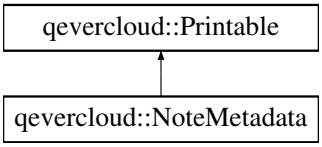
```
Optional< UserID > qevercloud::NoteMemberShareRelationship::sharerUserId
```

The user id of the user who most recently shared the note with this user. This field is used by the service to convey information to the user, so clients should treat it as read-only.

7.73 qevercloud::NoteMetadata Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NoteMetadata:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NoteMetadata &other) const
- bool operator!= (const NoteMetadata &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Guid guid
- Optional< QString > title
- Optional< qint32 > contentLength
- Optional< Timestamp > created
- Optional< Timestamp > updated
- Optional< Timestamp > deleted
- Optional< qint32 > updateSequenceNum
- Optional< QString > notebookGuid
- Optional< QList< Guid > > tagGuids
- Optional< NoteAttributes > attributes
- Optional< QString > largestResourceMime
- Optional< qint32 > largestResourceSize

Properties

- OptionalQList< Guid > tagGuids

7.73.1 Detailed Description

This structure is used in the set of results returned by the findNotesMetadata function. It represents the high-level information about a single Note, without some of the larger deep structure. This allows for the information about a list of Notes to be returned relatively quickly with less marshalling and data transfer to remote clients. Most fields in this structure are identical to the corresponding field in the Note structure, with the exception of:

7.73.2 Member Function Documentation

7.73.2.1 operator"!=()

```
bool qevercloud::NoteMetadata::operator!= (
            const NoteMetadata & other ) const  [inline]
```

7.73.2.2 operator==()

```
bool qevercloud::NoteMetadata::operator== (
            const NoteMetadata & other ) const  [inline]
```

7.73.2.3 print()

```
virtual void qevercloud::NoteMetadata::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.73.3 Member Data Documentation

7.73.3.1 attributes

```
Optional< NoteAttributes > qevercloud::NoteMetadata::attributes
```

NOT DOCUMENTED

7.73.3.2 contentLength

```
Optional< qint32 > qevercloud::NoteMetadata::contentLength
```

NOT DOCUMENTED

7.73.3.3 created

```
Optional< Timestamp > qevercloud::NoteMetadata::created
```

NOT DOCUMENTED

7.73.3.4 deleted

```
Optional< Timestamp > qevercloud::NoteMetadata::deleted
```

NOT DOCUMENTED

7.73.3.5 guid

```
Guid qevercloud::NoteMetadata::guid
```

NOT DOCUMENTED

7.73.3.6 largestResourceMime

```
Optional< QString > qevercloud::NoteMetadata::largestResourceMime
```

If set, then this will contain the MIME type of the largest Resource (in bytes) within the Note. This may be useful, for example, to choose an appropriate icon or thumbnail to represent the Note.

7.73.3.7 largestResourceSize

```
Optional< qint32 > qevercloud::NoteMetadata::largestResourceSize
```

If set, this will contain the size of the largest Resource file, in bytes, within the Note. This may be useful, for example, to decide whether to ask the server for a thumbnail to represent the Note.

7.73.3.8 localData

```
EverCloudLocalData qevercloud::NoteMetadata::localData
```

See the declaration of EverCloudLocalData for details

7.73.3.9 notebookGuid

```
Optional< QString > qevercloud::NoteMetadata::notebookGuid
```

NOT DOCUMENTED

7.73.3.10 tagGuids

```
Optional<QList<Guid> > qevercloud::NoteMetadata::tagGuids
```

NOT DOCUMENTED

7.73.3.11 title

```
Optional< QString > qevercloud::NoteMetadata::title
```

NOT DOCUMENTED

7.73.3.12 updated

```
Optional< Timestamp > qevercloud::NoteMetadata::updated
```

NOT DOCUMENTED

7.73.3.13 updateSequenceNum

```
Optional< qint32 > qevercloud::NoteMetadata::updateSequenceNum
```

NOT DOCUMENTED

7.73.4 Property Documentation

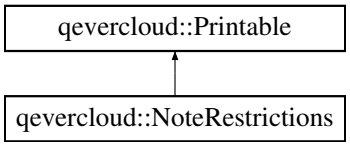
7.73.4.1 tagGuids

```
OptionalQList<Guid> qevercloud::NoteMetadata::tagGuids
```

7.74 qevercloud::NoteRestrictions Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NoteRestrictions:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NoteRestrictions &other) const
- bool operator!= (const NoteRestrictions &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< bool > noUpdateTitle
- Optional< bool > noUpdateContent
- Optional< bool > noEmail
- Optional< bool > noShare
- Optional< bool > noSharePublicly

7.74.1 Detailed Description

This structure captures information about the operations that cannot be performed on a given note that has been shared with a recipient via a SharedNote. The following operations are **never** allowed based on SharedNotes, and as such are left out of the NoteRestrictions structure for brevity:

- Expunging a note (NoteStore.expungeNote)
- Moving a note to the trash (Note.active)
- Updating a note's notebook (Note.notebookGuid)
- Updating a note's tags (Note.tagGuids, Note.tagNames)
- Updating a note's attributes (Note.attributes)
- Sharing a note with the business (NoteStore.shareNoteWithBusiness
- Getting a note's version history (NoteStore.listNoteVersions, NoteStore.getNoteVersion)

When a client has permission to update a note's title or content, it may also update the Note.updated timestamp.

This structure reflects only the privileges / restrictions conveyed by the SharedNote. It does not incorporate privileges conveyed by a potential SharedNotebook to the same recipient. As such, the actual permissions that the recipient has on the note may differ from the permissions expressed in this structure.

For example, consider a user with read-only access to a shared notebook, and a read-write share of a specific note in the notebook. The note restrictions would contain noUpdateTitle = false, while the notebook restrictions would contain noUpdateNotes = true. In this case, the user is allowed to update the note title based on the note restrictions.

Alternatively, consider a user with read-write access to a shared notebook, and a read-only share of a specific note in that notebook. The note restrictions would contain noUpdateTitle = true, while the notebook restrictions would contain noUpdateNotes = false. In this case, the user would have full edit permissions on the note based on the notebook restrictions.

7.74.2 Member Function Documentation

7.74.2.1 operator"!=()

```
bool qevercloud::NoteRestrictions::operator!= (
            const NoteRestrictions & other ) const  [inline]
```

7.74.2.2 operator==()

```
bool qevercloud::NoteRestrictions::operator== (
            const NoteRestrictions & other ) const  [inline]
```

7.74.2.3 print()

```
virtual void qevercloud::NoteRestrictions::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.74.3 Member Data Documentation

7.74.3.1 localData

```
EverCloudLocalData qevercloud::NoteRestrictions::localData
```

See the declaration of EverCloudLocalData for details

7.74.3.2 noEmail

```
Optional< bool > qevercloud::NoteRestrictions::noEmail
```

The client may not email the note (NoteStore.emailNote).

7.74.3.3 noShare

```
Optional< bool > qevercloud::NoteRestrictions::noShare
```

The client may not share the note with specific recipients (NoteStore.createOrUpdateSharedNotes).

7.74.3.4 noSharePublicly

```
Optional< bool > qevercloud::NoteRestrictions::noSharePublicly
```

The client may not make the note public (NoteStore.shareNote).

7.74.3.5 noUpdateContent

```
Optional< bool > qevercloud::NoteRestrictions::noUpdateContent
```

NOT DOCUMENTED

7.74.3.6 noUpdateTitle

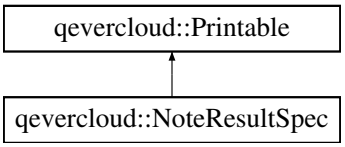
```
Optional< bool > qevercloud::NoteRestrictions::noUpdateTitle
```

The client may not update the note's title (Note.title).

7.75 qevercloud::NoteResultSpec Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NoteResultSpec:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NoteResultSpec &other) const
- bool operator!= (const NoteResultSpec &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< bool > includeContent
- Optional< bool > includeResourcesData
- Optional< bool > includeResourcesRecognition
- Optional< bool > includeResourcesAlternateData
- Optional< bool > includeSharedNotes
- Optional< bool > includeNoteAppDataValues
- Optional< bool > includeResourceAppDataValues
- Optional< bool > includeAccountLimits

7.75.1 Detailed Description

This structure is provided to the getNoteWithResultSpec function to specify the subset of fields that should be included in the Note that is returned. This allows clients to request the minimum set of information that they require when retrieving a note, reducing the size of the response and improving the response time.

If one of the fields in this spec is not set, then it will be treated as 'false' by the service, so that the default behavior is to include none of the fields below in the Note.

7.75.2 Member Function Documentation

7.75.2.1 operator"!=()

```
bool qevercloud::NoteResultSpec::operator!= (
            const NoteResultSpec & other ) const  [inline]
```

7.75.2.2 operator==()

```
bool qevercloud::NoteResultSpec::operator== (
            const NoteResultSpec & other ) const  [inline]
```

7.75.2.3 print()

```
virtual void qevercloud::NoteResultSpec::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.75.3 Member Data Documentation

7.75.3.1 includeAccountLimits

```
Optional< bool > qevercloud::NoteResultSpec::includeAccountLimits
```

If true, the Note.limits field will be populated with the note owner's account limits.

7.75.3.2 includeContent

```
Optional< bool > qevercloud::NoteResultSpec::includeContent
```

If true, the Note.content field will be populated with the note's ENML contents.

7.75.3.3 includeNoteAppDataValues

```
Optional< bool > qevercloud::NoteResultSpec::includeNoteAppDataValues
```

If true, the Note.attributes.applicationData.fullMap field will be populated.

7.75.3.4 includeResourceAppDataValues

```
Optional< bool > qevercloud::NoteResultSpec::includeResourceAppDataValues
```

If true, the Note.resource.attributes.applicationData.fullMap field will be populated.

7.75.3.5 includeResourcesAlternateData

```
Optional< bool > qevercloud::NoteResultSpec::includeResourcesAlternateData
```

If true, any Resource elements will include the binary contents of their 'alternateData' field's body, if an alternate form is available.

7.75.3.6 includeResourcesData

```
Optional< bool > qevercloud::NoteResultSpec::includeResourcesData
```

If true, any Resource elements will include the binary contents of their 'data' field's body.

7.75.3.7 includeResourcesRecognition

```
Optional< bool > qevercloud::NoteResultSpec::includeResourcesRecognition
```

If true, any Resource elements will include the binary contents of their 'recognition' field's body if recognition data is available.

7.75.3.8 includeSharedNotes

```
Optional< bool > qevercloud::NoteResultSpec::includeSharedNotes
```

If true, the Note.sharedNotes field will be populated with the note's shares.

7.75.3.9 localData

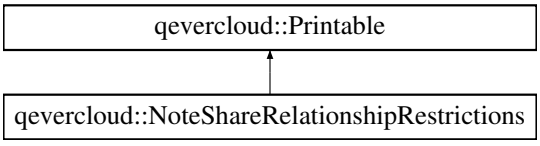
```
EverCloudLocalData qevercloud::NoteResultSpec::localData
```

See the declaration of EverCloudLocalData for details

7.76 qevercloud::NoteShareRelationshipRestrictions Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NoteShareRelationshipRestrictions:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NoteShareRelationshipRestrictions &other) const
- bool operator!= (const NoteShareRelationshipRestrictions &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< bool > noSetReadNote
- Optional< bool > noSetModifyNote
- Optional< bool > noSetFullAccess

7.76.1 Detailed Description

This structure is used by the service to communicate to clients, via getNoteShareRelationships, which privilege levels are assignable to the target of a note share relationship.

7.76.2 Member Function Documentation

7.76.2.1 operator"!=()

```
bool qevercloud::NoteShareRelationshipRestrictions::operator!= (
            const NoteShareRelationshipRestrictions & other ) const  [inline]
```

7.76.2.2 operator==()

```
bool qevercloud::NoteShareRelationshipRestrictions::operator== (
            const NoteShareRelationshipRestrictions & other ) const  [inline]
```

7.76.2.3 print()

```
virtual void qevercloud::NoteShareRelationshipRestrictions::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.76.3 Member Data Documentation

7.76.3.1 localData

```
EverCloudLocalData qevercloud::NoteShareRelationshipRestrictions::localData
```

See the declaration of EverCloudLocalData for details

7.76.3.2 noSetFullAccess

```
Optional< bool > qevercloud::NoteShareRelationshipRestrictions::noSetFullAccess
```

This value is true if the user is not allowed to set the privilege level to SharedNotePrivilegeLevel.FULL_ACCESS.

7.76.3.3 noSetModifyNote

```
Optional< bool > qevercloud::NoteShareRelationshipRestrictions::noSetModifyNote
```

This value is true if the user is not allowed to set the privilege level to SharedNotePrivilegeLevel.MODIFY_NOTE.

7.76.3.4 noSetReadNote

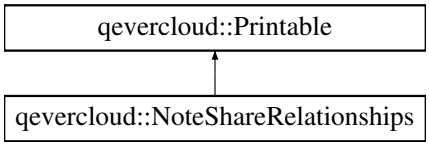
```
Optional< bool > qevercloud::NoteShareRelationshipRestrictions::noSetReadNote
```

This value is true if the user is not allowed to set the privilege level to SharedNotePrivilegeLevel.READ_NOTE.

7.77 qevercloud::NoteShareRelationships Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NoteShareRelationships:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NoteShareRelationships &other) const
- bool operator!= (const NoteShareRelationships &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QList< NoteInvitationShareRelationship > > invitations
- Optional< QList< NoteMemberShareRelationship > > memberships
- Optional< NoteShareRelationshipRestrictions > invitationRestrictions

Properties

- OptionalQList< NoteInvitationShareRelationship > invitations
- OptionalQList< NoteMemberShareRelationship > memberships

7.77.1 Detailed Description

Captures a collection of share relationships for a single note, for example, as returned by the getNoteShares method. The share relationships fall into two broad categories: members, and invitations that can be used to become members.

7.77.2 Member Function Documentation

7.77.2.1 operator"!=()

```
bool qevercloud::NoteShareRelationships::operator!= (
            const NoteShareRelationships & other ) const  [inline]
```

7.77.2.2 operator==()

```
bool qevercloud::NoteShareRelationships::operator== (
            const NoteShareRelationships & other ) const  [inline]
```

7.77.2.3 print()

```
virtual void qevercloud::NoteShareRelationships::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.77.3 Member Data Documentation

7.77.3.1 invitationRestrictions

```
Optional< NoteShareRelationshipRestrictions > qevercloud::NoteShareRelationships::invitation↩
Restrictions
```

NOT DOCUMENTED

7.77.3.2 invitations

```
Optional<QList<NoteInvitationShareRelationship> > qevercloud::NoteShareRelationships::invitations
```

A list of open invitations that can be redeemed into memberships to the note.

7.77.3.3 localData

```
EverCloudLocalData qevercloud::NoteShareRelationships::localData
```

See the declaration of EverCloudLocalData for details

7.77.3.4 memberships

```
Optional<QList<NoteMemberShareRelationship> > qevercloud::NoteShareRelationships::memberships
```

A list of memberships of the noteb. A member is identified by their Evernote UserID and has rights to access the note.

7.77.4 Property Documentation

7.77.4.1 invitations

```
OptionalQList<NoteInvitationShareRelationship> qevercloud::NoteShareRelationships::invitations
```

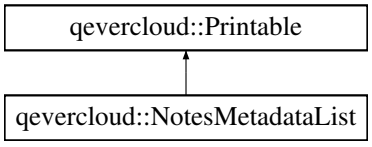
7.77.4.2 memberships

```
OptionalQList<NoteMemberShareRelationship> qevercloud::NoteShareRelationships::memberships
```

7.78 qevercloud::NotesMetadataList Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NotesMetadataList:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NotesMetadataList &other) const
- bool operator!= (const NotesMetadataList &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- qint32 startIndex = 0
- qint32 totalNotes = 0
- QList< NoteMetadata > notes
- Optional< QStringList > stoppedWords
- Optional< QStringList > searchedWords
- Optional< qint32 > updateCount
- Optional< QByteArray > searchContextBytes
- Optional< QString > debugInfo

7.78.1 Detailed Description

This structure is returned from calls to the findNotesMetadata function to give the high-level metadata about a subset of Notes that are found to match a specified NoteFilter in a search.

7.78.2 Member Function Documentation

7.78.2.1 operator"!=()

```
bool qevercloud::NotesMetadataList::operator!= (
            const NotesMetadataList & other ) const  [inline]
```

7.78.2.2 operator==()

```
bool qevercloud::NotesMetadataList::operator== (
            const NotesMetadataList & other ) const  [inline]
```

7.78.2.3 print()

```
virtual void qevercloud::NotesMetadataList::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.78.3 Member Data Documentation

7.78.3.1 debugInfo

```
Optional< QString > qevercloud::NotesMetadataList::debugInfo
```

Depends on the value of `context` in NoteFilter, this field may contain debug information if the service decides to do so.

7.78.3.2 localData

```
EverCloudLocalData qevercloud::NotesMetadataList::localData
```

See the declaration of EverCloudLocalData for details

7.78.3.3 notes

```
QList< NoteMetadata > qevercloud::NotesMetadataList::notes
```

The list of metadata for Notes in this range. The set of optional fields that are set in each metadata structure will depend on the NotesMetadataResultSpec provided by the caller when the search was performed. Only the 'guid' field will be guaranteed to be set in each Note.

7.78.3.4 searchContextBytes

```
Optional< QByteArray > qevercloud::NotesMetadataList::searchContextBytes
```

Specifies the correlating information about the current search session, in byte array.

7.78.3.5 searchedWords

```
Optional< QStringList > qevercloud::NotesMetadataList::searchedWords
```

If the NoteList was produced using a text based search query that included viable search words or quoted expressions, this will include a list of those words. Any stopped words will not be included in this list.

7.78.3.6 startIndex

```
qint32 qevercloud::NotesMetadataList::startIndex = 0
```

The starting index within the overall set of notes. This is also the number of notes that are "before" this list in the set.

7.78.3.7 stoppedWords

```
Optional< QStringList > qevercloud::NotesMetadataList::stoppedWords
```

If the NoteList was produced using a text based search query that included words that are not indexed or searched by the service, this will include a list of those ignored words.

7.78.3.8 totalNotes

```
qint32 qevercloud::NotesMetadataList::totalNotes = 0
```

The number of notes in the larger set. This can be used to calculate how many notes are "after" this note in the set. (I.e. remaining = totalNotes - (startIndex + notes.length))

7.78.3.9 updateCount

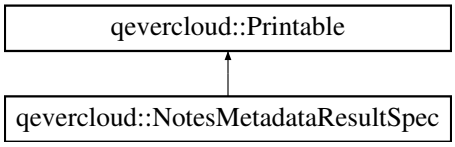
```
Optional< qint32 > qevercloud::NotesMetadataList::updateCount
```

Indicates the total number of transactions that have been committed within the account. This reflects (for example) the number of discrete additions or modifications that have been made to the data in this account (tags, notes, resources, etc.). This number is the "high water mark" for Update Sequence Numbers (USN) within the account.

7.79 qevercloud::NotesMetadataResultSpec Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::NotesMetadataResultSpec:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NotesMetadataResultSpec &other) const
- bool operator!= (const NotesMetadataResultSpec &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< bool > includeTitle
- Optional< bool > includeContentLength
- Optional< bool > includeCreated
- Optional< bool > includeUpdated
- Optional< bool > includeDeleted
- Optional< bool > includeUpdateSequenceNum
- Optional< bool > includeNotebookGuid
- Optional< bool > includeTagGuids
- Optional< bool > includeAttributes
- Optional< bool > includeLargestResourceMime
- Optional< bool > includeLargestResourceSize

7.79.1 Detailed Description

This structure is provided to the findNotesMetadata function to specify the subset of fields that should be included in each NoteMetadata element that is returned in the NotesMetadataList. Each field on this structure is a boolean flag that indicates whether the corresponding field should be included in the NoteMetadata structure when it is returned. For example, if the 'includeTitle' field is set on this structure when calling findNotesMetadata, then each NoteMetadata in the list should have its 'title' field set. If one of the fields in this spec is not set, then it will be treated as 'false' by the server, so the default behavior is to include nothing in replies (but the mandatory GUID)

7.79.2 Member Function Documentation

7.79.2.1 operator"!=()

```
bool qevercloud::NotesMetadataResultSpec::operator!= (
            const NotesMetadataResultSpec & other ) const  [inline]
```

7.79.2.2 operator==()

```
bool qevercloud::NotesMetadataResultSpec::operator== (
            const NotesMetadataResultSpec & other ) const  [inline]
```

7.79.2.3 print()

```
virtual void qevercloud::NotesMetadataResultSpec::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.79.3 Member Data Documentation

7.79.3.1 includeAttributes

```
Optional< bool > qevercloud::NotesMetadataResultSpec::includeAttributes
```

NOT DOCUMENTED

7.79.3.2 includeContentLength

```
Optional< bool > qevercloud::NotesMetadataResultSpec::includeContentLength
```

NOT DOCUMENTED

7.79.3.3 includeCreated

```
Optional< bool > qevercloud::NotesMetadataResultSpec::includeCreated
```

NOT DOCUMENTED

7.79.3.4 includeDeleted

```
Optional< bool > qevercloud::NotesMetadataResultSpec::includeDeleted
```

NOT DOCUMENTED

7.79.3.5 includeLargestResourceMime

```
Optional< bool > qevercloud::NotesMetadataResultSpec::includeLargestResourceMime
```

NOT DOCUMENTED

7.79.3.6 includeLargestResourceSize

```
Optional< bool > qevercloud::NotesMetadataResultSpec::includeLargestResourceSize
```

NOT DOCUMENTED

7.79.3.7 includeNotebookGuid

```
Optional< bool > qevercloud::NotesMetadataResultSpec::includeNotebookGuid
```

NOT DOCUMENTED

7.79.3.8 includeTagGuids

```
Optional< bool > qevercloud::NotesMetadataResultSpec::includeTagGuids
```

NOT DOCUMENTED

7.79.3.9 includeTitle

```
Optional< bool > qevercloud::NotesMetadataResultSpec::includeTitle
```

NOT DOCUMENTED

7.79.3.10 includeUpdated

```
Optional< bool > qevercloud::NotesMetadataResultSpec::includeUpdated
```

NOT DOCUMENTED

7.79.3.11 includeUpdateSequenceNum

```
Optional< bool > qevercloud::NotesMetadataResultSpec::includeUpdateSequenceNum
```

NOT DOCUMENTED

7.79.3.12 localData

```
EverCloudLocalData qevercloud::NotesMetadataResultSpec::localData
```

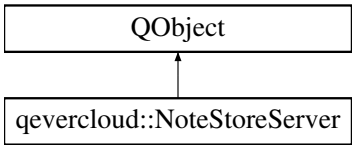
See the declaration of EverCloudLocalData for details

7.80 qevercloud::NoteStoreServer Class Reference

The NoteStoreServer class represents customizable server for NoteStore requests. It is primarily used for testing of QEverCloud.

```
#include <Servers.h>
```

Inheritance diagram for qevercloud::NoteStoreServer:



Public Slots

- void onRequest (QByteArray data)
- void onGetSyncStateRequestReady (SyncState value, EverCloudExceptionDataPtr exceptionData)
- void onGetFilteredSyncChunkRequestReady (SyncChunk value, EverCloudExceptionDataPtr exception↩ Data)
- void onGetLinkedNotebookSyncStateRequestReady (SyncState value, EverCloudExceptionDataPtr exceptionData)
- void onGetLinkedNotebookSyncChunkRequestReady (SyncChunk value, EverCloudExceptionDataPtr exceptionData)
- void onListNotebooksRequestReady (QList< Notebook > value, EverCloudExceptionDataPtr exceptionData)
- void onListAccessibleBusinessNotebooksRequestReady (QList< Notebook > value, EverCloudExceptionDataPtr exceptionData)
- void onGetNotebookRequestReady (Notebook value, EverCloudExceptionDataPtr exceptionData)
- void onGetDefaultNotebookRequestReady (Notebook value, EverCloudExceptionDataPtr exceptionData)
- void onCreateNotebookRequestReady (Notebook value, EverCloudExceptionDataPtr exceptionData)
- void onUpdateNotebookRequestReady (qint32 value, EverCloudExceptionDataPtr exceptionData)
- void onExpungeNotebookRequestReady (qint32 value, EverCloudExceptionDataPtr exceptionData)
- void onListTagsRequestReady (QList< Tag > value, EverCloudExceptionDataPtr exceptionData)
- void onListTagsByNotebookRequestReady (QList< Tag > value, EverCloudExceptionDataPtr exception↩ Data)
- void onGetTagRequestReady (Tag value, EverCloudExceptionDataPtr exceptionData)
- void onCreateTagRequestReady (Tag value, EverCloudExceptionDataPtr exceptionData)
- void onUpdateTagRequestReady (qint32 value, EverCloudExceptionDataPtr exceptionData)
- void onUntagAllRequestReady (EverCloudExceptionDataPtr exceptionData)
- void onExpungeTagRequestReady (qint32 value, EverCloudExceptionDataPtr exceptionData)
- void onListSearchesRequestReady (QList< SavedSearch > value, EverCloudExceptionDataPtr exception↩ Data)
- void onGetSearchRequestReady (SavedSearch value, EverCloudExceptionDataPtr exceptionData)
- void onCreateSearchRequestReady (SavedSearch value, EverCloudExceptionDataPtr exceptionData)
- void onUpdateSearchRequestReady (qint32 value, EverCloudExceptionDataPtr exceptionData)

- void onExpungeSearchRequestReady (qint32 value, EverCloudExceptionDataPtr exceptionData)
- void onFindNoteOffsetRequestReady (qint32 value, EverCloudExceptionDataPtr exceptionData)
- void onFindNotesMetadataRequestReady (NotesMetadataList value, EverCloudExceptionDataPtr exception↩ Data)
- void onFindNoteCountsRequestReady (NoteCollectionCounts value, EverCloudExceptionDataPtr exception↩ Data)
- void onGetNoteWithResultSpecRequestReady (Note value, EverCloudExceptionDataPtr exceptionData)
- void onGetNoteRequestReady (Note value, EverCloudExceptionDataPtr exceptionData)
- void onGetNoteApplicationDataRequestReady (LazyMap value, EverCloudExceptionDataPtr exceptionData)
- void onGetNoteApplicationDataEntryRequestReady (QString value, EverCloudExceptionDataPtr exception↩ Data)
- void onSetNoteApplicationDataEntryRequestReady (qint32 value, EverCloudExceptionDataPtr exception↩ Data)
- void onUnsetNoteApplicationDataEntryRequestReady (qint32 value, EverCloudExceptionDataPtr exception↩ Data)
- void onGetNoteContentRequestReady (QString value, EverCloudExceptionDataPtr exceptionData)
- void onGetNoteSearchTextRequestReady (QString value, EverCloudExceptionDataPtr exceptionData)
- void onGetResourceSearchTextRequestReady (QString value, EverCloudExceptionDataPtr exceptionData)
- void onGetNoteTagNamesRequestReady (QStringList value, EverCloudExceptionDataPtr exceptionData)
- void onCreateNoteRequestReady (Note value, EverCloudExceptionDataPtr exceptionData)
- void onUpdateNoteRequestReady (Note value, EverCloudExceptionDataPtr exceptionData)
- void onDeleteNoteRequestReady (qint32 value, EverCloudExceptionDataPtr exceptionData)
- void onExpungeNoteRequestReady (qint32 value, EverCloudExceptionDataPtr exceptionData)
- void onCopyNoteRequestReady (Note value, EverCloudExceptionDataPtr exceptionData)
- void onListNoteVersionsRequestReady (QList< NoteVersionId > value, EverCloudExceptionDataPtr exceptionData)
- void onGetNoteVersionRequestReady (Note value, EverCloudExceptionDataPtr exceptionData)
- void onGetResourceRequestReady (Resource value, EverCloudExceptionDataPtr exceptionData)
- void onGetResourceApplicationDataRequestReady (LazyMap value, EverCloudExceptionDataPtr exception↩ Data)
- void onGetResourceApplicationDataEntryRequestReady (QString value, EverCloudExceptionDataPtr exceptionData)
- void onSetResourceApplicationDataEntryRequestReady (qint32 value, EverCloudExceptionDataPtr exceptionData)
- void onUnsetResourceApplicationDataEntryRequestReady (qint32 value, EverCloudExceptionDataPtr exceptionData)
- void onUpdateResourceRequestReady (qint32 value, EverCloudExceptionDataPtr exceptionData)
- void onGetResourceDataRequestReady (QByteArray value, EverCloudExceptionDataPtr exceptionData)
- void onGetResourceByHashRequestReady (Resource value, EverCloudExceptionDataPtr exceptionData)
- void onGetResourceRecognitionRequestReady (QByteArray value, EverCloudExceptionDataPtr exception↩ Data)
- void onGetResourceAlternateDataRequestReady (QByteArray value, EverCloudExceptionDataPtr exceptionData)
- void onGetResourceAttributesRequestReady (ResourceAttributes value, EverCloudExceptionDataPtr exceptionData)
- void onGetPublicNotebookRequestReady (Notebook value, EverCloudExceptionDataPtr exceptionData)
- void onShareNotebookRequestReady (SharedNotebook value, EverCloudExceptionDataPtr exceptionData)
- void onCreateOrUpdateNotebookSharesRequestReady (CreateOrUpdateNotebookSharesResult value, EverCloudExceptionDataPtr exceptionData)
- void onUpdateSharedNotebookRequestReady (qint32 value, EverCloudExceptionDataPtr exceptionData)
- void onSetNotebookRecipientSettingsRequestReady (Notebook value, EverCloudExceptionDataPtr exceptionData)
- void onListSharedNotebooksRequestReady (QList< SharedNotebook > value, EverCloudExceptionDataPtr exceptionData)
- void onCreateLinkedNotebookRequestReady (LinkedNotebook value, EverCloudExceptionDataPtr exceptionData)

- void onUpdateLinkedNotebookRequestReady (qint32 value, EverCloudExceptionDataPtr exceptionData)
- void onListLinkedNotebooksRequestReady (QList< LinkedNotebook > value, EverCloudExceptionDataPtr exceptionData)
- void onExpungeLinkedNotebookRequestReady (qint32 value, EverCloudExceptionDataPtr exceptionData)
- void onAuthenticateToSharedNotebookRequestReady (AuthenticationResult value, EverCloudExceptionDataPtr exceptionData)
- void onGetSharedNotebookByAuthRequestReady (SharedNotebook value, EverCloudExceptionDataPtr exceptionData)
- void onEmailNoteRequestReady (EverCloudExceptionDataPtr exceptionData)
- void onShareNoteRequestReady (QString value, EverCloudExceptionDataPtr exceptionData)
- void onStopSharingNoteRequestReady (EverCloudExceptionDataPtr exceptionData)
- void onAuthenticateToSharedNoteRequestReady (AuthenticationResult value, EverCloudExceptionDataPtr exceptionData)
- void onFindRelatedRequestReady (RelatedResult value, EverCloudExceptionDataPtr exceptionData)
- void onUpdateNoteIfUsnMatchesRequestReady (UpdateNoteIfUsnMatchesResult value, EverCloudExceptionDataPtr exceptionData)
- void onManageNotebookSharesRequestReady (ManageNotebookSharesResult value, EverCloudExceptionDataPtr exceptionData)
- void onGetNotebookSharesRequestReady (ShareRelationships value, EverCloudExceptionDataPtr exceptionData)

Signals

- void getSyncStateRequest (IRequestContextPtr ctx)
- void getFilteredSyncChunkRequest (qint32 afterUSN, qint32 maxEntries, SyncChunkFilter filter, IRequestContextPtr ctx)
- void getLinkedNotebookSyncStateRequest (LinkedNotebook linkedNotebook, IRequestContextPtr ctx)
- void getLinkedNotebookSyncChunkRequest (LinkedNotebook linkedNotebook, qint32 afterUSN, qint32 maxEntries, bool fullSyncOnly, IRequestContextPtr ctx)
- void listNotebooksRequest (IRequestContextPtr ctx)
- void listAccessibleBusinessNotebooksRequest (IRequestContextPtr ctx)
- void getNotebookRequest (Guid guid, IRequestContextPtr ctx)
- void getDefaultNotebookRequest (IRequestContextPtr ctx)
- void createNotebookRequest (Notebook notebook, IRequestContextPtr ctx)
- void updateNotebookRequest (Notebook notebook, IRequestContextPtr ctx)
- void expungeNotebookRequest (Guid guid, IRequestContextPtr ctx)
- void listTagsRequest (IRequestContextPtr ctx)
- void listTagsByNotebookRequest (Guid notebookGuid, IRequestContextPtr ctx)
- void getTagRequest (Guid guid, IRequestContextPtr ctx)
- void createTagRequest (Tag tag, IRequestContextPtr ctx)
- void updateTagRequest (Tag tag, IRequestContextPtr ctx)
- void untagAllRequest (Guid guid, IRequestContextPtr ctx)
- void expungeTagRequest (Guid guid, IRequestContextPtr ctx)
- void listSearchesRequest (IRequestContextPtr ctx)
- void getSearchRequest (Guid guid, IRequestContextPtr ctx)
- void createSearchRequest (SavedSearch search, IRequestContextPtr ctx)
- void updateSearchRequest (SavedSearch search, IRequestContextPtr ctx)
- void expungeSearchRequest (Guid guid, IRequestContextPtr ctx)
- void findNoteOffsetRequest (NoteFilter filter, Guid guid, IRequestContextPtr ctx)
- void findNotesMetadataRequest (NoteFilter filter, qint32 offset, qint32 maxNotes, NotesMetadataResultSpec resultSpec, IRequestContextPtr ctx)
- void findNoteCountsRequest (NoteFilter filter, bool withTrash, IRequestContextPtr ctx)
- void getNoteWithResultSpecRequest (Guid guid, NoteResultSpec resultSpec, IRequestContextPtr ctx)

- void getNoteRequest (Guid guid, bool withContent, bool withResourcesData, bool withResourcesRecognition, bool withResourcesAlternateData, IRequestContextPtr ctx)
- void getNoteApplicationDataRequest (Guid guid, IRequestContextPtr ctx)
- void getNoteApplicationDataEntryRequest (Guid guid, QString key, IRequestContextPtr ctx)
- void setNoteApplicationDataEntryRequest (Guid guid, QString key, QString value, IRequestContextPtr ctx)
- void unsetNoteApplicationDataEntryRequest (Guid guid, QString key, IRequestContextPtr ctx)
- void getNoteContentRequest (Guid guid, IRequestContextPtr ctx)
- void getNoteSearchTextRequest (Guid guid, bool noteOnly, bool tokenizeForIndexing, IRequestContextPtr ctx)
- void getResourceSearchTextRequest (Guid guid, IRequestContextPtr ctx)
- void getNoteTagNamesRequest (Guid guid, IRequestContextPtr ctx)
- void createNoteRequest (Note note, IRequestContextPtr ctx)
- void updateNoteRequest (Note note, IRequestContextPtr ctx)
- void deleteNoteRequest (Guid guid, IRequestContextPtr ctx)
- void expungeNoteRequest (Guid guid, IRequestContextPtr ctx)
- void copyNoteRequest (Guid noteGuid, Guid toNotebookGuid, IRequestContextPtr ctx)
- void listNoteVersionsRequest (Guid noteGuid, IRequestContextPtr ctx)
- void getNoteVersionRequest (Guid noteGuid, qint32 updateSequenceNum, bool withResourcesData, bool withResourcesRecognition, bool withResourcesAlternateData, IRequestContextPtr ctx)
- void getResourceRequest (Guid guid, bool withData, bool withRecognition, bool withAttributes, bool withAlternateData, IRequestContextPtr ctx)
- void getResourceApplicationDataRequest (Guid guid, IRequestContextPtr ctx)
- void getResourceApplicationDataEntryRequest (Guid guid, QString key, IRequestContextPtr ctx)
- void setResourceApplicationDataEntryRequest (Guid guid, QString key, QString value, IRequestContextPtr ctx)
- void unsetResourceApplicationDataEntryRequest (Guid guid, QString key, IRequestContextPtr ctx)
- void updateResourceRequest (Resource resource, IRequestContextPtr ctx)
- void getResourceDataRequest (Guid guid, IRequestContextPtr ctx)
- void getResourceByHashRequest (Guid noteGuid, QByteArray contentHash, bool withData, bool withRecognition, bool withAlternateData, IRequestContextPtr ctx)
- void getResourceRecognitionRequest (Guid guid, IRequestContextPtr ctx)
- void getResourceAlternateDataRequest (Guid guid, IRequestContextPtr ctx)
- void getResourceAttributesRequest (Guid guid, IRequestContextPtr ctx)
- void getPublicNotebookRequest (UserID userId, QString publicUri, IRequestContextPtr ctx)
- void shareNotebookRequest (SharedNotebook sharedNotebook, QString message, IRequestContextPtr ctx)
- void createOrUpdateNotebookSharesRequest (NotebookShareTemplate shareTemplate, IRequestContextPtr ctx)
- void updateSharedNotebookRequest (SharedNotebook sharedNotebook, IRequestContextPtr ctx)
- void setNotebookRecipientSettingsRequest (QString notebookGuid, NotebookRecipientSettings recipient↩ Settings, IRequestContextPtr ctx)
- void listSharedNotebooksRequest (IRequestContextPtr ctx)
- void createLinkedNotebookRequest (LinkedNotebook linkedNotebook, IRequestContextPtr ctx)
- void updateLinkedNotebookRequest (LinkedNotebook linkedNotebook, IRequestContextPtr ctx)
- void listLinkedNotebooksRequest (IRequestContextPtr ctx)
- void expungeLinkedNotebookRequest (Guid guid, IRequestContextPtr ctx)
- void authenticateToSharedNotebookRequest (QString shareKeyOrGlobalId, IRequestContextPtr ctx)
- void getSharedNotebookByAuthRequest (IRequestContextPtr ctx)
- void emailNoteRequest (NoteEmailParameters parameters, IRequestContextPtr ctx)
- void shareNoteRequest (Guid guid, IRequestContextPtr ctx)
- void stopSharingNoteRequest (Guid guid, IRequestContextPtr ctx)
- void authenticateToSharedNoteRequest (QString guid, QString noteKey, IRequestContextPtr ctx)
- void findRelatedRequest (RelatedQuery query, RelatedResultSpec resultSpec, IRequestContextPtr ctx)
- void updateNoteIfUsnMatchesRequest (Note note, IRequestContextPtr ctx)
- void manageNotebookSharesRequest (ManageNotebookSharesParameters parameters, IRequestContextPtr ctx)

- void getNotebookSharesRequest (QString notebookGuid, IRequestContextPtr ctx)
- void getSyncStateRequestReady (QByteArray data)
- void getFilteredSyncChunkRequestReady (QByteArray data)
- void getLinkedNotebookSyncStateRequestReady (QByteArray data)
- void getLinkedNotebookSyncChunkRequestReady (QByteArray data)
- void listNotebooksRequestReady (QByteArray data)
- void listAccessibleBusinessNotebooksRequestReady (QByteArray data)
- void getNotebookRequestReady (QByteArray data)
- void getDefaultNotebookRequestReady (QByteArray data)
- void createNotebookRequestReady (QByteArray data)
- void updateNotebookRequestReady (QByteArray data)
- void expungeNotebookRequestReady (QByteArray data)
- void listTagsRequestReady (QByteArray data)
- void listTagsByNotebookRequestReady (QByteArray data)
- void getTagRequestReady (QByteArray data)
- void createTagRequestReady (QByteArray data)
- void updateTagRequestReady (QByteArray data)
- void untagAllRequestReady (QByteArray data)
- void expungeTagRequestReady (QByteArray data)
- void listSearchesRequestReady (QByteArray data)
- void getSearchRequestReady (QByteArray data)
- void createSearchRequestReady (QByteArray data)
- void updateSearchRequestReady (QByteArray data)
- void expungeSearchRequestReady (QByteArray data)
- void findNoteOffsetRequestReady (QByteArray data)
- void findNotesMetadataRequestReady (QByteArray data)
- void findNoteCountsRequestReady (QByteArray data)
- void getNoteWithResultSpecRequestReady (QByteArray data)
- void getNoteRequestReady (QByteArray data)
- void getNoteApplicationDataRequestReady (QByteArray data)
- void getNoteApplicationDataEntryRequestReady (QByteArray data)
- void setNoteApplicationDataEntryRequestReady (QByteArray data)
- void unsetNoteApplicationDataEntryRequestReady (QByteArray data)
- void getNoteContentRequestReady (QByteArray data)
- void getNoteSearchTextRequestReady (QByteArray data)
- void getResourceSearchTextRequestReady (QByteArray data)
- void getNoteTagNamesRequestReady (QByteArray data)
- void createNoteRequestReady (QByteArray data)
- void updateNoteRequestReady (QByteArray data)
- void deleteNoteRequestReady (QByteArray data)
- void expungeNoteRequestReady (QByteArray data)
- void copyNoteRequestReady (QByteArray data)
- void listNoteVersionsRequestReady (QByteArray data)
- void getNoteVersionRequestReady (QByteArray data)
- void getResourceRequestReady (QByteArray data)
- void getResourceApplicationDataRequestReady (QByteArray data)
- void getResourceApplicationDataEntryRequestReady (QByteArray data)
- void setResourceApplicationDataEntryRequestReady (QByteArray data)
- void unsetResourceApplicationDataEntryRequestReady (QByteArray data)
- void updateResourceRequestReady (QByteArray data)
- void getResourceDataRequestReady (QByteArray data)
- void getResourceByHashRequestReady (QByteArray data)
- void getResourceRecognitionRequestReady (QByteArray data)
- void getResourceAlternateDataRequestReady (QByteArray data)
- void getResourceAttributesRequestReady (QByteArray data)

- void getPublicNotebookRequestReady (QByteArray data)
- void shareNotebookRequestReady (QByteArray data)
- void createOrUpdateNotebookSharesRequestReady (QByteArray data)
- void updateSharedNotebookRequestReady (QByteArray data)
- void setNotebookRecipientSettingsRequestReady (QByteArray data)
- void listSharedNotebooksRequestReady (QByteArray data)
- void createLinkedNotebookRequestReady (QByteArray data)
- void updateLinkedNotebookRequestReady (QByteArray data)
- void listLinkedNotebooksRequestReady (QByteArray data)
- void expungeLinkedNotebookRequestReady (QByteArray data)
- void authenticateToSharedNotebookRequestReady (QByteArray data)
- void getSharedNotebookByAuthRequestReady (QByteArray data)
- void emailNoteRequestReady (QByteArray data)
- void shareNoteRequestReady (QByteArray data)
- void stopSharingNoteRequestReady (QByteArray data)
- void authenticateToSharedNoteRequestReady (QByteArray data)
- void findRelatedRequestReady (QByteArray data)
- void updateNoteIfUsnMatchesRequestReady (QByteArray data)
- void manageNotebookSharesRequestReady (QByteArray data)
- void getNotebookSharesRequestReady (QByteArray data)

Public Member Functions

- NoteStoreServer (QObject *parent=nullptr)

7.80.1 Detailed Description

The NoteStoreServer class represents customizable server for NoteStore requests. It is primarily used for testing of QEverCloud.

7.80.2 Constructor & Destructor Documentation

7.80.2.1 NoteStoreServer()

```
qevercloud::NoteStoreServer::NoteStoreServer (
            QObject * parent = nullptr )  [explicit]
```

7.80.3 Member Function Documentation

7.80.3.1 authenticateToSharedNotebookRequest

```
void qevercloud::NoteStoreServer::authenticateToSharedNotebookRequest (
            QString shareKeyOrGlobalId,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.2 authenticateToSharedNotebookRequestReady

```
void qevercloud::NoteStoreServer::authenticateToSharedNotebookRequestReady (
            QByteArray data )  [signal]
```

7.80.3.3 authenticateToSharedNoteRequest

```
void qevercloud::NoteStoreServer::authenticateToSharedNoteRequest (
            QString guid,
            QString noteKey,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.4 authenticateToSharedNoteRequestReady

```
void qevercloud::NoteStoreServer::authenticateToSharedNoteRequestReady (
            QByteArray data )  [signal]
```

7.80.3.5 copyNoteRequest

```
void qevercloud::NoteStoreServer::copyNoteRequest (
            Guid noteGuid,
            Guid toNotebookGuid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.6 copyNoteRequestReady

```
void qevercloud::NoteStoreServer::copyNoteRequestReady (
            QByteArray data )  [signal]
```

7.80.3.7 createLinkedNotebookRequest

```
void qevercloud::NoteStoreServer::createLinkedNotebookRequest (
            LinkedNotebook linkedNotebook,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.8 createLinkedNotebookRequestReady

```
void qevercloud::NoteStoreServer::createLinkedNotebookRequestReady (
            QByteArray data )  [signal]
```

7.80.3.9 createNotebookRequest

```
void qevercloud::NoteStoreServer::createNotebookRequest (
            Notebook notebook,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.10 createNotebookRequestReady

```
void qevercloud::NoteStoreServer::createNotebookRequestReady (
            QByteArray data )  [signal]
```

7.80.3.11 createNoteRequest

```
void qevercloud::NoteStoreServer::createNoteRequest (
            Note note,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.12 createNoteRequestReady

```
void qevercloud::NoteStoreServer::createNoteRequestReady (
            QByteArray data )  [signal]
```

7.80.3.13 createOrUpdateNotebookSharesRequest

```
void qevercloud::NoteStoreServer::createOrUpdateNotebookSharesRequest (
            NotebookShareTemplate shareTemplate,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.14 createOrUpdateNotebookSharesRequestReady

```
void qevercloud::NoteStoreServer::createOrUpdateNotebookSharesRequestReady (
            QByteArray data )  [signal]
```

7.80.3.15 createSearchRequest

```
void qevercloud::NoteStoreServer::createSearchRequest (
            SavedSearch search,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.16 createSearchRequestReady

```
void qevercloud::NoteStoreServer::createSearchRequestReady (
            QByteArray data )  [signal]
```

7.80.3.17 createTagRequest

```
void qevercloud::NoteStoreServer::createTagRequest (
            Tag tag,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.18 createTagRequestReady

```
void qevercloud::NoteStoreServer::createTagRequestReady (
            QByteArray data )  [signal]
```

7.80.3.19 deleteNoteRequest

```
void qevercloud::NoteStoreServer::deleteNoteRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.20 deleteNoteRequestReady

```
void qevercloud::NoteStoreServer::deleteNoteRequestReady (
            QByteArray data )  [signal]
```

7.80.3.21 emailNoteRequest

```
void qevercloud::NoteStoreServer::emailNoteRequest (
            NoteEmailParameters parameters,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.22 emailNoteRequestReady

```
void qevercloud::NoteStoreServer::emailNoteRequestReady (
            QByteArray data )  [signal]
```

7.80.3.23 expungeLinkedNotebookRequest

```
void qevercloud::NoteStoreServer::expungeLinkedNotebookRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.24 expungeLinkedNotebookRequestReady

```
void qevercloud::NoteStoreServer::expungeLinkedNotebookRequestReady (
            QByteArray data )  [signal]
```

7.80.3.25 expungeNotebookRequest

```
void qevercloud::NoteStoreServer::expungeNotebookRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.26 expungeNotebookRequestReady

```
void qevercloud::NoteStoreServer::expungeNotebookRequestReady (
            QByteArray data )  [signal]
```

7.80.3.27 expungeNoteRequest

```
void qevercloud::NoteStoreServer::expungeNoteRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.28 expungeNoteRequestReady

```
void qevercloud::NoteStoreServer::expungeNoteRequestReady (
            QByteArray data )  [signal]
```

7.80.3.29 expungeSearchRequest

```
void qevercloud::NoteStoreServer::expungeSearchRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.30 expungeSearchRequestReady

```
void qevercloud::NoteStoreServer::expungeSearchRequestReady (
            QByteArray data )  [signal]
```

7.80.3.31 expungeTagRequest

```
void qevercloud::NoteStoreServer::expungeTagRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.32 expungeTagRequestReady

```
void qevercloud::NoteStoreServer::expungeTagRequestReady (
            QByteArray data )  [signal]
```

7.80.3.33 findNoteCountsRequest

```
void qevercloud::NoteStoreServer::findNoteCountsRequest (
            NoteFilter filter,
            bool withTrash,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.34 findNoteCountsRequestReady

```
void qevercloud::NoteStoreServer::findNoteCountsRequestReady (
            QByteArray data )  [signal]
```

7.80.3.35 findNoteOffsetRequest

```
void qevercloud::NoteStoreServer::findNoteOffsetRequest (
            NoteFilter filter,
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.36 findNoteOffsetRequestReady

```
void qevercloud::NoteStoreServer::findNoteOffsetRequestReady (
            QByteArray data )  [signal]
```

7.80.3.37 findNotesMetadataRequest

```
void qevercloud::NoteStoreServer::findNotesMetadataRequest (
            NoteFilter filter,
            qint32 offset,
            qint32 maxNotes,
            NotesMetadataResultSpec resultSpec,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.38 findNotesMetadataRequestReady

```
void qevercloud::NoteStoreServer::findNotesMetadataRequestReady (
            QByteArray data )  [signal]
```

7.80.3.39 findRelatedRequest

```
void qevercloud::NoteStoreServer::findRelatedRequest (
            RelatedQuery query,
            RelatedResultSpec resultSpec,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.40 findRelatedRequestReady

```
void qevercloud::NoteStoreServer::findRelatedRequestReady (
            QByteArray data )  [signal]
```

7.80.3.41 getDefaultNotebookRequest

```
void qevercloud::NoteStoreServer::getDefaultNotebookRequest (
            IRequestContextPtr ctx )  [signal]
```

7.80.3.42 getDefaultNotebookRequestReady

```
void qevercloud::NoteStoreServer::getDefaultNotebookRequestReady (
            QByteArray data )  [signal]
```

7.80.3.43 getFilteredSyncChunkRequest

```
void qevercloud::NoteStoreServer::getFilteredSyncChunkRequest (
            qint32 afterUSN,
            qint32 maxEntries,
            SyncChunkFilter filter,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.44 getFilteredSyncChunkRequestReady

```
void qevercloud::NoteStoreServer::getFilteredSyncChunkRequestReady (
            QByteArray data )  [signal]
```

7.80.3.45 getLinkedNotebookSyncChunkRequest

```
void qevercloud::NoteStoreServer::getLinkedNotebookSyncChunkRequest (
            LinkedNotebook linkedNotebook,
            qint32 afterUSN,
            qint32 maxEntries,
            bool fullSyncOnly,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.46 getLinkedNotebookSyncChunkRequestReady

```
void qevercloud::NoteStoreServer::getLinkedNotebookSyncChunkRequestReady (
            QByteArray data )  [signal]
```

7.80.3.47 getLinkedNotebookSyncStateRequest

```
void qevercloud::NoteStoreServer::getLinkedNotebookSyncStateRequest (
            LinkedNotebook linkedNotebook,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.48 getLinkedNotebookSyncStateRequestReady

```
void qevercloud::NoteStoreServer::getLinkedNotebookSyncStateRequestReady (
            QByteArray data )  [signal]
```

7.80.3.49 getNoteApplicationDataEntryRequest

```
void qevercloud::NoteStoreServer::getNoteApplicationDataEntryRequest (
            Guid guid,
            QString key,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.50 getNoteApplicationDataEntryRequestReady

```
void qevercloud::NoteStoreServer::getNoteApplicationDataEntryRequestReady (
            QByteArray data )  [signal]
```

7.80.3.51 getNoteApplicationDataRequest

```
void qevercloud::NoteStoreServer::getNoteApplicationDataRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.52 getNoteApplicationDataRequestReady

```
void qevercloud::NoteStoreServer::getNoteApplicationDataRequestReady (
            QByteArray data )  [signal]
```

7.80.3.53 getNotebookRequest

```
void qevercloud::NoteStoreServer::getNotebookRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.54 getNotebookRequestReady

```
void qevercloud::NoteStoreServer::getNotebookRequestReady (
            QByteArray data )  [signal]
```

7.80.3.55 getNotebookSharesRequest

```
void qevercloud::NoteStoreServer::getNotebookSharesRequest (
            QString notebookGuid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.56 getNotebookSharesRequestReady

```
void qevercloud::NoteStoreServer::getNotebookSharesRequestReady (
            QByteArray data )  [signal]
```

7.80.3.57 getNoteContentRequest

```
void qevercloud::NoteStoreServer::getNoteContentRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.58 getNoteContentRequestReady

```
void qevercloud::NoteStoreServer::getNoteContentRequestReady (
            QByteArray data )  [signal]
```

7.80.3.59 getNoteRequest

```
void qevercloud::NoteStoreServer::getNoteRequest (
            Guid guid,
            bool withContent,
            bool withResourcesData,
            bool withResourcesRecognition,
            bool withResourcesAlternateData,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.60 getNoteRequestReady

```
void qevercloud::NoteStoreServer::getNoteRequestReady (
            QByteArray data )  [signal]
```

7.80.3.61 getNoteSearchTextRequest

```
void qevercloud::NoteStoreServer::getNoteSearchTextRequest (
            Guid guid,
            bool noteOnly,
            bool tokenizeForIndexing,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.62 getNoteSearchTextRequestReady

```
void qevercloud::NoteStoreServer::getNoteSearchTextRequestReady (
            QByteArray data )  [signal]
```

7.80.3.63 getNoteTagNamesRequest

```
void qevercloud::NoteStoreServer::getNoteTagNamesRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.64 getNoteTagNamesRequestReady

```
void qevercloud::NoteStoreServer::getNoteTagNamesRequestReady (
            QByteArray data )  [signal]
```

7.80.3.65 getNoteVersionRequest

```
void qevercloud::NoteStoreServer::getNoteVersionRequest (
            Guid noteGuid,
            qint32 updateSequenceNum,
            bool withResourcesData,
            bool withResourcesRecognition,
            bool withResourcesAlternateData,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.66 getNoteVersionRequestReady

```
void qevercloud::NoteStoreServer::getNoteVersionRequestReady (
            QByteArray data )  [signal]
```

7.80.3.67 getNoteWithResultSpecRequest

```
void qevercloud::NoteStoreServer::getNoteWithResultSpecRequest (
            Guid guid,
            NoteResultSpec resultSpec,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.68 getNoteWithResultSpecRequestReady

```
void qevercloud::NoteStoreServer::getNoteWithResultSpecRequestReady (
            QByteArray data )  [signal]
```

7.80.3.69 getPublicNotebookRequest

```
void qevercloud::NoteStoreServer::getPublicNotebookRequest (
            UserID userId,
            QString publicUri,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.70 getPublicNotebookRequestReady

```
void qevercloud::NoteStoreServer::getPublicNotebookRequestReady (
            QByteArray data )  [signal]
```

7.80.3.71 getResourceAlternateDataRequest

```
void qevercloud::NoteStoreServer::getResourceAlternateDataRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.72 getResourceAlternateDataRequestReady

```
void qevercloud::NoteStoreServer::getResourceAlternateDataRequestReady (
            QByteArray data )  [signal]
```

7.80.3.73 getResourceApplicationDataEntryRequest

```
void qevercloud::NoteStoreServer::getResourceApplicationDataEntryRequest (
            Guid guid,
            QString key,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.74 getResourceApplicationDataEntryRequestReady

```
void qevercloud::NoteStoreServer::getResourceApplicationDataEntryRequestReady (
            QByteArray data )  [signal]
```

7.80.3.75 getResourceApplicationDataRequest

```
void qevercloud::NoteStoreServer::getResourceApplicationDataRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.76 getResourceApplicationDataRequestReady

```
void qevercloud::NoteStoreServer::getResourceApplicationDataRequestReady (
            QByteArray data )  [signal]
```

7.80.3.77 getResourceAttributesRequest

```
void qevercloud::NoteStoreServer::getResourceAttributesRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.78 getResourceAttributesRequestReady

```
void qevercloud::NoteStoreServer::getResourceAttributesRequestReady (
            QByteArray data )  [signal]
```

7.80.3.79 getResourceByHashRequest

```
void qevercloud::NoteStoreServer::getResourceByHashRequest (
            Guid noteGuid,
            QByteArray contentHash,
            bool withData,
            bool withRecognition,
            bool withAlternateData,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.80 getResourceByHashRequestReady

```
void qevercloud::NoteStoreServer::getResourceByHashRequestReady (
            QByteArray data )  [signal]
```

7.80.3.81 getResourceDataRequest

```
void qevercloud::NoteStoreServer::getResourceDataRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.82 getResourceDataRequestReady

```
void qevercloud::NoteStoreServer::getResourceDataRequestReady (
            QByteArray data )  [signal]
```

7.80.3.83 getResourceRecognitionRequest

```
void qevercloud::NoteStoreServer::getResourceRecognitionRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.84 getResourceRecognitionRequestReady

```
void qevercloud::NoteStoreServer::getResourceRecognitionRequestReady (
            QByteArray data )  [signal]
```

7.80.3.85 getResourceRequest

```
void qevercloud::NoteStoreServer::getResourceRequest (
            Guid guid,
            bool withData,
            bool withRecognition,
            bool withAttributes,
            bool withAlternateData,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.86 getResourceRequestReady

```
void qevercloud::NoteStoreServer::getResourceRequestReady (
            QByteArray data )  [signal]
```

7.80.3.87 getResourceSearchTextRequest

```
void qevercloud::NoteStoreServer::getResourceSearchTextRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.88 getResourceSearchTextRequestReady

```
void qevercloud::NoteStoreServer::getResourceSearchTextRequestReady (
            QByteArray data )  [signal]
```

7.80.3.89 getSearchRequest

```
void qevercloud::NoteStoreServer::getSearchRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.90 getSearchRequestReady

```
void qevercloud::NoteStoreServer::getSearchRequestReady (
            QByteArray data )  [signal]
```

7.80.3.91 getSharedNotebookByAuthRequest

```
void qevercloud::NoteStoreServer::getSharedNotebookByAuthRequest (
            IRequestContextPtr ctx )  [signal]
```

7.80.3.92 getSharedNotebookByAuthRequestReady

```
void qevercloud::NoteStoreServer::getSharedNotebookByAuthRequestReady (
            QByteArray data )  [signal]
```

7.80.3.93 getSyncStateRequest

```
void qevercloud::NoteStoreServer::getSyncStateRequest (
            IRequestContextPtr ctx )  [signal]
```

7.80.3.94 getSyncStateRequestReady

```
void qevercloud::NoteStoreServer::getSyncStateRequestReady (
            QByteArray data )  [signal]
```

7.80.3.95 getTagRequest

```
void qevercloud::NoteStoreServer::getTagRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.96 getTagRequestReady

```
void qevercloud::NoteStoreServer::getTagRequestReady (
            QByteArray data )  [signal]
```

7.80.3.97 listAccessibleBusinessNotebooksRequest

```
void qevercloud::NoteStoreServer::listAccessibleBusinessNotebooksRequest (
            IRequestContextPtr ctx )  [signal]
```

7.80.3.98 listAccessibleBusinessNotebooksRequestReady

```
void qevercloud::NoteStoreServer::listAccessibleBusinessNotebooksRequestReady (
            QByteArray data )  [signal]
```

7.80.3.99 listLinkedNotebooksRequest

```
void qevercloud::NoteStoreServer::listLinkedNotebooksRequest (
            IRequestContextPtr ctx )  [signal]
```

7.80.3.100 listLinkedNotebooksRequestReady

```
void qevercloud::NoteStoreServer::listLinkedNotebooksRequestReady (
            QByteArray data )  [signal]
```

7.80.3.101 listNotebooksRequest

```
void qevercloud::NoteStoreServer::listNotebooksRequest (
            IRequestContextPtr ctx )  [signal]
```

7.80.3.102 listNotebooksRequestReady

```
void qevercloud::NoteStoreServer::listNotebooksRequestReady (
            QByteArray data )  [signal]
```

7.80.3.103 listNoteVersionsRequest

```
void qevercloud::NoteStoreServer::listNoteVersionsRequest (
            Guid noteGuid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.104 listNoteVersionsRequestReady

```
void qevercloud::NoteStoreServer::listNoteVersionsRequestReady (
            QByteArray data )  [signal]
```

7.80.3.105 listSearchesRequest

```
void qevercloud::NoteStoreServer::listSearchesRequest (
            IRequestContextPtr ctx )  [signal]
```

7.80.3.106 listSearchesRequestReady

```
void qevercloud::NoteStoreServer::listSearchesRequestReady (
            QByteArray data )  [signal]
```

7.80.3.107 listSharedNotebooksRequest

```
void qevercloud::NoteStoreServer::listSharedNotebooksRequest (
            IRequestContextPtr ctx )  [signal]
```

7.80.3.108 listSharedNotebooksRequestReady

```
void qevercloud::NoteStoreServer::listSharedNotebooksRequestReady (
            QByteArray data )  [signal]
```

7.80.3.109 listTagsByNotebookRequest

```
void qevercloud::NoteStoreServer::listTagsByNotebookRequest (
            Guid notebookGuid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.110 listTagsByNotebookRequestReady

```
void qevercloud::NoteStoreServer::listTagsByNotebookRequestReady (
            QByteArray data )  [signal]
```

7.80.3.111 listTagsRequest

```
void qevercloud::NoteStoreServer::listTagsRequest (
            IRequestContextPtr ctx )  [signal]
```

7.80.3.112 listTagsRequestReady

```
void qevercloud::NoteStoreServer::listTagsRequestReady (
            QByteArray data )  [signal]
```

7.80.3.113 manageNotebookSharesRequest

```
void qevercloud::NoteStoreServer::manageNotebookSharesRequest (
            ManageNotebookSharesParameters parameters,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.114 manageNotebookSharesRequestReady

```
void qevercloud::NoteStoreServer::manageNotebookSharesRequestReady (
            QByteArray data )  [signal]
```

7.80.3.115 onAuthenticateToSharedNotebookRequestReady

```
void qevercloud::NoteStoreServer::onAuthenticateToSharedNotebookRequestReady (
            AuthenticationResult value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.116 onAuthenticateToSharedNoteRequestReady

```
void qevercloud::NoteStoreServer::onAuthenticateToSharedNoteRequestReady (
            AuthenticationResult value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.117 onCopyNoteRequestReady

```
void qevercloud::NoteStoreServer::onCopyNoteRequestReady (
            Note value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.118 onCreateLinkedNotebookRequestReady

```
void qevercloud::NoteStoreServer::onCreateLinkedNotebookRequestReady (
            LinkedNotebook value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.119 onCreateNotebookRequestReady

```
void qevercloud::NoteStoreServer::onCreateNotebookRequestReady (
            Notebook value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.120 onCreateNoteRequestReady

```
void qevercloud::NoteStoreServer::onCreateNoteRequestReady (
            Note value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.121 onCreateOrUpdateNotebookSharesRequestReady

```
void qevercloud::NoteStoreServer::onCreateOrUpdateNotebookSharesRequestReady (
            CreateOrUpdateNotebookSharesResult value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.122 onCreateSearchRequestReady

```
void qevercloud::NoteStoreServer::onCreateSearchRequestReady (
            SavedSearch value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.123 onCreateTagRequestReady

```
void qevercloud::NoteStoreServer::onCreateTagRequestReady (
            Tag value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.124 onDeleteNoteRequestReady

```
void qevercloud::NoteStoreServer::onDeleteNoteRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.125 onEmailNoteRequestReady

```
void qevercloud::NoteStoreServer::onEmailNoteRequestReady (
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.126 onExpungeLinkedNotebookRequestReady

```
void qevercloud::NoteStoreServer::onExpungeLinkedNotebookRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.127 onExpungeNotebookRequestReady

```
void qevercloud::NoteStoreServer::onExpungeNotebookRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.128 onExpungeNoteRequestReady

```
void qevercloud::NoteStoreServer::onExpungeNoteRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.129 onExpungeSearchRequestReady

```
void qevercloud::NoteStoreServer::onExpungeSearchRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.130 onExpungeTagRequestReady

```
void qevercloud::NoteStoreServer::onExpungeTagRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.131 onFindNoteCountsRequestReady

```
void qevercloud::NoteStoreServer::onFindNoteCountsRequestReady (
            NoteCollectionCounts value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.132 onFindNoteOffsetRequestReady

```
void qevercloud::NoteStoreServer::onFindNoteOffsetRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.133 onFindNotesMetadataRequestReady

```
void qevercloud::NoteStoreServer::onFindNotesMetadataRequestReady (
            NotesMetadataList value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.134 onFindRelatedRequestReady

```
void qevercloud::NoteStoreServer::onFindRelatedRequestReady (
            RelatedResult value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.135 onGetDefaultNotebookRequestReady

```
void qevercloud::NoteStoreServer::onGetDefaultNotebookRequestReady (
            Notebook value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.136 onGetFilteredSyncChunkRequestReady

```
void qevercloud::NoteStoreServer::onGetFilteredSyncChunkRequestReady (
            SyncChunk value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.137 onGetLinkedNotebookSyncChunkRequestReady

```
void qevercloud::NoteStoreServer::onGetLinkedNotebookSyncChunkRequestReady (
            SyncChunk value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.138 onGetLinkedNotebookSyncStateRequestReady

```
void qevercloud::NoteStoreServer::onGetLinkedNotebookSyncStateRequestReady (
            SyncState value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.139 onGetNoteApplicationDataEntryRequestReady

```
void qevercloud::NoteStoreServer::onGetNoteApplicationDataEntryRequestReady (
            QString value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.140 onGetNoteApplicationDataRequestReady

```
void qevercloud::NoteStoreServer::onGetNoteApplicationDataRequestReady (
            LazyMap value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.141 onGetNotebookRequestReady

```
void qevercloud::NoteStoreServer::onGetNotebookRequestReady (
            Notebook value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.142 onGetNotebookSharesRequestReady

```
void qevercloud::NoteStoreServer::onGetNotebookSharesRequestReady (
            ShareRelationships value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.143 onGetNoteContentRequestReady

```
void qevercloud::NoteStoreServer::onGetNoteContentRequestReady (
            QString value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.144 onGetNoteRequestReady

```
void qevercloud::NoteStoreServer::onGetNoteRequestReady (
            Note value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.145 onGetNoteSearchTextRequestReady

```
void qevercloud::NoteStoreServer::onGetNoteSearchTextRequestReady (
            QString value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.146 onGetNoteTagNamesRequestReady

```
void qevercloud::NoteStoreServer::onGetNoteTagNamesRequestReady (
            QStringList value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.147 onGetNoteVersionRequestReady

```
void qevercloud::NoteStoreServer::onGetNoteVersionRequestReady (
            Note value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.148 onGetNoteWithResultSpecRequestReady

```
void qevercloud::NoteStoreServer::onGetNoteWithResultSpecRequestReady (
            Note value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.149 onGetPublicNotebookRequestReady

```
void qevercloud::NoteStoreServer::onGetPublicNotebookRequestReady (
            Notebook value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.150 onGetResourceAlternateDataRequestReady

```
void qevercloud::NoteStoreServer::onGetResourceAlternateDataRequestReady (
            QByteArray value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.151 onGetResourceApplicationDataEntryRequestReady

```
void qevercloud::NoteStoreServer::onGetResourceApplicationDataEntryRequestReady (
            QString value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.152 onGetResourceApplicationDataRequestReady

```
void qevercloud::NoteStoreServer::onGetResourceApplicationDataRequestReady (
            LazyMap value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.153 onGetResourceAttributesRequestReady

```
void qevercloud::NoteStoreServer::onGetResourceAttributesRequestReady (
            ResourceAttributes value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.154 onGetResourceByHashRequestReady

```
void qevercloud::NoteStoreServer::onGetResourceByHashRequestReady (
            Resource value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.155 onGetResourceDataRequestReady

```
void qevercloud::NoteStoreServer::onGetResourceDataRequestReady (
            QByteArray value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.156 onGetResourceRecognitionRequestReady

```
void qevercloud::NoteStoreServer::onGetResourceRecognitionRequestReady (
            QByteArray value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.157 onGetResourceRequestReady

```
void qevercloud::NoteStoreServer::onGetResourceRequestReady (
            Resource value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.158 onGetResourceSearchTextRequestReady

```
void qevercloud::NoteStoreServer::onGetResourceSearchTextRequestReady (
            QString value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.159 onGetSearchRequestReady

```
void qevercloud::NoteStoreServer::onGetSearchRequestReady (
            SavedSearch value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.160 onGetSharedNotebookByAuthRequestReady

```
void qevercloud::NoteStoreServer::onGetSharedNotebookByAuthRequestReady (
            SharedNotebook value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.161 onGetSyncStateRequestReady

```
void qevercloud::NoteStoreServer::onGetSyncStateRequestReady (
            SyncState value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.162 onGetTagRequestReady

```
void qevercloud::NoteStoreServer::onGetTagRequestReady (
            Tag value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.163 onListAccessibleBusinessNotebooksRequestReady

```
void qevercloud::NoteStoreServer::onListAccessibleBusinessNotebooksRequestReady (
            QList< Notebook > value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.164 onListLinkedNotebooksRequestReady

```
void qevercloud::NoteStoreServer::onListLinkedNotebooksRequestReady (
            QList< LinkedNotebook > value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.165 onListNotebooksRequestReady

```
void qevercloud::NoteStoreServer::onListNotebooksRequestReady (
            QList< Notebook > value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.166 onListNoteVersionsRequestReady

```
void qevercloud::NoteStoreServer::onListNoteVersionsRequestReady (
            QList< NoteVersionId > value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.167 onListSearchesRequestReady

```
void qevercloud::NoteStoreServer::onListSearchesRequestReady (
            QList< SavedSearch > value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.168 onListSharedNotebooksRequestReady

```
void qevercloud::NoteStoreServer::onListSharedNotebooksRequestReady (
            QList< SharedNotebook > value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.169 onListTagsByNotebookRequestReady

```
void qevercloud::NoteStoreServer::onListTagsByNotebookRequestReady (
            QList< Tag > value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.170 onListTagsRequestReady

```
void qevercloud::NoteStoreServer::onListTagsRequestReady (
            QList< Tag > value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.171 onManageNotebookSharesRequestReady

```
void qevercloud::NoteStoreServer::onManageNotebookSharesRequestReady (
            ManageNotebookSharesResult value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.172 onRequest

```
void qevercloud::NoteStoreServer::onRequest (
            QByteArray data )  [slot]
```

7.80.3.173 onSetNoteApplicationDataEntryRequestReady

```
void qevercloud::NoteStoreServer::onSetNoteApplicationDataEntryRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.174 onSetNotebookRecipientSettingsRequestReady

```
void qevercloud::NoteStoreServer::onSetNotebookRecipientSettingsRequestReady (
            Notebook value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.175 onSetResourceApplicationDataEntryRequestReady

```
void qevercloud::NoteStoreServer::onSetResourceApplicationDataEntryRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.176 onShareNotebookRequestReady

```
void qevercloud::NoteStoreServer::onShareNotebookRequestReady (
            SharedNotebook value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.177 onShareNoteRequestReady

```
void qevercloud::NoteStoreServer::onShareNoteRequestReady (
            QString value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.178 onStopSharingNoteRequestReady

```
void qevercloud::NoteStoreServer::onStopSharingNoteRequestReady (
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.179 onUnsetNoteApplicationDataEntryRequestReady

```
void qevercloud::NoteStoreServer::onUnsetNoteApplicationDataEntryRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.180 onUnsetResourceApplicationDataEntryRequestReady

```
void qevercloud::NoteStoreServer::onUnsetResourceApplicationDataEntryRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.181 onUntagAllRequestReady

```
void qevercloud::NoteStoreServer::onUntagAllRequestReady (
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.182 onUpdateLinkedNotebookRequestReady

```
void qevercloud::NoteStoreServer::onUpdateLinkedNotebookRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.183 onUpdateNotebookRequestReady

```
void qevercloud::NoteStoreServer::onUpdateNotebookRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.184 onUpdateNoteIfUsnMatchesRequestReady

```
void qevercloud::NoteStoreServer::onUpdateNoteIfUsnMatchesRequestReady (
            UpdateNoteIfUsnMatchesResult value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.185 onUpdateNoteRequestReady

```
void qevercloud::NoteStoreServer::onUpdateNoteRequestReady (
            Note value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.186 onUpdateResourceRequestReady

```
void qevercloud::NoteStoreServer::onUpdateResourceRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.187 onUpdateSearchRequestReady

```
void qevercloud::NoteStoreServer::onUpdateSearchRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.188 onUpdateSharedNotebookRequestReady

```
void qevercloud::NoteStoreServer::onUpdateSharedNotebookRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.189 onUpdateTagRequestReady

```
void qevercloud::NoteStoreServer::onUpdateTagRequestReady (
            qint32 value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.80.3.190 setNoteApplicationDataEntryRequest

```
void qevercloud::NoteStoreServer::setNoteApplicationDataEntryRequest (
            Guid guid,
            QString key,
            QString value,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.191 setNoteApplicationDataEntryRequestReady

```
void qevercloud::NoteStoreServer::setNoteApplicationDataEntryRequestReady (
            QByteArray data )  [signal]
```

7.80.3.192 setNotebookRecipientSettingsRequest

```
void qevercloud::NoteStoreServer::setNotebookRecipientSettingsRequest (
            QString notebookGuid,
            NotebookRecipientSettings recipientSettings,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.193 setNotebookRecipientSettingsRequestReady

```
void qevercloud::NoteStoreServer::setNotebookRecipientSettingsRequestReady (
            QByteArray data )  [signal]
```

7.80.3.194 setResourceApplicationDataEntryRequest

```
void qevercloud::NoteStoreServer::setResourceApplicationDataEntryRequest (
            Guid guid,
            QString key,
            QString value,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.195 setResourceApplicationDataEntryRequestReady

```
void qevercloud::NoteStoreServer::setResourceApplicationDataEntryRequestReady (
            QByteArray data )  [signal]
```

7.80.3.196 shareNotebookRequest

```
void qevercloud::NoteStoreServer::shareNotebookRequest (
            SharedNotebook sharedNotebook,
            QString message,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.197 shareNotebookRequestReady

```
void qevercloud::NoteStoreServer::shareNotebookRequestReady (
            QByteArray data )  [signal]
```

7.80.3.198 shareNoteRequest

```
void qevercloud::NoteStoreServer::shareNoteRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.199 shareNoteRequestReady

```
void qevercloud::NoteStoreServer::shareNoteRequestReady (
            QByteArray data )  [signal]
```

7.80.3.200 stopSharingNoteRequest

```
void qevercloud::NoteStoreServer::stopSharingNoteRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.201 stopSharingNoteRequestReady

```
void qevercloud::NoteStoreServer::stopSharingNoteRequestReady (
            QByteArray data )  [signal]
```

7.80.3.202 unsetNoteApplicationDataEntryRequest

```
void qevercloud::NoteStoreServer::unsetNoteApplicationDataEntryRequest (
            Guid guid,
            QString key,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.203 unsetNoteApplicationDataEntryRequestReady

```
void qevercloud::NoteStoreServer::unsetNoteApplicationDataEntryRequestReady (
            QByteArray data )  [signal]
```

7.80.3.204 unsetResourceApplicationDataEntryRequest

```
void qevercloud::NoteStoreServer::unsetResourceApplicationDataEntryRequest (
            Guid guid,
            QString key,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.205 unsetResourceApplicationDataEntryRequestReady

```
void qevercloud::NoteStoreServer::unsetResourceApplicationDataEntryRequestReady (
            QByteArray data )  [signal]
```

7.80.3.206 untagAllRequest

```
void qevercloud::NoteStoreServer::untagAllRequest (
            Guid guid,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.207 untagAllRequestReady

```
void qevercloud::NoteStoreServer::untagAllRequestReady (
            QByteArray data )  [signal]
```

7.80.3.208 updateLinkedNotebookRequest

```
void qevercloud::NoteStoreServer::updateLinkedNotebookRequest (
            LinkedNotebook linkedNotebook,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.209 updateLinkedNotebookRequestReady

```
void qevercloud::NoteStoreServer::updateLinkedNotebookRequestReady (
            QByteArray data )  [signal]
```

7.80.3.210 updateNotebookRequest

```
void qevercloud::NoteStoreServer::updateNotebookRequest (
            Notebook notebook,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.211 updateNotebookRequestReady

```
void qevercloud::NoteStoreServer::updateNotebookRequestReady (
            QByteArray data )  [signal]
```

7.80.3.212 updateNoteIfUsnMatchesRequest

```
void qevercloud::NoteStoreServer::updateNoteIfUsnMatchesRequest (
            Note note,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.213 updateNoteIfUsnMatchesRequestReady

```
void qevercloud::NoteStoreServer::updateNoteIfUsnMatchesRequestReady (
            QByteArray data )  [signal]
```

7.80.3.214 updateNoteRequest

```
void qevercloud::NoteStoreServer::updateNoteRequest (
            Note note,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.215 updateNoteRequestReady

```
void qevercloud::NoteStoreServer::updateNoteRequestReady (
            QByteArray data )  [signal]
```

7.80.3.216 updateResourceRequest

```
void qevercloud::NoteStoreServer::updateResourceRequest (
            Resource resource,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.217 updateResourceRequestReady

```
void qevercloud::NoteStoreServer::updateResourceRequestReady (
            QByteArray data )  [signal]
```

7.80.3.218 updateSearchRequest

```
void qevercloud::NoteStoreServer::updateSearchRequest (
            SavedSearch search,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.219 updateSearchRequestReady

```
void qevercloud::NoteStoreServer::updateSearchRequestReady (
            QByteArray data )  [signal]
```

7.80.3.220 updateSharedNotebookRequest

```
void qevercloud::NoteStoreServer::updateSharedNotebookRequest (
            SharedNotebook sharedNotebook,
            IRequestContextPtr ctx )  [signal]
```

7.80.3.221 updateSharedNotebookRequestReady

```
void qevercloud::NoteStoreServer::updateSharedNotebookRequestReady (
            QByteArray data )  [signal]
```

7.80.3.222 updateTagRequest

```
void qevercloud::NoteStoreServer::updateTagRequest (
            Tag tag,
            IRequestContextPtr ctx )  [signal]
```

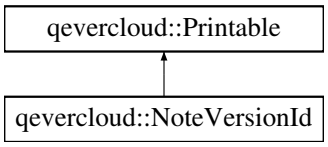
7.80.3.223 updateTagRequestReady

```
void qevercloud::NoteStoreServer::updateTagRequestReady (
            QByteArray data )  [signal]
```

7.81 qevercloud::NoteVersionId Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::NoteVersionId:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const NoteVersionId &other) const
- bool operator!= (const NoteVersionId &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- qint32 updateSequenceNum = 0
- Timestamp updated = 0
- Timestamp saved = 0
- QString title
- Optional< UserID > lastEditorId

7.81.1 Detailed Description

Identifying information about previous versions of a note that are backed up within Evernote's servers. Used in the return value of the listNoteVersions call.

7.81.2 Member Function Documentation

7.81.2.1 operator"!=()

```
bool qevercloud::NoteVersionId::operator!= (
            const NoteVersionId & other ) const  [inline]
```

7.81.2.2 operator==()

```
bool qevercloud::NoteVersionId::operator== (
            const NoteVersionId & other ) const  [inline]
```

7.81.2.3 print()

```
virtual void qevercloud::NoteVersionId::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.81.3 Member Data Documentation

7.81.3.1 lastEditorId

```
Optional< UserID > qevercloud::NoteVersionId::lastEditorId
```

The ID of the user who made the change to this version of the note. This will be unset if the note version was edited by the owner of the account.

7.81.3.2 localData

```
EverCloudLocalData qevercloud::NoteVersionId::localData
```

See the declaration of EverCloudLocalData for details

7.81.3.3 saved

```
Timestamp qevercloud::NoteVersionId::saved = 0
```

A timestamp that holds the date and time when this version of the note was backed up by Evernote's servers.

7.81.3.4 title

```
QString qevercloud::NoteVersionId::title
```

The title of the note when this particular version was saved. (The current title of the note may differ from this value.)

7.81.3.5 updated

```
Timestamp qevercloud::NoteVersionId::updated = 0
```

The 'updated' time that was set on the Note when it had this version of the content. This is the user-modifiable modification time on the note, so it's not reliable for guaranteeing the order of various versions. (E.g. if someone modifies the note, then changes this time manually into the past and then updates the note again.)

7.81.3.6 updateSequenceNum

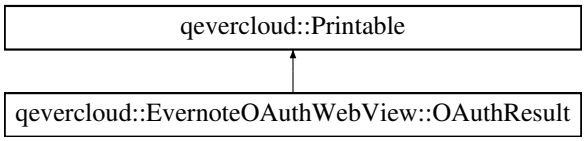
```
qint32 qevercloud::NoteVersionId::updateSequenceNum = 0
```

The update sequence number for the Note when it last had this content. This serves to uniquely identify each version of the note, since USN values are unique within an account for each update.

7.82 qevercloud::EvernoteOAuthWebView::OAuthResult Struct Reference

```
#include <OAuth.h>
```

Inheritance diagram for qevercloud::EvernoteOAuthWebView::OAuthResult:



Public Member Functions

- virtual void print (QTextStream &strm) const override

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- QString noteStoreUrl
- Timestamp expires

  authenticationToken time of expiration.
- QString shardId

  usually is not used
- UserID userId

  same as PublicUserInfo::userId
- QString webApiUrlPrefix

  see PublicUserInfo::webApiUrlPrefix
- QString authenticationToken

  This is what this all was for!
- QList< QNetworkCookie > cookies

7.82.1 Detailed Description

Holds data that is returned by Evernote on a successful authentication

7.82.2 Member Function Documentation

7.82.2.1 print()

```
virtual void qevercloud::EvernoteOAuthWebView::OAuthResult::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.82.3 Member Data Documentation

7.82.3.1 authenticationToken

```
QString qevercloud::EvernoteOAuthWebView::OAuthResult::authenticationToken
```

This is what this all was for!

Cookies set by Evernote during OAuth procedure. In April 2020 these cookies silently started to be required for UserStore API calls. Probably it was a bug on Evernote side which hopefully would be fixed at some point but nevertheless cookies set during OAuth procedure are now available as a part of OAuth result and can be used in subsequent calls to Evernote service. These cookies can be set when creating an instance of IRequestContext. Then this context can be used in QEver↩ Cloud calls. Cookies from context would propagate to HTTP requests performed by QEverCloud. See this thread on Evernote discussions for more details: https://discussion.evernote.↩ com/topic/124257-calls-to-userstore-from-evernote-api-stopped-working

7.82.3.2 cookies

```
QList<QNetworkCookie> qevercloud::EvernoteOAuthWebView::OAuthResult::cookies
```

7.82.3.3 expires

```
Timestamp qevercloud::EvernoteOAuthWebView::OAuthResult::expires
```

authenticationToken time of expiration.

7.82.3.4 noteStoreUrl

```
QString qevercloud::EvernoteOAuthWebView::OAuthResult::noteStoreUrl
```

note store url for the user; no need to question UserStore::getNoteStoreUrl for it.

7.82.3.5 shardId

```
QString qevercloud::EvernoteOAuthWebView::OAuthResult::shardId
```

usually is not used

7.82.3.6 userId

```
UserID qevercloud::EvernoteOAuthWebView::OAuthResult::userId
```

same as PublicUserInfo::userId

7.82.3.7 webApiUrlPrefix

```
QString qevercloud::EvernoteOAuthWebView::OAuthResult::webApiUrlPrefix
```

see PublicUserInfo::webApiUrlPrefix

7.83 qevercloud::Optional< T > Class Template Reference

```
#include <Optional.h>
```

Public Member Functions

- Optional ()
- Optional (const Optional &o)
- template<typename X >
  Optional (const Optional< X > &o)
- Optional (const T &value)
- template<typename X >
  Optional (const X &value)
- Optional & operator= (const Optional &o)
- template<typename X >
  Optional & operator= (const Optional< X > &o)
- Optional & operator= (const T &value)
- template<typename X >
  Optional & operator= (const X &value)
- operator const T & () const
- operator T& ()
- const T & ref () const
- T & ref ()
- bool isSet () const

  Checks if value is set.
- void clear ()
- Optional & init ()
- T * operator-> ()
- const T * operator-> () const
- T value (T defaultValue=T()) const
- bool isEqual (const Optional< T > &other) const
- bool operator== (const Optional< T > &other) const
- bool operator!= (const Optional< T > &other) const
- bool operator== (const T &other) const
- bool operator!= (const T &other) const
- Optional (Optional &&other)
- Optional & operator= (Optional &&other)
- Optional (T &&other)
- Optional & operator= (T &&other)

Friends

- template<typename X >
  class Optional
- void swap (Optional &first, Optional &second)

7.83.1 Detailed Description

template<typename T>
class qevercloud::Optional< T >

Supports optional values.

Most of the fields in the Evernote API structs are optional. But C++ does not support this notion directly.

To implement the concept of optional values conventional Thrift C++ wrapper uses a special field of a struct type where each field is of type bool with the same name as a field in the struct. This bool flag indicated was the field with the same name in the outer struct assigned or not.

While this method have its advantages (obviousness and simplicity) I found it very inconvenient to work with. You have to check by hand that both values (value itself and its __isset flag) are in sync. There is no checks whatsoever against an error and such an error is too easy to make.

So for my library I created a special class that supports the optional value notion explicitly. Basically Optional class just holds a bool value that tracks the fact that a value was assigned. But this tracking is done automatically and attempts to use unassigned values throw exceptions. In this way errors are much harder to make and it's harder for them to slip through testing unnoticed too.

7.83.2 Constructor & Destructor Documentation

7.83.2.1 Optional() [1/7]

```
template<typename T >
qevercloud::Optional< T >::Optional ( )  [inline]
```

Default constructor. Default Optional is not set.

7.83.2.2 Optional() [2/7]

```
template<typename T >
qevercloud::Optional< T >::Optional (
            const Optional< T > & o )  [inline]
```

Copy constructor.

7.83.2.3 Optional() [3/7]

```
template<typename T >
template<typename X >
qevercloud::Optional< T >::Optional (
            const Optional< X > & o )  [inline]
```

Template copy constructor. Allows to be initialized with Optional of any compatible type.

7.83.2.4 Optional() [4/7]

```
template<typename T >
qevercloud::Optional< T >::Optional (
            const T & value )  [inline]
```

Initialization with a value of the type T. Note: it's implicit.

7.83.2.5 Optional() [5/7]

```
template<typename T >
template<typename X >
qevercloud::Optional< T >::Optional (
            const X & value )  [inline]
```

Template initialization with a value of any compatible type.

7.83.2.6 Optional() [6/7]

```
template<typename T >
qevercloud::Optional< T >::Optional (
            Optional< T > && other )  [inline]
```

7.83.2.7 Optional() [7/7]

```
template<typename T >
qevercloud::Optional< T >::Optional (
            T && other )  [inline]
```

7.83.3 Member Function Documentation

7.83.3.1 clear()

```
template<typename T >
void qevercloud::Optional< T >::clear ( )  [inline]
```

Clears an Optional.

```
Optional<int> o(1);
o.clear();
cout « o; // exception
```

7.83.3.2 init()

```
template<typename T >
Optional & qevercloud::Optional< T >::init ( )  [inline]
```

Fast way to initialize an Optional with a default value.

It's very useful for structs.

```
struct S2 {int f;};
struct S {int f1; Optional<S2> f2};
Optional<S> o; // o.isSet() != ture

// without init() it's cumbersome to access struct fields
// it's especially true for nested Optionals
o = S(); // now o is set
o->f2 = S2(); // and o.f2 is set
o->f2->f = 1; // so at last it can be used

// with init() it's simpler
o.init()->f2.init()->f = 1;
```

Returns

reference to itself

7.83.3.3 isEqual()

```
template<typename T >
bool qevercloud::Optional< T >::isEqual (
            const Optional< T > & other ) const  [inline]
```

Two optionals are equal if they are both not set or have equal values.

7.83.3.4 isSet()

```
template<typename T >
bool qevercloud::Optional< T >::isSet ( ) const  [inline]
```

Checks if value is set.

Returns

true if Optional have been assigned a value and false otherwise.

Access to an unassigned ("not set") Optional lead to an exception.

7.83.3.5 operator const T &()

```
template<typename T >
qevercloud::Optional< T >::operator const T & ( ) const  [inline]
```

Implicit conversion of Optional<T> to T.

const version.

7.83.3.6 operator T&()

```
template<typename T >
qevercloud::Optional< T >::operator T& ( )  [inline]
```

Implicit conversion of Optional<T> to T.

Note: a reference is returned, not a copy.

7.83.3.7 operator"!=() [1/2]

```
template<typename T >
bool qevercloud::Optional< T >::operator!= (
            const Optional< T > & other ) const  [inline]
```

7.83.3.8 operator"!=() [2/2]

```
template<typename T >
bool qevercloud::Optional< T >::operator!= (
            const T & other ) const  [inline]
```

7.83.3.9 operator->() [1/2]

```
template<typename T >
T * qevercloud::Optional< T >::operator-> ( )  [inline]
```

Two syntatic constructs come to mind to use for implementation of access to a struct's/class's field directly from Optional.

One is the dereference operator. This is what boost::optional uses. While it's conceptually nice I found it to be not a very convenient way to refer to structs, especially nested ones. So I overloaded the operator-> and use smart pointer semantics.

```
struct S1 {int f1;};
struct S2 {Optional<S1> f2;};
Optional<S2> o;

*((*o).f2).f1; // boost way, not implemented

o->f2->f1;     // QEverCloud way
```

I admit, boost::optional is much more elegant overall. It uses pointer semantics quite clearly and in an instantly understandable way. It's universal (works for any type and not just structs). There is no need for implicit type concersions and so there is no subtleties because of it. And so on.

But then referring to struct fields is a chore. And this is the most common use case of Optionals in QEverCloud.

So I decided to use non-obvious-on-the-first-sight semantics for my Optional. IMO it's much more convenient when gotten used to.

7.83.3.10 operator->() [2/2]

```
template<typename T >
const T * qevercloud::Optional< T >::operator-> ( ) const  [inline]
```

const version.

7.83.3.11 operator=() [1/6]

```
template<typename T >
Optional & qevercloud::Optional< T >::operator= (
            const Optional< T > & o )  [inline]
```

Assignment.

7.83.3.12 operator=() [2/6]

```
template<typename T >
template<typename X >
Optional & qevercloud::Optional< T >::operator= (
            const Optional< X > & o )  [inline]
```

Template assignment with an Optional of any compatible value.

7.83.3.13 operator=() [3/6]

```
template<typename T >
Optional & qevercloud::Optional< T >::operator= (
            const T & value )  [inline]
```

Assignment with a value of the type T.

7.83.3.14 operator=() [4/6]

```
template<typename T >
template<typename X >
Optional & qevercloud::Optional< T >::operator= (
            const X & value )  [inline]
```

Template assignment with a value of any compatible type.

7.83.3.15 operator=() [5/6]

```
template<typename T >
Optional & qevercloud::Optional< T >::operator= (
            Optional< T > && other )  [inline]
```

7.83.3.16 operator=() [6/6]

```
template<typename T >
Optional & qevercloud::Optional< T >::operator= (
            T && other )  [inline]
```

7.83.3.17 operator==() [1/2]

```
template<typename T >
bool qevercloud::Optional< T >::operator== (
            const Optional< T > & other ) const  [inline]
```

7.83.3.18 operator==() [2/2]

```
template<typename T >
bool qevercloud::Optional< T >::operator== (
            const T & other ) const  [inline]
```

7.83.3.19 ref() [1/2]

```
template<typename T >
T & qevercloud::Optional< T >::ref ( )  [inline]
```

Returs reference to the holded value.

There are contexts in C++ where impicit type conversions can't help. For example:
```
Optional<QStringList> l;
for(auto s : l); // you will hear from your compiler
```

Explicit type conversion can be used...
```
Optional<QStringList> l;
for(auto s : static_cast<QStringList&>(l)); // ugh...
```

... but this is indeed ugly as hell.

So I implemented ref() function that returns a reference to the holded value.
```
Optional<QStringList> l;
for(auto s : l.ref()); // not ideal but OK
```

7.83.3.20 ref() [2/2]

```
template<typename T >
const T & qevercloud::Optional< T >::ref ( ) const  [inline]
```

Returs a reference to the holded value.

const version.

7.83.3.21 value()

```
template<typename T >
T qevercloud::Optional< T >::value (
            T defaultValue = T() ) const  [inline]
```

The function is sometimes useful to simplify checking for the value being set.

Parameters

defaultValue	The value to return if Optional is not set.

Returns

Optional value if set and defaultValue otherwise.

7.83.4 Friends And Related Symbol Documentation

7.83.4.1 Optional

```
template<typename T >
template<typename X >
friend class Optional  [friend]
```

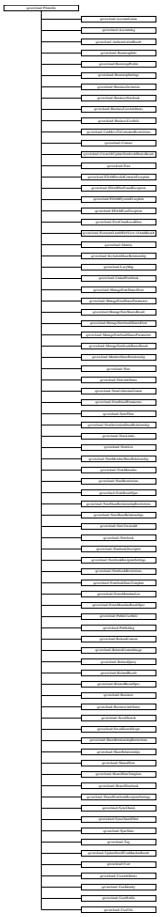
7.83.4.2 swap

```
template<typename T >
void swap (
            Optional< T > & first,
            Optional< T > & second )  [friend]
```

7.84 qevercloud::Printable Class Reference

```
#include <Printable.h>
```

Inheritance diagram for qevercloud::Printable:



Public Member Functions

- Printable ()=default
- virtual ~Printable ()=default
- virtual void print (QTextStream &strm) const =0
- virtual QString toString () const

Friends

- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &strm, const Printable &printable)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &dbg, const Printable &printable)

7.84.1 Constructor & Destructor Documentation

7.84.1.1 Printable()

```
qevercloud::Printable::Printable ( )  [default]
```

7.84.1.2 ~Printable()

```
virtual qevercloud::Printable::~Printable ( )  [virtual], [default]
```

7.84.2 Member Function Documentation

7.84.2.1 print()

```
virtual void qevercloud::Printable::print (
            QTextStream & strm ) const  [pure virtual]
```

Implemented in qevercloud::EverCloudLocalData, qevercloud::SyncState, qevercloud::SyncChunkFilter, qevercloud::NoteFilter, qevercloud::NotesMetadataResultSpec, qevercloud::NoteCollectionCounts, qevercloud::NoteResultSpec, qevercloud::NoteVersionId, qevercloud::RelatedQuery, qevercloud::RelatedResultSpec, qevercloud::ShareRelationshipRestrictions, qevercloud::MemberShareRe qevercloud::NoteShareRelationshipRestrictions, qevercloud::NoteMemberShareRelationship, qevercloud::NoteInvitationShareRelatio qevercloud::NoteShareRelationships, qevercloud::ManageNoteSharesParameters, qevercloud::Data, qevercloud::UserAttributes, qevercloud::BusinessUserAttributes, qevercloud::Accounting, qevercloud::BusinessUserInfo, qevercloud::AccountLimits, qevercloud::User, qevercloud::Contact, qevercloud::Identity, qevercloud::Tag, qevercloud::LazyMap, qevercloud::ResourceAttributes, qevercloud::Resource, qevercloud::NoteAttributes, qevercloud::SharedNote, qevercloud::NoteRestrictions, qevercloud::NoteLimits, qevercloud::Note, qevercloud::Publishing, qevercloud::BusinessNotebook, qevercloud::SavedSearchScope, qevercloud::SavedSearch, qevercloud::SharedNotebookRecipientSettings, qevercloud::NotebookRecipientSettings, qevercloud::SharedNotebook, qevercloud::CanMoveToContainerRestrictions, qevercloud::NotebookRestrictions, qevercloud::Notebook, qevercloud::LinkedNotebook, qevercloud::NotebookDescriptor, qevercloud::UserProfile, qevercloud::RelatedContentImage, qevercloud::RelatedContent, qevercloud::BusinessInvitation, qevercloud::UserIdentity, qevercloud::PublicUserInfo, qevercloud::UserUrls, qevercloud::AuthenticationResult, qevercloud::BootstrapSettings, qevercloud::BootstrapProfile, qevercloud::BootstrapInfo, qevercloud::EDAMUserException, qevercloud::EDAMSystemException, qevercloud::EDAMNotFoundException, qevercloud::EDAMInvalidContactsException, qevercloud::SyncChunk, qevercloud::NoteList, qevercloud::NoteMetadata, qevercloud::NotesMetadataList, qevercloud::NoteEmailParameters, qevercloud::RelatedResult, qevercloud::UpdateNoteIfUsnMatchesResult, qevercloud::InvitationShareRelationship, qevercloud::ShareRelationships, qevercloud::ManageNotebookSharesParameters, qevercloud::ManageNotebookSharesError, qevercloud::ManageNotebookSharesResult, qevercloud::SharedNoteTemplate, qevercloud::NotebookShareTemplate, qevercloud::CreateOrUpdateNotebookSharesResult, qevercloud::ManageNoteSharesError, qevercloud::ManageNoteSharesResult, and qevercloud::EvernoteOAuthWebView::OAuthResult.

7.84.2.2 toString()

```
virtual QString qevercloud::Printable::toString ( ) const  [virtual]
```

7.84.3 Friends And Related Symbol Documentation

7.84.3.1 operator<< [1/2]

```
QEVERCLOUD_EXPORT QDebug & operator<< (
            QDebug & dbg,
            const Printable & printable )  [friend]
```

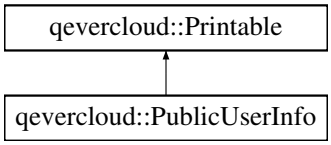
7.84.3.2 operator<< [2/2]

```
QEVERCLOUD_EXPORT QTextStream & operator<< (
            QTextStream & strm,
            const Printable & printable )  [friend]
```

7.85 qevercloud::PublicUserInfo Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::PublicUserInfo:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const PublicUserInfo &other) const
- bool operator!= (const PublicUserInfo &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- UserID userId = 0
- Optional< ServiceLevel > serviceLevel
- Optional< QString > username
- Optional< QString > noteStoreUrl
- Optional< QString > webApiUrlPrefix

7.85.1 Detailed Description

This structure is used to provide publicly-available user information about a particular account.

7.85.2 Member Function Documentation

7.85.2.1 operator"!=()

```
bool qevercloud::PublicUserInfo::operator!= (
            const PublicUserInfo & other ) const  [inline]
```

7.85.2.2 operator==()

```
bool qevercloud::PublicUserInfo::operator== (
            const PublicUserInfo & other ) const  [inline]
```

7.85.2.3 print()

```
virtual void qevercloud::PublicUserInfo::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.85.3 Member Data Documentation

7.85.3.1 localData

```
EverCloudLocalData qevercloud::PublicUserInfo::localData
```

See the declaration of EverCloudLocalData for details

7.85.3.2 noteStoreUrl

```
Optional< QString > qevercloud::PublicUserInfo::noteStoreUrl
```

This field will contain the full URL that clients should use to make NoteStore requests to the server shard that contains that user's data. I.e. this is the URL that should be used to create the Thrift HTTP client transport to send messages to the NoteStore service for the account.

7.85.3.3 serviceLevel

```
Optional< ServiceLevel > qevercloud::PublicUserInfo::serviceLevel
```

The service level of the account.

7.85.3.4 userId

```
UserID qevercloud::PublicUserInfo::userId = 0
```

The unique numeric user identifier for the user account.

7.85.3.5 username

```
Optional< QString > qevercloud::PublicUserInfo::username
```

NOT DOCUMENTED

7.85.3.6 webApiUrlPrefix

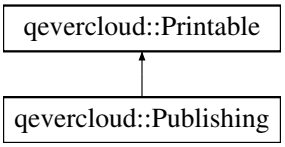
```
Optional< QString > qevercloud::PublicUserInfo::webApiUrlPrefix
```

This field will contain the initial part of the URLs that should be used to make requests to Evernote's thin client "web API", which provide optimized operations for clients that aren't capable of manipulating the full contents of accounts via the full Thrift data model. Clients should concatenate the relative path for the various servlets onto the end of this string to construct the full URL, as documented on our developer web site.

7.86 qevercloud::Publishing Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::Publishing:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const Publishing &other) const
- bool operator!= (const Publishing &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > uri
- Optional< NoteSortOrder > order
- Optional< bool > ascending
- Optional< QString > publicDescription

7.86.1 Detailed Description

If a Notebook has been opened to the public, the Notebook will have a reference to one of these structures, which gives the location and optional description of the externally-visible public Notebook.

7.86.2 Member Function Documentation

7.86.2.1 operator"!=()

```
bool qevercloud::Publishing::operator!= (
            const Publishing & other ) const  [inline]
```

7.86.2.2 operator==()

```
bool qevercloud::Publishing::operator== (
            const Publishing & other ) const  [inline]
```

7.86.2.3 print()

```
virtual void qevercloud::Publishing::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.86.3 Member Data Documentation

7.86.3.1 ascending

```
Optional< bool > qevercloud::Publishing::ascending
```

If this is set to true, then the public notes will be displayed in ascending order (e.g. from oldest to newest). Otherwise, the notes will be displayed in descending order (e.g. newest to oldest).

7.86.3.2 localData

```
EverCloudLocalData qevercloud::Publishing::localData
```

See the declaration of EverCloudLocalData for details

7.86.3.3 order

```
Optional< NoteSortOrder > qevercloud::Publishing::order
```

When the notes are publicly displayed, they will be sorted based on the requested criteria.

7.86.3.4 publicDescription

```
Optional< QString > qevercloud::Publishing::publicDescription
```

This field may be used to provide a short description of the notebook, which may be displayed when (e.g.) the notebook is shown in a public view. Can't begin or end with a space.
Length: EDAM_PUBLISHING_DESCRIPTION_LEN_MIN - EDAM_PUBLISHING_DESCRIPTION_LEN_MAX
Regex: EDAM_PUBLISHING_DESCRIPTION_REGEX

7.86.3.5 uri

```
Optional< QString > qevercloud::Publishing::uri
```

If this field is present, then the notebook is published for mass consumption on the Internet under the provided URI, which is relative to a defined base publishing URI defined by the service. This field can only be modified via the web service GUI ... publishing cannot be modified via an offline client.
Length: EDAM_PUBLISHING_URI_LEN_MIN - EDAM_PUBLISHING_URI_LEN_MAX
Regex: EDAM_PUBLISHING_URI_REGEX

7.87 qevercloud::QAssociativeContainerConstReferenceWrapper< Container > Class Template Reference

`#include <Helpers.h>`

Classes

- struct iterator

Public Member Functions

- QAssociativeContainerConstReferenceWrapper (const Container &container)
- iterator begin () const
- iterator end () const

7.87.1 Constructor & Destructor Documentation

7.87.1.1 QAssociativeContainerConstReferenceWrapper()

```
template<typename Container >
qevercloud::QAssociativeContainerConstReferenceWrapper< Container >::QAssociativeContainer↩
ConstReferenceWrapper (
            const Container & container )  [inline]
```

7.87.2 Member Function Documentation

7.87.2.1 begin()

```
template<typename Container >
iterator qevercloud::QAssociativeContainerConstReferenceWrapper< Container >::begin ( ) const
[inline]
```

7.87.2.2 end()

```
template<typename Container >
iterator qevercloud::QAssociativeContainerConstReferenceWrapper< Container >::end ( ) const
[inline]
```

7.88 qevercloud::QAssociativeContainerReferenceWrapper< Container > Class Template Reference

`#include <Helpers.h>`

Classes

- struct iterator

Public Member Functions

- QAssociativeContainerReferenceWrapper (Container &container)
- iterator begin ()
- iterator end ()

7.88.1 Constructor & Destructor Documentation

7.88.1.1 QAssociativeContainerReferenceWrapper()

```
template<typename Container >
qevercloud::QAssociativeContainerReferenceWrapper< Container >::QAssociativeContainerReference↩
Wrapper (
            Container & container )  [inline]
```

7.88.2 Member Function Documentation

7.88.2.1 begin()

```
template<typename Container >
iterator qevercloud::QAssociativeContainerReferenceWrapper< Container >::begin ( )  [inline]
```

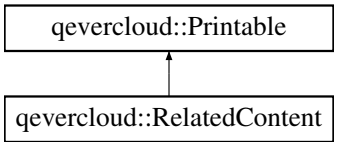
7.88.2.2 end()

```
template<typename Container >
iterator qevercloud::QAssociativeContainerReferenceWrapper< Container >::end ( )  [inline]
```

7.89 qevercloud::RelatedContent Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::RelatedContent:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const RelatedContent &other) const
- bool operator!= (const RelatedContent &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > contentId
- Optional< QString > title
- Optional< QString > url
- Optional< QString > sourceId
- Optional< QString > sourceUrl
- Optional< QString > sourceFaviconUrl
- Optional< QString > sourceName
- Optional< Timestamp > date
- Optional< QString > teaser
- Optional< QList< RelatedContentImage > > thumbnails
- Optional< RelatedContentType > contentType
- Optional< RelatedContentAccess > accessType
- Optional< QString > visibleUrl
- Optional< QString > clipUrl
- Optional< Contact > contact
- Optional< QStringList > authors

Properties

- OptionalQList< RelatedContentImage > thumbnails

7.89.1 Detailed Description

A structure identifying one snippet of related content (some information that is not part of an Evernote account but might still be relevant to the user).

7.89.2 Member Function Documentation

7.89.2.1 operator"!=()

```
bool qevercloud::RelatedContent::operator!= (
            const RelatedContent & other ) const  [inline]
```

7.89.2.2 operator==()

```
bool qevercloud::RelatedContent::operator== (
            const RelatedContent & other ) const  [inline]
```

7.89.2.3 print()

```
virtual void qevercloud::RelatedContent::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.89.3 Member Data Documentation

7.89.3.1 accessType

```
Optional< RelatedContentAccess > qevercloud::RelatedContent::accessType
```

An indication of how this content can be accessed. This type influences the semantics of the `url` parameter.

7.89.3.2 authors

```
Optional< QStringList > qevercloud::RelatedContent::authors
```

For News articles only. A list of names of the article authors, if available.

7.89.3.3 clipUrl

```
Optional< QString > qevercloud::RelatedContent::clipUrl
```

If set, the client should use this URL for clipping purposes, instead of the URL that was used to retrieve the content. The clipUrl may directly point to an .enex file, for example.

7.89.3.4 contact

```
Optional< Contact > qevercloud::RelatedContent::contact
```

If set, the client may use this Contact for messaging purposes. This will typically only be set for user profiles.

7.89.3.5 contentId

```
Optional< QString > qevercloud::RelatedContent::contentId
```

An identifier that uniquely identifies the content.

7.89.3.6 contentType

```
Optional< RelatedContentType > qevercloud::RelatedContent::contentType
```

The type of this related content.

7.89.3.7 date

```
Optional< Timestamp > qevercloud::RelatedContent::date
```

A timestamp telling the user about the recency of the content.

7.89.3.8 localData

```
EverCloudLocalData qevercloud::RelatedContent::localData
```

See the declaration of EverCloudLocalData for details

7.89.3.9 sourceFaviconUrl

```
Optional< QString > qevercloud::RelatedContent::sourceFaviconUrl
```

The favicon URL of the source which the content belongs to.

7.89.3.10 sourceId

```
Optional< QString > qevercloud::RelatedContent::sourceId
```

An identifier that uniquely identifies the source.

7.89.3.11 sourceName

```
Optional< QString > qevercloud::RelatedContent::sourceName
```

A human-readable name of the source that provided this content.

7.89.3.12 sourceUrl

```
Optional< QString > qevercloud::RelatedContent::sourceUrl
```

A URL the client can access to know more about the source.

7.89.3.13 teaser

```
Optional< QString > qevercloud::RelatedContent::teaser
```

A teaser text to show to the user; usually the first few sentences of the content, excluding the title.

7.89.3.14 thumbnails

```
Optional<QList<RelatedContentImage> > qevercloud::RelatedContent::thumbnails
```

A list of thumbnails the client can show in the snippet.

7.89.3.15 title

```
Optional< QString > qevercloud::RelatedContent::title
```

The main title to show.

7.89.3.16 url

```
Optional< QString > qevercloud::RelatedContent::url
```

The URL the client can use to retrieve the content.

7.89.3.17 visibleUrl

```
Optional< QString > qevercloud::RelatedContent::visibleUrl
```

If set, the client should show this URL to the user, instead of the URL that was used to retrieve the content. This URL should be used when opening the content in an external browser window, or when sharing with another person.

7.89.4 Property Documentation

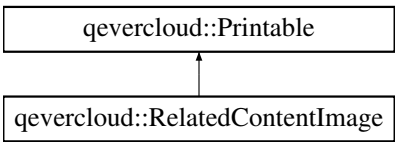
7.89.4.1 thumbnails

```
OptionalQList<RelatedContentImage> qevercloud::RelatedContent::thumbnails
```

7.90 qevercloud::RelatedContentImage Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::RelatedContentImage:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const RelatedContentImage &other) const
- bool operator!= (const RelatedContentImage &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > url
- Optional< qint32 > width
- Optional< qint32 > height
- Optional< double > pixelRatio
- Optional< qint32 > fileSize

7.90.1 Detailed Description

An external image that can be shown with a related content snippet, usually either a JPEG or PNG image. It is up to the client which image(s) are shown, depending on available screen real estate, resolution and aspect ratio.

7.90.2 Member Function Documentation

7.90.2.1 operator"!=()

```
bool qevercloud::RelatedContentImage::operator!= (
            const RelatedContentImage & other ) const  [inline]
```

7.90.2.2 operator==()

```
bool qevercloud::RelatedContentImage::operator== (
            const RelatedContentImage & other ) const  [inline]
```

7.90.2.3 print()

```
virtual void qevercloud::RelatedContentImage::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.90.3 Member Data Documentation

7.90.3.1 fileSize

```
Optional< qint32 > qevercloud::RelatedContentImage::fileSize
```

the size of the image file, in bytes

7.90.3.2 height

```
Optional< qint32 > qevercloud::RelatedContentImage::height
```

The height of the image, in pixels.

7.90.3.3 localData

```
EverCloudLocalData qevercloud::RelatedContentImage::localData
```

See the declaration of EverCloudLocalData for details

7.90.3.4 pixelRatio

```
Optional< double > qevercloud::RelatedContentImage::pixelRatio
```

the pixel ratio (usually either 1.0, 1.5 or 2.0)

7.90.3.5 url

```
Optional< QString > qevercloud::RelatedContentImage::url
```

The external URL of the image

7.90.3.6 width

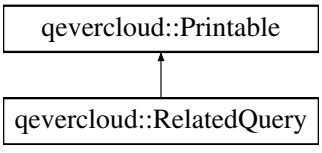
```
Optional< qint32 > qevercloud::RelatedContentImage::width
```

The width of the image, in pixels.

7.91 qevercloud::RelatedQuery Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::RelatedQuery:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const RelatedQuery &other) const
- bool operator!= (const RelatedQuery &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > noteGuid
- Optional< QString > plainText
- Optional< NoteFilter > filter
- Optional< QString > referenceUri
- Optional< QString > context
- Optional< QString > cacheKey

7.91.1 Detailed Description

A description of the thing for which we are searching for related entities.

You must specify either *noteGuid* or *plainText*, but not both. *filter* and *referenceUri* are optional.

7.91.2 Member Function Documentation

7.91.2.1 operator"!=()

```
bool qevercloud::RelatedQuery::operator!= (
            const RelatedQuery & other ) const  [inline]
```

7.91.2.2 operator==()

```
bool qevercloud::RelatedQuery::operator== (
            const RelatedQuery & other ) const  [inline]
```

7.91.2.3 print()

```
virtual void qevercloud::RelatedQuery::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.91.3 Member Data Documentation

7.91.3.1 cacheKey

```
Optional< QString > qevercloud::RelatedQuery::cacheKey
```

If set and non-empty, this is an indicator for the server whether it is actually necessary to perform a new findRelated call at all. Cache Keys are opaque strings which are returned by the server as part of "RelatedResult" in response to a "NoteStore.findRelated" query. Cache Keys are inherently query specific.

If set to an empty string, this indicates that the server should generate a cache key in the response as part of "RelatedResult".

If not set, the server will not attempt to generate a cache key at all.

7.91.3.2 context

```
Optional< QString > qevercloud::RelatedQuery::context
```

Specifies the context to consider when determining related results. Clients must leave this value unset unless they wish to explicitly specify a known non-default context.

7.91.3.3 filter

```
Optional< NoteFilter > qevercloud::RelatedQuery::filter
```

The list of criteria that will constrain the notes being considered related. Please note that some of the parameters may be ignored, such as *order* and *ascending*.

7.91.3.4 localData

```
EverCloudLocalData qevercloud::RelatedQuery::localData
```

See the declaration of EverCloudLocalData for details

7.91.3.5 noteGuid

```
Optional< QString > qevercloud::RelatedQuery::noteGuid
```

The GUID of an existing note in your account for which related entities will be found.

7.91.3.6 plainText

```
Optional< QString > qevercloud::RelatedQuery::plainText
```

A string of plain text for which to find related entities. You should provide a text block with a number of characters between EDAM_RELATED_PLAINTEXT_LEN_MIN and EDAM_RELATED_PLAINTEXT_LEN_MAX.

7.91.3.7 referenceUri

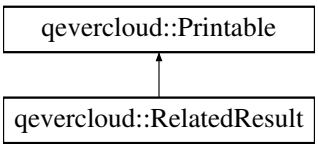
```
Optional< QString > qevercloud::RelatedQuery::referenceUri
```

A URI string specifying a reference entity, around which "relatedness" should be based. This can be an URL pointing to a web page, for example.

7.92 qevercloud::RelatedResult Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::RelatedResult:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const RelatedResult &other) const
- bool operator!= (const RelatedResult &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QList< Note > > notes
- Optional< QList< Notebook > > notebooks
- Optional< QList< Tag > > tags
- Optional< QList< NotebookDescriptor > > containingNotebooks
- Optional< QString > debugInfo
- Optional< QList< UserProfile > > experts
- Optional< QList< RelatedContent > > relatedContent
- Optional< QString > cacheKey
- Optional< qint32 > cacheExpires

Properties

- OptionalQList< Note > notes
- OptionalQList< Notebook > notebooks
- OptionalQList< Tag > tags
- OptionalQList< NotebookDescriptor > containingNotebooks
- OptionalQList< UserProfile > experts
- OptionalQList< RelatedContent > relatedContent

7.92.1 Detailed Description

The result of calling findRelated(). The contents of the notes, notebooks, and tags fields will be in decreasing order of expected relevance. It is possible that fewer results than requested will be returned even if there are enough distinct entities in the account in cases where the relevance is estimated to be low.

7.92.2 Member Function Documentation

7.92.2.1 operator"!=()

```
bool qevercloud::RelatedResult::operator!= (
            const RelatedResult & other ) const  [inline]
```

7.92.2.2 operator==()

```
bool qevercloud::RelatedResult::operator== (
            const RelatedResult & other ) const  [inline]
```

7.92.2.3 print()

```
virtual void qevercloud::RelatedResult::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.92.3 Member Data Documentation

7.92.3.1 cacheExpires

```
Optional< qint32 > qevercloud::RelatedResult::cacheExpires
```

If set, clients should reuse this response for any situations where the same input parameters are applicable for up to this many seconds after receiving this result.

After this time has passed, the client may request a new result from the service, but it should supply the stored cacheKey to the service when checking for an update.

7.92.3.2 cacheKey

```
Optional< QString > qevercloud::RelatedResult::cacheKey
```

If set and non-empty, this cache key may be used in subsequent "NoteStore.findRelated" calls (via "RelatedQuery") to re-use previous responses that were cached on the client-side, instead of actually performing another search.

If set to an empty string, this indicates that the server could not determine a specific key for this response, but the client should nevertheless remove any previously cached result for this request.

If unset/null, it is up to the client whether to re-use cached results or to use the server's response.

If set to the exact non-empty cache key that was specified in "RelatedQuery.cacheKey", this indicates that the server decided that cached results could be reused.

Depending on the cache key specified in the query, the "RelatedResult" may only be partially filled. For each set field, the client should replace the corresponding part in the previously cached result with the new partial result.

For example, for a specific query that has both "RelatedResultSpec.maxNotes" and "RelatedResultSpec.max↩ RelatedContent" set to positive values, the server may decide that the previously requested and cached *Related Content* are unchanged, but new results for *Related Notes* are available. The response will have a new cache key and have "RelatedResult.notes" set, but have "RelatedResult.relatedContent" unset (not just empty, but really unset).

In this situation, the client should replace any cached notes with the newly returned "RelatedResult.notes", but it can re-use the previously cached entries for "RelatedResult.relatedContent". List fields that are set, but empty indicate that no results could be found; the cache should be updated correspondingly.

7.92.3.3 containingNotebooks

```
Optional<QList<NotebookDescriptor> > qevercloud::RelatedResult::containingNotebooks
```

If `includeContainingNotebooks` is set to `true` in the RelatedResultSpec, return the list of notebooks to to which the returned related notes belong. The notebooks in this list will occur once per notebook GUID and are represented as NotebookDescriptor objects.

7.92.3.4 debugInfo

```
Optional< QString > qevercloud::RelatedResult::debugInfo
```

NOT DOCUMENTED

7.92.3.5 experts

```
Optional<QList<UserProfile> > qevercloud::RelatedResult::experts
```

If experts have been requested to be included, this will return a list of users within your business who have knowledge about the specified query.

7.92.3.6 localData

EverCloudLocalData qevercloud::RelatedResult::localData

See the declaration of EverCloudLocalData for details

7.92.3.7 notebooks

Optional<QList<Notebook> > qevercloud::RelatedResult::notebooks

If notebooks have been requested to be included, this will be the list of notebooks.

7.92.3.8 notes

Optional<QList<Note> > qevercloud::RelatedResult::notes

If notes have been requested to be included, this will be the list of notes.

7.92.3.9 relatedContent

Optional<QList<RelatedContent> > qevercloud::RelatedResult::relatedContent

If related content has been requested to be included, this will be the list of related content snippets.

7.92.3.10 tags

Optional<QList<Tag> > qevercloud::RelatedResult::tags

If tags have been requested to be included, this will be the list of tags.

7.92.4 Property Documentation

7.92.4.1 containingNotebooks

OptionalQList<NotebookDescriptor> qevercloud::RelatedResult::containingNotebooks

7.92.4.2 experts

OptionalQList<UserProfile> qevercloud::RelatedResult::experts

7.92.4.3 notebooks

OptionalQList<Notebook> qevercloud::RelatedResult::notebooks

7.92.4.4 notes

```
OptionalQList<Note> qevercloud::RelatedResult::notes
```

7.92.4.5 relatedContent

```
OptionalQList<RelatedContent> qevercloud::RelatedResult::relatedContent
```

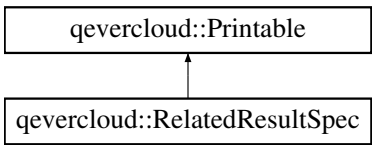
7.92.4.6 tags

```
OptionalQList<Tag> qevercloud::RelatedResult::tags
```

7.93 qevercloud::RelatedResultSpec Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::RelatedResultSpec:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const RelatedResultSpec &other) const
- bool operator!= (const RelatedResultSpec &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< qint32 > maxNotes
- Optional< qint32 > maxNotebooks
- Optional< qint32 > maxTags
- Optional< bool > writableNotebooksOnly
- Optional< bool > includeContainingNotebooks
- Optional< bool > includeDebugInfo
- Optional< qint32 > maxExperts
- Optional< qint32 > maxRelatedContent
- Optional< QSet< RelatedContentType > > relatedContentTypes

Properties

- OptionalQSet< RelatedContentType > relatedContentTypes

7.93.1 Detailed Description

A description of the thing for which the service will find related entities, via findRelated(), together with a description of what type of entities and how many you are seeking in the RelatedResult.

7.93.2 Member Function Documentation

7.93.2.1 operator"!=()

```
bool qevercloud::RelatedResultSpec::operator!= (
            const RelatedResultSpec & other ) const  [inline]
```

7.93.2.2 operator==()

```
bool qevercloud::RelatedResultSpec::operator== (
            const RelatedResultSpec & other ) const  [inline]
```

7.93.2.3 print()

```
virtual void qevercloud::RelatedResultSpec::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.93.3 Member Data Documentation

7.93.3.1 includeContainingNotebooks

```
Optional< bool > qevercloud::RelatedResultSpec::includeContainingNotebooks
```

If set to `true`, return the containingNotebooks field in the RelatedResult, which will contain the list of notebooks to to which the returned related notes belong.

7.93.3.2 includeDebugInfo

```
Optional< bool > qevercloud::RelatedResultSpec::includeDebugInfo
```

If set to `true`, indicate that debug information should be returned in the 'debugInfo' field of RelatedResult. Note that the call may be slower if this flag is set.

7.93.3.3 localData

```
EverCloudLocalData qevercloud::RelatedResultSpec::localData
```

See the declaration of EverCloudLocalData for details

7.93.3.4 maxExperts

```
Optional< qint32 > qevercloud::RelatedResultSpec::maxExperts
```

This can only be used when making a findRelated call against a business. Find users within your business who have knowledge about the specified query. No more than this many users will be returned. Any value greater than EDAM_RELATED_MAX_EXPERTS will be silently capped.

7.93.3.5 maxNotebooks

```
Optional< qint32 > qevercloud::RelatedResultSpec::maxNotebooks
```

Return notebooks that are related to the query, but no more than this many. Any value greater than EDAM↩ _RELATED_MAX_NOTEBOOKS will be silently capped. If you do not set this field, then no notebooks will be returned.

7.93.3.6 maxNotes

```
Optional< qint32 > qevercloud::RelatedResultSpec::maxNotes
```

Return notes that are related to the query, but no more than this many. Any value greater than EDAM_RELATED↩ _MAX_NOTES will be silently capped. If you do not set this field, then no notes will be returned.

7.93.3.7 maxRelatedContent

```
Optional< qint32 > qevercloud::RelatedResultSpec::maxRelatedContent
```

Return snippets of related content that is related to the query, but no more than this many. Any value greater than EDAM_RELATED_MAX_RELATED_CONTENT will be silently capped. If you do not set this field, then no related content will be returned.

7.93.3.8 maxTags

```
Optional< qint32 > qevercloud::RelatedResultSpec::maxTags
```

Return tags that are related to the query, but no more than this many. Any value greater than EDAM_RELATED_↩ MAX_TAGS will be silently capped. If you do not set this field, then no tags will be returned.

7.93.3.9 relatedContentTypes

```
Optional<QSet<RelatedContentType> > qevercloud::RelatedResultSpec::relatedContentTypes
```

Specifies the types of Related Content that should be returned.

7.93.3.10 writableNotebooksOnly

`Optional< bool > qevercloud::RelatedResultSpec::writableNotebooksOnly`

Require that all returned related notebooks are writable. The user will be able to create notes in all returned notebooks. However, individual notes returned may still belong to notebooks in which the user lacks the ability to create notes.

7.93.4 Property Documentation

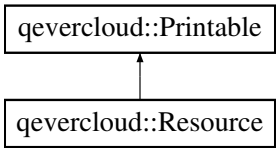
7.93.4.1 relatedContentTypes

`OptionalQSet<RelatedContentType> qevercloud::RelatedResultSpec::relatedContentTypes`

7.94 qevercloud::Resource Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::Resource:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const Resource &other) const
- bool operator!= (const Resource &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< Guid > guid
- Optional< Guid > noteGuid
- Optional< Data > data
- Optional< QString > mime
- Optional< qint16 > width
- Optional< qint16 > height
- Optional< qint16 > duration
- Optional< bool > active
- Optional< Data > recognition
- Optional< ResourceAttributes > attributes
- Optional< qint32 > updateSequenceNum
- Optional< Data > alternateData

7.94.1 Detailed Description

Every media file that is embedded or attached to a note is represented through a Resource entry.

7.94.2 Member Function Documentation

7.94.2.1 operator"!=()

```
bool qevercloud::Resource::operator!= (
            const Resource & other ) const  [inline]
```

7.94.2.2 operator==()

```
bool qevercloud::Resource::operator== (
            const Resource & other ) const  [inline]
```

7.94.2.3 print()

```
virtual void qevercloud::Resource::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.94.3 Member Data Documentation

7.94.3.1 active

```
Optional< bool > qevercloud::Resource::active
```

If the resource is active or not.

7.94.3.2 alternateData

```
Optional< Data > qevercloud::Resource::alternateData
```

Some Resources may be assigned an alternate data format by the service which may be more appropriate for indexing or rendering than the original data provided by the user. In these cases, the alternate data form will be available via this Data element. If a Resource has no alternate form, this field will be unset.

7.94.3.3 attributes

```
Optional< ResourceAttributes > qevercloud::Resource::attributes
```

A list of the attributes for this resource.

7.94.3.4 data

`Optional< Data > qevercloud::Resource::data`

The contents of the resource. Maximum length: The data.body is limited to EDAM_RESOURCE_SIZE_MAX_FREE for free accounts and EDAM_RESOURCE_SIZE_MAX_PREMIUM for premium accounts.

7.94.3.5 duration

`Optional< qint16 > qevercloud::Resource::duration`

DEPRECATED: ignored.

7.94.3.6 guid

`Optional< Guid > qevercloud::Resource::guid`

The unique identifier of this resource. Will be set whenever a resource is retrieved from the service, but may be null when a client is creating a resource.
Length: EDAM_GUID_LEN_MIN - EDAM_GUID_LEN_MAX
Regex: EDAM_GUID_REGEX

7.94.3.7 height

`Optional< qint16 > qevercloud::Resource::height`

If set, this contains the display height of this resource, in pixels.

7.94.3.8 localData

`EverCloudLocalData qevercloud::Resource::localData`

See the declaration of EverCloudLocalData for details

7.94.3.9 mime

`Optional< QString > qevercloud::Resource::mime`

The MIME type for the embedded resource. E.g. "image/gif"
Length: EDAM_MIME_LEN_MIN - EDAM_MIME_LEN_MAX
Regex: EDAM_MIME_REGEX

7.94.3.10 noteGuid

`Optional< Guid > qevercloud::Resource::noteGuid`

The unique identifier of the Note that holds this Resource. Will be set whenever the resource is retrieved from the service, but may be null when a client is creating a resource.
Length: EDAM_GUID_LEN_MIN - EDAM_GUID_LEN_MAX
Regex: EDAM_GUID_REGEX

7.94.3.11 recognition

```
Optional< Data > qevercloud::Resource::recognition
```

If set, this will hold the encoded data that provides information on search and recognition within this resource.

7.94.3.12 updateSequenceNum

```
Optional< qint32 > qevercloud::Resource::updateSequenceNum
```

A number identifying the last transaction to modify the state of this object. The USN values are sequential within an account, and can be used to compare the order of modifications within the service.

7.94.3.13 width

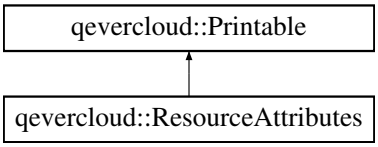
```
Optional< qint16 > qevercloud::Resource::width
```

If set, this contains the display width of this resource, in pixels.

7.95 qevercloud::ResourceAttributes Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::ResourceAttributes:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const ResourceAttributes &other) const
- bool operator!= (const ResourceAttributes &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > sourceURL
- Optional< Timestamp > timestamp
- Optional< double > latitude
- Optional< double > longitude
- Optional< double > altitude
- Optional< QString > cameraMake
- Optional< QString > cameraModel
- Optional< bool > clientWillIndex
- Optional< QString > recoType
- Optional< QString > fileName
- Optional< bool > attachment
- Optional< LazyMap > applicationData

7.95.1 Detailed Description

Structure holding the optional attributes of a Resource

7.95.2 Member Function Documentation

7.95.2.1 operator"!=()

```
bool qevercloud::ResourceAttributes::operator!= (
            const ResourceAttributes & other ) const  [inline]
```

7.95.2.2 operator==()

```
bool qevercloud::ResourceAttributes::operator== (
            const ResourceAttributes & other ) const  [inline]
```

7.95.2.3 print()

```
virtual void qevercloud::ResourceAttributes::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.95.3 Member Data Documentation

7.95.3.1 altitude

```
Optional< double > qevercloud::ResourceAttributes::altitude
```

the altitude where the resource was captured

7.95.3.2 applicationData

`Optional< LazyMap > qevercloud::ResourceAttributes::applicationData`

Provides a location for applications to store a relatively small (4kb) blob of data associated with a Resource that is not visible to the user and that is opaque to the Evernote service. A single application may use at most one entry in this map, using its API consumer key as the map key. See the documentation for LazyMap for a description of when the actual map values are returned by the service.

To safely add or modify your application's entry in the map, use NoteStore.setResourceApplicationDataEntry. To safely remove your application's entry from the map, use NoteStore.unsetResourceApplicationDataEntry.

Minimum length of a name (key): EDAM_APPLICATIONDATA_NAME_LEN_MIN
Sum max size of key and value: EDAM_APPLICATIONDATA_ENTRY_LEN_MAX
Syntax regex for name (key): EDAM_APPLICATIONDATA_NAME_REGEX

7.95.3.3 attachment

`Optional< bool > qevercloud::ResourceAttributes::attachment`

this will be true if the resource should be displayed as an attachment, or false if the resource should be displayed inline (if possible).

7.95.3.4 cameraMake

`Optional< QString > qevercloud::ResourceAttributes::cameraMake`

information about an image's camera, e.g. as embedded in the image's EXIF data
Length: EDAM_ATTRIBUTE_LEN_MIN - EDAM_ATTRIBUTE_LEN_MAX

7.95.3.5 cameraModel

`Optional< QString > qevercloud::ResourceAttributes::cameraModel`

information about an image's camera, e.g. as embedded in the image's EXIF data
Length: EDAM_ATTRIBUTE_LEN_MIN - EDAM_ATTRIBUTE_LEN_MAX

7.95.3.6 clientWillIndex

`Optional< bool > qevercloud::ResourceAttributes::clientWillIndex`

if true, then the original client that submitted the resource plans to submit the recognition index for this resource at a later time.

7.95.3.7 fileName

`Optional< QString > qevercloud::ResourceAttributes::fileName`

if the resource came from a source that provided an explicit file name, the original name will be stored here. Many resources come from unnamed sources, so this will not always be set.

7.95.3.8 latitude

```
Optional< double > qevercloud::ResourceAttributes::latitude
```

the latitude where the resource was captured

7.95.3.9 localData

```
EverCloudLocalData qevercloud::ResourceAttributes::localData
```

See the declaration of EverCloudLocalData for details

7.95.3.10 longitude

```
Optional< double > qevercloud::ResourceAttributes::longitude
```

the longitude where the resource was captured

7.95.3.11 recoType

```
Optional< QString > qevercloud::ResourceAttributes::recoType
```

DEPRECATED - this field is no longer set by the service, so should be ignored.

7.95.3.12 sourceURL

```
Optional< QString > qevercloud::ResourceAttributes::sourceURL
```

the original location where the resource was hosted
Length: EDAM_ATTRIBUTE_LEN_MIN - EDAM_ATTRIBUTE_LEN_MAX

7.95.3.13 timestamp

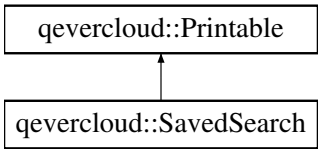
```
Optional< Timestamp > qevercloud::ResourceAttributes::timestamp
```

the date and time that is associated with this resource (e.g. the time embedded in an image from a digital camera with a clock)

7.96 qevercloud::SavedSearch Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::SavedSearch:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const SavedSearch &other) const
- bool operator!= (const SavedSearch &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< Guid > guid
- Optional< QString > name
- Optional< QString > query
- Optional< QueryFormat > format
- Optional< qint32 > updateSequenceNum
- Optional< SavedSearchScope > scope

7.96.1 Detailed Description

A named search associated with the account that can be quickly re-used.

7.96.2 Member Function Documentation

7.96.2.1 operator"!=()

```
bool qevercloud::SavedSearch::operator!= (
            const SavedSearch & other ) const  [inline]
```

7.96.2.2 operator==()

```
bool qevercloud::SavedSearch::operator== (
            const SavedSearch & other ) const  [inline]
```

7.96.2.3 print()

```
virtual void qevercloud::SavedSearch::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.96.3 Member Data Documentation

7.96.3.1 format

Optional< QueryFormat > qevercloud::SavedSearch::format

The format of the query string, to determine how to parse and process it.

7.96.3.2 guid

Optional< Guid > qevercloud::SavedSearch::guid

The unique identifier of this search. Will be set by the service, so may be omitted by the client when creating.
Length: EDAM_GUID_LEN_MIN - EDAM_GUID_LEN_MAX
Regex: EDAM_GUID_REGEX

7.96.3.3 localData

EverCloudLocalData qevercloud::SavedSearch::localData

See the declaration of EverCloudLocalData for details

7.96.3.4 name

Optional< QString > qevercloud::SavedSearch::name

The name of the saved search to display in the GUI. The account may only contain one search with a given name (case-insensitive compare). Can't begin or end with a space.
Length: EDAM_SAVED_SEARCH_NAME_LEN_MIN - EDAM_SAVED_SEARCH_NAME_LEN_MAX
Regex: EDAM_SAVED_SEARCH_NAME_REGEX

7.96.3.5 query

Optional< QString > qevercloud::SavedSearch::query

A string expressing the search to be performed.
Length: EDAM_SAVED_SEARCH_QUERY_LEN_MIN - EDAM_SAVED_SEARCH_QUERY_LEN_MAX

7.96.3.6 scope

Optional< SavedSearchScope > qevercloud::SavedSearch::scope

Specifies the set of notes that should be included in the search, if possible.

Clients are expected to search as much of the desired scope as possible, with the understanding that a given client may not be able to cover the full specified scope. For example, when executing a search that includes notes in both the owner's account and business notebooks, a mobile client may choose to only search within the user's account because it is not capable of searching both scopes simultaneously. When a search across multiple scopes is not possible, a client may choose which scope to search based on the current application context. If a client cannot search any of the desired scopes, it should refuse to execute the search.

7.96.3.7 updateSequenceNum

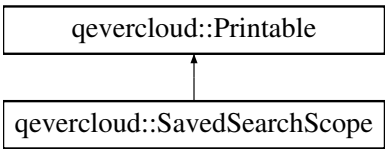
```
Optional< qint32 > qevercloud::SavedSearch::updateSequenceNum
```

A number identifying the last transaction to modify the state of this object. The USN values are sequential within an account, and can be used to compare the order of modifications within the service.

7.97 qevercloud::SavedSearchScope Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::SavedSearchScope:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const SavedSearchScope &other) const
- bool operator!= (const SavedSearchScope &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< bool > includeAccount
- Optional< bool > includePersonalLinkedNotebooks
- Optional< bool > includeBusinessLinkedNotebooks

7.97.1 Detailed Description

A structure defining the scope of a SavedSearch.

7.97.2 Member Function Documentation

7.97.2.1 operator"!=()

```
bool qevercloud::SavedSearchScope::operator!= (
            const SavedSearchScope & other ) const  [inline]
```

7.97.2.2 operator==()

```
bool qevercloud::SavedSearchScope::operator== (
            const SavedSearchScope & other ) const  [inline]
```

7.97.2.3 print()

```
virtual void qevercloud::SavedSearchScope::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.97.3 Member Data Documentation

7.97.3.1 includeAccount

```
Optional< bool > qevercloud::SavedSearchScope::includeAccount
```

The search should include notes from the account that contains the SavedSearch.

7.97.3.2 includeBusinessLinkedNotebooks

```
Optional< bool > qevercloud::SavedSearchScope::includeBusinessLinkedNotebooks
```

The search should include notes within those shared notebooks that the user has joined that are business notebooks in the business that the user is currently a member of.

7.97.3.3 includePersonalLinkedNotebooks

```
Optional< bool > qevercloud::SavedSearchScope::includePersonalLinkedNotebooks
```

The search should include notes within those shared notebooks that the user has joined that are NOT business notebooks.

7.97.3.4 localData

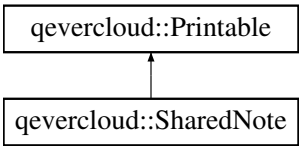
```
EverCloudLocalData qevercloud::SavedSearchScope::localData
```

See the declaration of EverCloudLocalData for details

7.98 qevercloud::SharedNote Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::SharedNote:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const SharedNote &other) const
- bool operator!= (const SharedNote &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< UserID > sharerUserID
- Optional< Identity > recipientIdentity
- Optional< SharedNotePrivilegeLevel > privilege
- Optional< Timestamp > serviceCreated
- Optional< Timestamp > serviceUpdated
- Optional< Timestamp > serviceAssigned

7.98.1 Detailed Description

Represents a relationship between a note and a single share invitation recipient. The recipient is identified via an Identity, and has a given privilege that specifies what actions they may take on the note.

7.98.2 Member Function Documentation

7.98.2.1 operator"!=()

```
bool qevercloud::SharedNote::operator!= (
            const SharedNote & other ) const  [inline]
```

7.98.2.2 operator==()

```
bool qevercloud::SharedNote::operator== (
            const SharedNote & other ) const  [inline]
```

7.98.2.3 print()

```
virtual void qevercloud::SharedNote::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.98.3 Member Data Documentation

7.98.3.1 localData

```
EverCloudLocalData qevercloud::SharedNote::localData
```

See the declaration of EverCloudLocalData for details

7.98.3.2 privilege

```
Optional< SharedNotePrivilegeLevel > qevercloud::SharedNote::privilege
```

The privilege level that the share grants to the recipient.

7.98.3.3 recipientIdentity

```
Optional< Identity > qevercloud::SharedNote::recipientIdentity
```

The identity of the recipient of the share. For a given note, there may be only one SharedNote per recipient identity. Only recipientIdentity.id is guaranteed to be set. Other fields on the Identity may or my not be set based on the requesting user's relationship with the recipient.

7.98.3.4 serviceAssigned

```
Optional< Timestamp > qevercloud::SharedNote::serviceAssigned
```

The time at which the share was assigned to a specific recipient user ID.

7.98.3.5 serviceCreated

```
Optional< Timestamp > qevercloud::SharedNote::serviceCreated
```

The time at which the share was created.

7.98.3.6 serviceUpdated

```
Optional< Timestamp > qevercloud::SharedNote::serviceUpdated
```

The time at which the share was last updated.

7.98.3.7 sharerUserID

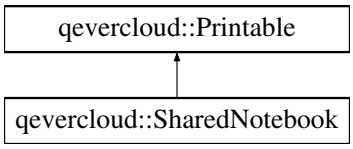
```
Optional< UserID > qevercloud::SharedNote::sharerUserID
```

The user ID of the user who shared the note with the recipient.

7.99 qevercloud::SharedNotebook Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::SharedNotebook:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const SharedNotebook &other) const
- bool operator!= (const SharedNotebook &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< qint64 > id
- Optional< UserID > userId
- Optional< Guid > notebookGuid
- Optional< QString > email
- Optional< IdentityID > recipientIdentityId
- Optional< bool > notebookModifiable
- Optional< Timestamp > serviceCreated
- Optional< Timestamp > serviceUpdated
- Optional< QString > globalId
- Optional< QString > username
- Optional< SharedNotebookPrivilegeLevel > privilege
- Optional< SharedNotebookRecipientSettings > recipientSettings
- Optional< UserID > sharerUserId
- Optional< QString > recipientUsername
- Optional< UserID > recipientUserId
- Optional< Timestamp > serviceAssigned

7.99.1 Detailed Description

Shared notebooks represent a relationship between a notebook and a single share invitation recipient.

7.99.2 Member Function Documentation

7.99.2.1 operator"!=()

```
bool qevercloud::SharedNotebook::operator!= (
            const SharedNotebook & other ) const  [inline]
```

7.99.2.2 operator==()

```
bool qevercloud::SharedNotebook::operator== (
            const SharedNotebook & other ) const  [inline]
```

7.99.2.3 print()

```
virtual void qevercloud::SharedNotebook::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.99.3 Member Data Documentation

7.99.3.1 email

```
Optional< QString > qevercloud::SharedNotebook::email
```

A string containing a display name for the recipient of the share. This may be an email address, a phone number, a full name, or some other descriptive string This field is read-only to clients. It will be filled in by the service when returning shared notebooks.

7.99.3.2 globalId

```
Optional< QString > qevercloud::SharedNotebook::globalId
```

An immutable, opaque string that acts as a globally unique identifier for this shared notebook record. You can use this field to match linked notebook and shared notebook records as well as to create new LinkedNotebook records. This field replaces the deprecated shareKey field.

7.99.3.3 id

```
Optional< qint64 > qevercloud::SharedNotebook::id
```

The primary identifier of the share, which is not globally unique.

7.99.3.4 localData

```
EverCloudLocalData qevercloud::SharedNotebook::localData
```

See the declaration of EverCloudLocalData for details

7.99.3.5 notebookGuid

Optional< Guid > qevercloud::SharedNotebook::notebookGuid

The GUID of the notebook that has been shared.

7.99.3.6 notebookModifiable

Optional< bool > qevercloud::SharedNotebook::notebookModifiable

DEPRECATED

7.99.3.7 privilege

Optional< SharedNotebookPrivilegeLevel > qevercloud::SharedNotebook::privilege

The privilege level granted to the notebook, activity stream, and invitations. See the corresponding enumeration for details.

7.99.3.8 recipientIdentityId

Optional< IdentityID > qevercloud::SharedNotebook::recipientIdentityId

The IdentityID of the share recipient. If present, only the user who has claimed that identity may access this share.

7.99.3.9 recipientSettings

Optional< SharedNotebookRecipientSettings > qevercloud::SharedNotebook::recipientSettings

Settings intended for use only by the recipient of this shared notebook. You should skip setting this value unless you want to change the value contained inside the structure, and only if you are the recipient.

7.99.3.10 recipientUserId

Optional< UserID > qevercloud::SharedNotebook::recipientUserId

The id of the user who can access this share. This is the id for the user with the username in recipientUsername. This value is read-only and set by the service. Value set by clients will be ignored. This field may be unset for unjoined notebooks and is always set if serviceAssigned is set. Clients should prefer this field over recipient↩ Username unless they need to use usernames directly.

7.99.3.11 recipientUsername

Optional< QString > qevercloud::SharedNotebook::recipientUsername

The username of the user who can access this share. This is the username for the user with the id in recipientUserId. This value can be set by clients when calling shareNotebook(...), and that will result in the created SharedNotebook being assigned to a user. This value is always set if serviceAssigned is set.

7.99.3.12 serviceAssigned

`Optional< Timestamp > qevercloud::SharedNotebook::serviceAssigned`

The date this SharedNotebook was assigned (i.e. has been associated with an Evernote user whose user ID is set in recipientUserId). Unset if the SharedNotebook is not assigned. This field is a read-only value that is set by the service.

7.99.3.13 serviceCreated

`Optional< Timestamp > qevercloud::SharedNotebook::serviceCreated`

The date that the owner first created the share with the specific email address.

7.99.3.14 serviceUpdated

`Optional< Timestamp > qevercloud::SharedNotebook::serviceUpdated`

The date the shared notebook was last updated on the service. This will be updated when authenticateToShared↩ Notebook is called the first time with a shared notebook (i.e. when the username is bound to that shared notebook), and also when the SharedNotebook privilege is updated as part of a shareNotebook(...) call, as well as on any calls to updateSharedNotebook(...).

7.99.3.15 sharerUserId

`Optional< UserID > qevercloud::SharedNotebook::sharerUserId`

The user id of the user who shared a notebook via this shared notebook instance. This may not be the same as userId, since a user with full access to a notebook may have created a new share for that notebook. For Business, this represents the user who shared the business notebook. This field is currently unset for a SharedNotebook created by joining a notebook that has been published to the business.

7.99.3.16 userId

`Optional< UserID > qevercloud::SharedNotebook::userId`

The user id of the owner of the notebook.

7.99.3.17 username

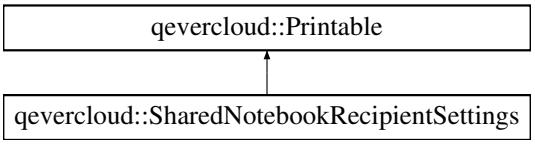
`Optional< QString > qevercloud::SharedNotebook::username`

DEPRECATED. The username of the user who can access this share. This value is read-only to clients. It will be filled in by the service when returning shared notebooks.

7.100 qevercloud::SharedNotebookRecipientSettings Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::SharedNotebookRecipientSettings:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const SharedNotebookRecipientSettings &other) const
- bool operator!= (const SharedNotebookRecipientSettings &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< bool > reminderNotifyEmail
- Optional< bool > reminderNotifyInApp

7.100.1 Detailed Description

Settings meant for the recipient of a shared notebook, such as for indicating which types of notifications the recipient wishes for reminders, etc.

The reminderNotifyEmail and reminderNotifyInApp fields have a 3-state read value but a 2-state write value. On read, it is possible to observe "unset", true, or false. The initial state is "unset". When you choose to set a value, you may set it to either true or false, but you cannot unset the value. Once one of these members has a true/false value, it will always have a true/false value.

7.100.2 Member Function Documentation

7.100.2.1 operator"!=()

```
bool qevercloud::SharedNotebookRecipientSettings::operator!= (
            const SharedNotebookRecipientSettings & other ) const  [inline]
```

7.100.2.2 operator==()

```
bool qevercloud::SharedNotebookRecipientSettings::operator== (
            const SharedNotebookRecipientSettings & other ) const  [inline]
```

7.100.2.3 print()

```
virtual void qevercloud::SharedNotebookRecipientSettings::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.100.3 Member Data Documentation

7.100.3.1 localData

```
EverCloudLocalData qevercloud::SharedNotebookRecipientSettings::localData
```

See the declaration of EverCloudLocalData for details

7.100.3.2 reminderNotifyEmail

```
Optional< bool > qevercloud::SharedNotebookRecipientSettings::reminderNotifyEmail
```

Indicates that the user wishes to receive daily e-mail notifications for reminders associated with the notebook. This may be true only for business notebooks that belong to the business of which the user is a member. You may only set this value on a notebook in your business.

7.100.3.3 reminderNotifyInApp

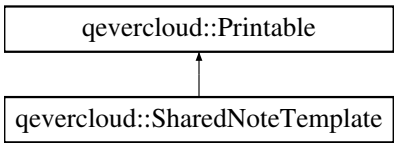
```
Optional< bool > qevercloud::SharedNotebookRecipientSettings::reminderNotifyInApp
```

Indicates that the user wishes to receive notifications for reminders by applications that support providing such notifications. The exact nature of the notification is defined by the individual applications.

7.101 qevercloud::SharedNoteTemplate Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::SharedNoteTemplate:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const SharedNoteTemplate &other) const
- bool operator!= (const SharedNoteTemplate &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< Guid > noteGuid
- Optional< MessageThreadID > recipientThreadId
- Optional< QList< Contact > > recipientContacts
- Optional< SharedNotePrivilegeLevel > privilege

Properties

- OptionalQList< Contact > recipientContacts

7.101.1 Detailed Description

A structure used to share a note with one or more recipients at a given privilege.

7.101.2 Member Function Documentation

7.101.2.1 operator"!=()

```
bool qevercloud::SharedNoteTemplate::operator!= (
            const SharedNoteTemplate & other ) const  [inline]
```

7.101.2.2 operator==()

```
bool qevercloud::SharedNoteTemplate::operator== (
            const SharedNoteTemplate & other ) const  [inline]
```

7.101.2.3 print()

```
virtual void qevercloud::SharedNoteTemplate::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.101.3 Member Data Documentation

7.101.3.1 localData

```
EverCloudLocalData qevercloud::SharedNoteTemplate::localData
```

See the declaration of EverCloudLocalData for details

7.101.3.2 noteGuid

```
Optional< Guid > qevercloud::SharedNoteTemplate::noteGuid
```

The GUID of the note.

7.101.3.3 privilege

```
Optional< SharedNotePrivilegeLevel > qevercloud::SharedNoteTemplate::privilege
```

The privilege level to be granted.

7.101.3.4 recipientContacts

```
Optional<QList<Contact> > qevercloud::SharedNoteTemplate::recipientContacts
```

The recipients of the note share specified as a list of contacts. This should only be set if the sharing takes place before the thread is created. Use recipientThreadId instead when sharing with an existing thread. Either this field or recipientThreadId must be set.

7.101.3.5 recipientThreadId

```
Optional< MessageThreadID > qevercloud::SharedNoteTemplate::recipientThreadId
```

The recipients of the note share specified as a messaging thread ID. If you have an existing messaging thread to share the note with, specify its ID here instead of recipientContacts in order to properly support defunct identities. The sharer must be a participant of the thread. Either this field or recipientContacts must be set.

7.101.4 Property Documentation

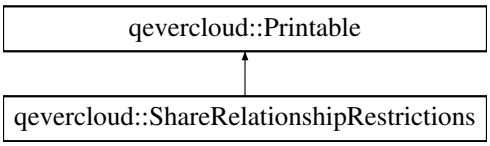
7.101.4.1 recipientContacts

```
OptionalQList<Contact> qevercloud::SharedNoteTemplate::recipientContacts
```

7.102 qevercloud::ShareRelationshipRestrictions Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::ShareRelationshipRestrictions:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const ShareRelationshipRestrictions &other) const
- bool operator!= (const ShareRelationshipRestrictions &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< bool > noSetReadOnly
- Optional< bool > noSetReadPlusActivity
- Optional< bool > noSetModify
- Optional< bool > noSetFullAccess

7.102.1 Detailed Description

NO DOC COMMENT ID FOUND

7.102.2 Member Function Documentation

7.102.2.1 operator"!=()

```
bool qevercloud::ShareRelationshipRestrictions::operator!= (
            const ShareRelationshipRestrictions & other ) const  [inline]
```

7.102.2.2 operator==()

```
bool qevercloud::ShareRelationshipRestrictions::operator== (
            const ShareRelationshipRestrictions & other ) const  [inline]
```

7.102.2.3 print()

```
virtual void qevercloud::ShareRelationshipRestrictions::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.102.3 Member Data Documentation

7.102.3.1 localData

```
EverCloudLocalData qevercloud::ShareRelationshipRestrictions::localData
```

See the declaration of EverCloudLocalData for details

7.102.3.2 noSetFullAccess

```
Optional< bool > qevercloud::ShareRelationshipRestrictions::noSetFullAccess
```

NOT DOCUMENTED

7.102.3.3 noSetModify

```
Optional< bool > qevercloud::ShareRelationshipRestrictions::noSetModify
```

NOT DOCUMENTED

7.102.3.4 noSetReadOnly

```
Optional< bool > qevercloud::ShareRelationshipRestrictions::noSetReadOnly
```

NOT DOCUMENTED

7.102.3.5 noSetReadPlusActivity

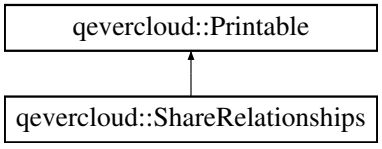
```
Optional< bool > qevercloud::ShareRelationshipRestrictions::noSetReadPlusActivity
```

NOT DOCUMENTED

7.103 qevercloud::ShareRelationships Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::ShareRelationships:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const ShareRelationships &other) const
- bool operator!= (const ShareRelationships &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QList< InvitationShareRelationship > > invitations
- Optional< QList< MemberShareRelationship > > memberships
- Optional< ShareRelationshipRestrictions > invitationRestrictions

Properties

- OptionalQList< InvitationShareRelationship > invitations
- OptionalQList< MemberShareRelationship > memberships

7.103.1 Detailed Description

Captures a collection of share relationships for a notebook, for example, as returned by the getNotebookShares method. The share relationships fall into two broad categories: members, and invitations that can be used to become members.

7.103.2 Member Function Documentation

7.103.2.1 operator"!=()

```
bool qevercloud::ShareRelationships::operator!= (
            const ShareRelationships & other ) const  [inline]
```

7.103.2.2 operator==()

```
bool qevercloud::ShareRelationships::operator== (
            const ShareRelationships & other ) const  [inline]
```

7.103.2.3 print()

```
virtual void qevercloud::ShareRelationships::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.103.3 Member Data Documentation

7.103.3.1 invitationRestrictions

```
Optional< ShareRelationshipRestrictions > qevercloud::ShareRelationships::invitationRestrictions
```

The restrictions on what privileges may be granted to invitees to this notebook. These restrictions may be specific to the calling user or to the notebook itself. They represent the union of all possible invite cases, so it is possible that once the recipient of the invitation has been identified by the service, such as by a business auto-join, the actual assigned privilege may change.

7.103.3.2 invitations

```
Optional<QList<InvitationShareRelationship> > qevercloud::ShareRelationships::invitations
```

A list of open invitations that can be redeemed into memberships to the notebook.

7.103.3.3 localData

```
EverCloudLocalData qevercloud::ShareRelationships::localData
```

See the declaration of EverCloudLocalData for details

7.103.3.4 memberships

```
Optional<QList<MemberShareRelationship> > qevercloud::ShareRelationships::memberships
```

A list of memberships of the notebook. A member is identified by their Evernote UserID and has rights to access the notebook.

7.103.4 Property Documentation

7.103.4.1 invitations

```
OptionalQList<InvitationShareRelationship> qevercloud::ShareRelationships::invitations
```

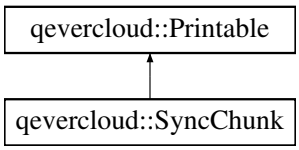
7.103.4.2 memberships

```
OptionalQList<MemberShareRelationship> qevercloud::ShareRelationships::memberships
```

7.104 qevercloud::SyncChunk Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::SyncChunk:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const SyncChunk &other) const
- bool operator!= (const SyncChunk &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Timestamp currentTime = 0
- Optional< qint32 > chunkHighUSN
- qint32 updateCount = 0
- Optional< QList< Note > > notes
- Optional< QList< Notebook > > notebooks
- Optional< QList< Tag > > tags
- Optional< QList< SavedSearch > > searches
- Optional< QList< Resource > > resources
- Optional< QList< Guid > > expungedNotes
- Optional< QList< Guid > > expungedNotebooks
- Optional< QList< Guid > > expungedTags
- Optional< QList< Guid > > expungedSearches
- Optional< QList< LinkedNotebook > > linkedNotebooks
- Optional< QList< Guid > > expungedLinkedNotebooks

Properties

- OptionalQList< Note > notes
- OptionalQList< Notebook > notebooks
- OptionalQList< Tag > tags
- OptionalQList< SavedSearch > searches
- OptionalQList< Resource > resources
- OptionalQList< Guid > expungedNotes
- OptionalQList< Guid > expungedNotebooks
- OptionalQList< Guid > expungedTags
- OptionalQList< Guid > expungedSearches
- OptionalQList< LinkedNotebook > linkedNotebooks
- OptionalQList< Guid > expungedLinkedNotebooks

7.104.1 Detailed Description

This structure is given out by the NoteStore when a client asks to receive the current state of an account. The client asks for the server's state one chunk at a time in order to allow clients to retrieve the state of a large account without needing to transfer the entire account in a single message.

The server always gives SyncChunks using an ascending series of Update Sequence Numbers (USNs).

7.104.2 Member Function Documentation

7.104.2.1 operator"!=()

```
bool qevercloud::SyncChunk::operator!= (
            const SyncChunk & other ) const  [inline]
```

7.104.2.2 operator==()

```
bool qevercloud::SyncChunk::operator== (
            const SyncChunk & other ) const  [inline]
```

7.104.2.3 print()

```
virtual void qevercloud::SyncChunk::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.104.3 Member Data Documentation

7.104.3.1 chunkHighUSN

```
Optional< qint32 > qevercloud::SyncChunk::chunkHighUSN
```

The highest USN for any of the data objects represented in this sync chunk. If there are no objects in the chunk, this will not be set.

7.104.3.2 currentTime

```
Timestamp qevercloud::SyncChunk::currentTime = 0
```

The server's current date and time.

7.104.3.3 expungedLinkedNotebooks

```
Optional<QList<Guid> > qevercloud::SyncChunk::expungedLinkedNotebooks
```

If present, the GUIDs of all of the LinkedNotebooks that were permanently expunged in this chunk.

7.104.3.4 expungedNotebooks

```
Optional<QList<Guid> > qevercloud::SyncChunk::expungedNotebooks
```

If present, the GUIDs of all of the notebooks that were permanently expunged in this chunk. When a notebook is expunged, this implies that all of its child notes (and their resources) were also expunged.

7.104.3.5 expungedNotes

```
Optional<QList<Guid> > qevercloud::SyncChunk::expungedNotes
```

If present, the GUIDs of all of the notes that were permanently expunged in this chunk.

7.104.3.6 expungedSearches

```
Optional<QList<Guid> > qevercloud::SyncChunk::expungedSearches
```

If present, the GUIDs of all of the saved searches that were permanently expunged in this chunk.

7.104.3.7 expungedTags

```
Optional<QList<Guid> > qevercloud::SyncChunk::expungedTags
```

If present, the GUIDs of all of the tags that were permanently expunged in this chunk.

7.104.3.8 linkedNotebooks

```
Optional<QList<LinkedNotebook> > qevercloud::SyncChunk::linkedNotebooks
```

If present, this is a list of non-expunged LinkedNotebooks that have a USN in this chunk.

7.104.3.9 localData

```
EverCloudLocalData qevercloud::SyncChunk::localData
```

See the declaration of EverCloudLocalData for details

7.104.3.10 notebooks

```
Optional<QList<Notebook> > qevercloud::SyncChunk::notebooks
```

If present, this is a list of non-expunged notebooks that have a USN in this chunk.

7.104.3.11 notes

```
Optional<QList<Note> > qevercloud::SyncChunk::notes
```

If present, this is a list of non-expunged notes that have a USN in this chunk. This will include notes that are "deleted" but not expunged (i.e. in the trash). The notes will include their list of tags and resources, but the note content, resource content, resource recognition data and resource alternate data will not be supplied.

7.104.3.12 resources

```
Optional<QList<Resource> > qevercloud::SyncChunk::resources
```

If present, this is a list of the non-expunged resources that have a USN in this chunk. This will include the metadata for each resource, but not its binary contents or recognition data, which must be retrieved separately.

7.104.3.13 searches

```
Optional<QList<SavedSearch> > qevercloud::SyncChunk::searches
```

If present, this is a list of non-expunged searches that have a USN in this chunk.

7.104.3.14 tags

```
Optional<QList<Tag> > qevercloud::SyncChunk::tags
```

If present, this is a list of the non-expunged tags that have a USN in this chunk.

7.104.3.15 updateCount

```
qint32 qevercloud::SyncChunk::updateCount = 0
```

The total number of updates that have been performed in the service for this account. This is equal to the highest USN within the account at the point that this SyncChunk was generated. If updateCount and chunkHighUSN are identical, that means that this is the last chunk in the account ... there is no more recent information.

7.104.4 Property Documentation

7.104.4.1 expungedLinkedNotebooks

```
OptionalQList<Guid> qevercloud::SyncChunk::expungedLinkedNotebooks
```

7.104.4.2 expungedNotebooks

```
OptionalQList<Guid> qevercloud::SyncChunk::expungedNotebooks
```

7.104.4.3 expungedNotes

OptionalQList<Guid> qevercloud::SyncChunk::expungedNotes

7.104.4.4 expungedSearches

OptionalQList<Guid> qevercloud::SyncChunk::expungedSearches

7.104.4.5 expungedTags

OptionalQList<Guid> qevercloud::SyncChunk::expungedTags

7.104.4.6 linkedNotebooks

OptionalQList<LinkedNotebook> qevercloud::SyncChunk::linkedNotebooks

7.104.4.7 notebooks

OptionalQList<Notebook> qevercloud::SyncChunk::notebooks

7.104.4.8 notes

OptionalQList<Note> qevercloud::SyncChunk::notes

7.104.4.9 resources

OptionalQList<Resource> qevercloud::SyncChunk::resources

7.104.4.10 searches

OptionalQList<SavedSearch> qevercloud::SyncChunk::searches

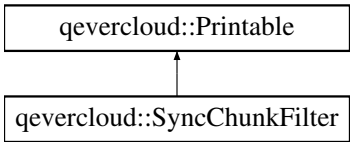
7.104.4.11 tags

OptionalQList<Tag> qevercloud::SyncChunk::tags

7.105 qevercloud::SyncChunkFilter Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::SyncChunkFilter:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const SyncChunkFilter &other) const
- bool operator!= (const SyncChunkFilter &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< bool > includeNotes
- Optional< bool > includeNoteResources
- Optional< bool > includeNoteAttributes
- Optional< bool > includeNotebooks
- Optional< bool > includeTags
- Optional< bool > includeSearches
- Optional< bool > includeResources
- Optional< bool > includeLinkedNotebooks
- Optional< bool > includeExpunged
- Optional< bool > includeNoteApplicationDataFullMap
- Optional< bool > includeResourceApplicationDataFullMap
- Optional< bool > includeNoteResourceApplicationDataFullMap
- Optional< bool > includeSharedNotes
- Optional< bool > omitSharedNotebooks
- Optional< QString > requireNoteContentClass
- Optional< QSet< QString > > notebookGuids

Properties

- OptionalQSet< QString > notebookGuids

7.105.1 Detailed Description

This structure is used with the 'getFilteredSyncChunk' call to provide fine-grained control over the data that's returned when a client needs to synchronize with the service. Each flag in this structure specifies whether to include one class of data in the results of that call.

7.105.2 Member Function Documentation

7.105.2.1 operator"!=()

```
bool qevercloud::SyncChunkFilter::operator!= (
            const SyncChunkFilter & other ) const  [inline]
```

7.105.2.2 operator==()

```
bool qevercloud::SyncChunkFilter::operator== (
            const SyncChunkFilter & other ) const  [inline]
```

7.105.2.3 print()

```
virtual void qevercloud::SyncChunkFilter::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.105.3 Member Data Documentation

7.105.3.1 includeExpunged

```
Optional< bool > qevercloud::SyncChunkFilter::includeExpunged
```

If true, then the server will include the 'expunged' data for any type of included data. For example, if 'includeTags' and 'includeExpunged' are both true, then the SyncChunks.expungedTags field will be set with the GUIDs of tags that have been expunged from the server.

7.105.3.2 includeLinkedNotebooks

```
Optional< bool > qevercloud::SyncChunkFilter::includeLinkedNotebooks
```

If true, then the server will include the SyncChunks.linkedNotebooks field.

7.105.3.3 includeNoteApplicationDataFullMap

```
Optional< bool > qevercloud::SyncChunkFilter::includeNoteApplicationDataFullMap
```

If true, then the values for the applicationData map will be filled in, assuming notes and note attributes are being returned. Otherwise, only the keysOnly field will be filled in.

7.105.3.4 includeNoteAttributes

```
Optional< bool > qevercloud::SyncChunkFilter::includeNoteAttributes
```

If true, then the server will include the 'attributes' field on all of the Notes that are in SyncChunks.notes. If 'include↩ Notes' is false, then this will have no effect.

7.105.3.5 includeNotebooks

```
Optional< bool > qevercloud::SyncChunkFilter::includeNotebooks
```

If true, then the server will include the SyncChunks.notebooks field

7.105.3.6 includeNoteResourceApplicationDataFullMap

```
Optional< bool > qevercloud::SyncChunkFilter::includeNoteResourceApplicationDataFullMap
```

If true, then the fullMap values for the applicationData map will be filled in for resources found inside of notes, assuming resources are being returned in notes (includeNoteResources is true). Otherwise, only the keysOnly field will be filled in.

7.105.3.7 includeNoteResources

```
Optional< bool > qevercloud::SyncChunkFilter::includeNoteResources
```

If true, then the server will include the 'resources' field on all of the Notes that are in SyncChunk.notes. If 'include↩ Notes' is false, then this will have no effect.

7.105.3.8 includeNotes

```
Optional< bool > qevercloud::SyncChunkFilter::includeNotes
```

If true, then the server will include the SyncChunks.notes field

7.105.3.9 includeResourceApplicationDataFullMap

```
Optional< bool > qevercloud::SyncChunkFilter::includeResourceApplicationDataFullMap
```

If true, then the fullMap values for the applicationData map will be filled in, assuming resources and resource attributes are being returned (includeResources is true). Otherwise, only the keysOnly field will be filled in.

7.105.3.10 includeResources

```
Optional< bool > qevercloud::SyncChunkFilter::includeResources
```

If true, then the server will include the SyncChunks.resources field. Since the Resources are also provided with their Note (in the Notes.resources list), this is primarily useful for clients that want to watch for changes to individual Resources due to recognition data being added.

7.105.3.11 includeSearches

```
Optional< bool > qevercloud::SyncChunkFilter::includeSearches
```

If true, then the server will include the SyncChunks.searches field

7.105.3.12 includeSharedNotes

```
Optional< bool > qevercloud::SyncChunkFilter::includeSharedNotes
```

If true, then the service will include the sharedNotes field on all notes that are in SyncChunk.notes. If 'includeNotes' is false, then this will have no effect.

7.105.3.13 includeTags

```
Optional< bool > qevercloud::SyncChunkFilter::includeTags
```

If true, then the server will include the SyncChunks.tags field

7.105.3.14 localData

```
EverCloudLocalData qevercloud::SyncChunkFilter::localData
```

See the declaration of EverCloudLocalData for details

7.105.3.15 notebookGuids

```
Optional<QSet<QString> > qevercloud::SyncChunkFilter::notebookGuids
```

If set, then restrict the returned notebooks, notes, and resources to those associated with one of the notebooks whose GUID is provided in this list. If not set, then no filtering on notebook GUID will be performed. If you set this field, you may not also set includeExpunged else an EDAMUserException with an error code of DATA_CONFLICT will be thrown. You only need to set this field if you want to restrict the returned entities more than what your authentication token allows you to access. For example, there is no need to set this field for single notebook tokens such as for shared notebooks. You can use this field to synchronize a newly discovered business notebook while incrementally synchronizing a business account, in which case you will only need to consider setting includeNotes, includeNotebooks, includeNoteAttributes, includeNoteResources, and maybe some of the "FullMap" fields.

7.105.3.16 omitSharedNotebooks

```
Optional< bool > qevercloud::SyncChunkFilter::omitSharedNotebooks
```

NOT DOCUMENTED

7.105.3.17 requireNoteContentClass

```
Optional< QString > qevercloud::SyncChunkFilter::requireNoteContentClass
```

If set, then only send notes whose content class matches this value. The value can be a literal match or, if the last character is an asterisk, a prefix match.

7.105.4 Property Documentation

7.105.4.1 notebookGuids

```
OptionalQSet<QString> qevercloud::SyncChunkFilter::notebookGuids
```

7.106 qevercloud::IDurableService::SyncRequest Struct Reference

```
#include <DurableService.h>
```

Public Member Functions

- SyncRequest (const char *name, QString description, SyncServiceCall &&call)

Public Attributes

- const char * m_name
- QString m_description
- SyncServiceCall m_call

7.106.1 Constructor & Destructor Documentation

7.106.1.1 SyncRequest()

```
qevercloud::IDurableService::SyncRequest::SyncRequest (
            const char * name,
            QString description,
            SyncServiceCall && call )  [inline]
```

7.106.2 Member Data Documentation

7.106.2.1 m_call

```
SyncServiceCall qevercloud::IDurableService::SyncRequest::m_call
```

7.106.2.2 m_description

```
QString qevercloud::IDurableService::SyncRequest::m_description
```

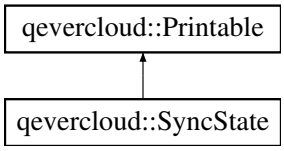
7.106.2.3 m_name

```
const char* qevercloud::IDurableService::SyncRequest::m_name
```

7.107 qevercloud::SyncState Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::SyncState:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const SyncState &other) const
- bool operator!= (const SyncState &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Timestamp currentTime = 0
- Timestamp fullSyncBefore = 0
- qint32 updateCount = 0
- Optional< qint64 > uploaded
- Optional< Timestamp > userLastUpdated
- Optional< MessageEventID > userMaxMessageEventId

7.107.1 Detailed Description

This structure encapsulates the information about the state of the user's account for the purpose of "state based" synchronization.

7.107.2 Member Function Documentation

7.107.2.1 operator"!=()

```
bool qevercloud::SyncState::operator!= (
            const SyncState & other ) const  [inline]
```

7.107.2.2 operator==()

```
bool qevercloud::SyncState::operator== (
            const SyncState & other ) const  [inline]
```

7.107.2.3 print()

```
virtual void qevercloud::SyncState::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.107.3 Member Data Documentation

7.107.3.1 currentTime

```
Timestamp qevercloud::SyncState::currentTime = 0
```

The server's current date and time.

7.107.3.2 fullSyncBefore

```
Timestamp qevercloud::SyncState::fullSyncBefore = 0
```

The cutoff date and time for client caches to be updated via incremental synchronization. Any clients that were last synched with the server before this date/time must do a full resync of all objects. This cutoff point will change over time as archival data is deleted or special circumstances on the service require resynchronization.

7.107.3.3 localData

```
EverCloudLocalData qevercloud::SyncState::localData
```

See the declaration of EverCloudLocalData for details

7.107.3.4 updateCount

```
qint32 qevercloud::SyncState::updateCount = 0
```

Indicates the total number of transactions that have been committed within the account. This reflects (for example) the number of discrete additions or modifications that have been made to the data in this account (tags, notes, resources, etc.). This number is the "high water mark" for Update Sequence Numbers (USN) within the account.

7.107.3.5 uploaded

```
Optional< qint64 > qevercloud::SyncState::uploaded
```

The total number of bytes that have been uploaded to this account in the current monthly period. This can be compared against Accounting.uploadLimit (from the UserStore) to determine how close the user is to their monthly upload limit. This value may not be present if the SyncState has been retrieved by a caller that only has read access to the account.

7.107.3.6 userLastUpdated

```
Optional< Timestamp > qevercloud::SyncState::userLastUpdated
```

The last time when a user's account level information was changed. This value is the latest time when a modification was made to any of the following: accounting information (billing, quota, premium status, etc.), user attributes and business user information (business name, business user attributes, etc.) if the user is in a business. Clients who need to maintain account information about a User should watch this field for updates rather than polling User↩ Store.getUser for updates. Here is the basic flow that clients should follow:

1. Call NoteStore.getSyncState to retrieve the SyncState object
2. Compare SyncState.userLastUpdated to previously stored value: if (SyncState.userLastUpdated > previousValue) call UserStore.getUser to get the latest User object; else do nothing;
3. Update previousValue = SyncState.userLastUpdated

7.107.3.7 userMaxMessageEventId

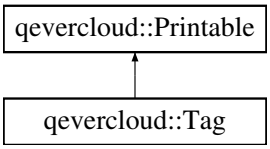
```
Optional< MessageEventID > qevercloud::SyncState::userMaxMessageEventId
```

The greatest MessageEventID for this user's account. Clients that do a full sync should store this value locally and compare their local copy to the value returned by getSyncState to determine if they need to sync with MessageStore. This value will be omitted if the user has never sent or received a message.

7.108 qevercloud::Tag Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::Tag:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const Tag &other) const
- bool operator!= (const Tag &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< Guid > guid
- Optional< QString > name
- Optional< Guid > parentGuid
- Optional< qint32 > updateSequenceNum

7.108.1 Detailed Description

A tag within a user's account is a unique name which may be organized a simple hierarchy.

7.108.2 Member Function Documentation

7.108.2.1 operator"!=()

```
bool qevercloud::Tag::operator!= (
            const Tag & other ) const  [inline]
```

7.108.2.2 operator==()

```
bool qevercloud::Tag::operator== (
            const Tag & other ) const  [inline]
```

7.108.2.3 print()

```
virtual void qevercloud::Tag::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.108.3 Member Data Documentation

7.108.3.1 guid

```
Optional< Guid > qevercloud::Tag::guid
```

The unique identifier of this tag. Will be set by the service, so may be omitted by the client when creating the Tag.
Length: EDAM_GUID_LEN_MIN - EDAM_GUID_LEN_MAX
Regex: EDAM_GUID_REGEX

7.108.3.2 localData

```
EverCloudLocalData qevercloud::Tag::localData
```

See the declaration of EverCloudLocalData for details

7.108.3.3 name

`Optional< QString > qevercloud::Tag::name`

A sequence of characters representing the tag's identifier. Case is preserved, but is ignored for comparisons. This means that an account may only have one tag with a given name, via case-insensitive comparison, so an account may not have both "food" and "Food" tags. May not contain a comma (','), and may not begin or end with a space.
Length: EDAM_TAG_NAME_LEN_MIN - EDAM_TAG_NAME_LEN_MAX
Regex: EDAM_TAG_NAME_REGEX

7.108.3.4 parentGuid

`Optional< Guid > qevercloud::Tag::parentGuid`

If this is set, then this is the GUID of the tag that holds this tag within the tag organizational hierarchy. If this is not set, then the tag has no parent and it is a "top level" tag. Cycles are not allowed (e.g. a->parent->parent == a) and will be rejected by the service.
Length: EDAM_GUID_LEN_MIN - EDAM_GUID_LEN_MAX
Regex: EDAM_GUID_REGEX

7.108.3.5 updateSequenceNum

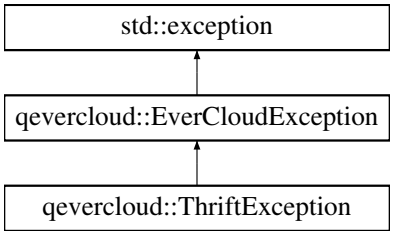
`Optional< qint32 > qevercloud::Tag::updateSequenceNum`

A number identifying the last transaction to modify the state of this object. The USN values are sequential within an account, and can be used to compare the order of modifications within the service.

7.109 qevercloud::ThriftException Class Reference

`#include <Exceptions.h>`

Inheritance diagram for qevercloud::ThriftException:



Public Types

- enum class Type {
  UNKNOWN = 0 , UNKNOWN_METHOD = 1 , INVALID_MESSAGE_TYPE = 2 , WRONG_METHOD_NAME = 3 ,
  BAD_SEQUENCE_ID = 4 , MISSING_RESULT = 5 , INTERNAL_ERROR = 6 , PROTOCOL_ERROR = 7 ,
  INVALID_DATA = 8 }

Public Member Functions

- ThriftException ()
- ThriftException (Type type)
- ThriftException (Type type, QString message)
- virtual ~ThriftException () noexcept override
- bool operator== (const ThriftException &other) const
- bool operator!= (const ThriftException &other) const
- Type type () const
- const char * what () const noexcept override
- virtual EverCloudExceptionDataPtr exceptionData () const override

Public Member Functions inherited from qevercloud::EverCloudException

- EverCloudException ()
- EverCloudException (QString error)
- EverCloudException (const std::string &error)
- EverCloudException (const char *error)
- virtual ~EverCloudException () noexcept override

Protected Attributes

- Type m_type

Protected Attributes inherited from qevercloud::EverCloudException

- QByteArray m_error

Friends

- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &strm, const Type type)

7.109.1 Detailed Description

Errors of the Thrift protocol level. It could be wrongly formatted parameters or return values for example.

7.109.2 Member Enumeration Documentation

7.109.2.1 Type

```
enum class qevercloud::ThriftException::Type  [strong]
```

Enumerator

UNKNOWN	
UNKNOWN_METHOD	
INVALID_MESSAGE_TYPE	
WRONG_METHOD_NAME	
BAD_SEQUENCE_ID	
MISSING_RESULT	
INTERNAL_ERROR	
PROTOCOL_ERROR	

7.109.3 Constructor & Destructor Documentation

7.109.3.1 ThriftException() [1/3]

```
qevercloud::ThriftException::ThriftException ( )
```

7.109.3.2 ThriftException() [2/3]

```
qevercloud::ThriftException::ThriftException (
            Type type )
```

7.109.3.3 ThriftException() [3/3]

```
qevercloud::ThriftException::ThriftException (
            Type type,
            QString message )
```

7.109.3.4 ~ThriftException()

```
virtual qevercloud::ThriftException::~ThriftException ( )  [override], [virtual], [noexcept]
```

7.109.4 Member Function Documentation

7.109.4.1 exceptionData()

```
virtual EverCloudExceptionDataPtr qevercloud::ThriftException::exceptionData ( ) const  [override],
[virtual]
```

Reimplemented from qevercloud::EverCloudException.

7.109.4.2 operator"!=()

```
bool qevercloud::ThriftException::operator!= (
            const ThriftException & other ) const
```

7.109.4.3 operator==()

```
bool qevercloud::ThriftException::operator== (
            const ThriftException & other ) const
```

7.109.4.4 type()

```
Type qevercloud::ThriftException::type ( ) const
```

7.109.4.5 what()

```
const char * qevercloud::ThriftException::what ( ) const  [override], [virtual], [noexcept]
```

Reimplemented from qevercloud::EverCloudException.

7.109.5 Friends And Related Symbol Documentation

7.109.5.1 operator<<

```
QEVERCLOUD_EXPORT QTextStream & operator<< (
            QTextStream & strm,
            const Type type )  [friend]
```

7.109.6 Member Data Documentation

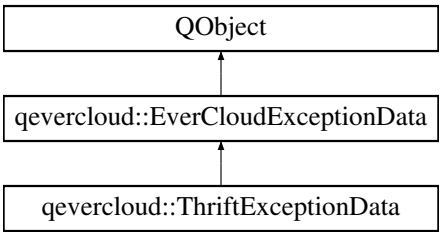
7.109.6.1 m_type

```
Type qevercloud::ThriftException::m_type  [protected]
```

7.110 qevercloud::ThriftExceptionData Class Reference

```
#include <Exceptions.h>
```

Inheritance diagram for qevercloud::ThriftExceptionData:



Public Member Functions

- ThriftExceptionData (QString error, ThriftException::Type type)
- virtual void throwException () const override

Public Member Functions inherited from qevercloud::EverCloudExceptionData

- EverCloudExceptionData (QString error)

Protected Attributes

- ThriftException::Type m_type

Additional Inherited Members

Public Attributes inherited from qevercloud::EverCloudExceptionData

- QString errorMessage

7.110.1 Detailed Description

Asynchronous API counterpart of ThriftException. See EverCloudExceptionData for more details.

7.110.2 Constructor & Destructor Documentation

7.110.2.1 ThriftExceptionData()

```
qevercloud::ThriftExceptionData::ThriftExceptionData (
            QString error,
            ThriftException::Type type )  [explicit]
```

7.110.3 Member Function Documentation

7.110.3.1 throwException()

```
virtual void qevercloud::ThriftExceptionData::throwException ( ) const  [override], [virtual]
```

If you want to throw an exception that corresponds to a received EverCloudExceptionData descendant than call this function. Do not use `throw` statement, it's not polymorphic.

Reimplemented from qevercloud::EverCloudExceptionData.

7.110.4 Member Data Documentation

7.110.4.1 m_type

```
ThriftException::Type qevercloud::ThriftExceptionData::m_type  [protected]
```

7.111 qevercloud::Thumbnail Class Reference

The class is for downloading thumbnails for notes and resources from Evernote servers.

```
#include <Thumbnail.h>
```

Public Types

- enum class ImageType { PNG , JPEG , GIF , BMP }

Public Member Functions

- Thumbnail ()

  Default constructor.
- Thumbnail (QString host, QString shardId, QString authenticationToken, int size=300, ImageType imageType=ImageType::PNG)

  Constructs Thumbnail.
- virtual ~Thumbnail ()
- Thumbnail & setHost (QString host)
- Thumbnail & setShardId (QString shardId)
- Thumbnail & setAuthenticationToken (QString authenticationToken)
- Thumbnail & setSize (int size)
- Thumbnail & setImageType (ImageType imageType)
- QByteArray download (Guid guid, const bool isPublic=false, const bool isResourceGuid=false, const qint64 timeoutMsec=30000)

  Downloads the thumbnail for a resource or a note.
- AsyncResult * downloadAsync (Guid guid, const bool isPublic=false, const bool isResourceGuid=false, const qint64 timeoutMsec=30000)
- std::pair< QNetworkRequest, QByteArray > createPostRequest (qevercloud::Guid guid, bool isPublic=false, bool isResourceGuid=false)

  Prepares a POST request for a thumbnail download.

Friends

- QEVERCLOUD_EXPORT QTextStream & operator<< (QTextStream &strm, const ImageType imageType)
- QEVERCLOUD_EXPORT QDebug & operator<< (QDebug &dbg, const ImageType imageType)

7.111.1 Detailed Description

The class is for downloading thumbnails for notes and resources from Evernote servers.

These thumbnails are not available with general EDAM Thrift interface as explained in the documentation.

Usage:

```
Thumbnail thumb("www.evernote.com", sharId, authenticationToken);
QByteArray pngImage = thumb.download(noteGuid);
```

By default 300x300 PNG images are requested.

7.111.2 Member Enumeration Documentation

7.111.2.1 ImageType

```
enum class qevercloud::Thumbnail::ImageType  [strong]
```

Specifies image type of the returned thumbnail.

Can be PNG, JPEG, GIF or BMP.

Enumerator

PNG	
JPEG	
GIF	
BMP	

7.111.3 Constructor & Destructor Documentation

7.111.3.1 Thumbnail() [1/2]

```
qevercloud::Thumbnail::Thumbnail ( )
```

Default constructor.

host, shardId, authenticationToken have to be specified before calling download or createPostRequest

7.111.3.2 Thumbnail() [2/2]

```
qevercloud::Thumbnail::Thumbnail (
            QString host,
            QString shardId,
            QString authenticationToken,
            int size = 300,
            ImageType imageType = ImageType::PNG )
```

Constructs Thumbnail.

Parameters

host	www.evernote.com or sandbox.evernote.com
shardId	You can get the value from UserStore service or as a result of an authentication.
authenticationToken	For working private notes/resources you must supply a valid authentication token. For public resources the value specified is not used.
size	The size of the thumbnail. Evernote supports values from from 1 to 300. By default 300 is used.
imageType	Thumbnail image type. See ImageType. By default PNG is used.

7.111.3.3 ~Thumbnail()

```
virtual qevercloud::Thumbnail::~Thumbnail ( )  [virtual]
```

7.111.4 Member Function Documentation

7.111.4.1 createPostRequest()

```
std::pair< QNetworkRequest, QByteArray > qevercloud::Thumbnail::createPostRequest (
            qevercloud::Guid guid,
```

```
            bool isPublic = false,
            bool isResourceGuid = false )
```

Prepares a POST request for a thumbnail download.

Parameters

guid	The note or resource guid
isPublic	Specify true for public notes/resources. In this case authentication token is not sent to with the request as it shoud be according to the docs.
isResourceGuid	true if guid denotes a resource and false if it denotes a note.

Returns

a pair of QNetworkRequest for the POST request and data that must be posted with the request.

7.111.4.2 download()

```
QByteArray qevercloud::Thumbnail::download (
            Guid guid,
            const bool isPublic = false,
            const bool isResourceGuid = false,
            const qint64 timeoutMsec = 30000 )
```

Downloads the thumbnail for a resource or a note.

Parameters

guid	The note or resource guid
isPublic	Specify true for public notes/resources. In this case authentication token is not sent to with the request as it shoud be according to the docs.
isResourceGuid	true if guid denotes a resource and false if it denotes a note.
timeoutMsec	Timeout for download request in milliseconds

Returns

downloaded data.

7.111.4.3 downloadAsync()

```
AsyncResult * qevercloud::Thumbnail::downloadAsync (
            Guid guid,
            const bool isPublic = false,
            const bool isResourceGuid = false,
            const qint64 timeoutMsec = 30000 )
```

Asynchronous version of download function

7.111.4.4 setAuthenticationToken()

```
Thumbnail & qevercloud::Thumbnail::setAuthenticationToken (
            QString authenticationToken )
```

Parameters

authenticationToken	For working private notes/resources you must supply a valid authentication token. For public resources the value specified is not used.

7.111.4.5 setHost()

```
Thumbnail & qevercloud::Thumbnail::setHost (
            QString host )
```

Parameters

host	www.evernote.com or sandbox.evernote.com

7.111.4.6 setImageType()

```
Thumbnail & qevercloud::Thumbnail::setImageType (
            ImageType imageType )
```

Parameters

imageType	Thumbnail image type. See ImageType. By default PNG is used.

7.111.4.7 setShardId()

```
Thumbnail & qevercloud::Thumbnail::setShardId (
            QString shardId )
```

Parameters

shardId	You can get the value from UserStore service or as a result of an authentication.

7.111.4.8 setSize()

```
Thumbnail & qevercloud::Thumbnail::setSize (
            int size )
```

Parameters

size	The size of the thumbnail. Evernote supports values from from 1 to 300. By default 300 is used.

7.111.5 Friends And Related Symbol Documentation

7.111.5.1 operator<< [1/2]

```
QEVERCLOUD_EXPORT QDebug & operator<< (
            QDebug & dbg,
            const ImageType imageType )  [friend]
```

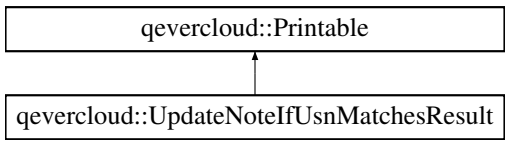
7.111.5.2 operator<< [2/2]

```
QEVERCLOUD_EXPORT QTextStream & operator<< (
            QTextStream & strm,
            const ImageType imageType )  [friend]
```

7.112 qevercloud::UpdateNoteIfUsnMatchesResult Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::UpdateNoteIfUsnMatchesResult:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const UpdateNoteIfUsnMatchesResult &other) const
- bool operator!= (const UpdateNoteIfUsnMatchesResult &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< Note > note
- Optional< bool > updated

7.112.1 Detailed Description

The result of a call to updateNoteIfUsnMatches, which optionally updates a note based on the current value of the note's update sequence number on the service.

7.112.2 Member Function Documentation

7.112.2.1 operator"!=()

```
bool qevercloud::UpdateNoteIfUsnMatchesResult::operator!= (
            const UpdateNoteIfUsnMatchesResult & other ) const  [inline]
```

7.112.2.2 operator==()

```
bool qevercloud::UpdateNoteIfUsnMatchesResult::operator== (
            const UpdateNoteIfUsnMatchesResult & other ) const  [inline]
```

7.112.2.3 print()

```
virtual void qevercloud::UpdateNoteIfUsnMatchesResult::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.112.3 Member Data Documentation

7.112.3.1 localData

```
EverCloudLocalData qevercloud::UpdateNoteIfUsnMatchesResult::localData
```

See the declaration of EverCloudLocalData for details

7.112.3.2 note

```
Optional< Note > qevercloud::UpdateNoteIfUsnMatchesResult::note
```

Either the current state of the note if `updated` is false or the result of updating the note as would be done via the `updateNote` method. If the note was not updated, you will receive a Note that does not include note content, resources data, resources recognition data, or resources alternate data. You can check for updates to these large objects by checking the Data.bodyHash values and downloading accordingly.

7.112.3.3 updated

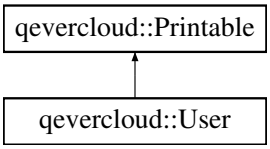
```
Optional< bool > qevercloud::UpdateNoteIfUsnMatchesResult::updated
```

Whether or not the note was updated by the operation.

7.113 qevercloud::User Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::User:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const User &other) const
- bool operator!= (const User &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< UserID > id
- Optional< QString > username
- Optional< QString > email
- Optional< QString > name
- Optional< QString > timezone
- Optional< PrivilegeLevel > privilege
- Optional< ServiceLevel > serviceLevel
- Optional< Timestamp > created
- Optional< Timestamp > updated
- Optional< Timestamp > deleted
- Optional< bool > active
- Optional< QString > shardId
- Optional< UserAttributes > attributes
- Optional< Accounting > accounting
- Optional< BusinessUserInfo > businessUserInfo
- Optional< QString > photoUrl
- Optional< Timestamp > photoLastUpdated
- Optional< AccountLimits > accountLimits

7.113.1 Detailed Description

This represents the information about a single user account.

7.113.2 Member Function Documentation

7.113.2.1 operator"!=()

```
bool qevercloud::User::operator!= (
            const User & other ) const  [inline]
```

7.113.2.2 operator==()

```
bool qevercloud::User::operator== (
            const User & other ) const  [inline]
```

7.113.2.3 print()

```
virtual void qevercloud::User::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.113.3 Member Data Documentation

7.113.3.1 accounting

```
Optional< Accounting > qevercloud::User::accounting
```

Bookkeeping information for the user's subscription.

7.113.3.2 accountLimits

```
Optional< AccountLimits > qevercloud::User::accountLimits
```

Account limits applicable for this user.

7.113.3.3 active

```
Optional< bool > qevercloud::User::active
```

If the user account is available for login and synchronization, this flag will be set to true.

7.113.3.4 attributes

```
Optional< UserAttributes > qevercloud::User::attributes
```

If present, this will contain a list of the attributes for this user account.

7.113.3.5 businessUserInfo

```
Optional< BusinessUserInfo > qevercloud::User::businessUserInfo
```

If present, this will contain a set of business information relating to the user's business membership. If not present, the user is not currently part of a business.

7.113.3.6 created

```
Optional< Timestamp > qevercloud::User::created
```

The date and time when this user account was created in the service.

7.113.3.7 deleted

```
Optional< Timestamp > qevercloud::User::deleted
```

If the account has been deleted from the system (e.g. as the result of a legal request by the user), the date and time of the deletion will be represented here. If not, this value will not be set.

7.113.3.8 email

```
Optional< QString > qevercloud::User::email
```

The email address registered for the user. Must comply with RFC 2821 and RFC 2822.
Third party applications that authenticate using OAuth do not have access to this field. Length: EDAM_EMAIL_↩ LEN_MIN - EDAM_EMAIL_LEN_MAX
Regex: EDAM_EMAIL_REGEX

7.113.3.9 id

```
Optional< UserID > qevercloud::User::id
```

The unique numeric identifier for the account, which will not change for the lifetime of the account.

7.113.3.10 localData

```
EverCloudLocalData qevercloud::User::localData
```

See the declaration of EverCloudLocalData for details

7.113.3.11 name

```
Optional< QString > qevercloud::User::name
```

The printable name of the user, which may be a combination of given and family names. This is used instead of separate "first" and "last" names due to variations in international name format/order. May not start or end with a whitespace character. May contain any character but carriage return or newline (Unicode classes Zl and Zp).
Length: EDAM_USER_NAME_LEN_MIN - EDAM_USER_NAME_LEN_MAX
Regex: EDAM_USER_NAME_REGEX

7.113.3.12 photoLastUpdated

`Optional< Timestamp > qevercloud::User::photoLastUpdated`

The time at which the photo at 'photoUrl' was last updated by this User. This field will be null if the User never set a profile photo. This field is filled in by the service and is read-only to clients.

7.113.3.13 photoUrl

`Optional< QString > qevercloud::User::photoUrl`

The URL of the photo that represents this User. This field is filled in by the service and is read-only to clients. If `photoLastUpdated` is not set, this url will point to a placeholder user photo generated by the service.

7.113.3.14 privilege

`Optional< PrivilegeLevel > qevercloud::User::privilege`

NOT DOCUMENTED

7.113.3.15 serviceLevel

`Optional< ServiceLevel > qevercloud::User::serviceLevel`

The level of service the user currently receives. This will always be populated for users retrieved from the Evernote service.

7.113.3.16 shardId

`Optional< QString > qevercloud::User::shardId`

DEPRECATED - Client applications should have no need to use this field.

7.113.3.17 timezone

`Optional< QString > qevercloud::User::timezone`

The zone ID for the user's default location. If present, this may be used to localize the display of any timestamp for which no other timezone is available. The format must be encoded as a standard zone ID such as "America/Los↩
_Angeles" or "GMT+08:00"
Length: EDAM_TIMEZONE_LEN_MIN - EDAM_TIMEZONE_LEN_MAX
Regex: EDAM_TIMEZONE_REGEX

7.113.3.18 updated

`Optional< Timestamp > qevercloud::User::updated`

The date and time when this user account was last modified in the service.

7.113.3.19 username

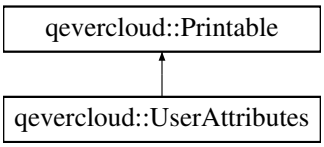
```
Optional< QString > qevercloud::User::username
```

The name that uniquely identifies a single user account. This name may be presented by the user, along with their password, to log into their account. May only contain a-z, 0-9, or '-', and may not start or end with the '-'
Length: EDAM_USER_USERNAME_LEN_MIN - EDAM_USER_USERNAME_LEN_MAX
Regex: EDAM_USER_USERNAME_REGEX

7.114 qevercloud::UserAttributes Struct Reference

```
#include <Types.h>
```

Inheritance diagram for qevercloud::UserAttributes:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const UserAttributes &other) const
- bool operator!= (const UserAttributes &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > defaultLocationName
- Optional< double > defaultLatitude
- Optional< double > defaultLongitude
- Optional< bool > preactivation
- Optional< QStringList > viewedPromotions
- Optional< QString > incomingEmailAddress
- Optional< QStringList > recentMailedAddresses
- Optional< QString > comments
- Optional< Timestamp > dateAgreedToTermsOfService
- Optional< qint32 > maxReferrals
- Optional< qint32 > referralCount
- Optional< QString > refererCode
- Optional< Timestamp > sentEmailDate

- Optional< qint32 > sentEmailCount
- Optional< qint32 > dailyEmailLimit
- Optional< Timestamp > emailOptOutDate
- Optional< Timestamp > partnerEmailOptInDate
- Optional< QString > preferredLanguage
- Optional< QString > preferredCountry
- Optional< bool > clipFullPage
- Optional< QString > twitterUserName
- Optional< QString > twitterId
- Optional< QString > groupName
- Optional< QString > recognitionLanguage
- Optional< QString > referralProof
- Optional< bool > educationalDiscount
- Optional< QString > businessAddress
- Optional< bool > hideSponsorBilling
- Optional< bool > useEmailAutoFiling
- Optional< ReminderEmailConfig > reminderEmailConfig
- Optional< Timestamp > emailAddressLastConfirmed
- Optional< Timestamp > passwordUpdated
- Optional< bool > salesforcePushEnabled
- Optional< bool > shouldLogClientEvent
- Optional< bool > optOutMachineLearning

7.114.1 Detailed Description

A structure holding the optional attributes that can be stored on a User. These are generally less critical than the core User fields.

7.114.2 Member Function Documentation

7.114.2.1 operator"!=()

```
bool qevercloud::UserAttributes::operator!= (
            const UserAttributes & other ) const  [inline]
```

7.114.2.2 operator==()

```
bool qevercloud::UserAttributes::operator== (
            const UserAttributes & other ) const  [inline]
```

7.114.2.3 print()

```
virtual void qevercloud::UserAttributes::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.114.3 Member Data Documentation

7.114.3.1 businessAddress

```
Optional< QString > qevercloud::UserAttributes::businessAddress
```

A string recording the business address of a Sponsored Account user who has requested invoicing.

7.114.3.2 clipFullPage

```
Optional< bool > qevercloud::UserAttributes::clipFullPage
```

Boolean flag set to true if the user wants to clip full pages by default when they use the web clipper without a selection.

7.114.3.3 comments

```
Optional< QString > qevercloud::UserAttributes::comments
```

Free-form text field that may hold general support information, etc.
Length: EDAM_ATTRIBUTE_LEN_MIN - EDAM_ATTRIBUTE_LEN_MAX

7.114.3.4 dailyEmailLimit

```
Optional< qint32 > qevercloud::UserAttributes::dailyEmailLimit
```

If set, this is the maximum number of emails that may be sent in a given day from this account. If unset, the server will use the configured default limit.

7.114.3.5 dateAgreedToTermsOfService

```
Optional< Timestamp > qevercloud::UserAttributes::dateAgreedToTermsOfService
```

The date/time when the user agreed to the terms of service. This can be used as the effective "start date" for the account.

7.114.3.6 defaultLatitude

```
Optional< double > qevercloud::UserAttributes::defaultLatitude
```

if set, this is the latitude that should be assigned to any notes that have no other latitude information.

7.114.3.7 defaultLocationName

```
Optional< QString > qevercloud::UserAttributes::defaultLocationName
```

the location string that should be associated with the user in order to determine where notes are taken if not otherwise specified.
Length: EDAM_ATTRIBUTE_LEN_MIN - EDAM_ATTRIBUTE_LEN_MAX

7.114.3.8 defaultLongitude

```
Optional< double > qevercloud::UserAttributes::defaultLongitude
```

if set, this is the longitude that should be assigned to any notes that have no other longitude information.

7.114.3.9 educationalDiscount

```
Optional< bool > qevercloud::UserAttributes::educationalDiscount
```

NOT DOCUMENTED

7.114.3.10 emailAddressLastConfirmed

```
Optional< Timestamp > qevercloud::UserAttributes::emailAddressLastConfirmed
```

If set, this contains the time at which the user last confirmed that the configured email address for this account is correct and up-to-date. If this is unset that indicates that the user's email address is unverified.

7.114.3.11 emailOptOutDate

```
Optional< Timestamp > qevercloud::UserAttributes::emailOptOutDate
```

If set, this is the date when the user asked to be excluded from offers and promotions sent by Evernote. If not set, then the user currently agrees to receive these messages.

7.114.3.12 groupName

```
Optional< QString > qevercloud::UserAttributes::groupName
```

A name identifier used to identify a particular set of branding and light customization.

7.114.3.13 hideSponsorBilling

```
Optional< bool > qevercloud::UserAttributes::hideSponsorBilling
```

A flag indicating whether to hide the billing information on a sponsored account owner's settings page

7.114.3.14 incomingEmailAddress

```
Optional< QString > qevercloud::UserAttributes::incomingEmailAddress
```

if set, this is the email address that the user may send email to in order to add an email note directly into the account via the SMTP email gateway. This is the part of the email address before the '@' symbol ... our domain is not included. If this is not set, the user may not add notes via the gateway.
Length: EDAM_ATTRIBUTE_LEN_MIN - EDAM_ATTRIBUTE_LEN_MAX

7.114.3.15 localData

```
EverCloudLocalData qevercloud::UserAttributes::localData
```

See the declaration of EverCloudLocalData for details

7.114.3.16 maxReferrals

```
Optional< qint32 > qevercloud::UserAttributes::maxReferrals
```

The number of referrals that the user is permitted to make.

7.114.3.17 optOutMachineLearning

```
Optional< bool > qevercloud::UserAttributes::optOutMachineLearning
```

If set to True, no Machine Learning nor human review will be done to this user's note contents.

7.114.3.18 partnerEmailOptInDate

```
Optional< Timestamp > qevercloud::UserAttributes::partnerEmailOptInDate
```

If set, this is the date when the user asked to be included in offers and promotions sent by Evernote's partners. If not sent, then the user currently does not agree to receive these emails.

7.114.3.19 passwordUpdated

```
Optional< Timestamp > qevercloud::UserAttributes::passwordUpdated
```

If set, this contains the time at which the user's password last changed. This will be unset for users created before the addition of this field who have not changed their passwords since the addition of this field.

7.114.3.20 preactivation

```
Optional< bool > qevercloud::UserAttributes::preactivation
```

if set, the user account is not yet confirmed for login. I.e. the account has been created, but we are still waiting for the user to complete the activation step.

7.114.3.21 preferredCountry

```
Optional< QString > qevercloud::UserAttributes::preferredCountry
```

Preferred country code based on ISO 3166-1-alpha-2 indicating the users preferred country

7.114.3.22 preferredLanguage

```
Optional< QString > qevercloud::UserAttributes::preferredLanguage
```

a 2 character language codes based on: http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt used for localization purposes to determine what language to use for the web interface and for other direct communication (e.g. emails).

7.114.3.23 recentMailedAddresses

```
Optional< QStringList > qevercloud::UserAttributes::recentMailedAddresses
```

if set, this will contain a list of email addresses that have recently been used as recipients of outbound emails by the user. This can be used to pre-populate a list of possible destinations when a user wishes to send a note via email.
Length: EDAM_ATTRIBUTE_LEN_MIN - EDAM_ATTRIBUTE_LEN_MAX each
Max: EDAM_USER_RECENT_MAILED_ADDRESSES_MAX entries

7.114.3.24 recognitionLanguage

```
Optional< QString > qevercloud::UserAttributes::recognitionLanguage
```

a 2 character language codes based on: http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt If set, this is used to determine the language that should be used when processing images and PDF files to find text. If not set, then the 'preferredLanguage' will be used.

7.114.3.25 refererCode

```
Optional< QString > qevercloud::UserAttributes::refererCode
```

A code indicating where the user was sent from. AKA promotion code

7.114.3.26 referralCount

```
Optional< qint32 > qevercloud::UserAttributes::referralCount
```

The number of referrals sent from this account.

7.114.3.27 referralProof

```
Optional< QString > qevercloud::UserAttributes::referralProof
```

NOT DOCUMENTED

7.114.3.28 reminderEmailConfig

```
Optional< ReminderEmailConfig > qevercloud::UserAttributes::reminderEmailConfig
```

Configuration state for whether or not the user wishes to receive reminder e-mail. This setting applies to both the reminder e-mail sent for personal reminder notes and for the reminder e-mail sent for reminder notes in the user's business notebooks that the user has configured for e-mail notifications.

7.114.3.29 salesforcePushEnabled

```
Optional< bool > qevercloud::UserAttributes::salesforcePushEnabled
```

NOT DOCUMENTED

7.114.3.30 sentEmailCount

```
Optional< qint32 > qevercloud::UserAttributes::sentEmailCount
```

The number of emails that were sent from the user via the service on sentEmailDate. Used to enforce a limit on the number of emails per user per day to prevent spamming.

7.114.3.31 sentEmailDate

```
Optional< Timestamp > qevercloud::UserAttributes::sentEmailDate
```

The most recent date when the user sent outbound emails from the service. Used with sentEmailCount to limit the number of emails that can be sent per day.

7.114.3.32 shouldLogClientEvent

```
Optional< bool > qevercloud::UserAttributes::shouldLogClientEvent
```

If set to True, the server will record LogRequest send from clients of this user as ClientEventLog.

7.114.3.33 twitterId

```
Optional< QString > qevercloud::UserAttributes::twitterId
```

The unique identifier of the user's Twitter account if that user has chosen to enable Twittering into Evernote.

7.114.3.34 twitterUserName

```
Optional< QString > qevercloud::UserAttributes::twitterUserName
```

The username of the account of someone who has chosen to enable Twittering into Evernote. This value is subject to change, since users may change their Twitter user name.

7.114.3.35 useEmailAutoFiling

```
Optional< bool > qevercloud::UserAttributes::useEmailAutoFiling
```

A flag indicating whether the user chooses to allow Evernote to automatically file and tag emailed notes

7.114.3.36 viewedPromotions

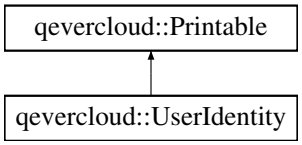
`Optional< QStringList > qevercloud::UserAttributes::viewedPromotions`

a list of promotions the user has seen. This list may occasionally be modified by the system when promotions are no longer available.
Length: EDAM_ATTRIBUTE_LEN_MIN - EDAM_ATTRIBUTE_LEN_MAX

7.115 qevercloud::UserIdentity Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::UserIdentity:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const UserIdentity &other) const
- bool operator!= (const UserIdentity &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< UserIdentityType > type
- Optional< QString > stringIdentifier
- Optional< qint64 > longIdentifier

7.115.1 Detailed Description

A structure that holds user identifying information such as an email address, Evernote user ID, or an identifier from a 3rd party service. An instance consists of a type and a value, where the value will be stored in one of the value fields depending upon the data type required for the identity type.

When used with shared notebook invitations, a UserIdentity identifies a particular person who may not (yet) have an Evernote UserID UserIdentity but who has (almost) unique access to the service endpoint described by the UserIdentity. For example, an e-mail UserIdentity can identify the person who receives e-mail at the given address, and who can therefore read the share key that has a cryptographic signature from the Evernote service. With the share key, this person can supply their Evernote UserID via an authentication token to join the notebook (authenticateToSharedNotebook), at which time we have associated the e-mail UserIdentity with an Evernote UserID UserIdentity. Note that using shared notebook records, the relationship between Evernote UserIDs and e-mail addresses is many to many.

Note that the identifier may not directly identify a particular Evernote UserID UserIdentity without further verification. For example, an e-mail UserIdentity may be associated with an invitation to join a notebook (via a shared notebook record), but until a user uses a share key, that was sent to that e-mail address, to join the notebook, we do not know an Evernote UserID UserIdentity ID to match the e-mail address.

7.115.2 Member Function Documentation

7.115.2.1 operator"!=()

```
bool qevercloud::UserIdentity::operator!= (
            const UserIdentity & other ) const  [inline]
```

7.115.2.2 operator==()

```
bool qevercloud::UserIdentity::operator== (
            const UserIdentity & other ) const  [inline]
```

7.115.2.3 print()

```
virtual void qevercloud::UserIdentity::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.115.3 Member Data Documentation

7.115.3.1 localData

```
EverCloudLocalData qevercloud::UserIdentity::localData
```

See the declaration of EverCloudLocalData for details

7.115.3.2 longIdentifier

```
Optional< qint64 > qevercloud::UserIdentity::longIdentifier
```

NOT DOCUMENTED

7.115.3.3 stringIdentifier

```
Optional< QString > qevercloud::UserIdentity::stringIdentifier
```

NOT DOCUMENTED

7.115.3.4 type

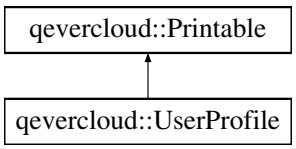
```
Optional< UserIdentityType > qevercloud::UserIdentity::type
```

NOT DOCUMENTED

7.116 qevercloud::UserProfile Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::UserProfile:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const UserProfile &other) const
- bool operator!= (const UserProfile &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< UserID > id
- Optional< QString > name
- Optional< QString > email
- Optional< QString > username
- Optional< BusinessUserAttributes > attributes
- Optional< Timestamp > joined
- Optional< Timestamp > photoLastUpdated
- Optional< QString > photoUrl
- Optional< BusinessUserRole > role
- Optional< BusinessUserStatus > status

7.116.1 Detailed Description

This structure represents profile information for a user in a business.

7.116.2 Member Function Documentation

7.116.2.1 operator"!=()

```
bool qevercloud::UserProfile::operator!= (
            const UserProfile & other ) const  [inline]
```

7.116.2.2 operator==()

```
bool qevercloud::UserProfile::operator== (
            const UserProfile & other ) const  [inline]
```

7.116.2.3 print()

```
virtual void qevercloud::UserProfile::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.116.3 Member Data Documentation

7.116.3.1 attributes

```
Optional< BusinessUserAttributes > qevercloud::UserProfile::attributes
```

The user's business specific attributes.

7.116.3.2 email

```
Optional< QString > qevercloud::UserProfile::email
```

The user's business email address. If the user has not registered their business email address, this field will be empty.

7.116.3.3 id

```
Optional< UserID > qevercloud::UserProfile::id
```

The numeric identifier that uniquely identifies a user.

7.116.3.4 joined

```
Optional< Timestamp > qevercloud::UserProfile::joined
```

The time when the user joined the business

7.116.3.5 localData

```
EverCloudLocalData qevercloud::UserProfile::localData
```

See the declaration of EverCloudLocalData for details

7.116.3.6 name

```
Optional< QString > qevercloud::UserProfile::name
```

The full name of the user.

7.116.3.7 photoLastUpdated

```
Optional< Timestamp > qevercloud::UserProfile::photoLastUpdated
```

The time when the user's profile photo was most recently updated

7.116.3.8 photoUrl

```
Optional< QString > qevercloud::UserProfile::photoUrl
```

A URL identifying a copy of the user's current profile photo

7.116.3.9 role

```
Optional< BusinessUserRole > qevercloud::UserProfile::role
```

The BusinessUserRole for the user

7.116.3.10 status

```
Optional< BusinessUserStatus > qevercloud::UserProfile::status
```

The BusinessUserStatus for the user

7.116.3.11 username

```
Optional< QString > qevercloud::UserProfile::username
```

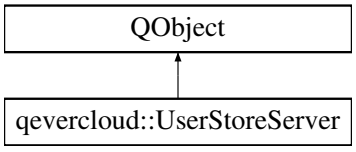
The user's Evernote username.

7.117 qevercloud::UserStoreServer Class Reference

The UserStoreServer class represents customizable server for UserStore requests. It is primarily used for testing of QEverCloud.

```
#include <Servers.h>
```

Inheritance diagram for qevercloud::UserStoreServer:



Public Slots

- void onRequest (QByteArray data)
- void onCheckVersionRequestReady (bool value, EverCloudExceptionDataPtr exceptionData)
- void onGetBootstrapInfoRequestReady (BootstrapInfo value, EverCloudExceptionDataPtr exceptionData)
- void onAuthenticateLongSessionRequestReady (AuthenticationResult value, EverCloudExceptionDataPtr exceptionData)
- void onCompleteTwoFactorAuthenticationRequestReady (AuthenticationResult value, EverCloudExceptionDataPtr exceptionData)
- void onRevokeLongSessionRequestReady (EverCloudExceptionDataPtr exceptionData)
- void onAuthenticateToBusinessRequestReady (AuthenticationResult value, EverCloudExceptionDataPtr exceptionData)
- void onGetUserRequestReady (User value, EverCloudExceptionDataPtr exceptionData)
- void onGetPublicUserInfoRequestReady (PublicUserInfo value, EverCloudExceptionDataPtr exceptionData)
- void onGetUserUrlsRequestReady (UserUrls value, EverCloudExceptionDataPtr exceptionData)
- void onInviteToBusinessRequestReady (EverCloudExceptionDataPtr exceptionData)
- void onRemoveFromBusinessRequestReady (EverCloudExceptionDataPtr exceptionData)
- void onUpdateBusinessUserIdentifierRequestReady (EverCloudExceptionDataPtr exceptionData)
- void onListBusinessUsersRequestReady (QList< UserProfile > value, EverCloudExceptionDataPtr exceptionData)
- void onListBusinessInvitationsRequestReady (QList< BusinessInvitation > value, EverCloudExceptionDataPtr exceptionData)
- void onGetAccountLimitsRequestReady (AccountLimits value, EverCloudExceptionDataPtr exceptionData)

Signals

- void checkVersionRequest (QString clientName, qint16 edamVersionMajor, qint16 edamVersionMinor, IRequestContextPtr ctx)
- void getBootstrapInfoRequest (QString locale, IRequestContextPtr ctx)
- void authenticateLongSessionRequest (QString username, QString password, QString consumerKey, QString consumerSecret, QString deviceIdentifier, QString deviceDescription, bool supportsTwoFactor, IRequestContextPtr ctx)
- void completeTwoFactorAuthenticationRequest (QString oneTimeCode, QString deviceIdentifier, QString deviceDescription, IRequestContextPtr ctx)
- void revokeLongSessionRequest (IRequestContextPtr ctx)
- void authenticateToBusinessRequest (IRequestContextPtr ctx)
- void getUserRequest (IRequestContextPtr ctx)
- void getPublicUserInfoRequest (QString username, IRequestContextPtr ctx)
- void getUserUrlsRequest (IRequestContextPtr ctx)
- void inviteToBusinessRequest (QString emailAddress, IRequestContextPtr ctx)
- void removeFromBusinessRequest (QString emailAddress, IRequestContextPtr ctx)
- void updateBusinessUserIdentifierRequest (QString oldEmailAddress, QString newEmailAddress, IRequestContextPtr ctx)
- void listBusinessUsersRequest (IRequestContextPtr ctx)
- void listBusinessInvitationsRequest (bool includeRequestedInvitations, IRequestContextPtr ctx)
- void getAccountLimitsRequest (ServiceLevel serviceLevel, IRequestContextPtr ctx)
- void checkVersionRequestReady (QByteArray data)
- void getBootstrapInfoRequestReady (QByteArray data)
- void authenticateLongSessionRequestReady (QByteArray data)
- void completeTwoFactorAuthenticationRequestReady (QByteArray data)
- void revokeLongSessionRequestReady (QByteArray data)
- void authenticateToBusinessRequestReady (QByteArray data)
- void getUserRequestReady (QByteArray data)
- void getPublicUserInfoRequestReady (QByteArray data)

- void getUserUrlsRequestReady (QByteArray data)
- void inviteToBusinessRequestReady (QByteArray data)
- void removeFromBusinessRequestReady (QByteArray data)
- void updateBusinessUserIdentifierRequestReady (QByteArray data)
- void listBusinessUsersRequestReady (QByteArray data)
- void listBusinessInvitationsRequestReady (QByteArray data)
- void getAccountLimitsRequestReady (QByteArray data)

Public Member Functions

- UserStoreServer (QObject *parent=nullptr)

7.117.1 Detailed Description

The UserStoreServer class represents customizable server for UserStore requests. It is primarily used for testing of QEverCloud.

7.117.2 Constructor & Destructor Documentation

7.117.2.1 UserStoreServer()

```
qevercloud::UserStoreServer::UserStoreServer (
            QObject * parent = nullptr )  [explicit]
```

7.117.3 Member Function Documentation

7.117.3.1 authenticateLongSessionRequest

```
void qevercloud::UserStoreServer::authenticateLongSessionRequest (
            QString username,
            QString password,
            QString consumerKey,
            QString consumerSecret,
            QString deviceIdentifier,
            QString deviceDescription,
            bool supportsTwoFactor,
            IRequestContextPtr ctx )  [signal]
```

7.117.3.2 authenticateLongSessionRequestReady

```
void qevercloud::UserStoreServer::authenticateLongSessionRequestReady (
            QByteArray data )  [signal]
```

7.117.3.3 authenticateToBusinessRequest

```
void qevercloud::UserStoreServer::authenticateToBusinessRequest (
            IRequestContextPtr ctx )  [signal]
```

7.117.3.4 authenticateToBusinessRequestReady

```
void qevercloud::UserStoreServer::authenticateToBusinessRequestReady (
            QByteArray data )  [signal]
```

7.117.3.5 checkVersionRequest

```
void qevercloud::UserStoreServer::checkVersionRequest (
            QString clientName,
            qint16 edamVersionMajor,
            qint16 edamVersionMinor,
            IRequestContextPtr ctx )  [signal]
```

7.117.3.6 checkVersionRequestReady

```
void qevercloud::UserStoreServer::checkVersionRequestReady (
            QByteArray data )  [signal]
```

7.117.3.7 completeTwoFactorAuthenticationRequest

```
void qevercloud::UserStoreServer::completeTwoFactorAuthenticationRequest (
            QString oneTimeCode,
            QString deviceIdentifier,
            QString deviceDescription,
            IRequestContextPtr ctx )  [signal]
```

7.117.3.8 completeTwoFactorAuthenticationRequestReady

```
void qevercloud::UserStoreServer::completeTwoFactorAuthenticationRequestReady (
            QByteArray data )  [signal]
```

7.117.3.9 getAccountLimitsRequest

```
void qevercloud::UserStoreServer::getAccountLimitsRequest (
            ServiceLevel serviceLevel,
            IRequestContextPtr ctx )  [signal]
```

7.117.3.10 getAccountLimitsRequestReady

```
void qevercloud::UserStoreServer::getAccountLimitsRequestReady (
            QByteArray data )  [signal]
```

7.117.3.11 getBootstrapInfoRequest

```
void qevercloud::UserStoreServer::getBootstrapInfoRequest (
            QString locale,
            IRequestContextPtr ctx )  [signal]
```

7.117.3.12 getBootstrapInfoRequestReady

```
void qevercloud::UserStoreServer::getBootstrapInfoRequestReady (
            QByteArray data )  [signal]
```

7.117.3.13 getPublicUserInfoRequest

```
void qevercloud::UserStoreServer::getPublicUserInfoRequest (
            QString username,
            IRequestContextPtr ctx )  [signal]
```

7.117.3.14 getPublicUserInfoRequestReady

```
void qevercloud::UserStoreServer::getPublicUserInfoRequestReady (
            QByteArray data )  [signal]
```

7.117.3.15 getUserRequest

```
void qevercloud::UserStoreServer::getUserRequest (
            IRequestContextPtr ctx )  [signal]
```

7.117.3.16 getUserRequestReady

```
void qevercloud::UserStoreServer::getUserRequestReady (
            QByteArray data )  [signal]
```

7.117.3.17 getUserUrlsRequest

```
void qevercloud::UserStoreServer::getUserUrlsRequest (
            IRequestContextPtr ctx )  [signal]
```

7.117.3.18 getUserUrlsRequestReady

```
void qevercloud::UserStoreServer::getUserUrlsRequestReady (
            QByteArray data )  [signal]
```

7.117.3.19 inviteToBusinessRequest

```
void qevercloud::UserStoreServer::inviteToBusinessRequest (
            QString emailAddress,
            IRequestContextPtr ctx )  [signal]
```

7.117.3.20 inviteToBusinessRequestReady

```
void qevercloud::UserStoreServer::inviteToBusinessRequestReady (
            QByteArray data )  [signal]
```

7.117.3.21 listBusinessInvitationsRequest

```
void qevercloud::UserStoreServer::listBusinessInvitationsRequest (
            bool includeRequestedInvitations,
            IRequestContextPtr ctx )  [signal]
```

7.117.3.22 listBusinessInvitationsRequestReady

```
void qevercloud::UserStoreServer::listBusinessInvitationsRequestReady (
            QByteArray data )  [signal]
```

7.117.3.23 listBusinessUsersRequest

```
void qevercloud::UserStoreServer::listBusinessUsersRequest (
            IRequestContextPtr ctx )  [signal]
```

7.117.3.24 listBusinessUsersRequestReady

```
void qevercloud::UserStoreServer::listBusinessUsersRequestReady (
            QByteArray data )  [signal]
```

7.117.3.25 onAuthenticateLongSessionRequestReady

```
void qevercloud::UserStoreServer::onAuthenticateLongSessionRequestReady (
            AuthenticationResult value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.117.3.26 onAuthenticateToBusinessRequestReady

```
void qevercloud::UserStoreServer::onAuthenticateToBusinessRequestReady (
            AuthenticationResult value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.117.3.27 onCheckVersionRequestReady

```
void qevercloud::UserStoreServer::onCheckVersionRequestReady (
            bool value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.117.3.28 onCompleteTwoFactorAuthenticationRequestReady

```
void qevercloud::UserStoreServer::onCompleteTwoFactorAuthenticationRequestReady (
            AuthenticationResult value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.117.3.29 onGetAccountLimitsRequestReady

```
void qevercloud::UserStoreServer::onGetAccountLimitsRequestReady (
            AccountLimits value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.117.3.30 onGetBootstrapInfoRequestReady

```
void qevercloud::UserStoreServer::onGetBootstrapInfoRequestReady (
            BootstrapInfo value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.117.3.31 onGetPublicUserInfoRequestReady

```
void qevercloud::UserStoreServer::onGetPublicUserInfoRequestReady (
            PublicUserInfo value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.117.3.32 onGetUserRequestReady

```
void qevercloud::UserStoreServer::onGetUserRequestReady (
            User value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.117.3.33 onGetUserUrlsRequestReady

```
void qevercloud::UserStoreServer::onGetUserUrlsRequestReady (
            UserUrls value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.117.3.34 onInviteToBusinessRequestReady

```
void qevercloud::UserStoreServer::onInviteToBusinessRequestReady (
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.117.3.35 onListBusinessInvitationsRequestReady

```
void qevercloud::UserStoreServer::onListBusinessInvitationsRequestReady (
            QList< BusinessInvitation > value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.117.3.36 onListBusinessUsersRequestReady

```
void qevercloud::UserStoreServer::onListBusinessUsersRequestReady (
            QList< UserProfile > value,
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.117.3.37 onRemoveFromBusinessRequestReady

```
void qevercloud::UserStoreServer::onRemoveFromBusinessRequestReady (
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.117.3.38 onRequest

```
void qevercloud::UserStoreServer::onRequest (
            QByteArray data )  [slot]
```

7.117.3.39 onRevokeLongSessionRequestReady

```
void qevercloud::UserStoreServer::onRevokeLongSessionRequestReady (
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.117.3.40 onUpdateBusinessUserIdentifierRequestReady

```
void qevercloud::UserStoreServer::onUpdateBusinessUserIdentifierRequestReady (
            EverCloudExceptionDataPtr exceptionData )  [slot]
```

7.117.3.41 removeFromBusinessRequest

```
void qevercloud::UserStoreServer::removeFromBusinessRequest (
            QString emailAddress,
            IRequestContextPtr ctx )  [signal]
```

7.117.3.42 removeFromBusinessRequestReady

```
void qevercloud::UserStoreServer::removeFromBusinessRequestReady (
            QByteArray data )  [signal]
```

7.117.3.43 revokeLongSessionRequest

```
void qevercloud::UserStoreServer::revokeLongSessionRequest (
            IRequestContextPtr ctx )  [signal]
```

7.117.3.44 revokeLongSessionRequestReady

```
void qevercloud::UserStoreServer::revokeLongSessionRequestReady (
            QByteArray data )  [signal]
```

7.117.3.45 updateBusinessUserIdentifierRequest

```
void qevercloud::UserStoreServer::updateBusinessUserIdentifierRequest (
            QString oldEmailAddress,
            QString newEmailAddress,
            IRequestContextPtr ctx )  [signal]
```

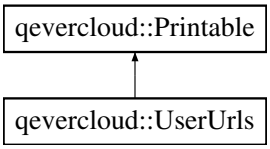
7.117.3.46 updateBusinessUserIdentifierRequestReady

```
void qevercloud::UserStoreServer::updateBusinessUserIdentifierRequestReady (
            QByteArray data )  [signal]
```

7.118 qevercloud::UserUrls Struct Reference

`#include <Types.h>`

Inheritance diagram for qevercloud::UserUrls:



Public Member Functions

- virtual void print (QTextStream &strm) const override
- bool operator== (const UserUrls &other) const
- bool operator!= (const UserUrls &other) const

Public Member Functions inherited from qevercloud::Printable

- Printable ()=default
- virtual ~Printable ()=default
- virtual QString toString () const

Public Attributes

- EverCloudLocalData localData
- Optional< QString > noteStoreUrl
- Optional< QString > webApiUrlPrefix
- Optional< QString > userStoreUrl
- Optional< QString > utilityUrl
- Optional< QString > messageStoreUrl
- Optional< QString > userWebSocketUrl

7.118.1 Member Function Documentation

7.118.1.1 operator"!=()

```
bool qevercloud::UserUrls::operator!= (
            const UserUrls & other ) const  [inline]
```

7.118.1.2 operator==()

```
bool qevercloud::UserUrls::operator== (
            const UserUrls & other ) const  [inline]
```

7.118.1.3 print()

```
virtual void qevercloud::UserUrls::print (
            QTextStream & strm ) const  [override], [virtual]
```

Implements qevercloud::Printable.

7.118.2 Member Data Documentation

7.118.2.1 localData

```
EverCloudLocalData qevercloud::UserUrls::localData
```

See the declaration of EverCloudLocalData for details

7.118.2.2 messageStoreUrl

```
Optional< QString > qevercloud::UserUrls::messageStoreUrl
```

This field will contain the full URL that clients should use to make MessageStore requests to the server. I.e. this is the URL that should be used to create the Thrift HTTP client transport to send messages to the MessageStore service for the account.

7.118.2.3 noteStoreUrl

```
Optional< QString > qevercloud::UserUrls::noteStoreUrl
```

This field will contain the full URL that clients should use to make NoteStore requests to the server shard that contains that user's data. I.e. this is the URL that should be used to create the Thrift HTTP client transport to send messages to the NoteStore service for the account.

7.118.2.4 userStoreUrl

```
Optional< QString > qevercloud::UserUrls::userStoreUrl
```

This field will contain the full URL that clients should use to make UserStore requests after successfully authenticating. I.e. this is the URL that should be used to create the Thrift HTTP client transport to send messages to the UserStore service for this account.

7.118.2.5 userWebSocketUrl

```
Optional< QString > qevercloud::UserUrls::userWebSocketUrl
```

This field will contain the full URL that clients should use when opening a persistent web socket to receive notification of events for the authenticated user.

7.118.2.6 utilityUrl

```
Optional< QString > qevercloud::UserUrls::utilityUrl
```

This field will contain the full URL that clients should use to make Utility requests to the server shard that contains that user's data. I.e. this is the URL that should be used to create the Thrift HTTP client transport to send messages to the Utility service for the account.

7.118.2.7 webApiUrlPrefix

```
Optional< QString > qevercloud::UserUrls::webApiUrlPrefix
```

This field will contain the initial part of the URLs that should be used to make requests to Evernote's thin client "web API", which provide optimized operations for clients that aren't capable of manipulating the full contents of accounts via the full Thrift data model. Clients should concatenate the relative path for the various servlets onto the end of this string to construct the full URL, as documented on our developer web site.

Chapter 8

File Documentation

8.1 AsyncResult.h File Reference

```
#include "EverCloudException.h"
#include "Export.h"
#include "Helpers.h"
#include "RequestContext.h"
#include <QNetworkRequest>
#include <QObject>
#include <QUuid>
```

Classes

- class qevercloud::AsyncResult

  Returned by asynchonous versions of functions.

Namespaces

- namespace qevercloud

8.2 AsyncResult.h

Go to the documentation of this file.

```
00001
00009 #ifndef QEVERCLOUD_ASYNC_RESULT_H
00010 #define QEVERCLOUD_ASYNC_RESULT_H
00011
00012 #include "EverCloudException.h"
00013 #include "Export.h"
00014 #include "Helpers.h"
00015 #include "RequestContext.h"
00016
00017 #include <QNetworkRequest>
00018 #include <QObject>
00019 #include <QUuid>
00020
00021 namespace qevercloud {
00022
00023 QT_FORWARD_DECLARE_CLASS(AsyncResultPrivate)
00024 QT_FORWARD_DECLARE_CLASS(DurableService)
```

```
00025
00026
00053 class QEVERCLOUD_EXPORT AsyncResult: public QObject
00054 {
00055     Q_OBJECT
00056     Q_DISABLE_COPY(AsyncResult)
00057 public:
00058     static QVariant asIs(QByteArray replyData);
00059
00060     typedef QVariant (*ReadFunctionType)(QByteArray replyData);
00061
00062     AsyncResult(QString url, QByteArray postData,
00063                 IRequestContextPtr ctx,
00064                 ReadFunctionType readFunction = AsyncResult::asIs,
00065                 bool autoDelete = true, QObject * parent = nullptr);
00066
00067     AsyncResult(QNetworkRequest request, QByteArray postData,
00068                 IRequestContextPtr ctx,
00069                 ReadFunctionType readFunction = AsyncResult::asIs,
00070                 bool autoDelete = true, QObject * parent = nullptr);
00071
00076     AsyncResult(QVariant result, EverCloudExceptionDataPtr error,
00077                 IRequestContextPtr ctx, bool autoDelete = true,
00078                 QObject * parent = nullptr);
00079
00080     ~AsyncResult();
00081
00088     bool waitForFinished(int timeout = -1);
00089
00090 Q_SIGNALS:
00102     void finished(
00103         QVariant result,
00104         EverCloudExceptionDataPtr error,
00105         IRequestContextPtr ctx);
00106
00107 private:
00108     friend class DurableService;
00109
00110 private:
00111     AsyncResultPrivate * const d_ptr;
00112     Q_DECLARE_PRIVATE(AsyncResult)
00113 };
00114
00115 } // namespace qevercloud
00116
00117 #endif // QEVERCLOUD_ASYNC_RESULT_H
```

8.3 DurableService.h File Reference

```
#include "AsyncResult.h"
#include "Export.h"
#include "RequestContext.h"
#include <QDateTime>
#include <QVariant>
#include <functional>
#include <memory>
#include <utility>
```

Classes

- struct qevercloud::IRetryPolicy
- class qevercloud::IDurableService
- struct qevercloud::IDurableService::SyncRequest
- struct qevercloud::IDurableService::AsyncRequest

Namespaces

- namespace qevercloud

Typedefs

- using qevercloud::IRetryPolicyPtr = std::shared_ptr< IRetryPolicy >
- using qevercloud::IDurableServicePtr = std::shared_ptr< IDurableService >

Functions

- QEVERCLOUD_EXPORT IRetryPolicyPtr qevercloud::newRetryPolicy ()
- QEVERCLOUD_EXPORT IRetryPolicyPtr qevercloud::nullRetryPolicy ()
- QEVERCLOUD_EXPORT IDurableServicePtr qevercloud::newDurableService (IRetryPolicyPtr={}, IRequestContextPtr={})

8.4 DurableService.h

Go to the documentation of this file.

```
00001
00008 #ifndef QEVERCLOUD_DURABLE_SERVICE_H
00009 #define QEVERCLOUD_DURABLE_SERVICE_H
00010
00011 #include "AsyncResult.h"
00012 #include "Export.h"
00013 #include "RequestContext.h"
00014
00015 #include <QDateTime>
00016 #include <QVariant>
00017
00018 #include <functional>
00019 #include <memory>
00020 #include <utility>
00021
00022 namespace qevercloud {
00023
00025
00026 struct QEVERCLOUD_EXPORT IRetryPolicy
00027 {
00028     virtual bool shouldRetry(
00029         const EverCloudExceptionDataPtr & exceptionData) = 0;
00030 };
00031
00032 using IRetryPolicyPtr = std::shared_ptr<IRetryPolicy>;
00033
00035
00036 QT_FORWARD_DECLARE_CLASS(DurableServicePrivate)
00037
00038 class QEVERCLOUD_EXPORT IDurableService
00039 {
00040 public:
00041     using SyncResult = std::pair<QVariant,EverCloudExceptionDataPtr>;
00042     using SyncServiceCall = std::function<SyncResult(IRequestContextPtr)>;
00043     using AsyncServiceCall = std::function<AsyncResult*(IRequestContextPtr)>;
00044
00045     struct QEVERCLOUD_EXPORT SyncRequest
00046     {
00047         const char *    m_name;
00048         QString         m_description;
00049         SyncServiceCall m_call;
00050
00051         SyncRequest(const char * name, QString description,
00052                     SyncServiceCall && call) :
00053             m_name(name),
00054             m_description(std::move(description)),
00055             m_call(std::move(call))
00056         {}
00057     };
00058
00059     struct QEVERCLOUD_EXPORT AsyncRequest
00060     {
00061         const char *    m_name;
00062         QString         m_description;
00063         AsyncServiceCall    m_call;
00064
00065         AsyncRequest(const char * name, QString description,
00066                      AsyncServiceCall && call) :
00067             m_name(name),
00068             m_description(std::move(description)),
00069             m_call(std::move(call))
```

```
00070         {}
00071     };
00072
00073 public:
00074     virtual SyncResult executeSyncRequest(
00075         SyncRequest && syncRequest, IRequestContextPtr ctx) = 0;
00076
00077     virtual AsyncResult * executeAsyncRequest(
00078         AsyncRequest && asyncRequest, IRequestContextPtr ctx) = 0;
00079 };
00080
00081 using IDurableServicePtr = std::shared_ptr<IDurableService>;
00082
00084
00085 QEVERCLOUD_EXPORT IRetryPolicyPtr newRetryPolicy();
00086
00087 QEVERCLOUD_EXPORT IRetryPolicyPtr nullRetryPolicy();
00088
00089 QEVERCLOUD_EXPORT IDurableServicePtr newDurableService(
00090     IRetryPolicyPtr = {},
00091     IRequestContextPtr = {});
00092
00093 } // namespace qevercloud
00094
00095 #endif // QEVERCLOUD_DURABLE_SERVICE_H
```

8.5 EventLoopFinisher.h File Reference

```
#include "Export.h"
#include "Helpers.h"
#include <QEventLoop>
#include <QObject>
```

Classes

- class qevercloud::EventLoopFinisher

Namespaces

- namespace qevercloud

8.6 EventLoopFinisher.h

Go to the documentation of this file.

```
00001
00009 #ifndef QEVERCLOUD_EVENT_LOOP_FINISHER_H
00010 #define QEVERCLOUD_EVENT_LOOP_FINISHER_H
00011
00012 #include "Export.h"
00013 #include "Helpers.h"
00014
00015 #include <QEventLoop>
00016 #include <QObject>
00017
00018 namespace qevercloud {
00019
00020 QT_FORWARD_DECLARE_CLASS(EventLoopFinisherPrivate)
00021
00022 class QEVERCLOUD_EXPORT EventLoopFinisher: public QObject
00023 {
00024     Q_OBJECT
00025 public:
00026     explicit EventLoopFinisher(
00027         QEventLoop * loop, int exitCode, QObject * parent = Q_NULLPTR);
00028
```

```
00029     ~EventLoopFinisher();
00030
00031 public Q_SLOTS:
00032     void stopEventLoop();
00033
00034 private:
00035     EventLoopFinisherPrivate * const d_ptr;
00036     Q_DECLARE_PRIVATE(EventLoopFinisher)
00037 };
00038
00039 } // namespace qevercloud
00040
00041 #endif // QEVERCLOUD_EVENT_LOOP_FINISHER_H
```

8.7 EverCloudException.h File Reference

```
#include "Export.h"
#include "Helpers.h"
#include <QObject>
#include <QString>
#include <exception>
#include <memory>
```

Classes

- class qevercloud::EverCloudException
- class qevercloud::EverCloudExceptionData

  EverCloudException counterpart for asynchronous API.
- class qevercloud::EvernoteException
- class qevercloud::EvernoteExceptionData

Namespaces

- namespace qevercloud

Typedefs

- using qevercloud::EverCloudExceptionDataPtr = std::shared_ptr< EverCloudExceptionData >

Variables

- class QEVERCLOUD_EXPORT qevercloud::EverCloudExceptionData

8.8 EverCloudException.h

Go to the documentation of this file.

```
00001
00009 #ifndef QEVERCLOUD_EVER_CLOUD_EXCEPTION_H
00010 #define QEVERCLOUD_EVER_CLOUD_EXCEPTION_H
00011
00012 #include "Export.h"
00013 #include "Helpers.h"
00014
00015 #include <QObject>
00016 #include <QString>
00017
00018 #include <exception>
00019 #include <memory>
00020
00021 namespace qevercloud {
00022
00024
00025 class QEVERCLOUD_EXPORT EverCloudExceptionData;
00026
00028
00032 class QEVERCLOUD_EXPORT EverCloudException: public std::exception
00033 {
00034 protected:
00035     mutable QByteArray m_error;
00036
00037 public:
00038     explicit EverCloudException();
00039     explicit EverCloudException(QString error);
00040     explicit EverCloudException(const std::string & error);
00041     explicit EverCloudException(const char * error);
00042
00043     virtual ~EverCloudException() noexcept override;
00044
00045     virtual const char * what() const noexcept override;
00046
00047     virtual std::shared_ptr<EverCloudExceptionData> exceptionData() const;
00048 };
00049
00051
00129 class QEVERCLOUD_EXPORT EverCloudExceptionData: public QObject
00130 {
00131     Q_OBJECT
00132     Q_DISABLE_COPY(EverCloudExceptionData)
00133 public:
00137     QString errorMessage;
00138
00139     explicit EverCloudExceptionData(QString error);
00140
00146     virtual void throwException() const;
00147 };
00148
00149 using EverCloudExceptionDataPtr = std::shared_ptr<EverCloudExceptionData>;
00150
00152
00157 class QEVERCLOUD_EXPORT EvernoteException: public EverCloudException
00158 {
00159 public:
00160     explicit EvernoteException();
00161     explicit EvernoteException(QString error);
00162     explicit EvernoteException(const std::string & error);
00163     explicit EvernoteException(const char * error);
00164
00165     virtual EverCloudExceptionDataPtr exceptionData() const override;
00166 };
00167
00169
00174 class QEVERCLOUD_EXPORT EvernoteExceptionData: public EverCloudExceptionData
00175 {
00176     Q_OBJECT
00177     Q_DISABLE_COPY(EvernoteExceptionData)
00178 public:
00179     explicit EvernoteExceptionData(QString error);
00180     virtual void throwException() const override;
00181 };
00182
00183 } // namespace qevercloud
00184
00185 #endif // QEVERCLOUD_EVER_CLOUD_EXCEPTION_H
```

8.9 Exceptions.h File Reference

```
#include "EverCloudException.h"
#include "Export.h"
#include "Optional.h"
#include "generated/EDAMErrorCode.h"
#include "generated/Types.h"
#include <QNetworkReply>
#include <QObject>
#include <QString>
```

Classes

- class qevercloud::NetworkException

  The NetworkException class represents QNetworkReply level errors.
- class qevercloud::NetworkExceptionData
- class qevercloud::ThriftException
- class qevercloud::ThriftExceptionData
- class qevercloud::EDAMUserExceptionData
- class qevercloud::EDAMSystemExceptionData
- class qevercloud::EDAMNotFoundExceptionData
- class qevercloud::EDAMInvalidContactsExceptionData
- class qevercloud::EDAMSystemExceptionRateLimitReached
- class qevercloud::EDAMSystemExceptionRateLimitReachedData
- class qevercloud::EDAMSystemExceptionAuthExpired
- class qevercloud::EDAMSystemExceptionAuthExpiredData

Namespaces

- namespace qevercloud

8.10 Exceptions.h

Go to the documentation of this file.

```
00001
00009 #ifndef QEVERCLOUD_EXCEPTIONS_H
00010 #define QEVERCLOUD_EXCEPTIONS_H
00011
00012 #include "EverCloudException.h"
00013 #include "Export.h"
00014 #include "Optional.h"
00015 #include "generated/EDAMErrorCode.h"
00016 #include "generated/Types.h"
00017
00018 #include <QNetworkReply>
00019 #include <QObject>
00020 #include <QString>
00021
00022 namespace qevercloud {
00023
00025
00029 class QEVERCLOUD_EXPORT NetworkException: public EverCloudException
00030 {
00031 public:
00032     NetworkException();
00033     NetworkException(QNetworkReply::NetworkError error);
00034     NetworkException(QNetworkReply::NetworkError error, QString message);
00035     virtual ~NetworkException() noexcept override;
00036
```

```
00037     bool operator==(const NetworkException & other) const;
00038     bool operator!=(const NetworkException & other) const;
00039
00040     QNetworkReply::NetworkError type() const;
00041
00042     const char * what() const noexcept override;
00043
00044     virtual EverCloudExceptionDataPtr exceptionData() const override;
00045
00046 protected:
00047     QNetworkReply::NetworkError m_type;
00048 };
00049
00054 class QEVERCLOUD_EXPORT NetworkExceptionData: public EverCloudExceptionData
00055 {
00056     Q_OBJECT
00057     Q_DISABLE_COPY(NetworkExceptionData)
00058 public:
00059     explicit NetworkExceptionData(QString error, QNetworkReply::NetworkError type);
00060     virtual void throwException() const override;
00061
00062 protected:
00063     QNetworkReply::NetworkError m_type;
00064 };
00065
00067
00072 class QEVERCLOUD_EXPORT ThriftException: public EverCloudException
00073 {
00074 public:
00075     enum class Type {
00076         UNKNOWN = 0,
00077         UNKNOWN_METHOD = 1,
00078         INVALID_MESSAGE_TYPE = 2,
00079         WRONG_METHOD_NAME = 3,
00080         BAD_SEQUENCE_ID = 4,
00081         MISSING_RESULT = 5,
00082         INTERNAL_ERROR = 6,
00083         PROTOCOL_ERROR = 7,
00084         INVALID_DATA = 8
00085     };
00086
00087     friend QEVERCLOUD_EXPORT QTextStream & operator<<(
00088         QTextStream & strm, const Type type);
00089
00090     ThriftException();
00091     ThriftException(Type type);
00092     ThriftException(Type type, QString message);
00093     virtual ~ThriftException() noexcept override;
00094
00095     bool operator==(const ThriftException & other) const;
00096     bool operator!=(const ThriftException & other) const;
00097
00098     Type type() const;
00099
00100     const char * what() const noexcept override;
00101
00102     virtual EverCloudExceptionDataPtr exceptionData() const override;
00103
00104 protected:
00105     Type m_type;
00106 };
00107
00112 class QEVERCLOUD_EXPORT ThriftExceptionData: public EverCloudExceptionData
00113 {
00114     Q_OBJECT
00115     Q_DISABLE_COPY(ThriftExceptionData)
00116 public:
00117     explicit ThriftExceptionData(QString error, ThriftException::Type type);
00118     virtual void throwException() const override;
00119
00120 protected:
00121     ThriftException::Type m_type;
00122 };
00123
00125
00130 class QEVERCLOUD_EXPORT EDAMUserExceptionData: public EvernoteExceptionData
00131 {
00132     Q_OBJECT
00133     Q_DISABLE_COPY(EDAMUserExceptionData)
00134 public:
00135     explicit EDAMUserExceptionData(
00136         QString error, EDAMErrorCode errorCode, Optional<QString> parameter);
00137
00138     virtual void throwException() const override;
00139
00140 protected:
00141     EDAMErrorCode       m_errorCode;
```

```
00142     Optional<QString>   m_parameter;
00143 };
00144
00146
00151 class QEVERCLOUD_EXPORT EDAMSystemExceptionData: public EvernoteExceptionData
00152 {
00153     Q_OBJECT
00154     Q_DISABLE_COPY(EDAMSystemExceptionData)
00155 public:
00156     explicit EDAMSystemExceptionData(
00157         QString err, EDAMErrorCode errorCode, Optional<QString> message,
00158         Optional<qint32> rateLimitDuration);
00159
00160     virtual void throwException() const override;
00161
00162 protected:
00163     EDAMErrorCode       m_errorCode;
00164     Optional<QString>   m_message;
00165     Optional<qint32>    m_rateLimitDuration;
00166 };
00167
00169
00174 class QEVERCLOUD_EXPORT EDAMNotFoundExceptionData: public EvernoteExceptionData
00175 {
00176     Q_OBJECT
00177     Q_DISABLE_COPY(EDAMNotFoundExceptionData)
00178 public:
00179     explicit EDAMNotFoundExceptionData(
00180         QString error, Optional<QString> identifier, Optional<QString> key);
00181
00182     virtual void throwException() const override;
00183
00184 protected:
00185     Optional<QString> m_identifier;
00186     Optional<QString> m_key;
00187 };
00188
00190
00195 class QEVERCLOUD_EXPORT EDAMInvalidContactsExceptionData:
00196     public EvernoteExceptionData
00197 {
00198     Q_OBJECT
00199     Q_DISABLE_COPY(EDAMInvalidContactsExceptionData)
00200 public:
00201     explicit EDAMInvalidContactsExceptionData(
00202         QList<Contact> contacts, Optional<QString> parameter,
00203         Optional<QList<EDAMInvalidContactReason> > reasons);
00204
00205     virtual void throwException() const override;
00206
00207 protected:
00208     QList<Contact>      m_contacts;
00209     Optional<QString>   m_parameter;
00210     Optional<QList<EDAMInvalidContactReason>> m_reasons;
00211 };
00212
00214
00218 class QEVERCLOUD_EXPORT EDAMSystemExceptionRateLimitReached:
00219     public EDAMSystemException
00220 {
00221 public:
00222     virtual EverCloudExceptionDataPtr exceptionData() const override;
00223 };
00224
00226
00231 class QEVERCLOUD_EXPORT EDAMSystemExceptionRateLimitReachedData:
00232     public EDAMSystemExceptionData
00233 {
00234     Q_OBJECT
00235     Q_DISABLE_COPY(EDAMSystemExceptionRateLimitReachedData)
00236 public:
00237     explicit EDAMSystemExceptionRateLimitReachedData(
00238         QString error, EDAMErrorCode errorCode, Optional<QString> message,
00239         Optional<qint32> rateLimitDuration);
00240
00241     virtual void throwException() const override;
00242 };
00243
00245
00249 class QEVERCLOUD_EXPORT EDAMSystemExceptionAuthExpired: public EDAMSystemException
00250 {
00251 public:
00252     virtual EverCloudExceptionDataPtr exceptionData() const override;
00253 };
00254
00256
00261 class QEVERCLOUD_EXPORT EDAMSystemExceptionAuthExpiredData:
```

```
00262     public EDAMSystemExceptionData
00263 {
00264     Q_OBJECT
00265     Q_DISABLE_COPY(EDAMSystemExceptionAuthExpiredData)
00266 public:
00267     explicit EDAMSystemExceptionAuthExpiredData(
00268         QString error, EDAMErrorCode errorCode, Optional<QString> message,
00269         Optional<qint32> rateLimitDuration);
00270
00271     virtual void throwException() const override;
00272 };
00273
00274 } // namespace qevercloud
00275
00276 #endif // QEVERCLOUD_EXCEPTIONS_H
```

8.11 Export.h File Reference

```
#include <QtCore/QtGlobal>
```

Macros

- #define QEVERCLOUD_EXPORT Q_DECL_IMPORT

8.11.1 Macro Definition Documentation

8.11.1.1 QEVERCLOUD_EXPORT

```
#define QEVERCLOUD_EXPORT Q_DECL_IMPORT
```

Original work: Copyright (c) 2014 Sergey Skoblikov Modified work: Copyright (c) 2015-2019 Dmitry Ivanov

This file is a part of QEverCloud project and is distributed under the terms of MIT license: https://opensource.org/licenses/MIT

8.12 Export.h

Go to the documentation of this file.

```
00001
00009 #ifndef QEVERCLOUD_EXPORT_H
00010 #define QEVERCLOUD_EXPORT_H
00011
00012 #include <QtCore/QtGlobal>
00013
00014 #if defined(QEVERCLOUD_SHARED_LIBRARY)
00015 #  define QEVERCLOUD_EXPORT Q_DECL_EXPORT
00016 #elif defined(QEVERCLOUD_STATIC_LIBRARY)
00017 #  define QEVERCLOUD_EXPORT Q_DECL_EXPORT
00018 #else
00019 #  define QEVERCLOUD_EXPORT Q_DECL_IMPORT
00020 #endif
00021
00022 #endif // QEVERCLOUD_EXPORT_H
```

8.13 Constants.h File Reference

```
#include "../Export.h"
```

Namespaces

- namespace qevercloud

Variables

- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_ATTRIBUTE_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_ATTRIBUTE_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_ATTRIBUTE_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_ATTRIBUTE_LIST_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_ATTRIBUTE_MAP_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_GUID_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_GUID_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_GUID_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_EMAIL_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_EMAIL_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_EMAIL_LOCAL_REGEX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_EMAIL_DOMAIN_REGEX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_EMAIL_REGEX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_VAT_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_TIMEZONE_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_TIMEZONE_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_TIMEZONE_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MIME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MIME_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_REGEX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_GIF
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_JPEG
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_PNG
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_TIFF
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_BMP
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_WAV
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_MP3
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_AMR
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_AAC
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_M4A
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_MP4_VIDEO
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_INK
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_PDF
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MIME_TYPE_DEFAULT
- QEVERCLOUD_EXPORT const QSet< QString > qevercloud::EDAM_MIME_TYPES
- QEVERCLOUD_EXPORT const QSet< QString > qevercloud::EDAM_INDEXABLE_RESOURCE_MIME_TYPES
- QEVERCLOUD_EXPORT const QSet< QString > qevercloud::EDAM_INDEXABLE_PLAINTEXT_MIME_TYPES
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_SEARCH_QUERY_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_SEARCH_QUERY_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SEARCH_QUERY_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_HASH_LEN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_USERNAME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_USERNAME_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_USER_USERNAME_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_NAME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_NAME_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_USER_NAME_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_TAG_NAME_LEN_MIN

- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_TAG_NAME_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_TAG_NAME_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_TITLE_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_TITLE_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_NOTE_TITLE_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_CONTENT_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_CONTENT_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_APPLICATIONDATA_NAME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_APPLICATIONDATA_NAME_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_APPLICATIONDATA_VALUE_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_APPLICATIONDATA_VALUE_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_APPLICATIONDATA_ENTRY_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_APPLICATIONDATA_NAME_REGEX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_APPLICATIONDATA_VALUE_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTEBOOK_NAME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTEBOOK_NAME_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_NOTEBOOK_NAME_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTEBOOK_STACK_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTEBOOK_STACK_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_NOTEBOOK_STACK_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_WORKSPACE_NAME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_WORKSPACE_NAME_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_WORKSPACE_DESCRIPTION_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_WORKSPACE_NAME_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PUBLISHING_URI_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PUBLISHING_URI_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PUBLISHING_URI_REGEX
- QEVERCLOUD_EXPORT const QSet< QString > qevercloud::EDAM_PUBLISHING_URI_PROHIBITED
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PUBLISHING_DESCRIPTION_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PUBLISHING_DESCRIPTION_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PUBLISHING_DESCRIPTION_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_SAVED_SEARCH_NAME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_SAVED_SEARCH_NAME_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SAVED_SEARCH_NAME_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_PASSWORD_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_PASSWORD_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_USER_PASSWORD_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_BUSINESS_URI_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_BUSINESS_MARKETING_CODE_REGEX_PATTERN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_TAGS_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_RESOURCES_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_TAGS_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_BUSINESS_TAGS_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_SAVED_SEARCHES_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_NOTES_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_BUSINESS_NOTES_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_NOTEBOOKS_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_WORKSPACES_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_BUSINESS_NOTEBOOKS_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_BUSINESS_WORKSPACES_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_RECENT_MAILED_ADDRESSES_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_MAIL_LIMIT_DAILY_FREE
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_MAIL_LIMIT_DAILY_PREMIUM
- QEVERCLOUD_EXPORT const qint64 qevercloud::EDAM_USER_UPLOAD_LIMIT_FREE
- QEVERCLOUD_EXPORT const qint64 qevercloud::EDAM_USER_UPLOAD_LIMIT_PREMIUM

- QEVERCLOUD_EXPORT const qint64 qevercloud::EDAM_USER_UPLOAD_LIMIT_BUSINESS_FIRST_MONTH
- QEVERCLOUD_EXPORT const qint64 qevercloud::EDAM_USER_UPLOAD_LIMIT_BUSINESS_NEXT_MONTH
- QEVERCLOUD_EXPORT const qint64 qevercloud::EDAM_USER_UPLOAD_LIMIT_PLUS
- QEVERCLOUD_EXPORT const qint64 qevercloud::EDAM_USER_UPLOAD_SURVEY_THRESHOLD
- QEVERCLOUD_EXPORT const qint64 qevercloud::EDAM_USER_UPLOAD_LIMIT_BUSINESS
- QEVERCLOUD_EXPORT const qint64 qevercloud::EDAM_USER_UPLOAD_LIMIT_BUSINESS_PER_USER
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_SIZE_MAX_FREE
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_SIZE_MAX_PREMIUM
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RESOURCE_SIZE_MAX_FREE
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RESOURCE_SIZE_MAX_PREMIUM
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_LINKED_NOTEBOOK_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_LINKED_NOTEBOOK_MAX_PREMIUM
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTEBOOK_BUSINESS_SHARED_NOTEBOOK_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTEBOOK_PERSONAL_SHARED_NOTEBOOK_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_BUSINESS_SHARED_NOTE_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_PERSONAL_SHARED_NOTE_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_CONTENT_CLASS_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_CONTENT_CLASS_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_NOTE_CONTENT_CLASS_REGEX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_HELLO_APP_CONTENT_CLASS_PREFIX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_FOOD_APP_CONTENT_CLASS_PREFIX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_CONTENT_CLASS_HELLO_ENCOUNTER
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_CONTENT_CLASS_HELLO_PROFILE
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_CONTENT_CLASS_FOOD_MEAL
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_CONTENT_CLASS_SKITCH_PREFIX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_CONTENT_CLASS_SKITCH
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_CONTENT_CLASS_SKITCH_PDF
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_CONTENT_CLASS_PENULTIMATE_PREFIX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_CONTENT_CLASS_PENULTIMATE_NOTEBOOK
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SOURCE_APPLICATION_POSTIT
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SOURCE_APPLICATION_MOLESKINE
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SOURCE_APPLICATION_EN_SCANSNAP
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SOURCE_APPLICATION_EWC
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SOURCE_APPLICATION_ANDROID_SHARE_EXTENSION
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SOURCE_APPLICATION_IOS_SHARE_EXTENSION
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SOURCE_APPLICATION_WEB_CLIPPER
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_SOURCE_OUTLOOK_CLIPPER
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_TITLE_QUALITY_UNTITLED
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_TITLE_QUALITY_LOW
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_TITLE_QUALITY_MEDIUM
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_TITLE_QUALITY_HIGH
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RELATED_PLAINTEXT_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RELATED_PLAINTEXT_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RELATED_MAX_NOTES
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RELATED_MAX_NOTEBOOKS
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RELATED_MAX_TAGS
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RELATED_MAX_EXPERTS
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_RELATED_MAX_RELATED_CONTENT
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_BUSINESS_PHONE_NUMBER_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PREFERENCE_NAME_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PREFERENCE_NAME_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PREFERENCE_VALUE_LEN_MIN

- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PREFERENCE_VALUE_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MAX_PREFERENCES
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MAX_VALUES_PER_PREFERENCE
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PREFERENCE_ONLY_ONE_VALUE_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PREFERENCE_NAME_REGEX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PREFERENCE_VALUE_REGEX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PREFERENCE_ONLY_ONE_VALUE_REGEX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PREFERENCE_SHORTCUTS
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PREFERENCE_BUSINESS_DEFAULT_NOTEBOOK
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PREFERENCE_BUSINESS_QUICKNOTE
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PREFERENCE_SHORTCUTS_MAX_VALUES
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_DEVICE_ID_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_DEVICE_ID_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_DEVICE_DESCRIPTION_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_DEVICE_DESCRIPTION_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_SEARCH_SUGGESTIONS_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_SEARCH_SUGGESTIONS_PREFIX_LEN_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_SEARCH_SUGGESTIONS_PREFIX_LEN_MIN
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_FIND_CONTACT_DEFAULT_MAX_RESULTS
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_FIND_CONTACT_MAX_RESULTS
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_NOTE_LOCK_VIEWERS_NOTES_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_GET_ORDERS_MAX_RESULTS
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MESSAGE_BODY_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MESSAGE_BODY_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MESSAGE_RECIPIENTS_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MESSAGE_ATTACHMENTS_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MESSAGE_ATTACHMENT_TITLE_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MESSAGE_ATTACHMENT_TITLE_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_MESSAGE_ATTACHMENT_SNIPPET_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_MESSAGE_ATTACHMENT_SNIPPET_REGEX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_USER_PROFILE_PHOTO_MAX_BYTES
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_PROMOTION_ID_LEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_PROMOTION_ID_REGEX
- QEVERCLOUD_EXPORT const qint16 qevercloud::EDAM_APP_RATING_MIN
- QEVERCLOUD_EXPORT const qint16 qevercloud::EDAM_APP_RATING_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_SNIPPETS_NOTES_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_CONNECTED_IDENTITY_REQUEST_MAX
- QEVERCLOUD_EXPORT const qint32 qevercloud::EDAM_OPEN_ID_ACCESS_TOKEN_MAX
- QEVERCLOUD_EXPORT const QString qevercloud::CLASSIFICATION_RECIPE_USER_NON_RECIPE
- QEVERCLOUD_EXPORT const QString qevercloud::CLASSIFICATION_RECIPE_USER_RECIPE
- QEVERCLOUD_EXPORT const QString qevercloud::CLASSIFICATION_RECIPE_SERVICE_RECIPE
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_NOTE_SOURCE_WEB_CLIP
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_NOTE_SOURCE_WEB_CLIP_SIMPLIFIED
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_NOTE_SOURCE_MAIL_CLIP
- QEVERCLOUD_EXPORT const QString qevercloud::EDAM_NOTE_SOURCE_MAIL_SMTP_GATEWAY
- QEVERCLOUD_EXPORT const qint16 qevercloud::EDAM_VERSION_MAJOR
- QEVERCLOUD_EXPORT const qint16 qevercloud::EDAM_VERSION_MINOR

8.14 Constants.h

Go to the documentation of this file.

```
00001
00012 #ifndef QEVERCLOUD_GENERATED_CONSTANTS_H
00013 #define QEVERCLOUD_GENERATED_CONSTANTS_H
00014
00015 #include "../Export.h"
00016
00017 namespace qevercloud {
00018
00020
00021 // Limits.thrift
00025 QEVERCLOUD_EXPORT extern const qint32 EDAM_ATTRIBUTE_LEN_MIN;
00026
00027 // Limits.thrift
00031 QEVERCLOUD_EXPORT extern const qint32 EDAM_ATTRIBUTE_LEN_MAX;
00032
00033 // Limits.thrift
00038 QEVERCLOUD_EXPORT extern const QString EDAM_ATTRIBUTE_REGEX;
00039
00040 // Limits.thrift
00045 QEVERCLOUD_EXPORT extern const qint32 EDAM_ATTRIBUTE_LIST_MAX;
00046
00047 // Limits.thrift
00052 QEVERCLOUD_EXPORT extern const qint32 EDAM_ATTRIBUTE_MAP_MAX;
00053
00054 // Limits.thrift
00058 QEVERCLOUD_EXPORT extern const qint32 EDAM_GUID_LEN_MIN;
00059
00060 // Limits.thrift
00064 QEVERCLOUD_EXPORT extern const qint32 EDAM_GUID_LEN_MAX;
00065
00066 // Limits.thrift
00070 QEVERCLOUD_EXPORT extern const QString EDAM_GUID_REGEX;
00071
00072 // Limits.thrift
00076 QEVERCLOUD_EXPORT extern const qint32 EDAM_EMAIL_LEN_MIN;
00077
00078 // Limits.thrift
00082 QEVERCLOUD_EXPORT extern const qint32 EDAM_EMAIL_LEN_MAX;
00083
00084 // Limits.thrift
00089 QEVERCLOUD_EXPORT extern const QString EDAM_EMAIL_LOCAL_REGEX;
00090
00091 // Limits.thrift
00096 QEVERCLOUD_EXPORT extern const QString EDAM_EMAIL_DOMAIN_REGEX;
00097
00098 // Limits.thrift
00103 QEVERCLOUD_EXPORT extern const QString EDAM_EMAIL_REGEX;
00104
00105 // Limits.thrift
00111 QEVERCLOUD_EXPORT extern const QString EDAM_VAT_REGEX;
00112
00113 // Limits.thrift
00117 QEVERCLOUD_EXPORT extern const qint32 EDAM_TIMEZONE_LEN_MIN;
00118
00119 // Limits.thrift
00123 QEVERCLOUD_EXPORT extern const qint32 EDAM_TIMEZONE_LEN_MAX;
00124
00125 // Limits.thrift
00134 QEVERCLOUD_EXPORT extern const QString EDAM_TIMEZONE_REGEX;
00135
00136 // Limits.thrift
00140 QEVERCLOUD_EXPORT extern const qint32 EDAM_MIME_LEN_MIN;
00141
00142 // Limits.thrift
00146 QEVERCLOUD_EXPORT extern const qint32 EDAM_MIME_LEN_MAX;
00147
00148 // Limits.thrift
00153 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_REGEX;
00154
00155 // Limits.thrift
00157 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_GIF;
00158
00159 // Limits.thrift
00161 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_JPEG;
00162
00163 // Limits.thrift
00165 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_PNG;
00166
00167 // Limits.thrift
00169 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_TIFF;
00170
00171 // Limits.thrift
```

```
00173 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_BMP;
00174
00175 // Limits.thrift
00177 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_WAV;
00178
00179 // Limits.thrift
00181 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_MP3;
00182
00183 // Limits.thrift
00185 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_AMR;
00186
00187 // Limits.thrift
00189 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_AAC;
00190
00191 // Limits.thrift
00193 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_M4A;
00194
00195 // Limits.thrift
00197 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_MP4_VIDEO;
00198
00199 // Limits.thrift
00201 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_INK;
00202
00203 // Limits.thrift
00205 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_PDF;
00206
00207 // Limits.thrift
00209 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_DEFAULT;
00210
00211 // Limits.thrift
00216 QEVERCLOUD_EXPORT extern const QSet<QString> EDAM_MIME_TYPES;
00217
00218 // Limits.thrift
00224 QEVERCLOUD_EXPORT extern const QSet<QString> EDAM_INDEXABLE_RESOURCE_MIME_TYPES;
00225
00226 // Limits.thrift
00232 QEVERCLOUD_EXPORT extern const QSet<QString> EDAM_INDEXABLE_PLAINTEXT_MIME_TYPES;
00233
00234 // Limits.thrift
00238 QEVERCLOUD_EXPORT extern const qint32 EDAM_SEARCH_QUERY_LEN_MIN;
00239
00240 // Limits.thrift
00244 QEVERCLOUD_EXPORT extern const qint32 EDAM_SEARCH_QUERY_LEN_MAX;
00245
00246 // Limits.thrift
00252 QEVERCLOUD_EXPORT extern const QString EDAM_SEARCH_QUERY_REGEX;
00253
00254 // Limits.thrift
00260 QEVERCLOUD_EXPORT extern const qint32 EDAM_HASH_LEN;
00261
00262 // Limits.thrift
00266 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_USERNAME_LEN_MIN;
00267
00268 // Limits.thrift
00272 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_USERNAME_LEN_MAX;
00273
00274 // Limits.thrift
00280 QEVERCLOUD_EXPORT extern const QString EDAM_USER_USERNAME_REGEX;
00281
00282 // Limits.thrift
00286 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_NAME_LEN_MIN;
00287
00288 // Limits.thrift
00292 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_NAME_LEN_MAX;
00293
00294 // Limits.thrift
00299 QEVERCLOUD_EXPORT extern const QString EDAM_USER_NAME_REGEX;
00300
00301 // Limits.thrift
00305 QEVERCLOUD_EXPORT extern const qint32 EDAM_TAG_NAME_LEN_MIN;
00306
00307 // Limits.thrift
00311 QEVERCLOUD_EXPORT extern const qint32 EDAM_TAG_NAME_LEN_MAX;
00312
00313 // Limits.thrift
00319 QEVERCLOUD_EXPORT extern const QString EDAM_TAG_NAME_REGEX;
00320
00321 // Limits.thrift
00325 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_TITLE_LEN_MIN;
00326
00327 // Limits.thrift
00331 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_TITLE_LEN_MAX;
00332
00333 // Limits.thrift
00339 QEVERCLOUD_EXPORT extern const QString EDAM_NOTE_TITLE_REGEX;
00340
00341 // Limits.thrift
```

```
00346 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_CONTENT_LEN_MIN;
00347
00348 // Limits.thrift
00353 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_CONTENT_LEN_MAX;
00354
00355 // Limits.thrift
00361 QEVERCLOUD_EXPORT extern const qint32 EDAM_APPLICATIONDATA_NAME_LEN_MIN;
00362
00363 // Limits.thrift
00369 QEVERCLOUD_EXPORT extern const qint32 EDAM_APPLICATIONDATA_NAME_LEN_MAX;
00370
00371 // Limits.thrift
00376 QEVERCLOUD_EXPORT extern const qint32 EDAM_APPLICATIONDATA_VALUE_LEN_MIN;
00377
00378 // Limits.thrift
00386 QEVERCLOUD_EXPORT extern const qint32 EDAM_APPLICATIONDATA_VALUE_LEN_MAX;
00387
00388 // Limits.thrift
00393 QEVERCLOUD_EXPORT extern const qint32 EDAM_APPLICATIONDATA_ENTRY_LEN_MAX;
00394
00395 // Limits.thrift
00404 QEVERCLOUD_EXPORT extern const QString EDAM_APPLICATIONDATA_NAME_REGEX;
00405
00406 // Limits.thrift
00413 QEVERCLOUD_EXPORT extern const QString EDAM_APPLICATIONDATA_VALUE_REGEX;
00414
00415 // Limits.thrift
00419 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTEBOOK_NAME_LEN_MIN;
00420
00421 // Limits.thrift
00425 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTEBOOK_NAME_LEN_MAX;
00426
00427 // Limits.thrift
00433 QEVERCLOUD_EXPORT extern const QString EDAM_NOTEBOOK_NAME_REGEX;
00434
00435 // Limits.thrift
00439 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTEBOOK_STACK_LEN_MIN;
00440
00441 // Limits.thrift
00445 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTEBOOK_STACK_LEN_MAX;
00446
00447 // Limits.thrift
00453 QEVERCLOUD_EXPORT extern const QString EDAM_NOTEBOOK_STACK_REGEX;
00454
00455 // Limits.thrift
00459 QEVERCLOUD_EXPORT extern const qint32 EDAM_WORKSPACE_NAME_LEN_MIN;
00460
00461 // Limits.thrift
00465 QEVERCLOUD_EXPORT extern const qint32 EDAM_WORKSPACE_NAME_LEN_MAX;
00466
00467 // Limits.thrift
00471 QEVERCLOUD_EXPORT extern const qint32 EDAM_WORKSPACE_DESCRIPTION_LEN_MAX;
00472
00473 // Limits.thrift
00479 QEVERCLOUD_EXPORT extern const QString EDAM_WORKSPACE_NAME_REGEX;
00480
00481 // Limits.thrift
00485 QEVERCLOUD_EXPORT extern const qint32 EDAM_PUBLISHING_URI_LEN_MIN;
00486
00487 // Limits.thrift
00491 QEVERCLOUD_EXPORT extern const qint32 EDAM_PUBLISHING_URI_LEN_MAX;
00492
00493 // Limits.thrift
00497 QEVERCLOUD_EXPORT extern const QString EDAM_PUBLISHING_URI_REGEX;
00498
00499 // Limits.thrift
00503 QEVERCLOUD_EXPORT extern const QSet<QString> EDAM_PUBLISHING_URI_PROHIBITED;
00504
00505 // Limits.thrift
00509 QEVERCLOUD_EXPORT extern const qint32 EDAM_PUBLISHING_DESCRIPTION_LEN_MIN;
00510
00511 // Limits.thrift
00515 QEVERCLOUD_EXPORT extern const qint32 EDAM_PUBLISHING_DESCRIPTION_LEN_MAX;
00516
00517 // Limits.thrift
00524 QEVERCLOUD_EXPORT extern const QString EDAM_PUBLISHING_DESCRIPTION_REGEX;
00525
00526 // Limits.thrift
00530 QEVERCLOUD_EXPORT extern const qint32 EDAM_SAVED_SEARCH_NAME_LEN_MIN;
00531
00532 // Limits.thrift
00536 QEVERCLOUD_EXPORT extern const qint32 EDAM_SAVED_SEARCH_NAME_LEN_MAX;
00537
00538 // Limits.thrift
00544 QEVERCLOUD_EXPORT extern const QString EDAM_SAVED_SEARCH_NAME_REGEX;
00545
00546 // Limits.thrift
```

```
00550 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_PASSWORD_LEN_MIN;
00551
00552 // Limits.thrift
00556 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_PASSWORD_LEN_MAX;
00557
00558 // Limits.thrift
00562 QEVERCLOUD_EXPORT extern const QString EDAM_USER_PASSWORD_REGEX;
00563
00564 // Limits.thrift
00568 QEVERCLOUD_EXPORT extern const qint32 EDAM_BUSINESS_URI_LEN_MAX;
00569
00570 // Limits.thrift
00574 QEVERCLOUD_EXPORT extern const QString EDAM_BUSINESS_MARKETING_CODE_REGEX_PATTERN;
00575
00576 // Limits.thrift
00580 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_TAGS_MAX;
00581
00582 // Limits.thrift
00586 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_RESOURCES_MAX;
00587
00588 // Limits.thrift
00592 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_TAGS_MAX;
00593
00594 // Limits.thrift
00598 QEVERCLOUD_EXPORT extern const qint32 EDAM_BUSINESS_TAGS_MAX;
00599
00600 // Limits.thrift
00604 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_SAVED_SEARCHES_MAX;
00605
00606 // Limits.thrift
00610 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_NOTES_MAX;
00611
00612 // Limits.thrift
00616 QEVERCLOUD_EXPORT extern const qint32 EDAM_BUSINESS_NOTES_MAX;
00617
00618 // Limits.thrift
00622 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_NOTEBOOKS_MAX;
00623
00624 // Limits.thrift
00628 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_WORKSPACES_MAX;
00629
00630 // Limits.thrift
00634 QEVERCLOUD_EXPORT extern const qint32 EDAM_BUSINESS_NOTEBOOKS_MAX;
00635
00636 // Limits.thrift
00640 QEVERCLOUD_EXPORT extern const qint32 EDAM_BUSINESS_WORKSPACES_MAX;
00641
00642 // Limits.thrift
00647 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_RECENT_MAILED_ADDRESSES_MAX;
00648
00649 // Limits.thrift
00655 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_MAIL_LIMIT_DAILY_FREE;
00656
00657 // Limits.thrift
00663 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_MAIL_LIMIT_DAILY_PREMIUM;
00664
00665 // Limits.thrift
00670 QEVERCLOUD_EXPORT extern const qint64 EDAM_USER_UPLOAD_LIMIT_FREE;
00671
00672 // Limits.thrift
00677 QEVERCLOUD_EXPORT extern const qint64 EDAM_USER_UPLOAD_LIMIT_PREMIUM;
00678
00679 // Limits.thrift
00684 QEVERCLOUD_EXPORT extern const qint64 EDAM_USER_UPLOAD_LIMIT_BUSINESS_FIRST_MONTH;
00685
00686 // Limits.thrift
00691 QEVERCLOUD_EXPORT extern const qint64 EDAM_USER_UPLOAD_LIMIT_BUSINESS_NEXT_MONTH;
00692
00693 // Limits.thrift
00698 QEVERCLOUD_EXPORT extern const qint64 EDAM_USER_UPLOAD_LIMIT_PLUS;
00699
00700 // Limits.thrift
00706 QEVERCLOUD_EXPORT extern const qint64 EDAM_USER_UPLOAD_SURVEY_THRESHOLD;
00707
00708 // Limits.thrift
00714 QEVERCLOUD_EXPORT extern const qint64 EDAM_USER_UPLOAD_LIMIT_BUSINESS;
00715
00716 // Limits.thrift
00722 QEVERCLOUD_EXPORT extern const qint64 EDAM_USER_UPLOAD_LIMIT_BUSINESS_PER_USER;
00723
00724 // Limits.thrift
00731 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_SIZE_MAX_FREE;
00732
00733 // Limits.thrift
00740 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_SIZE_MAX_PREMIUM;
00741
00742 // Limits.thrift
```

```
00746 QEVERCLOUD_EXPORT extern const qint32 EDAM_RESOURCE_SIZE_MAX_FREE;
00747
00748 // Limits.thrift
00752 QEVERCLOUD_EXPORT extern const qint32 EDAM_RESOURCE_SIZE_MAX_PREMIUM;
00753
00754 // Limits.thrift
00759 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_LINKED_NOTEBOOK_MAX;
00760
00761 // Limits.thrift
00767 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_LINKED_NOTEBOOK_MAX_PREMIUM;
00768
00769 // Limits.thrift
00773 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTEBOOK_BUSINESS_SHARED_NOTEBOOK_MAX;
00774
00775 // Limits.thrift
00779 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTEBOOK_PERSONAL_SHARED_NOTEBOOK_MAX;
00780
00781 // Limits.thrift
00785 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_BUSINESS_SHARED_NOTE_MAX;
00786
00787 // Limits.thrift
00791 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_PERSONAL_SHARED_NOTE_MAX;
00792
00793 // Limits.thrift
00797 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_CONTENT_CLASS_LEN_MIN;
00798
00799 // Limits.thrift
00803 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_CONTENT_CLASS_LEN_MAX;
00804
00805 // Limits.thrift
00810 QEVERCLOUD_EXPORT extern const QString EDAM_NOTE_CONTENT_CLASS_REGEX;
00811
00812 // Limits.thrift
00820 QEVERCLOUD_EXPORT extern const QString EDAM_HELLO_APP_CONTENT_CLASS_PREFIX;
00821
00822 // Limits.thrift
00830 QEVERCLOUD_EXPORT extern const QString EDAM_FOOD_APP_CONTENT_CLASS_PREFIX;
00831
00832 // Limits.thrift
00839 QEVERCLOUD_EXPORT extern const QString EDAM_CONTENT_CLASS_HELLO_ENCOUNTER;
00840
00841 // Limits.thrift
00848 QEVERCLOUD_EXPORT extern const QString EDAM_CONTENT_CLASS_HELLO_PROFILE;
00849
00850 // Limits.thrift
00857 QEVERCLOUD_EXPORT extern const QString EDAM_CONTENT_CLASS_FOOD_MEAL;
00858
00859 // Limits.thrift
00865 QEVERCLOUD_EXPORT extern const QString EDAM_CONTENT_CLASS_SKITCH_PREFIX;
00866
00867 // Limits.thrift
00872 QEVERCLOUD_EXPORT extern const QString EDAM_CONTENT_CLASS_SKITCH;
00873
00874 // Limits.thrift
00879 QEVERCLOUD_EXPORT extern const QString EDAM_CONTENT_CLASS_SKITCH_PDF;
00880
00881 // Limits.thrift
00887 QEVERCLOUD_EXPORT extern const QString EDAM_CONTENT_CLASS_PENULTIMATE_PREFIX;
00888
00889 // Limits.thrift
00894 QEVERCLOUD_EXPORT extern const QString EDAM_CONTENT_CLASS_PENULTIMATE_NOTEBOOK;
00895
00896 // Limits.thrift
00901 QEVERCLOUD_EXPORT extern const QString EDAM_SOURCE_APPLICATION_POSTIT;
00902
00903 // Limits.thrift
00908 QEVERCLOUD_EXPORT extern const QString EDAM_SOURCE_APPLICATION_MOLESKINE;
00909
00910 // Limits.thrift
00915 QEVERCLOUD_EXPORT extern const QString EDAM_SOURCE_APPLICATION_EN_SCANSNAP;
00916
00917 // Limits.thrift
00922 QEVERCLOUD_EXPORT extern const QString EDAM_SOURCE_APPLICATION_EWC;
00923
00924 // Limits.thrift
00929 QEVERCLOUD_EXPORT extern const QString EDAM_SOURCE_APPLICATION_ANDROID_SHARE_EXTENSION;
00930
00931 // Limits.thrift
00936 QEVERCLOUD_EXPORT extern const QString EDAM_SOURCE_APPLICATION_IOS_SHARE_EXTENSION;
00937
00938 // Limits.thrift
00943 QEVERCLOUD_EXPORT extern const QString EDAM_SOURCE_APPLICATION_WEB_CLIPPER;
00944
00945 // Limits.thrift
00949 QEVERCLOUD_EXPORT extern const QString EDAM_SOURCE_OUTLOOK_CLIPPER;
00950
00951 // Limits.thrift
```

```
00956 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_TITLE_QUALITY_UNTITLED;
00957
00958 // Limits.thrift
00964 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_TITLE_QUALITY_LOW;
00965
00966 // Limits.thrift
00972 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_TITLE_QUALITY_MEDIUM;
00973
00974 // Limits.thrift
00980 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_TITLE_QUALITY_HIGH;
00981
00982 // Limits.thrift
00987 QEVERCLOUD_EXPORT extern const qint32 EDAM_RELATED_PLAINTEXT_LEN_MIN;
00988
00989 // Limits.thrift
00994 QEVERCLOUD_EXPORT extern const qint32 EDAM_RELATED_PLAINTEXT_LEN_MAX;
00995
00996 // Limits.thrift
01001 QEVERCLOUD_EXPORT extern const qint32 EDAM_RELATED_MAX_NOTES;
01002
01003 // Limits.thrift
01008 QEVERCLOUD_EXPORT extern const qint32 EDAM_RELATED_MAX_NOTEBOOKS;
01009
01010 // Limits.thrift
01014 QEVERCLOUD_EXPORT extern const qint32 EDAM_RELATED_MAX_TAGS;
01015
01016 // Limits.thrift
01020 QEVERCLOUD_EXPORT extern const qint32 EDAM_RELATED_MAX_EXPERTS;
01021
01022 // Limits.thrift
01027 QEVERCLOUD_EXPORT extern const qint32 EDAM_RELATED_MAX_RELATED_CONTENT;
01028
01029 // Limits.thrift
01034 QEVERCLOUD_EXPORT extern const qint32 EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_LEN_MIN;
01035
01036 // Limits.thrift
01041 QEVERCLOUD_EXPORT extern const qint32 EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_LEN_MAX;
01042
01043 // Limits.thrift
01049 QEVERCLOUD_EXPORT extern const QString EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_REGEX;
01050
01051 // Limits.thrift
01055 QEVERCLOUD_EXPORT extern const qint32 EDAM_BUSINESS_PHONE_NUMBER_LEN_MAX;
01056
01057 // Limits.thrift
01061 QEVERCLOUD_EXPORT extern const qint32 EDAM_PREFERENCE_NAME_LEN_MIN;
01062
01063 // Limits.thrift
01067 QEVERCLOUD_EXPORT extern const qint32 EDAM_PREFERENCE_NAME_LEN_MAX;
01068
01069 // Limits.thrift
01073 QEVERCLOUD_EXPORT extern const qint32 EDAM_PREFERENCE_VALUE_LEN_MIN;
01074
01075 // Limits.thrift
01079 QEVERCLOUD_EXPORT extern const qint32 EDAM_PREFERENCE_VALUE_LEN_MAX;
01080
01081 // Limits.thrift
01085 QEVERCLOUD_EXPORT extern const qint32 EDAM_MAX_PREFERENCES;
01086
01087 // Limits.thrift
01092 QEVERCLOUD_EXPORT extern const qint32 EDAM_MAX_VALUES_PER_PREFERENCE;
01093
01094 // Limits.thrift
01102 QEVERCLOUD_EXPORT extern const qint32 EDAM_PREFERENCE_ONLY_ONE_VALUE_LEN_MAX;
01103
01104 // Limits.thrift
01108 QEVERCLOUD_EXPORT extern const QString EDAM_PREFERENCE_NAME_REGEX;
01109
01110 // Limits.thrift
01115 QEVERCLOUD_EXPORT extern const QString EDAM_PREFERENCE_VALUE_REGEX;
01116
01117 // Limits.thrift
01122 QEVERCLOUD_EXPORT extern const QString EDAM_PREFERENCE_ONLY_ONE_VALUE_REGEX;
01123
01124 // Limits.thrift
01128 QEVERCLOUD_EXPORT extern const QString EDAM_PREFERENCE_SHORTCUTS;
01129
01130 // Limits.thrift
01141 QEVERCLOUD_EXPORT extern const QString EDAM_PREFERENCE_BUSINESS_DEFAULT_NOTEBOOK;
01142
01143 // Limits.thrift
01158 QEVERCLOUD_EXPORT extern const QString EDAM_PREFERENCE_BUSINESS_QUICKNOTE;
01159
01160 // Limits.thrift
01164 QEVERCLOUD_EXPORT extern const qint32 EDAM_PREFERENCE_SHORTCUTS_MAX_VALUES;
01165
01166 // Limits.thrift
```

```
01170 QEVERCLOUD_EXPORT extern const qint32 EDAM_DEVICE_ID_LEN_MAX;
01171
01172 // Limits.thrift
01176 QEVERCLOUD_EXPORT extern const QString EDAM_DEVICE_ID_REGEX;
01177
01178 // Limits.thrift
01182 QEVERCLOUD_EXPORT extern const qint32 EDAM_DEVICE_DESCRIPTION_LEN_MAX;
01183
01184 // Limits.thrift
01188 QEVERCLOUD_EXPORT extern const QString EDAM_DEVICE_DESCRIPTION_REGEX;
01189
01190 // Limits.thrift
01194 QEVERCLOUD_EXPORT extern const qint32 EDAM_SEARCH_SUGGESTIONS_MAX;
01195
01196 // Limits.thrift
01200 QEVERCLOUD_EXPORT extern const qint32 EDAM_SEARCH_SUGGESTIONS_PREFIX_LEN_MAX;
01201
01202 // Limits.thrift
01206 QEVERCLOUD_EXPORT extern const qint32 EDAM_SEARCH_SUGGESTIONS_PREFIX_LEN_MIN;
01207
01208 // Limits.thrift
01212 QEVERCLOUD_EXPORT extern const qint32 EDAM_FIND_CONTACT_DEFAULT_MAX_RESULTS;
01213
01214 // Limits.thrift
01218 QEVERCLOUD_EXPORT extern const qint32 EDAM_FIND_CONTACT_MAX_RESULTS;
01219
01220 // Limits.thrift
01225 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_LOCK_VIEWERS_NOTES_MAX;
01226
01227 // Limits.thrift
01231 QEVERCLOUD_EXPORT extern const qint32 EDAM_GET_ORDERS_MAX_RESULTS;
01232
01233 // Limits.thrift
01237 QEVERCLOUD_EXPORT extern const qint32 EDAM_MESSAGE_BODY_LEN_MAX;
01238
01239 // Limits.thrift
01243 QEVERCLOUD_EXPORT extern const QString EDAM_MESSAGE_BODY_REGEX;
01244
01245 // Limits.thrift
01249 QEVERCLOUD_EXPORT extern const qint32 EDAM_MESSAGE_RECIPIENTS_MAX;
01250
01251 // Limits.thrift
01255 QEVERCLOUD_EXPORT extern const qint32 EDAM_MESSAGE_ATTACHMENTS_MAX;
01256
01257 // Limits.thrift
01261 QEVERCLOUD_EXPORT extern const qint32 EDAM_MESSAGE_ATTACHMENT_TITLE_LEN_MAX;
01262
01263 // Limits.thrift
01267 QEVERCLOUD_EXPORT extern const QString EDAM_MESSAGE_ATTACHMENT_TITLE_REGEX;
01268
01269 // Limits.thrift
01273 QEVERCLOUD_EXPORT extern const qint32 EDAM_MESSAGE_ATTACHMENT_SNIPPET_LEN_MAX;
01274
01275 // Limits.thrift
01279 QEVERCLOUD_EXPORT extern const QString EDAM_MESSAGE_ATTACHMENT_SNIPPET_REGEX;
01280
01281 // Limits.thrift
01286 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_PROFILE_PHOTO_MAX_BYTES;
01287
01288 // Limits.thrift
01292 QEVERCLOUD_EXPORT extern const qint32 EDAM_PROMOTION_ID_LEN_MAX;
01293
01294 // Limits.thrift
01298 QEVERCLOUD_EXPORT extern const QString EDAM_PROMOTION_ID_REGEX;
01299
01300 // Limits.thrift
01302 QEVERCLOUD_EXPORT extern const qint16 EDAM_APP_RATING_MIN;
01303
01304 // Limits.thrift
01305 QEVERCLOUD_EXPORT extern const qint16 EDAM_APP_RATING_MAX;
01306
01307 // Limits.thrift
01311 QEVERCLOUD_EXPORT extern const qint32 EDAM_SNIPPETS_NOTES_MAX;
01312
01313 // Limits.thrift
01317 QEVERCLOUD_EXPORT extern const qint32 EDAM_CONNECTED_IDENTITY_REQUEST_MAX;
01318
01319 // Limits.thrift
01324 QEVERCLOUD_EXPORT extern const qint32 EDAM_OPEN_ID_ACCESS_TOKEN_MAX;
01325
01326 // Types.thrift
01331 QEVERCLOUD_EXPORT extern const QString CLASSIFICATION_RECIPE_USER_NON_RECIPE;
01332
01333 // Types.thrift
01338 QEVERCLOUD_EXPORT extern const QString CLASSIFICATION_RECIPE_USER_RECIPE;
01339
01340 // Types.thrift
```

```
01345 QEVERCLOUD_EXPORT extern const QString CLASSIFICATION_RECIPE_SERVICE_RECIPE;
01346
01347 // Types.thrift
01352 QEVERCLOUD_EXPORT extern const QString EDAM_NOTE_SOURCE_WEB_CLIP;
01353
01354 // Types.thrift
01359 QEVERCLOUD_EXPORT extern const QString EDAM_NOTE_SOURCE_WEB_CLIP_SIMPLIFIED;
01360
01361 // Types.thrift
01366 QEVERCLOUD_EXPORT extern const QString EDAM_NOTE_SOURCE_MAIL_CLIP;
01367
01368 // Types.thrift
01373 QEVERCLOUD_EXPORT extern const QString EDAM_NOTE_SOURCE_MAIL_SMTP_GATEWAY;
01374
01375 // UserStore.thrift
01381 QEVERCLOUD_EXPORT extern const qint16 EDAM_VERSION_MAJOR;
01382
01383 // UserStore.thrift
01389 QEVERCLOUD_EXPORT extern const qint16 EDAM_VERSION_MINOR;
01390
01391 } // namespace qevercloud
01392
01393 #endif // QEVERCLOUD_GENERATED_CONSTANTS_H
```

8.15 EDAMErrorCode.h File Reference

```
#include "../Export.h"
#include "../Helpers.h"
#include <QDebug>
#include <QMetaType>
#include <QTextStream>
```

Namespaces

- namespace qevercloud

Enumerations

- enum class qevercloud::EDAMErrorCode {
  qevercloud::UNKNOWN = 1 , qevercloud::BAD_DATA_FORMAT = 2 , qevercloud::PERMISSION_DENIED = 3 , qevercloud::INTERNAL_ERROR = 4 ,
  qevercloud::DATA_REQUIRED = 5 , qevercloud::LIMIT_REACHED = 6 , qevercloud::QUOTA_REACHED = 7 , qevercloud::INVALID_AUTH = 8 ,
  qevercloud::AUTH_EXPIRED = 9 , qevercloud::DATA_CONFLICT = 10 , qevercloud::ENML_VALIDATION = 11 , qevercloud::SHARD_UNAVAILABLE = 12 ,
  qevercloud::LEN_TOO_SHORT = 13 , qevercloud::LEN_TOO_LONG = 14 , qevercloud::TOO_FEW = 15 , qevercloud::TOO_MANY = 16 ,
  qevercloud::UNSUPPORTED_OPERATION = 17 , qevercloud::TAKEN_DOWN = 18 , qevercloud::RATE_LIMIT_REACHED = 19 , qevercloud::BUSINESS_SECURITY_LOGIN_REQUIRED = 20 ,
  qevercloud::DEVICE_LIMIT_REACHED = 21 , qevercloud::OPENID_ALREADY_TAKEN = 22 , qevercloud::INVALID_OPENID_ = 23 , qevercloud::USER_NOT_ASSOCIATED = 24 ,
  qevercloud::USER_NOT_REGISTERED = 25 , qevercloud::USER_ALREADY_ASSOCIATED = 26 ,
  qevercloud::ACCOUNT_CLEAR = 27 , qevercloud::SSO_AUTHENTICATION_REQUIRED = 28 }
- enum class qevercloud::EDAMInvalidContactReason { qevercloud::BAD_ADDRESS , qevercloud::DUPLICATE_CONTACT , qevercloud::NO_CONNECTION }
- enum class qevercloud::ShareRelationshipPrivilegeLevel { qevercloud::READ_NOTEBOOK = 0 ,
  qevercloud::READ_NOTEBOOK_PLUS_ACTIVITY = 10 , qevercloud::MODIFY_NOTEBOOK_PLUS_ACTIVITY = 20 , qevercloud::FULL_ACCESS = 30 }

- enum class qevercloud::PrivilegeLevel { qevercloud::NORMAL = 1 , qevercloud::PREMIUM = 3 , qevercloud::VIP = 5 , qevercloud::MANAGER = 7 , qevercloud::SUPPORT = 8 , qevercloud::ADMIN = 9 }
- enum class qevercloud::ServiceLevel { qevercloud::BASIC = 1 , qevercloud::PLUS = 2 , qevercloud::PREMIUM = 3 , qevercloud::BUSINESS = 4 }
- enum class qevercloud::QueryFormat { qevercloud::USER = 1 , qevercloud::SEXP = 2 }
- enum class qevercloud::NoteSortOrder { qevercloud::CREATED = 1 , qevercloud::UPDATED = 2 , qevercloud::RELEVANCE = 3 , qevercloud::UPDATE_SEQUENCE_NU = 4 , qevercloud::TITLE = 5 }
- enum class qevercloud::PremiumOrderStatus { qevercloud::NONE = 0 , qevercloud::PENDING = 1 , qevercloud::ACTIVE = 2 , qevercloud::FAILED = 3 , qevercloud::CANCELLATION_PENDING = 4 , qevercloud::CANCELED = 5 }
- enum class qevercloud::SharedNotebookPrivilegeLevel { qevercloud::READ_NOTEBOOK = 0 , qevercloud::MODIFY_NOTEBOOK_PLUS_ACTIVITY = 1 , qevercloud::READ_NOTEBOOK_PLUS_ACTIVITY = 2 , qevercloud::GROUP = 3 , qevercloud::FULL_ACCESS = 4 , qevercloud::BUSINESS_FULL_ACCESS = 5 }
- enum class qevercloud::SharedNotePrivilegeLevel { qevercloud::READ_NOTE = 0 , qevercloud::MODIFY_NOTE = 1 , qevercloud::FULL_ACCESS = 2 }
- enum class qevercloud::SponsoredGroupRole { qevercloud::GROUP_MEMBER = 1 , qevercloud::GROUP_ADMIN = 2 , qevercloud::GROUP_OWNER = 3 }
- enum class qevercloud::BusinessUserRole { qevercloud::ADMIN = 1 , qevercloud::NORMAL = 2 }
- enum class qevercloud::BusinessUserStatus { qevercloud::ACTIVE = 1 , qevercloud::DEACTIVATED = 2 }
- enum class qevercloud::SharedNotebookInstanceRestrictions { qevercloud::ASSIGNED = 1 , qevercloud::NO_SHARED_NOTE = 2 }
- enum class qevercloud::ReminderEmailConfig { qevercloud::DO_NOT_SEND = 1 , qevercloud::SEND_DAILY_EMAIL = 2 }
- enum class qevercloud::BusinessInvitationStatus { qevercloud::APPROVED = 0 , qevercloud::REQUESTED = 1 , qevercloud::REDEEMED = 2 }
- enum class qevercloud::ContactType { qevercloud::EVERNOTE = 1 , qevercloud::SMS = 2 , qevercloud::FACEBOOK = 3 , qevercloud::EMAIL = 4 , qevercloud::TWITTER = 5 , qevercloud::LINKEDIN = 6 }
- enum class qevercloud::EntityType { qevercloud::NOTE = 1 , qevercloud::NOTEBOOK = 2 , qevercloud::WORKSPACE = 3 }
- enum class qevercloud::RecipientStatus { qevercloud::NOT_IN_MY_LIST = 1 , qevercloud::IN_MY_LIST = 2 , qevercloud::IN_MY_LIST_AND_DEFAULT_NOTEBOOK = 3 }
- enum class qevercloud::CanMoveToContainerStatus { qevercloud::CAN_BE_MOVED = 1 , qevercloud::INSUFFICIENT_ENTITY = 2 , qevercloud::INSUFFICIENT_CONTAINER_PRIVILEGE = 3 }
- enum class qevercloud::RelatedContentType { qevercloud::NEWS_ARTICLE = 1 , qevercloud::PROFILE_PERSON = 2 , qevercloud::PROFILE_ORGANIZATION = 3 , qevercloud::REFERENCE_MATERIAL = 4 }
- enum class qevercloud::RelatedContentAccess { qevercloud::NOT_ACCESSIBLE = 0 , qevercloud::DIRECT_LINK_ACCESS_C = 1 , qevercloud::DIRECT_LINK_LOGIN_REQUIRED = 2 , qevercloud::DIRECT_LINK_EMBEDDED_VIEW = 3 }
- enum class qevercloud::UserIdentityType { qevercloud::EVERNOTE_USERID = 1 , qevercloud::EMAIL = 2 , qevercloud::IDENTITYID = 3 }

Functions

- uint qevercloud::qHash (EDAMErrorCode value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const EDAMErrorCode value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const EDAMErrorCode value)
- uint qevercloud::qHash (EDAMInvalidContactReason value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const EDAMInvalidContactReason value)

- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const EDAMInvalidContactReason value)
- uint qevercloud::qHash (ShareRelationshipPrivilegeLevel value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const ShareRelationshipPrivilegeLevel value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const ShareRelationshipPrivilegeLevel value)
- uint qevercloud::qHash (PrivilegeLevel value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const PrivilegeLevel value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const PrivilegeLevel value)
- uint qevercloud::qHash (ServiceLevel value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const ServiceLevel value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const ServiceLevel value)
- uint qevercloud::qHash (QueryFormat value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const QueryFormat value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const QueryFormat value)
- uint qevercloud::qHash (NoteSortOrder value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const NoteSortOrder value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const NoteSortOrder value)
- uint qevercloud::qHash (PremiumOrderStatus value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const PremiumOrderStatus value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const PremiumOrderStatus value)
- uint qevercloud::qHash (SharedNotebookPrivilegeLevel value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const SharedNotebookPrivilegeLevel value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const SharedNotebookPrivilegeLevel value)
- uint qevercloud::qHash (SharedNotePrivilegeLevel value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const SharedNotePrivilegeLevel value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const SharedNotePrivilegeLevel value)
- uint qevercloud::qHash (SponsoredGroupRole value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const SponsoredGroupRole value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const SponsoredGroupRole value)
- uint qevercloud::qHash (BusinessUserRole value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const BusinessUserRole value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const BusinessUserRole value)
- uint qevercloud::qHash (BusinessUserStatus value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const BusinessUserStatus value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const BusinessUserStatus value)
- uint qevercloud::qHash (SharedNotebookInstanceRestrictions value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const SharedNotebookInstanceRestric value)

- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const SharedNotebookInstanceRestrictions value)
- uint qevercloud::qHash (ReminderEmailConfig value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const ReminderEmailConfig value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const ReminderEmailConfig value)
- uint qevercloud::qHash (BusinessInvitationStatus value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const BusinessInvitationStatus value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const BusinessInvitationStatus value)
- uint qevercloud::qHash (ContactType value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const ContactType value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const ContactType value)
- uint qevercloud::qHash (EntityType value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const EntityType value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const EntityType value)
- uint qevercloud::qHash (RecipientStatus value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const RecipientStatus value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const RecipientStatus value)
- uint qevercloud::qHash (CanMoveToContainerStatus value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const CanMoveToContainerStatus value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const CanMoveToContainerStatus value)
- uint qevercloud::qHash (RelatedContentType value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const RelatedContentType value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const RelatedContentType value)
- uint qevercloud::qHash (RelatedContentAccess value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const RelatedContentAccess value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const RelatedContentAccess value)
- uint qevercloud::qHash (UserIdentityType value)
- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const UserIdentityType value)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const UserIdentityType value)

8.16 EDAMErrorCode.h

Go to the documentation of this file.

```
00001
00012 #ifndef QEVERCLOUD_GENERATED_EDAMERRORCODE_H
00013 #define QEVERCLOUD_GENERATED_EDAMERRORCODE_H
00014
00015 #include "../Export.h"
00016
00017 #include "../Helpers.h"
00018 #include <QDebug>
00019 #include <QMetaType>
00020 #include <QTextStream>
```

```
00021
00022 namespace qevercloud {
00023
00025
00026 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00027 #if QEVERCLOUD_USES_Q_NAMESPACE
00028 Q_NAMESPACE
00029 #endif
00030 #endif
00031
00033
00112 enum class EDAMErrorCode
00113 {
00114     UNKNOWN = 1,
00115     BAD_DATA_FORMAT = 2,
00116     PERMISSION_DENIED = 3,
00117     INTERNAL_ERROR = 4,
00118     DATA_REQUIRED = 5,
00119     LIMIT_REACHED = 6,
00120     QUOTA_REACHED = 7,
00121     INVALID_AUTH = 8,
00122     AUTH_EXPIRED = 9,
00123     DATA_CONFLICT = 10,
00124     ENML_VALIDATION = 11,
00125     SHARD_UNAVAILABLE = 12,
00126     LEN_TOO_SHORT = 13,
00127     LEN_TOO_LONG = 14,
00128     TOO_FEW = 15,
00129     TOO_MANY = 16,
00130     UNSUPPORTED_OPERATION = 17,
00131     TAKEN_DOWN = 18,
00132     RATE_LIMIT_REACHED = 19,
00133     BUSINESS_SECURITY_LOGIN_REQUIRED = 20,
00134     DEVICE_LIMIT_REACHED = 21,
00135     OPENID_ALREADY_TAKEN = 22,
00136     INVALID_OPENID_TOKEN = 23,
00137     USER_NOT_ASSOCIATED = 24,
00138     USER_NOT_REGISTERED = 25,
00139     USER_ALREADY_ASSOCIATED = 26,
00140     ACCOUNT_CLEAR = 27,
00141     SSO_AUTHENTICATION_REQUIRED = 28
00142 };
00143
00144 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00145 #if QEVERCLOUD_USES_Q_NAMESPACE
00146 Q_ENUM_NS(EDAMErrorCode)
00147 #endif
00148 #endif
00149
00150 inline uint qHash(EDAMErrorCode value)
00151 {
00152     return static_cast<uint>(value);
00153 }
00154
00156
00157 QEVERCLOUD_EXPORT QTextStream & operator<<(
00158     QTextStream & out, const EDAMErrorCode value);
00159
00161
00162 QEVERCLOUD_EXPORT QDebug & operator<<(
00163     QDebug & out, const EDAMErrorCode value);
00164
00166
00200 enum class EDAMInvalidContactReason
00201 {
00202     BAD_ADDRESS,
00203     DUPLICATE_CONTACT,
00204     NO_CONNECTION
00205 };
00206
00207 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00208 #if QEVERCLOUD_USES_Q_NAMESPACE
00209 Q_ENUM_NS(EDAMInvalidContactReason)
00210 #endif
00211 #endif
00212
00213 inline uint qHash(EDAMInvalidContactReason value)
00214 {
00215     return static_cast<uint>(value);
00216 }
00217
00219
00220 QEVERCLOUD_EXPORT QTextStream & operator<<(
00221     QTextStream & out, const EDAMInvalidContactReason value);
00222
00224
00225 QEVERCLOUD_EXPORT QDebug & operator<<(
```

```
00226     QDebug & out, const EDAMInvalidContactReason value);
00227
00229
00251 enum class ShareRelationshipPrivilegeLevel
00252 {
00253     READ_NOTEBOOK = 0,
00254     READ_NOTEBOOK_PLUS_ACTIVITY = 10,
00255     MODIFY_NOTEBOOK_PLUS_ACTIVITY = 20,
00256     FULL_ACCESS = 30
00257 };
00258
00259 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00260 #if QEVERCLOUD_USES_Q_NAMESPACE
00261 Q_ENUM_NS(ShareRelationshipPrivilegeLevel)
00262 #endif
00263 #endif
00264
00265 inline uint qHash(ShareRelationshipPrivilegeLevel value)
00266 {
00267     return static_cast<uint>(value);
00268 }
00269
00271
00272 QEVERCLOUD_EXPORT QTextStream & operator<<(
00273     QTextStream & out, const ShareRelationshipPrivilegeLevel value);
00274
00276
00277 QEVERCLOUD_EXPORT QDebug & operator<<(
00278     QDebug & out, const ShareRelationshipPrivilegeLevel value);
00279
00281
00287 enum class PrivilegeLevel
00288 {
00289     NORMAL = 1,
00290     PREMIUM = 3,
00291     VIP = 5,
00292     MANAGER = 7,
00293     SUPPORT = 8,
00294     ADMIN = 9
00295 };
00296
00297 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00298 #if QEVERCLOUD_USES_Q_NAMESPACE
00299 Q_ENUM_NS(PrivilegeLevel)
00300 #endif
00301 #endif
00302
00303 inline uint qHash(PrivilegeLevel value)
00304 {
00305     return static_cast<uint>(value);
00306 }
00307
00309
00310 QEVERCLOUD_EXPORT QTextStream & operator<<(
00311     QTextStream & out, const PrivilegeLevel value);
00312
00314
00315 QEVERCLOUD_EXPORT QDebug & operator<<(
00316     QDebug & out, const PrivilegeLevel value);
00317
00319
00325 enum class ServiceLevel
00326 {
00327     BASIC = 1,
00328     PLUS = 2,
00329     PREMIUM = 3,
00330     BUSINESS = 4
00331 };
00332
00333 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00334 #if QEVERCLOUD_USES_Q_NAMESPACE
00335 Q_ENUM_NS(ServiceLevel)
00336 #endif
00337 #endif
00338
00339 inline uint qHash(ServiceLevel value)
00340 {
00341     return static_cast<uint>(value);
00342 }
00343
00345
00346 QEVERCLOUD_EXPORT QTextStream & operator<<(
00347     QTextStream & out, const ServiceLevel value);
00348
00350
00351 QEVERCLOUD_EXPORT QDebug & operator<<(
00352     QDebug & out, const ServiceLevel value);
```

```
00353
00355
00360 enum class QueryFormat
00361 {
00362     USER = 1,
00363     SEXP = 2
00364 };
00365
00366 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00367 #if QEVERCLOUD_USES_Q_NAMESPACE
00368 Q_ENUM_NS(QueryFormat)
00369 #endif
00370 #endif
00371
00372 inline uint qHash(QueryFormat value)
00373 {
00374     return static_cast<uint>(value);
00375 }
00376
00378
00379 QEVERCLOUD_EXPORT QTextStream & operator<<(
00380     QTextStream & out, const QueryFormat value);
00381
00383
00384 QEVERCLOUD_EXPORT QDebug & operator<<(
00385     QDebug & out, const QueryFormat value);
00386
00388
00393 enum class NoteSortOrder
00394 {
00395     CREATED = 1,
00396     UPDATED = 2,
00397     RELEVANCE = 3,
00398     UPDATE_SEQUENCE_NUMBER = 4,
00399     TITLE = 5
00400 };
00401
00402 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00403 #if QEVERCLOUD_USES_Q_NAMESPACE
00404 Q_ENUM_NS(NoteSortOrder)
00405 #endif
00406 #endif
00407
00408 inline uint qHash(NoteSortOrder value)
00409 {
00410     return static_cast<uint>(value);
00411 }
00412
00414
00415 QEVERCLOUD_EXPORT QTextStream & operator<<(
00416     QTextStream & out, const NoteSortOrder value);
00417
00419
00420 QEVERCLOUD_EXPORT QDebug & operator<<(
00421     QDebug & out, const NoteSortOrder value);
00422
00424
00445 enum class PremiumOrderStatus
00446 {
00447     NONE = 0,
00448     PENDING = 1,
00449     ACTIVE = 2,
00450     FAILED = 3,
00451     CANCELLATION_PENDING = 4,
00452     CANCELED = 5
00453 };
00454
00455 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00456 #if QEVERCLOUD_USES_Q_NAMESPACE
00457 Q_ENUM_NS(PremiumOrderStatus)
00458 #endif
00459 #endif
00460
00461 inline uint qHash(PremiumOrderStatus value)
00462 {
00463     return static_cast<uint>(value);
00464 }
00465
00467
00468 QEVERCLOUD_EXPORT QTextStream & operator<<(
00469     QTextStream & out, const PremiumOrderStatus value);
00470
00472
00473 QEVERCLOUD_EXPORT QDebug & operator<<(
00474     QDebug & out, const PremiumOrderStatus value);
00475
00477
```

```
00513 enum class SharedNotebookPrivilegeLevel
00514 {
00515     READ_NOTEBOOK = 0,
00516     MODIFY_NOTEBOOK_PLUS_ACTIVITY = 1,
00517     READ_NOTEBOOK_PLUS_ACTIVITY = 2,
00518     GROUP = 3,
00519     FULL_ACCESS = 4,
00520     BUSINESS_FULL_ACCESS = 5
00521 };
00522
00523 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00524 #if QEVERCLOUD_USES_Q_NAMESPACE
00525 Q_ENUM_NS(SharedNotebookPrivilegeLevel)
00526 #endif
00527 #endif
00528
00529 inline uint qHash(SharedNotebookPrivilegeLevel value)
00530 {
00531     return static_cast<uint>(value);
00532 }
00533
00535
00536 QEVERCLOUD_EXPORT QTextStream & operator<<(
00537     QTextStream & out, const SharedNotebookPrivilegeLevel value);
00538
00540
00541 QEVERCLOUD_EXPORT QDebug & operator<<(
00542     QDebug & out, const SharedNotebookPrivilegeLevel value);
00543
00545
00560 enum class SharedNotePrivilegeLevel
00561 {
00562     READ_NOTE = 0,
00563     MODIFY_NOTE = 1,
00564     FULL_ACCESS = 2
00565 };
00566
00567 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00568 #if QEVERCLOUD_USES_Q_NAMESPACE
00569 Q_ENUM_NS(SharedNotePrivilegeLevel)
00570 #endif
00571 #endif
00572
00573 inline uint qHash(SharedNotePrivilegeLevel value)
00574 {
00575     return static_cast<uint>(value);
00576 }
00577
00579
00580 QEVERCLOUD_EXPORT QTextStream & operator<<(
00581     QTextStream & out, const SharedNotePrivilegeLevel value);
00582
00584
00585 QEVERCLOUD_EXPORT QDebug & operator<<(
00586     QDebug & out, const SharedNotePrivilegeLevel value);
00587
00589
00599 enum class SponsoredGroupRole
00600 {
00601     GROUP_MEMBER = 1,
00602     GROUP_ADMIN = 2,
00603     GROUP_OWNER = 3
00604 };
00605
00606 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00607 #if QEVERCLOUD_USES_Q_NAMESPACE
00608 Q_ENUM_NS(SponsoredGroupRole)
00609 #endif
00610 #endif
00611
00612 inline uint qHash(SponsoredGroupRole value)
00613 {
00614     return static_cast<uint>(value);
00615 }
00616
00618
00619 QEVERCLOUD_EXPORT QTextStream & operator<<(
00620     QTextStream & out, const SponsoredGroupRole value);
00621
00623
00624 QEVERCLOUD_EXPORT QDebug & operator<<(
00625     QDebug & out, const SponsoredGroupRole value);
00626
00628
00636 enum class BusinessUserRole
00637 {
00638     ADMIN = 1,
```

```
00639     NORMAL = 2
00640 };
00641
00642 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00643 #if QEVERCLOUD_USES_Q_NAMESPACE
00644 Q_ENUM_NS(BusinessUserRole)
00645 #endif
00646 #endif
00647
00648 inline uint qHash(BusinessUserRole value)
00649 {
00650     return static_cast<uint>(value);
00651 }
00652
00654
00655 QEVERCLOUD_EXPORT QTextStream & operator<<(
00656     QTextStream & out, const BusinessUserRole value);
00657
00659
00660 QEVERCLOUD_EXPORT QDebug & operator<<(
00661     QDebug & out, const BusinessUserRole value);
00662
00664
00678 enum class BusinessUserStatus
00679 {
00680     ACTIVE = 1,
00681     DEACTIVATED = 2
00682 };
00683
00684 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00685 #if QEVERCLOUD_USES_Q_NAMESPACE
00686 Q_ENUM_NS(BusinessUserStatus)
00687 #endif
00688 #endif
00689
00690 inline uint qHash(BusinessUserStatus value)
00691 {
00692     return static_cast<uint>(value);
00693 }
00694
00696
00697 QEVERCLOUD_EXPORT QTextStream & operator<<(
00698     QTextStream & out, const BusinessUserStatus value);
00699
00701
00702 QEVERCLOUD_EXPORT QDebug & operator<<(
00703     QDebug & out, const BusinessUserStatus value);
00704
00706
00717 enum class SharedNotebookInstanceRestrictions
00718 {
00719     ASSIGNED = 1,
00720     NO_SHARED_NOTEBOOKS = 2
00721 };
00722
00723 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00724 #if QEVERCLOUD_USES_Q_NAMESPACE
00725 Q_ENUM_NS(SharedNotebookInstanceRestrictions)
00726 #endif
00727 #endif
00728
00729 inline uint qHash(SharedNotebookInstanceRestrictions value)
00730 {
00731     return static_cast<uint>(value);
00732 }
00733
00735
00736 QEVERCLOUD_EXPORT QTextStream & operator<<(
00737     QTextStream & out, const SharedNotebookInstanceRestrictions value);
00738
00740
00741 QEVERCLOUD_EXPORT QDebug & operator<<(
00742     QDebug & out, const SharedNotebookInstanceRestrictions value);
00743
00745
00756 enum class ReminderEmailConfig
00757 {
00758     DO_NOT_SEND = 1,
00759     SEND_DAILY_EMAIL = 2
00760 };
00761
00762 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00763 #if QEVERCLOUD_USES_Q_NAMESPACE
00764 Q_ENUM_NS(ReminderEmailConfig)
00765 #endif
00766 #endif
00767
```

```
00768 inline uint qHash(ReminderEmailConfig value)
00769 {
00770     return static_cast<uint>(value);
00771 }
00772
00774
00775 QEVERCLOUD_EXPORT QTextStream & operator<<(
00776     QTextStream & out, const ReminderEmailConfig value);
00777
00779
00780 QEVERCLOUD_EXPORT QDebug & operator<<(
00781     QDebug & out, const ReminderEmailConfig value);
00782
00784
00798 enum class BusinessInvitationStatus
00799 {
00800     APPROVED = 0,
00801     REQUESTED = 1,
00802     REDEEMED = 2
00803 };
00804
00805 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00806 #if QEVERCLOUD_USES_Q_NAMESPACE
00807 Q_ENUM_NS(BusinessInvitationStatus)
00808 #endif
00809 #endif
00810
00811 inline uint qHash(BusinessInvitationStatus value)
00812 {
00813     return static_cast<uint>(value);
00814 }
00815
00817
00818 QEVERCLOUD_EXPORT QTextStream & operator<<(
00819     QTextStream & out, const BusinessInvitationStatus value);
00820
00822
00823 QEVERCLOUD_EXPORT QDebug & operator<<(
00824     QDebug & out, const BusinessInvitationStatus value);
00825
00827
00831 enum class ContactType
00832 {
00833     EVERNOTE = 1,
00834     SMS = 2,
00835     FACEBOOK = 3,
00836     EMAIL = 4,
00837     TWITTER = 5,
00838     LINKEDIN = 6
00839 };
00840
00841 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00842 #if QEVERCLOUD_USES_Q_NAMESPACE
00843 Q_ENUM_NS(ContactType)
00844 #endif
00845 #endif
00846
00847 inline uint qHash(ContactType value)
00848 {
00849     return static_cast<uint>(value);
00850 }
00851
00853
00854 QEVERCLOUD_EXPORT QTextStream & operator<<(
00855     QTextStream & out, const ContactType value);
00856
00858
00859 QEVERCLOUD_EXPORT QDebug & operator<<(
00860     QDebug & out, const ContactType value);
00861
00863
00867 enum class EntityType
00868 {
00869     NOTE = 1,
00870     NOTEBOOK = 2,
00871     WORKSPACE = 3
00872 };
00873
00874 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00875 #if QEVERCLOUD_USES_Q_NAMESPACE
00876 Q_ENUM_NS(EntityType)
00877 #endif
00878 #endif
00879
00880 inline uint qHash(EntityType value)
00881 {
00882     return static_cast<uint>(value);
```

```
00883 }
00884
00886
00887 QEVERCLOUD_EXPORT QTextStream & operator<<(
00888     QTextStream & out, const EntityType value);
00889
00891
00892 QEVERCLOUD_EXPORT QDebug & operator<<(
00893     QDebug & out, const EntityType value);
00894
00896
00911 enum class RecipientStatus
00912 {
00913     NOT_IN_MY_LIST = 1,
00914     IN_MY_LIST = 2,
00915     IN_MY_LIST_AND_DEFAULT_NOTEBOOK = 3
00916 };
00917
00918 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00919 #if QEVERCLOUD_USES_Q_NAMESPACE
00920 Q_ENUM_NS(RecipientStatus)
00921 #endif
00922 #endif
00923
00924 inline uint qHash(RecipientStatus value)
00925 {
00926     return static_cast<uint>(value);
00927 }
00928
00930
00931 QEVERCLOUD_EXPORT QTextStream & operator<<(
00932     QTextStream & out, const RecipientStatus value);
00933
00935
00936 QEVERCLOUD_EXPORT QDebug & operator<<(
00937     QDebug & out, const RecipientStatus value);
00938
00940
00959 enum class CanMoveToContainerStatus
00960 {
00961     CAN_BE_MOVED = 1,
00962     INSUFFICIENT_ENTITY_PRIVILEGE = 2,
00963     INSUFFICIENT_CONTAINER_PRIVILEGE = 3
00964 };
00965
00966 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
00967 #if QEVERCLOUD_USES_Q_NAMESPACE
00968 Q_ENUM_NS(CanMoveToContainerStatus)
00969 #endif
00970 #endif
00971
00972 inline uint qHash(CanMoveToContainerStatus value)
00973 {
00974     return static_cast<uint>(value);
00975 }
00976
00978
00979 QEVERCLOUD_EXPORT QTextStream & operator<<(
00980     QTextStream & out, const CanMoveToContainerStatus value);
00981
00983
00984 QEVERCLOUD_EXPORT QDebug & operator<<(
00985     QDebug & out, const CanMoveToContainerStatus value);
00986
00988
00997 enum class RelatedContentType
00998 {
00999     NEWS_ARTICLE = 1,
01000     PROFILE_PERSON = 2,
01001     PROFILE_ORGANIZATION = 3,
01002     REFERENCE_MATERIAL = 4
01003 };
01004
01005 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
01006 #if QEVERCLOUD_USES_Q_NAMESPACE
01007 Q_ENUM_NS(RelatedContentType)
01008 #endif
01009 #endif
01010
01011 inline uint qHash(RelatedContentType value)
01012 {
01013     return static_cast<uint>(value);
01014 }
01015
01017
01018 QEVERCLOUD_EXPORT QTextStream & operator<<(
01019     QTextStream & out, const RelatedContentType value);
```

```
01020
01022
01023 QEVERCLOUD_EXPORT QDebug & operator<<(
01024     QDebug & out, const RelatedContentType value);
01025
01027
01047 enum class RelatedContentAccess
01048 {
01049     NOT_ACCESSIBLE = 0,
01050     DIRECT_LINK_ACCESS_OK = 1,
01051     DIRECT_LINK_LOGIN_REQUIRED = 2,
01052     DIRECT_LINK_EMBEDDED_VIEW = 3
01053 };
01054
01055 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
01056 #if QEVERCLOUD_USES_Q_NAMESPACE
01057 Q_ENUM_NS(RelatedContentAccess)
01058 #endif
01059 #endif
01060
01061 inline uint qHash(RelatedContentAccess value)
01062 {
01063     return static_cast<uint>(value);
01064 }
01065
01067
01068 QEVERCLOUD_EXPORT QTextStream & operator<<(
01069     QTextStream & out, const RelatedContentAccess value);
01070
01072
01073 QEVERCLOUD_EXPORT QDebug & operator<<(
01074     QDebug & out, const RelatedContentAccess value);
01075
01077
01081 enum class UserIdentityType
01082 {
01083     EVERNOTE_USERID = 1,
01084     EMAIL = 2,
01085     IDENTITYID = 3
01086 };
01087
01088 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
01089 #if QEVERCLOUD_USES_Q_NAMESPACE
01090 Q_ENUM_NS(UserIdentityType)
01091 #endif
01092 #endif
01093
01094 inline uint qHash(UserIdentityType value)
01095 {
01096     return static_cast<uint>(value);
01097 }
01098
01100
01101 QEVERCLOUD_EXPORT QTextStream & operator<<(
01102     QTextStream & out, const UserIdentityType value);
01103
01105
01106 QEVERCLOUD_EXPORT QDebug & operator<<(
01107     QDebug & out, const UserIdentityType value);
01108
01109 } // namespace qevercloud
01110
01111 Q_DECLARE_METATYPE(qevercloud::EDAMErrorCode)
01112 Q_DECLARE_METATYPE(qevercloud::EDAMInvalidContactReason)
01113 Q_DECLARE_METATYPE(qevercloud::ShareRelationshipPrivilegeLevel)
01114 Q_DECLARE_METATYPE(qevercloud::PrivilegeLevel)
01115 Q_DECLARE_METATYPE(qevercloud::ServiceLevel)
01116 Q_DECLARE_METATYPE(qevercloud::QueryFormat)
01117 Q_DECLARE_METATYPE(qevercloud::NoteSortOrder)
01118 Q_DECLARE_METATYPE(qevercloud::PremiumOrderStatus)
01119 Q_DECLARE_METATYPE(qevercloud::SharedNotebookPrivilegeLevel)
01120 Q_DECLARE_METATYPE(qevercloud::SharedNotePrivilegeLevel)
01121 Q_DECLARE_METATYPE(qevercloud::SponsoredGroupRole)
01122 Q_DECLARE_METATYPE(qevercloud::BusinessUserRole)
01123 Q_DECLARE_METATYPE(qevercloud::BusinessUserStatus)
01124 Q_DECLARE_METATYPE(qevercloud::SharedNotebookInstanceRestrictions)
01125 Q_DECLARE_METATYPE(qevercloud::ReminderEmailConfig)
01126 Q_DECLARE_METATYPE(qevercloud::BusinessInvitationStatus)
01127 Q_DECLARE_METATYPE(qevercloud::ContactType)
01128 Q_DECLARE_METATYPE(qevercloud::EntityType)
01129 Q_DECLARE_METATYPE(qevercloud::RecipientStatus)
01130 Q_DECLARE_METATYPE(qevercloud::CanMoveToContainerStatus)
01131 Q_DECLARE_METATYPE(qevercloud::RelatedContentType)
01132 Q_DECLARE_METATYPE(qevercloud::RelatedContentAccess)
01133 Q_DECLARE_METATYPE(qevercloud::UserIdentityType)
01134
01135 #endif // QEVERCLOUD_GENERATED_EDAMERRORCODE_H
```

8.17 Servers.h File Reference

```
#include "../Export.h"
#include "../Optional.h"
#include "../RequestContext.h"
#include "Constants.h"
#include "Types.h"
#include <QObject>
#include <functional>
```

Classes

- class qevercloud::NoteStoreServer

  The NoteStoreServer class represents customizable server for NoteStore requests. It is primarily used for testing of QEverCloud.
- class qevercloud::UserStoreServer

  The UserStoreServer class represents customizable server for UserStore requests. It is primarily used for testing of QEverCloud.

Namespaces

- namespace qevercloud

8.18 Servers.h

Go to the documentation of this file.

```
00001
00012 #ifndef QEVERCLOUD_GENERATED_SERVERS_H
00013 #define QEVERCLOUD_GENERATED_SERVERS_H
00014
00015 #include "../Export.h"
00016
00017 #include "../Optional.h"
00018 #include "../RequestContext.h"
00019 #include "Constants.h"
00020 #include "Types.h"
00021 #include <QObject>
00022 #include <functional>
00023
00024 namespace qevercloud {
00025
00027
00033 class QEVERCLOUD_EXPORT NoteStoreServer: public QObject
00034 {
00035     Q_OBJECT
00036     Q_DISABLE_COPY(NoteStoreServer)
00037 public:
00038     explicit NoteStoreServer(QObject * parent = nullptr);
00039
00040 Q_SIGNALS:
00041     // Signals notifying listeners about incoming requests
00042     void getSyncStateRequest(
00043         IRequestContextPtr ctx);
00044
00045     void getFilteredSyncChunkRequest(
00046         qint32 afterUSN,
00047         qint32 maxEntries,
00048         SyncChunkFilter filter,
00049         IRequestContextPtr ctx);
00050
00051     void getLinkedNotebookSyncStateRequest(
00052         LinkedNotebook linkedNotebook,
00053         IRequestContextPtr ctx);
00054
```

```
00055     void getLinkedNotebookSyncChunkRequest(
00056         LinkedNotebook linkedNotebook,
00057         qint32 afterUSN,
00058         qint32 maxEntries,
00059         bool fullSyncOnly,
00060         IRequestContextPtr ctx);
00061
00062     void listNotebooksRequest(
00063         IRequestContextPtr ctx);
00064
00065     void listAccessibleBusinessNotebooksRequest(
00066         IRequestContextPtr ctx);
00067
00068     void getNotebookRequest(
00069         Guid guid,
00070         IRequestContextPtr ctx);
00071
00072     void getDefaultNotebookRequest(
00073         IRequestContextPtr ctx);
00074
00075     void createNotebookRequest(
00076         Notebook notebook,
00077         IRequestContextPtr ctx);
00078
00079     void updateNotebookRequest(
00080         Notebook notebook,
00081         IRequestContextPtr ctx);
00082
00083     void expungeNotebookRequest(
00084         Guid guid,
00085         IRequestContextPtr ctx);
00086
00087     void listTagsRequest(
00088         IRequestContextPtr ctx);
00089
00090     void listTagsByNotebookRequest(
00091         Guid notebookGuid,
00092         IRequestContextPtr ctx);
00093
00094     void getTagRequest(
00095         Guid guid,
00096         IRequestContextPtr ctx);
00097
00098     void createTagRequest(
00099         Tag tag,
00100         IRequestContextPtr ctx);
00101
00102     void updateTagRequest(
00103         Tag tag,
00104         IRequestContextPtr ctx);
00105
00106     void untagAllRequest(
00107         Guid guid,
00108         IRequestContextPtr ctx);
00109
00110     void expungeTagRequest(
00111         Guid guid,
00112         IRequestContextPtr ctx);
00113
00114     void listSearchesRequest(
00115         IRequestContextPtr ctx);
00116
00117     void getSearchRequest(
00118         Guid guid,
00119         IRequestContextPtr ctx);
00120
00121     void createSearchRequest(
00122         SavedSearch search,
00123         IRequestContextPtr ctx);
00124
00125     void updateSearchRequest(
00126         SavedSearch search,
00127         IRequestContextPtr ctx);
00128
00129     void expungeSearchRequest(
00130         Guid guid,
00131         IRequestContextPtr ctx);
00132
00133     void findNoteOffsetRequest(
00134         NoteFilter filter,
00135         Guid guid,
00136         IRequestContextPtr ctx);
00137
00138     void findNotesMetadataRequest(
00139         NoteFilter filter,
00140         qint32 offset,
00141         qint32 maxNotes,
```

```
00142         NotesMetadataResultSpec resultSpec,
00143         IRequestContextPtr ctx);
00144
00145     void findNoteCountsRequest(
00146         NoteFilter filter,
00147         bool withTrash,
00148         IRequestContextPtr ctx);
00149
00150     void getNoteWithResultSpecRequest(
00151         Guid guid,
00152         NoteResultSpec resultSpec,
00153         IRequestContextPtr ctx);
00154
00155     void getNoteRequest(
00156         Guid guid,
00157         bool withContent,
00158         bool withResourcesData,
00159         bool withResourcesRecognition,
00160         bool withResourcesAlternateData,
00161         IRequestContextPtr ctx);
00162
00163     void getNoteApplicationDataRequest(
00164         Guid guid,
00165         IRequestContextPtr ctx);
00166
00167     void getNoteApplicationDataEntryRequest(
00168         Guid guid,
00169         QString key,
00170         IRequestContextPtr ctx);
00171
00172     void setNoteApplicationDataEntryRequest(
00173         Guid guid,
00174         QString key,
00175         QString value,
00176         IRequestContextPtr ctx);
00177
00178     void unsetNoteApplicationDataEntryRequest(
00179         Guid guid,
00180         QString key,
00181         IRequestContextPtr ctx);
00182
00183     void getNoteContentRequest(
00184         Guid guid,
00185         IRequestContextPtr ctx);
00186
00187     void getNoteSearchTextRequest(
00188         Guid guid,
00189         bool noteOnly,
00190         bool tokenizeForIndexing,
00191         IRequestContextPtr ctx);
00192
00193     void getResourceSearchTextRequest(
00194         Guid guid,
00195         IRequestContextPtr ctx);
00196
00197     void getNoteTagNamesRequest(
00198         Guid guid,
00199         IRequestContextPtr ctx);
00200
00201     void createNoteRequest(
00202         Note note,
00203         IRequestContextPtr ctx);
00204
00205     void updateNoteRequest(
00206         Note note,
00207         IRequestContextPtr ctx);
00208
00209     void deleteNoteRequest(
00210         Guid guid,
00211         IRequestContextPtr ctx);
00212
00213     void expungeNoteRequest(
00214         Guid guid,
00215         IRequestContextPtr ctx);
00216
00217     void copyNoteRequest(
00218         Guid noteGuid,
00219         Guid toNotebookGuid,
00220         IRequestContextPtr ctx);
00221
00222     void listNoteVersionsRequest(
00223         Guid noteGuid,
00224         IRequestContextPtr ctx);
00225
00226     void getNoteVersionRequest(
00227         Guid noteGuid,
00228         qint32 updateSequenceNum,
```

```
00229         bool withResourcesData,
00230         bool withResourcesRecognition,
00231         bool withResourcesAlternateData,
00232         IRequestContextPtr ctx);
00233
00234     void getResourceRequest(
00235         Guid guid,
00236         bool withData,
00237         bool withRecognition,
00238         bool withAttributes,
00239         bool withAlternateData,
00240         IRequestContextPtr ctx);
00241
00242     void getResourceApplicationDataRequest(
00243         Guid guid,
00244         IRequestContextPtr ctx);
00245
00246     void getResourceApplicationDataEntryRequest(
00247         Guid guid,
00248         QString key,
00249         IRequestContextPtr ctx);
00250
00251     void setResourceApplicationDataEntryRequest(
00252         Guid guid,
00253         QString key,
00254         QString value,
00255         IRequestContextPtr ctx);
00256
00257     void unsetResourceApplicationDataEntryRequest(
00258         Guid guid,
00259         QString key,
00260         IRequestContextPtr ctx);
00261
00262     void updateResourceRequest(
00263         Resource resource,
00264         IRequestContextPtr ctx);
00265
00266     void getResourceDataRequest(
00267         Guid guid,
00268         IRequestContextPtr ctx);
00269
00270     void getResourceByHashRequest(
00271         Guid noteGuid,
00272         QByteArray contentHash,
00273         bool withData,
00274         bool withRecognition,
00275         bool withAlternateData,
00276         IRequestContextPtr ctx);
00277
00278     void getResourceRecognitionRequest(
00279         Guid guid,
00280         IRequestContextPtr ctx);
00281
00282     void getResourceAlternateDataRequest(
00283         Guid guid,
00284         IRequestContextPtr ctx);
00285
00286     void getResourceAttributesRequest(
00287         Guid guid,
00288         IRequestContextPtr ctx);
00289
00290     void getPublicNotebookRequest(
00291         UserID userId,
00292         QString publicUri,
00293         IRequestContextPtr ctx);
00294
00295     void shareNotebookRequest(
00296         SharedNotebook sharedNotebook,
00297         QString message,
00298         IRequestContextPtr ctx);
00299
00300     void createOrUpdateNotebookSharesRequest(
00301         NotebookShareTemplate shareTemplate,
00302         IRequestContextPtr ctx);
00303
00304     void updateSharedNotebookRequest(
00305         SharedNotebook sharedNotebook,
00306         IRequestContextPtr ctx);
00307
00308     void setNotebookRecipientSettingsRequest(
00309         QString notebookGuid,
00310         NotebookRecipientSettings recipientSettings,
00311         IRequestContextPtr ctx);
00312
00313     void listSharedNotebooksRequest(
00314         IRequestContextPtr ctx);
00315
```

```
00316     void createLinkedNotebookRequest(
00317         LinkedNotebook linkedNotebook,
00318         IRequestContextPtr ctx);
00319
00320     void updateLinkedNotebookRequest(
00321         LinkedNotebook linkedNotebook,
00322         IRequestContextPtr ctx);
00323
00324     void listLinkedNotebooksRequest(
00325         IRequestContextPtr ctx);
00326
00327     void expungeLinkedNotebookRequest(
00328         Guid guid,
00329         IRequestContextPtr ctx);
00330
00331     void authenticateToSharedNotebookRequest(
00332         QString shareKeyOrGlobalId,
00333         IRequestContextPtr ctx);
00334
00335     void getSharedNotebookByAuthRequest(
00336         IRequestContextPtr ctx);
00337
00338     void emailNoteRequest(
00339         NoteEmailParameters parameters,
00340         IRequestContextPtr ctx);
00341
00342     void shareNoteRequest(
00343         Guid guid,
00344         IRequestContextPtr ctx);
00345
00346     void stopSharingNoteRequest(
00347         Guid guid,
00348         IRequestContextPtr ctx);
00349
00350     void authenticateToSharedNoteRequest(
00351         QString guid,
00352         QString noteKey,
00353         IRequestContextPtr ctx);
00354
00355     void findRelatedRequest(
00356         RelatedQuery query,
00357         RelatedResultSpec resultSpec,
00358         IRequestContextPtr ctx);
00359
00360     void updateNoteIfUsnMatchesRequest(
00361         Note note,
00362         IRequestContextPtr ctx);
00363
00364     void manageNotebookSharesRequest(
00365         ManageNotebookSharesParameters parameters,
00366         IRequestContextPtr ctx);
00367
00368     void getNotebookSharesRequest(
00369         QString notebookGuid,
00370         IRequestContextPtr ctx);
00371
00372     // Signals used to send encoded response data
00373     void getSyncStateRequestReady(
00374         QByteArray data);
00375
00376     void getFilteredSyncChunkRequestReady(
00377         QByteArray data);
00378
00379     void getLinkedNotebookSyncStateRequestReady(
00380         QByteArray data);
00381
00382     void getLinkedNotebookSyncChunkRequestReady(
00383         QByteArray data);
00384
00385     void listNotebooksRequestReady(
00386         QByteArray data);
00387
00388     void listAccessibleBusinessNotebooksRequestReady(
00389         QByteArray data);
00390
00391     void getNotebookRequestReady(
00392         QByteArray data);
00393
00394     void getDefaultNotebookRequestReady(
00395         QByteArray data);
00396
00397     void createNotebookRequestReady(
00398         QByteArray data);
00399
00400     void updateNotebookRequestReady(
00401         QByteArray data);
00402
```

```
00403     void expungeNotebookRequestReady(
00404         QByteArray data);
00405
00406     void listTagsRequestReady(
00407         QByteArray data);
00408
00409     void listTagsByNotebookRequestReady(
00410         QByteArray data);
00411
00412     void getTagRequestReady(
00413         QByteArray data);
00414
00415     void createTagRequestReady(
00416         QByteArray data);
00417
00418     void updateTagRequestReady(
00419         QByteArray data);
00420
00421     void untagAllRequestReady(
00422         QByteArray data);
00423
00424     void expungeTagRequestReady(
00425         QByteArray data);
00426
00427     void listSearchesRequestReady(
00428         QByteArray data);
00429
00430     void getSearchRequestReady(
00431         QByteArray data);
00432
00433     void createSearchRequestReady(
00434         QByteArray data);
00435
00436     void updateSearchRequestReady(
00437         QByteArray data);
00438
00439     void expungeSearchRequestReady(
00440         QByteArray data);
00441
00442     void findNoteOffsetRequestReady(
00443         QByteArray data);
00444
00445     void findNotesMetadataRequestReady(
00446         QByteArray data);
00447
00448     void findNoteCountsRequestReady(
00449         QByteArray data);
00450
00451     void getNoteWithResultSpecRequestReady(
00452         QByteArray data);
00453
00454     void getNoteRequestReady(
00455         QByteArray data);
00456
00457     void getNoteApplicationDataRequestReady(
00458         QByteArray data);
00459
00460     void getNoteApplicationDataEntryRequestReady(
00461         QByteArray data);
00462
00463     void setNoteApplicationDataEntryRequestReady(
00464         QByteArray data);
00465
00466     void unsetNoteApplicationDataEntryRequestReady(
00467         QByteArray data);
00468
00469     void getNoteContentRequestReady(
00470         QByteArray data);
00471
00472     void getNoteSearchTextRequestReady(
00473         QByteArray data);
00474
00475     void getResourceSearchTextRequestReady(
00476         QByteArray data);
00477
00478     void getNoteTagNamesRequestReady(
00479         QByteArray data);
00480
00481     void createNoteRequestReady(
00482         QByteArray data);
00483
00484     void updateNoteRequestReady(
00485         QByteArray data);
00486
00487     void deleteNoteRequestReady(
00488         QByteArray data);
00489
```

```
00490     void expungeNoteRequestReady(
00491         QByteArray data);
00492
00493     void copyNoteRequestReady(
00494         QByteArray data);
00495
00496     void listNoteVersionsRequestReady(
00497         QByteArray data);
00498
00499     void getNoteVersionRequestReady(
00500         QByteArray data);
00501
00502     void getResourceRequestReady(
00503         QByteArray data);
00504
00505     void getResourceApplicationDataRequestReady(
00506         QByteArray data);
00507
00508     void getResourceApplicationDataEntryRequestReady(
00509         QByteArray data);
00510
00511     void setResourceApplicationDataEntryRequestReady(
00512         QByteArray data);
00513
00514     void unsetResourceApplicationDataEntryRequestReady(
00515         QByteArray data);
00516
00517     void updateResourceRequestReady(
00518         QByteArray data);
00519
00520     void getResourceDataRequestReady(
00521         QByteArray data);
00522
00523     void getResourceByHashRequestReady(
00524         QByteArray data);
00525
00526     void getResourceRecognitionRequestReady(
00527         QByteArray data);
00528
00529     void getResourceAlternateDataRequestReady(
00530         QByteArray data);
00531
00532     void getResourceAttributesRequestReady(
00533         QByteArray data);
00534
00535     void getPublicNotebookRequestReady(
00536         QByteArray data);
00537
00538     void shareNotebookRequestReady(
00539         QByteArray data);
00540
00541     void createOrUpdateNotebookSharesRequestReady(
00542         QByteArray data);
00543
00544     void updateSharedNotebookRequestReady(
00545         QByteArray data);
00546
00547     void setNotebookRecipientSettingsRequestReady(
00548         QByteArray data);
00549
00550     void listSharedNotebooksRequestReady(
00551         QByteArray data);
00552
00553     void createLinkedNotebookRequestReady(
00554         QByteArray data);
00555
00556     void updateLinkedNotebookRequestReady(
00557         QByteArray data);
00558
00559     void listLinkedNotebooksRequestReady(
00560         QByteArray data);
00561
00562     void expungeLinkedNotebookRequestReady(
00563         QByteArray data);
00564
00565     void authenticateToSharedNotebookRequestReady(
00566         QByteArray data);
00567
00568     void getSharedNotebookByAuthRequestReady(
00569         QByteArray data);
00570
00571     void emailNoteRequestReady(
00572         QByteArray data);
00573
00574     void shareNoteRequestReady(
00575         QByteArray data);
00576
```

```
00577     void stopSharingNoteRequestReady(
00578         QByteArray data);
00579
00580     void authenticateToSharedNoteRequestReady(
00581         QByteArray data);
00582
00583     void findRelatedRequestReady(
00584         QByteArray data);
00585
00586     void updateNoteIfUsnMatchesRequestReady(
00587         QByteArray data);
00588
00589     void manageNotebookSharesRequestReady(
00590         QByteArray data);
00591
00592     void getNotebookSharesRequestReady(
00593         QByteArray data);
00594
00595 public Q_SLOTS:
00596     // Slot used to deliver requests to the server
00597     void onRequest(QByteArray data);
00598
00599     // Slots for replies to requests
00600     void onGetSyncStateRequestReady(
00601         SyncState value,
00602         EverCloudExceptionDataPtr exceptionData);
00603
00604     void onGetFilteredSyncChunkRequestReady(
00605         SyncChunk value,
00606         EverCloudExceptionDataPtr exceptionData);
00607
00608     void onGetLinkedNotebookSyncStateRequestReady(
00609         SyncState value,
00610         EverCloudExceptionDataPtr exceptionData);
00611
00612     void onGetLinkedNotebookSyncChunkRequestReady(
00613         SyncChunk value,
00614         EverCloudExceptionDataPtr exceptionData);
00615
00616     void onListNotebooksRequestReady(
00617         QList<Notebook> value,
00618         EverCloudExceptionDataPtr exceptionData);
00619
00620     void onListAccessibleBusinessNotebooksRequestReady(
00621         QList<Notebook> value,
00622         EverCloudExceptionDataPtr exceptionData);
00623
00624     void onGetNotebookRequestReady(
00625         Notebook value,
00626         EverCloudExceptionDataPtr exceptionData);
00627
00628     void onGetDefaultNotebookRequestReady(
00629         Notebook value,
00630         EverCloudExceptionDataPtr exceptionData);
00631
00632     void onCreateNotebookRequestReady(
00633         Notebook value,
00634         EverCloudExceptionDataPtr exceptionData);
00635
00636     void onUpdateNotebookRequestReady(
00637         qint32 value,
00638         EverCloudExceptionDataPtr exceptionData);
00639
00640     void onExpungeNotebookRequestReady(
00641         qint32 value,
00642         EverCloudExceptionDataPtr exceptionData);
00643
00644     void onListTagsRequestReady(
00645         QList<Tag> value,
00646         EverCloudExceptionDataPtr exceptionData);
00647
00648     void onListTagsByNotebookRequestReady(
00649         QList<Tag> value,
00650         EverCloudExceptionDataPtr exceptionData);
00651
00652     void onGetTagRequestReady(
00653         Tag value,
00654         EverCloudExceptionDataPtr exceptionData);
00655
00656     void onCreateTagRequestReady(
00657         Tag value,
00658         EverCloudExceptionDataPtr exceptionData);
00659
00660     void onUpdateTagRequestReady(
00661         qint32 value,
00662         EverCloudExceptionDataPtr exceptionData);
00663
```

```
00664     void onUntagAllRequestReady(
00665         EverCloudExceptionDataPtr exceptionData);
00666
00667     void onExpungeTagRequestReady(
00668         qint32 value,
00669         EverCloudExceptionDataPtr exceptionData);
00670
00671     void onListSearchesRequestReady(
00672         QList<SavedSearch> value,
00673         EverCloudExceptionDataPtr exceptionData);
00674
00675     void onGetSearchRequestReady(
00676         SavedSearch value,
00677         EverCloudExceptionDataPtr exceptionData);
00678
00679     void onCreateSearchRequestReady(
00680         SavedSearch value,
00681         EverCloudExceptionDataPtr exceptionData);
00682
00683     void onUpdateSearchRequestReady(
00684         qint32 value,
00685         EverCloudExceptionDataPtr exceptionData);
00686
00687     void onExpungeSearchRequestReady(
00688         qint32 value,
00689         EverCloudExceptionDataPtr exceptionData);
00690
00691     void onFindNoteOffsetRequestReady(
00692         qint32 value,
00693         EverCloudExceptionDataPtr exceptionData);
00694
00695     void onFindNotesMetadataRequestReady(
00696         NotesMetadataList value,
00697         EverCloudExceptionDataPtr exceptionData);
00698
00699     void onFindNoteCountsRequestReady(
00700         NoteCollectionCounts value,
00701         EverCloudExceptionDataPtr exceptionData);
00702
00703     void onGetNoteWithResultSpecRequestReady(
00704         Note value,
00705         EverCloudExceptionDataPtr exceptionData);
00706
00707     void onGetNoteRequestReady(
00708         Note value,
00709         EverCloudExceptionDataPtr exceptionData);
00710
00711     void onGetNoteApplicationDataRequestReady(
00712         LazyMap value,
00713         EverCloudExceptionDataPtr exceptionData);
00714
00715     void onGetNoteApplicationDataEntryRequestReady(
00716         QString value,
00717         EverCloudExceptionDataPtr exceptionData);
00718
00719     void onSetNoteApplicationDataEntryRequestReady(
00720         qint32 value,
00721         EverCloudExceptionDataPtr exceptionData);
00722
00723     void onUnsetNoteApplicationDataEntryRequestReady(
00724         qint32 value,
00725         EverCloudExceptionDataPtr exceptionData);
00726
00727     void onGetNoteContentRequestReady(
00728         QString value,
00729         EverCloudExceptionDataPtr exceptionData);
00730
00731     void onGetNoteSearchTextRequestReady(
00732         QString value,
00733         EverCloudExceptionDataPtr exceptionData);
00734
00735     void onGetResourceSearchTextRequestReady(
00736         QString value,
00737         EverCloudExceptionDataPtr exceptionData);
00738
00739     void onGetNoteTagNamesRequestReady(
00740         QStringList value,
00741         EverCloudExceptionDataPtr exceptionData);
00742
00743     void onCreateNoteRequestReady(
00744         Note value,
00745         EverCloudExceptionDataPtr exceptionData);
00746
00747     void onUpdateNoteRequestReady(
00748         Note value,
00749         EverCloudExceptionDataPtr exceptionData);
00750
```

```
00751     void onDeleteNoteRequestReady(
00752         qint32 value,
00753         EverCloudExceptionDataPtr exceptionData);
00754
00755     void onExpungeNoteRequestReady(
00756         qint32 value,
00757         EverCloudExceptionDataPtr exceptionData);
00758
00759     void onCopyNoteRequestReady(
00760         Note value,
00761         EverCloudExceptionDataPtr exceptionData);
00762
00763     void onListNoteVersionsRequestReady(
00764         QList<NoteVersionId> value,
00765         EverCloudExceptionDataPtr exceptionData);
00766
00767     void onGetNoteVersionRequestReady(
00768         Note value,
00769         EverCloudExceptionDataPtr exceptionData);
00770
00771     void onGetResourceRequestReady(
00772         Resource value,
00773         EverCloudExceptionDataPtr exceptionData);
00774
00775     void onGetResourceApplicationDataRequestReady(
00776         LazyMap value,
00777         EverCloudExceptionDataPtr exceptionData);
00778
00779     void onGetResourceApplicationDataEntryRequestReady(
00780         QString value,
00781         EverCloudExceptionDataPtr exceptionData);
00782
00783     void onSetResourceApplicationDataEntryRequestReady(
00784         qint32 value,
00785         EverCloudExceptionDataPtr exceptionData);
00786
00787     void onUnsetResourceApplicationDataEntryRequestReady(
00788         qint32 value,
00789         EverCloudExceptionDataPtr exceptionData);
00790
00791     void onUpdateResourceRequestReady(
00792         qint32 value,
00793         EverCloudExceptionDataPtr exceptionData);
00794
00795     void onGetResourceDataRequestReady(
00796         QByteArray value,
00797         EverCloudExceptionDataPtr exceptionData);
00798
00799     void onGetResourceByHashRequestReady(
00800         Resource value,
00801         EverCloudExceptionDataPtr exceptionData);
00802
00803     void onGetResourceRecognitionRequestReady(
00804         QByteArray value,
00805         EverCloudExceptionDataPtr exceptionData);
00806
00807     void onGetResourceAlternateDataRequestReady(
00808         QByteArray value,
00809         EverCloudExceptionDataPtr exceptionData);
00810
00811     void onGetResourceAttributesRequestReady(
00812         ResourceAttributes value,
00813         EverCloudExceptionDataPtr exceptionData);
00814
00815     void onGetPublicNotebookRequestReady(
00816         Notebook value,
00817         EverCloudExceptionDataPtr exceptionData);
00818
00819     void onShareNotebookRequestReady(
00820         SharedNotebook value,
00821         EverCloudExceptionDataPtr exceptionData);
00822
00823     void onCreateOrUpdateNotebookSharesRequestReady(
00824         CreateOrUpdateNotebookSharesResult value,
00825         EverCloudExceptionDataPtr exceptionData);
00826
00827     void onUpdateSharedNotebookRequestReady(
00828         qint32 value,
00829         EverCloudExceptionDataPtr exceptionData);
00830
00831     void onSetNotebookRecipientSettingsRequestReady(
00832         Notebook value,
00833         EverCloudExceptionDataPtr exceptionData);
00834
00835     void onListSharedNotebooksRequestReady(
00836         QList<SharedNotebook> value,
00837         EverCloudExceptionDataPtr exceptionData);
```

```
00838
00839     void onCreateLinkedNotebookRequestReady(
00840         LinkedNotebook value,
00841         EverCloudExceptionDataPtr exceptionData);
00842
00843     void onUpdateLinkedNotebookRequestReady(
00844         qint32 value,
00845         EverCloudExceptionDataPtr exceptionData);
00846
00847     void onListLinkedNotebooksRequestReady(
00848         QList<LinkedNotebook> value,
00849         EverCloudExceptionDataPtr exceptionData);
00850
00851     void onExpungeLinkedNotebookRequestReady(
00852         qint32 value,
00853         EverCloudExceptionDataPtr exceptionData);
00854
00855     void onAuthenticateToSharedNotebookRequestReady(
00856         AuthenticationResult value,
00857         EverCloudExceptionDataPtr exceptionData);
00858
00859     void onGetSharedNotebookByAuthRequestReady(
00860         SharedNotebook value,
00861         EverCloudExceptionDataPtr exceptionData);
00862
00863     void onEmailNoteRequestReady(
00864         EverCloudExceptionDataPtr exceptionData);
00865
00866     void onShareNoteRequestReady(
00867         QString value,
00868         EverCloudExceptionDataPtr exceptionData);
00869
00870     void onStopSharingNoteRequestReady(
00871         EverCloudExceptionDataPtr exceptionData);
00872
00873     void onAuthenticateToSharedNoteRequestReady(
00874         AuthenticationResult value,
00875         EverCloudExceptionDataPtr exceptionData);
00876
00877     void onFindRelatedRequestReady(
00878         RelatedResult value,
00879         EverCloudExceptionDataPtr exceptionData);
00880
00881     void onUpdateNoteIfUsnMatchesRequestReady(
00882         UpdateNoteIfUsnMatchesResult value,
00883         EverCloudExceptionDataPtr exceptionData);
00884
00885     void onManageNotebookSharesRequestReady(
00886         ManageNotebookSharesResult value,
00887         EverCloudExceptionDataPtr exceptionData);
00888
00889     void onGetNotebookSharesRequestReady(
00890         ShareRelationships value,
00891         EverCloudExceptionDataPtr exceptionData);
00892
00893 };
00894
00896
00902 class QEVERCLOUD_EXPORT UserStoreServer: public QObject
00903 {
00904     Q_OBJECT
00905     Q_DISABLE_COPY(UserStoreServer)
00906 public:
00907     explicit UserStoreServer(QObject * parent = nullptr);
00908
00909 Q_SIGNALS:
00910     // Signals notifying listeners about incoming requests
00911     void checkVersionRequest(
00912         QString clientName,
00913         qint16 edamVersionMajor,
00914         qint16 edamVersionMinor,
00915         IRequestContextPtr ctx);
00916
00917     void getBootstrapInfoRequest(
00918         QString locale,
00919         IRequestContextPtr ctx);
00920
00921     void authenticateLongSessionRequest(
00922         QString username,
00923         QString password,
00924         QString consumerKey,
00925         QString consumerSecret,
00926         QString deviceIdentifier,
00927         QString deviceDescription,
00928         bool supportsTwoFactor,
00929         IRequestContextPtr ctx);
00930
```

```
00931     void completeTwoFactorAuthenticationRequest(
00932         QString oneTimeCode,
00933         QString deviceIdentifier,
00934         QString deviceDescription,
00935         IRequestContextPtr ctx);
00936
00937     void revokeLongSessionRequest(
00938         IRequestContextPtr ctx);
00939
00940     void authenticateToBusinessRequest(
00941         IRequestContextPtr ctx);
00942
00943     void getUserRequest(
00944         IRequestContextPtr ctx);
00945
00946     void getPublicUserInfoRequest(
00947         QString username,
00948         IRequestContextPtr ctx);
00949
00950     void getUserUrlsRequest(
00951         IRequestContextPtr ctx);
00952
00953     void inviteToBusinessRequest(
00954         QString emailAddress,
00955         IRequestContextPtr ctx);
00956
00957     void removeFromBusinessRequest(
00958         QString emailAddress,
00959         IRequestContextPtr ctx);
00960
00961     void updateBusinessUserIdentifierRequest(
00962         QString oldEmailAddress,
00963         QString newEmailAddress,
00964         IRequestContextPtr ctx);
00965
00966     void listBusinessUsersRequest(
00967         IRequestContextPtr ctx);
00968
00969     void listBusinessInvitationsRequest(
00970         bool includeRequestedInvitations,
00971         IRequestContextPtr ctx);
00972
00973     void getAccountLimitsRequest(
00974         ServiceLevel serviceLevel,
00975         IRequestContextPtr ctx);
00976
00977     // Signals used to send encoded response data
00978     void checkVersionRequestReady(
00979         QByteArray data);
00980
00981     void getBootstrapInfoRequestReady(
00982         QByteArray data);
00983
00984     void authenticateLongSessionRequestReady(
00985         QByteArray data);
00986
00987     void completeTwoFactorAuthenticationRequestReady(
00988         QByteArray data);
00989
00990     void revokeLongSessionRequestReady(
00991         QByteArray data);
00992
00993     void authenticateToBusinessRequestReady(
00994         QByteArray data);
00995
00996     void getUserRequestReady(
00997         QByteArray data);
00998
00999     void getPublicUserInfoRequestReady(
01000         QByteArray data);
01001
01002     void getUserUrlsRequestReady(
01003         QByteArray data);
01004
01005     void inviteToBusinessRequestReady(
01006         QByteArray data);
01007
01008     void removeFromBusinessRequestReady(
01009         QByteArray data);
01010
01011     void updateBusinessUserIdentifierRequestReady(
01012         QByteArray data);
01013
01014     void listBusinessUsersRequestReady(
01015         QByteArray data);
01016
01017     void listBusinessInvitationsRequestReady(
```

```
01018         QByteArray data);
01019
01020     void getAccountLimitsRequestReady(
01021         QByteArray data);
01022
01023 public Q_SLOTS:
01024     // Slot used to deliver requests to the server
01025     void onRequest(QByteArray data);
01026
01027     // Slots for replies to requests
01028     void onCheckVersionRequestReady(
01029         bool value,
01030         EverCloudExceptionDataPtr exceptionData);
01031
01032     void onGetBootstrapInfoRequestReady(
01033         BootstrapInfo value,
01034         EverCloudExceptionDataPtr exceptionData);
01035
01036     void onAuthenticateLongSessionRequestReady(
01037         AuthenticationResult value,
01038         EverCloudExceptionDataPtr exceptionData);
01039
01040     void onCompleteTwoFactorAuthenticationRequestReady(
01041         AuthenticationResult value,
01042         EverCloudExceptionDataPtr exceptionData);
01043
01044     void onRevokeLongSessionRequestReady(
01045         EverCloudExceptionDataPtr exceptionData);
01046
01047     void onAuthenticateToBusinessRequestReady(
01048         AuthenticationResult value,
01049         EverCloudExceptionDataPtr exceptionData);
01050
01051     void onGetUserRequestReady(
01052         User value,
01053         EverCloudExceptionDataPtr exceptionData);
01054
01055     void onGetPublicUserInfoRequestReady(
01056         PublicUserInfo value,
01057         EverCloudExceptionDataPtr exceptionData);
01058
01059     void onGetUserUrlsRequestReady(
01060         UserUrls value,
01061         EverCloudExceptionDataPtr exceptionData);
01062
01063     void onInviteToBusinessRequestReady(
01064         EverCloudExceptionDataPtr exceptionData);
01065
01066     void onRemoveFromBusinessRequestReady(
01067         EverCloudExceptionDataPtr exceptionData);
01068
01069     void onUpdateBusinessUserIdentifierRequestReady(
01070         EverCloudExceptionDataPtr exceptionData);
01071
01072     void onListBusinessUsersRequestReady(
01073         QList<UserProfile> value,
01074         EverCloudExceptionDataPtr exceptionData);
01075
01076     void onListBusinessInvitationsRequestReady(
01077         QList<BusinessInvitation> value,
01078         EverCloudExceptionDataPtr exceptionData);
01079
01080     void onGetAccountLimitsRequestReady(
01081         AccountLimits value,
01082         EverCloudExceptionDataPtr exceptionData);
01083
01084 };
01085
01086 } // namespace qevercloud
01087
01088 #endif // QEVERCLOUD_GENERATED_SERVERS_H
```

8.19 Services.h File Reference

```
#include "../Export.h"
#include "../AsyncResult.h"
#include "../DurableService.h"
#include "../Optional.h"
#include "../RequestContext.h"
```

```
#include "Constants.h"
#include "Types.h"
#include <QObject>
```

Classes

- class qevercloud::INoteStore
- class qevercloud::IUserStore

Namespaces

- namespace qevercloud

Typedefs

- using qevercloud::INoteStorePtr = std::shared_ptr< INoteStore >
- using qevercloud::IUserStorePtr = std::shared_ptr< IUserStore >

Functions

- QEVERCLOUD_EXPORT INoteStore * qevercloud::newNoteStore (QString noteStoreUrl={}, IRequestContextPtr ctx={}, QObject *parent=nullptr, IRetryPolicyPtr retryPolicy={})
- QEVERCLOUD_EXPORT IUserStore * qevercloud::newUserStore (QString userStoreUrl={}, IRequestContextPtr ctx={}, QObject *parent=nullptr, IRetryPolicyPtr retryPolicy={})

8.20 Services.h

Go to the documentation of this file.

```
00001
00012 #ifndef QEVERCLOUD_GENERATED_SERVICES_H
00013 #define QEVERCLOUD_GENERATED_SERVICES_H
00014
00015 #include "../Export.h"
00016
00017 #include "../AsyncResult.h"
00018 #include "../DurableService.h"
00019 #include "../Optional.h"
00020 #include "../RequestContext.h"
00021 #include "Constants.h"
00022 #include "Types.h"
00023 #include <QObject>
00024
00025 namespace qevercloud {
00026
00028
00055 class QEVERCLOUD_EXPORT INoteStore: public QObject
00056 {
00057     Q_OBJECT
00058     Q_DISABLE_COPY(INoteStore)
00059 protected:
00060     INoteStore(QObject * parent) :
00061         QObject(parent)
00062     {}
00063
00064 public:
00065     virtual QString noteStoreUrl() const = 0;
00066     virtual void setNoteStoreUrl(QString url) = 0;
00067
00072     virtual SyncState getSyncState(
00073         IRequestContextPtr ctx = {}) = 0;
00074
```

```
00076     virtual AsyncResult * getSyncStateAsync(
00077         IRequestContextPtr ctx = {}) = 0;
00078
00112     virtual SyncChunk getFilteredSyncChunk(
00113         qint32 afterUSN,
00114         qint32 maxEntries,
00115         const SyncChunkFilter & filter,
00116         IRequestContextPtr ctx = {}) = 0;
00117
00119     virtual AsyncResult * getFilteredSyncChunkAsync(
00120         qint32 afterUSN,
00121         qint32 maxEntries,
00122         const SyncChunkFilter & filter,
00123         IRequestContextPtr ctx = {}) = 0;
00124
00166     virtual SyncState getLinkedNotebookSyncState(
00167         const LinkedNotebook & linkedNotebook,
00168         IRequestContextPtr ctx = {}) = 0;
00169
00171     virtual AsyncResult * getLinkedNotebookSyncStateAsync(
00172         const LinkedNotebook & linkedNotebook,
00173         IRequestContextPtr ctx = {}) = 0;
00174
00239     virtual SyncChunk getLinkedNotebookSyncChunk(
00240         const LinkedNotebook & linkedNotebook,
00241         qint32 afterUSN,
00242         qint32 maxEntries,
00243         bool fullSyncOnly,
00244         IRequestContextPtr ctx = {}) = 0;
00245
00247     virtual AsyncResult * getLinkedNotebookSyncChunkAsync(
00248         const LinkedNotebook & linkedNotebook,
00249         qint32 afterUSN,
00250         qint32 maxEntries,
00251         bool fullSyncOnly,
00252         IRequestContextPtr ctx = {}) = 0;
00253
00257     virtual QList<Notebook> listNotebooks(
00258         IRequestContextPtr ctx = {}) = 0;
00259
00261     virtual AsyncResult * listNotebooksAsync(
00262         IRequestContextPtr ctx = {}) = 0;
00263
00277     virtual QList<Notebook> listAccessibleBusinessNotebooks(
00278         IRequestContextPtr ctx = {}) = 0;
00279
00281     virtual AsyncResult * listAccessibleBusinessNotebooksAsync(
00282         IRequestContextPtr ctx = {}) = 0;
00283
00303     virtual Notebook getNotebook(
00304         Guid guid,
00305         IRequestContextPtr ctx = {}) = 0;
00306
00308     virtual AsyncResult * getNotebookAsync(
00309         Guid guid,
00310         IRequestContextPtr ctx = {}) = 0;
00311
00316     virtual Notebook getDefaultNotebook(
00317         IRequestContextPtr ctx = {}) = 0;
00318
00320     virtual AsyncResult * getDefaultNotebookAsync(
00321         IRequestContextPtr ctx = {}) = 0;
00322
00357     virtual Notebook createNotebook(
00358         const Notebook & notebook,
00359         IRequestContextPtr ctx = {}) = 0;
00360
00362     virtual AsyncResult * createNotebookAsync(
00363         const Notebook & notebook,
00364         IRequestContextPtr ctx = {}) = 0;
00365
00405     virtual qint32 updateNotebook(
00406         const Notebook & notebook,
00407         IRequestContextPtr ctx = {}) = 0;
00408
00410     virtual AsyncResult * updateNotebookAsync(
00411         const Notebook & notebook,
00412         IRequestContextPtr ctx = {}) = 0;
00413
00439     virtual qint32 expungeNotebook(
00440         Guid guid,
00441         IRequestContextPtr ctx = {}) = 0;
00442
00444     virtual AsyncResult * expungeNotebookAsync(
00445         Guid guid,
00446         IRequestContextPtr ctx = {}) = 0;
00447
```

```
00452     virtual QList<Tag> listTags(
00453         IRequestContextPtr ctx = {}) = 0;
00454
00456     virtual AsyncResult * listTagsAsync(
00457         IRequestContextPtr ctx = {}) = 0;
00458
00472     virtual QList<Tag> listTagsByNotebook(
00473         Guid notebookGuid,
00474         IRequestContextPtr ctx = {}) = 0;
00475
00477     virtual AsyncResult * listTagsByNotebookAsync(
00478         Guid notebookGuid,
00479         IRequestContextPtr ctx = {}) = 0;
00480
00499     virtual Tag getTag(
00500         Guid guid,
00501         IRequestContextPtr ctx = {}) = 0;
00502
00504     virtual AsyncResult * getTagAsync(
00505         Guid guid,
00506         IRequestContextPtr ctx = {}) = 0;
00507
00536     virtual Tag createTag(
00537         const Tag & tag,
00538         IRequestContextPtr ctx = {}) = 0;
00539
00541     virtual AsyncResult * createTagAsync(
00542         const Tag & tag,
00543         IRequestContextPtr ctx = {}) = 0;
00544
00576     virtual qint32 updateTag(
00577         const Tag & tag,
00578         IRequestContextPtr ctx = {}) = 0;
00579
00581     virtual AsyncResult * updateTagAsync(
00582         const Tag & tag,
00583         IRequestContextPtr ctx = {}) = 0;
00584
00611     virtual void untagAll(
00612         Guid guid,
00613         IRequestContextPtr ctx = {}) = 0;
00614
00616     virtual AsyncResult * untagAllAsync(
00617         Guid guid,
00618         IRequestContextPtr ctx = {}) = 0;
00619
00645     virtual qint32 expungeTag(
00646         Guid guid,
00647         IRequestContextPtr ctx = {}) = 0;
00648
00650     virtual AsyncResult * expungeTagAsync(
00651         Guid guid,
00652         IRequestContextPtr ctx = {}) = 0;
00653
00658     virtual QList<SavedSearch> listSearches(
00659         IRequestContextPtr ctx = {}) = 0;
00660
00662     virtual AsyncResult * listSearchesAsync(
00663         IRequestContextPtr ctx = {}) = 0;
00664
00682     virtual SavedSearch getSearch(
00683         Guid guid,
00684         IRequestContextPtr ctx = {}) = 0;
00685
00687     virtual AsyncResult * getSearchAsync(
00688         Guid guid,
00689         IRequestContextPtr ctx = {}) = 0;
00690
00715     virtual SavedSearch createSearch(
00716         const SavedSearch & search,
00717         IRequestContextPtr ctx = {}) = 0;
00718
00720     virtual AsyncResult * createSearchAsync(
00721         const SavedSearch & search,
00722         IRequestContextPtr ctx = {}) = 0;
00723
00751     virtual qint32 updateSearch(
00752         const SavedSearch & search,
00753         IRequestContextPtr ctx = {}) = 0;
00754
00756     virtual AsyncResult * updateSearchAsync(
00757         const SavedSearch & search,
00758         IRequestContextPtr ctx = {}) = 0;
00759
00785     virtual qint32 expungeSearch(
00786         Guid guid,
00787         IRequestContextPtr ctx = {}) = 0;
```

```
00788
00790     virtual AsyncResult * expungeSearchAsync(
00791         Guid guid,
00792         IRequestContextPtr ctx = {}) = 0;
00793
00835     virtual qint32 findNoteOffset(
00836         const NoteFilter & filter,
00837         Guid guid,
00838         IRequestContextPtr ctx = {}) = 0;
00839
00841     virtual AsyncResult * findNoteOffsetAsync(
00842         const NoteFilter & filter,
00843         Guid guid,
00844         IRequestContextPtr ctx = {}) = 0;
00845
00903     virtual NotesMetadataList findNotesMetadata(
00904         const NoteFilter & filter,
00905         qint32 offset,
00906         qint32 maxNotes,
00907         const NotesMetadataResultSpec & resultSpec,
00908         IRequestContextPtr ctx = {}) = 0;
00909
00911     virtual AsyncResult * findNotesMetadataAsync(
00912         const NoteFilter & filter,
00913         qint32 offset,
00914         qint32 maxNotes,
00915         const NotesMetadataResultSpec & resultSpec,
00916         IRequestContextPtr ctx = {}) = 0;
00917
00949     virtual NoteCollectionCounts findNoteCounts(
00950         const NoteFilter & filter,
00951         bool withTrash,
00952         IRequestContextPtr ctx = {}) = 0;
00953
00955     virtual AsyncResult * findNoteCountsAsync(
00956         const NoteFilter & filter,
00957         bool withTrash,
00958         IRequestContextPtr ctx = {}) = 0;
00959
00991     virtual Note getNoteWithResultSpec(
00992         Guid guid,
00993         const NoteResultSpec & resultSpec,
00994         IRequestContextPtr ctx = {}) = 0;
00995
00997     virtual AsyncResult * getNoteWithResultSpecAsync(
00998         Guid guid,
00999         const NoteResultSpec & resultSpec,
01000         IRequestContextPtr ctx = {}) = 0;
01001
01009     virtual Note getNote(
01010         Guid guid,
01011         bool withContent,
01012         bool withResourcesData,
01013         bool withResourcesRecognition,
01014         bool withResourcesAlternateData,
01015         IRequestContextPtr ctx = {}) = 0;
01016
01018     virtual AsyncResult * getNoteAsync(
01019         Guid guid,
01020         bool withContent,
01021         bool withResourcesData,
01022         bool withResourcesRecognition,
01023         bool withResourcesAlternateData,
01024         IRequestContextPtr ctx = {}) = 0;
01025
01034     virtual LazyMap getNoteApplicationData(
01035         Guid guid,
01036         IRequestContextPtr ctx = {}) = 0;
01037
01039     virtual AsyncResult * getNoteApplicationDataAsync(
01040         Guid guid,
01041         IRequestContextPtr ctx = {}) = 0;
01042
01052     virtual QString getNoteApplicationDataEntry(
01053         Guid guid,
01054         QString key,
01055         IRequestContextPtr ctx = {}) = 0;
01056
01058     virtual AsyncResult * getNoteApplicationDataEntryAsync(
01059         Guid guid,
01060         QString key,
01061         IRequestContextPtr ctx = {}) = 0;
01062
01067     virtual qint32 setNoteApplicationDataEntry(
01068         Guid guid,
01069         QString key,
01070         QString value,
```

```
01071         IRequestContextPtr ctx = {}) = 0;
01072
01074     virtual AsyncResult * setNoteApplicationDataEntryAsync(
01075         Guid guid,
01076         QString key,
01077         QString value,
01078         IRequestContextPtr ctx = {}) = 0;
01079
01085     virtual qint32 unsetNoteApplicationDataEntry(
01086         Guid guid,
01087         QString key,
01088         IRequestContextPtr ctx = {}) = 0;
01089
01091     virtual AsyncResult * unsetNoteApplicationDataEntryAsync(
01092         Guid guid,
01093         QString key,
01094         IRequestContextPtr ctx = {}) = 0;
01095
01116     virtual QString getNoteContent(
01117         Guid guid,
01118         IRequestContextPtr ctx = {}) = 0;
01119
01121     virtual AsyncResult * getNoteContentAsync(
01122         Guid guid,
01123         IRequestContextPtr ctx = {}) = 0;
01124
01159     virtual QString getNoteSearchText(
01160         Guid guid,
01161         bool noteOnly,
01162         bool tokenizeForIndexing,
01163         IRequestContextPtr ctx = {}) = 0;
01164
01166     virtual AsyncResult * getNoteSearchTextAsync(
01167         Guid guid,
01168         bool noteOnly,
01169         bool tokenizeForIndexing,
01170         IRequestContextPtr ctx = {}) = 0;
01171
01196     virtual QString getResourceSearchText(
01197         Guid guid,
01198         IRequestContextPtr ctx = {}) = 0;
01199
01201     virtual AsyncResult * getResourceSearchTextAsync(
01202         Guid guid,
01203         IRequestContextPtr ctx = {}) = 0;
01204
01223     virtual QStringList getNoteTagNames(
01224         Guid guid,
01225         IRequestContextPtr ctx = {}) = 0;
01226
01228     virtual AsyncResult * getNoteTagNamesAsync(
01229         Guid guid,
01230         IRequestContextPtr ctx = {}) = 0;
01231
01295     virtual Note createNote(
01296         const Note & note,
01297         IRequestContextPtr ctx = {}) = 0;
01298
01300     virtual AsyncResult * createNoteAsync(
01301         const Note & note,
01302         IRequestContextPtr ctx = {}) = 0;
01303
01375     virtual Note updateNote(
01376         const Note & note,
01377         IRequestContextPtr ctx = {}) = 0;
01378
01380     virtual AsyncResult * updateNoteAsync(
01381         const Note & note,
01382         IRequestContextPtr ctx = {}) = 0;
01383
01410     virtual qint32 deleteNote(
01411         Guid guid,
01412         IRequestContextPtr ctx = {}) = 0;
01413
01415     virtual AsyncResult * deleteNoteAsync(
01416         Guid guid,
01417         IRequestContextPtr ctx = {}) = 0;
01418
01443     virtual qint32 expungeNote(
01444         Guid guid,
01445         IRequestContextPtr ctx = {}) = 0;
01446
01448     virtual AsyncResult * expungeNoteAsync(
01449         Guid guid,
01450         IRequestContextPtr ctx = {}) = 0;
01451
01494     virtual Note copyNote(
```

```
01495         Guid noteGuid,
01496         Guid toNotebookGuid,
01497         IRequestContextPtr ctx = {}) = 0;
01498
01500     virtual AsyncResult * copyNoteAsync(
01501         Guid noteGuid,
01502         Guid toNotebookGuid,
01503         IRequestContextPtr ctx = {}) = 0;
01504
01527     virtual QList<NoteVersionId> listNoteVersions(
01528         Guid noteGuid,
01529         IRequestContextPtr ctx = {}) = 0;
01530
01532     virtual AsyncResult * listNoteVersionsAsync(
01533         Guid noteGuid,
01534         IRequestContextPtr ctx = {}) = 0;
01535
01579     virtual Note getNoteVersion(
01580         Guid noteGuid,
01581         qint32 updateSequenceNum,
01582         bool withResourcesData,
01583         bool withResourcesRecognition,
01584         bool withResourcesAlternateData,
01585         IRequestContextPtr ctx = {}) = 0;
01586
01588     virtual AsyncResult * getNoteVersionAsync(
01589         Guid noteGuid,
01590         qint32 updateSequenceNum,
01591         bool withResourcesData,
01592         bool withResourcesRecognition,
01593         bool withResourcesAlternateData,
01594         IRequestContextPtr ctx = {}) = 0;
01595
01633     virtual Resource getResource(
01634         Guid guid,
01635         bool withData,
01636         bool withRecognition,
01637         bool withAttributes,
01638         bool withAlternateData,
01639         IRequestContextPtr ctx = {}) = 0;
01640
01642     virtual AsyncResult * getResourceAsync(
01643         Guid guid,
01644         bool withData,
01645         bool withRecognition,
01646         bool withAttributes,
01647         bool withAlternateData,
01648         IRequestContextPtr ctx = {}) = 0;
01649
01658     virtual LazyMap getResourceApplicationData(
01659         Guid guid,
01660         IRequestContextPtr ctx = {}) = 0;
01661
01663     virtual AsyncResult * getResourceApplicationDataAsync(
01664         Guid guid,
01665         IRequestContextPtr ctx = {}) = 0;
01666
01676     virtual QString getResourceApplicationDataEntry(
01677         Guid guid,
01678         QString key,
01679         IRequestContextPtr ctx = {}) = 0;
01680
01682     virtual AsyncResult * getResourceApplicationDataEntryAsync(
01683         Guid guid,
01684         QString key,
01685         IRequestContextPtr ctx = {}) = 0;
01686
01691     virtual qint32 setResourceApplicationDataEntry(
01692         Guid guid,
01693         QString key,
01694         QString value,
01695         IRequestContextPtr ctx = {}) = 0;
01696
01698     virtual AsyncResult * setResourceApplicationDataEntryAsync(
01699         Guid guid,
01700         QString key,
01701         QString value,
01702         IRequestContextPtr ctx = {}) = 0;
01703
01708     virtual qint32 unsetResourceApplicationDataEntry(
01709         Guid guid,
01710         QString key,
01711         IRequestContextPtr ctx = {}) = 0;
01712
01714     virtual AsyncResult * unsetResourceApplicationDataEntryAsync(
01715         Guid guid,
01716         QString key,
```

```
01717         IRequestContextPtr ctx = {}) = 0;
01718
01768     virtual qint32 updateResource(
01769         const Resource & resource,
01770         IRequestContextPtr ctx = {}) = 0;
01771
01773     virtual AsyncResult * updateResourceAsync(
01774         const Resource & resource,
01775         IRequestContextPtr ctx = {}) = 0;
01776
01799     virtual QByteArray getResourceData(
01800         Guid guid,
01801         IRequestContextPtr ctx = {}) = 0;
01802
01804     virtual AsyncResult * getResourceDataAsync(
01805         Guid guid,
01806         IRequestContextPtr ctx = {}) = 0;
01807
01849     virtual Resource getResourceByHash(
01850         Guid noteGuid,
01851         QByteArray contentHash,
01852         bool withData,
01853         bool withRecognition,
01854         bool withAlternateData,
01855         IRequestContextPtr ctx = {}) = 0;
01856
01858     virtual AsyncResult * getResourceByHashAsync(
01859         Guid noteGuid,
01860         QByteArray contentHash,
01861         bool withData,
01862         bool withRecognition,
01863         bool withAlternateData,
01864         IRequestContextPtr ctx = {}) = 0;
01865
01890     virtual QByteArray getResourceRecognition(
01891         Guid guid,
01892         IRequestContextPtr ctx = {}) = 0;
01893
01895     virtual AsyncResult * getResourceRecognitionAsync(
01896         Guid guid,
01897         IRequestContextPtr ctx = {}) = 0;
01898
01923     virtual QByteArray getResourceAlternateData(
01924         Guid guid,
01925         IRequestContextPtr ctx = {}) = 0;
01926
01928     virtual AsyncResult * getResourceAlternateDataAsync(
01929         Guid guid,
01930         IRequestContextPtr ctx = {}) = 0;
01931
01952     virtual ResourceAttributes getResourceAttributes(
01953         Guid guid,
01954         IRequestContextPtr ctx = {}) = 0;
01955
01957     virtual AsyncResult * getResourceAttributesAsync(
01958         Guid guid,
01959         IRequestContextPtr ctx = {}) = 0;
01960
01995     virtual Notebook getPublicNotebook(
01996         UserID userId,
01997         QString publicUri,
01998         IRequestContextPtr ctx = {}) = 0;
01999
02001     virtual AsyncResult * getPublicNotebookAsync(
02002         UserID userId,
02003         QString publicUri,
02004         IRequestContextPtr ctx = {}) = 0;
02005
02083     virtual SharedNotebook shareNotebook(
02084         const SharedNotebook & sharedNotebook,
02085         QString message,
02086         IRequestContextPtr ctx = {}) = 0;
02087
02089     virtual AsyncResult * shareNotebookAsync(
02090         const SharedNotebook & sharedNotebook,
02091         QString message,
02092         IRequestContextPtr ctx = {}) = 0;
02093
02148     virtual CreateOrUpdateNotebookSharesResult createOrUpdateNotebookShares(
02149         const NotebookShareTemplate & shareTemplate,
02150         IRequestContextPtr ctx = {}) = 0;
02151
02153     virtual AsyncResult * createOrUpdateNotebookSharesAsync(
02154         const NotebookShareTemplate & shareTemplate,
02155         IRequestContextPtr ctx = {}) = 0;
02156
02160     virtual qint32 updateSharedNotebook(
```

```
02161         const SharedNotebook & sharedNotebook,
02162         IRequestContextPtr ctx = {}) = 0;
02163
02165     virtual AsyncResult * updateSharedNotebookAsync(
02166         const SharedNotebook & sharedNotebook,
02167         IRequestContextPtr ctx = {}) = 0;
02168
02205     virtual Notebook setNotebookRecipientSettings(
02206         QString notebookGuid,
02207         const NotebookRecipientSettings & recipientSettings,
02208         IRequestContextPtr ctx = {}) = 0;
02209
02211     virtual AsyncResult * setNotebookRecipientSettingsAsync(
02212         QString notebookGuid,
02213         const NotebookRecipientSettings & recipientSettings,
02214         IRequestContextPtr ctx = {}) = 0;
02215
02223     virtual QList<SharedNotebook> listSharedNotebooks(
02224         IRequestContextPtr ctx = {}) = 0;
02225
02227     virtual AsyncResult * listSharedNotebooksAsync(
02228         IRequestContextPtr ctx = {}) = 0;
02229
02267     virtual LinkedNotebook createLinkedNotebook(
02268         const LinkedNotebook & linkedNotebook,
02269         IRequestContextPtr ctx = {}) = 0;
02270
02272     virtual AsyncResult * createLinkedNotebookAsync(
02273         const LinkedNotebook & linkedNotebook,
02274         IRequestContextPtr ctx = {}) = 0;
02275
02292     virtual qint32 updateLinkedNotebook(
02293         const LinkedNotebook & linkedNotebook,
02294         IRequestContextPtr ctx = {}) = 0;
02295
02297     virtual AsyncResult * updateLinkedNotebookAsync(
02298         const LinkedNotebook & linkedNotebook,
02299         IRequestContextPtr ctx = {}) = 0;
02300
02304     virtual QList<LinkedNotebook> listLinkedNotebooks(
02305         IRequestContextPtr ctx = {}) = 0;
02306
02308     virtual AsyncResult * listLinkedNotebooksAsync(
02309         IRequestContextPtr ctx = {}) = 0;
02310
02322     virtual qint32 expungeLinkedNotebook(
02323         Guid guid,
02324         IRequestContextPtr ctx = {}) = 0;
02325
02327     virtual AsyncResult * expungeLinkedNotebookAsync(
02328         Guid guid,
02329         IRequestContextPtr ctx = {}) = 0;
02330
02381     virtual AuthenticationResult authenticateToSharedNotebook(
02382         QString shareKeyOrGlobalId,
02383         IRequestContextPtr ctx = {}) = 0;
02384
02386     virtual AsyncResult * authenticateToSharedNotebookAsync(
02387         QString shareKeyOrGlobalId,
02388         IRequestContextPtr ctx = {}) = 0;
02389
02415     virtual SharedNotebook getSharedNotebookByAuth(
02416         IRequestContextPtr ctx = {}) = 0;
02417
02419     virtual AsyncResult * getSharedNotebookByAuthAsync(
02420         IRequestContextPtr ctx = {}) = 0;
02421
02471     virtual void emailNote(
02472         const NoteEmailParameters & parameters,
02473         IRequestContextPtr ctx = {}) = 0;
02474
02476     virtual AsyncResult * emailNoteAsync(
02477         const NoteEmailParameters & parameters,
02478         IRequestContextPtr ctx = {}) = 0;
02479
02503     virtual QString shareNote(
02504         Guid guid,
02505         IRequestContextPtr ctx = {}) = 0;
02506
02508     virtual AsyncResult * shareNoteAsync(
02509         Guid guid,
02510         IRequestContextPtr ctx = {}) = 0;
02511
02534     virtual void stopSharingNote(
02535         Guid guid,
02536         IRequestContextPtr ctx = {}) = 0;
02537
```

```
02539     virtual AsyncResult * stopSharingNoteAsync(
02540         Guid guid,
02541         IRequestContextPtr ctx = {}) = 0;
02542
02585     virtual AuthenticationResult authenticateToSharedNote(
02586         QString guid,
02587         QString noteKey,
02588         IRequestContextPtr ctx = {}) = 0;
02589
02591     virtual AsyncResult * authenticateToSharedNoteAsync(
02592         QString guid,
02593         QString noteKey,
02594         IRequestContextPtr ctx = {}) = 0;
02595
02645     virtual RelatedResult findRelated(
02646         const RelatedQuery & query,
02647         const RelatedResultSpec & resultSpec,
02648         IRequestContextPtr ctx = {}) = 0;
02649
02651     virtual AsyncResult * findRelatedAsync(
02652         const RelatedQuery & query,
02653         const RelatedResultSpec & resultSpec,
02654         IRequestContextPtr ctx = {}) = 0;
02655
02683     virtual UpdateNoteIfUsnMatchesResult updateNoteIfUsnMatches(
02684         const Note & note,
02685         IRequestContextPtr ctx = {}) = 0;
02686
02688     virtual AsyncResult * updateNoteIfUsnMatchesAsync(
02689         const Note & note,
02690         IRequestContextPtr ctx = {}) = 0;
02691
02708     virtual ManageNotebookSharesResult manageNotebookShares(
02709         const ManageNotebookSharesParameters & parameters,
02710         IRequestContextPtr ctx = {}) = 0;
02711
02713     virtual AsyncResult * manageNotebookSharesAsync(
02714         const ManageNotebookSharesParameters & parameters,
02715         IRequestContextPtr ctx = {}) = 0;
02716
02725     virtual ShareRelationships getNotebookShares(
02726         QString notebookGuid,
02727         IRequestContextPtr ctx = {}) = 0;
02728
02730     virtual AsyncResult * getNotebookSharesAsync(
02731         QString notebookGuid,
02732         IRequestContextPtr ctx = {}) = 0;
02733
02734 };
02735
02736 using INoteStorePtr = std::shared_ptr<INoteStore>;
02737
02739
02759 class QEVERCLOUD_EXPORT IUserStore: public QObject
02760 {
02761     Q_OBJECT
02762     Q_DISABLE_COPY(IUserStore)
02763 protected:
02764     IUserStore(QObject * parent) :
02765         QObject(parent)
02766     {}
02767
02768 public:
02769     virtual QString userStoreUrl() const = 0;
02770     virtual void setUserStoreUrl(QString url) = 0;
02771
02798     virtual bool checkVersion(
02799         QString clientName,
02800         qint16 edamVersionMajor = EDAM_VERSION_MAJOR,
02801         qint16 edamVersionMinor = EDAM_VERSION_MINOR,
02802         IRequestContextPtr ctx = {}) = 0;
02803
02805     virtual AsyncResult * checkVersionAsync(
02806         QString clientName,
02807         qint16 edamVersionMajor = EDAM_VERSION_MAJOR,
02808         qint16 edamVersionMinor = EDAM_VERSION_MINOR,
02809         IRequestContextPtr ctx = {}) = 0;
02810
02823     virtual BootstrapInfo getBootstrapInfo(
02824         QString locale,
02825         IRequestContextPtr ctx = {}) = 0;
02826
02828     virtual AsyncResult * getBootstrapInfoAsync(
02829         QString locale,
02830         IRequestContextPtr ctx = {}) = 0;
02831
02918     virtual AuthenticationResult authenticateLongSession(
```

```
02919         QString username,
02920         QString password,
02921         QString consumerKey,
02922         QString consumerSecret,
02923         QString deviceIdentifier,
02924         QString deviceDescription,
02925         bool supportsTwoFactor,
02926         IRequestContextPtr ctx = {}) = 0;
02927
02929     virtual AsyncResult * authenticateLongSessionAsync(
02930         QString username,
02931         QString password,
02932         QString consumerKey,
02933         QString consumerSecret,
02934         QString deviceIdentifier,
02935         QString deviceDescription,
02936         bool supportsTwoFactor,
02937         IRequestContextPtr ctx = {}) = 0;
02938
02977     virtual AuthenticationResult completeTwoFactorAuthentication(
02978         QString oneTimeCode,
02979         QString deviceIdentifier,
02980         QString deviceDescription,
02981         IRequestContextPtr ctx = {}) = 0;
02982
02984     virtual AsyncResult * completeTwoFactorAuthenticationAsync(
02985         QString oneTimeCode,
02986         QString deviceIdentifier,
02987         QString deviceDescription,
02988         IRequestContextPtr ctx = {}) = 0;
02989
03008     virtual void revokeLongSession(
03009         IRequestContextPtr ctx = {}) = 0;
03010
03012     virtual AsyncResult * revokeLongSessionAsync(
03013         IRequestContextPtr ctx = {}) = 0;
03014
03048     virtual AuthenticationResult authenticateToBusiness(
03049         IRequestContextPtr ctx = {}) = 0;
03050
03052     virtual AsyncResult * authenticateToBusinessAsync(
03053         IRequestContextPtr ctx = {}) = 0;
03054
03062     virtual User getUser(
03063         IRequestContextPtr ctx = {}) = 0;
03064
03066     virtual AsyncResult * getUserAsync(
03067         IRequestContextPtr ctx = {}) = 0;
03068
03077     virtual PublicUserInfo getPublicUserInfo(
03078         QString username,
03079         IRequestContextPtr ctx = {}) = 0;
03080
03082     virtual AsyncResult * getPublicUserInfoAsync(
03083         QString username,
03084         IRequestContextPtr ctx = {}) = 0;
03085
03095     virtual UserUrls getUserUrls(
03096         IRequestContextPtr ctx = {}) = 0;
03097
03099     virtual AsyncResult * getUserUrlsAsync(
03100         IRequestContextPtr ctx = {}) = 0;
03101
03145     virtual void inviteToBusiness(
03146         QString emailAddress,
03147         IRequestContextPtr ctx = {}) = 0;
03148
03150     virtual AsyncResult * inviteToBusinessAsync(
03151         QString emailAddress,
03152         IRequestContextPtr ctx = {}) = 0;
03153
03178     virtual void removeFromBusiness(
03179         QString emailAddress,
03180         IRequestContextPtr ctx = {}) = 0;
03181
03183     virtual AsyncResult * removeFromBusinessAsync(
03184         QString emailAddress,
03185         IRequestContextPtr ctx = {}) = 0;
03186
03229     virtual void updateBusinessUserIdentifier(
03230         QString oldEmailAddress,
03231         QString newEmailAddress,
03232         IRequestContextPtr ctx = {}) = 0;
03233
03235     virtual AsyncResult * updateBusinessUserIdentifierAsync(
03236         QString oldEmailAddress,
03237         QString newEmailAddress,
```

```
03238         IRequestContextPtr ctx = {}) = 0;
03239
03258     virtual QList<UserProfile> listBusinessUsers(
03259         IRequestContextPtr ctx = {}) = 0;
03260
03262     virtual AsyncResult * listBusinessUsersAsync(
03263         IRequestContextPtr ctx = {}) = 0;
03264
03279     virtual QList<BusinessInvitation> listBusinessInvitations(
03280         bool includeRequestedInvitations,
03281         IRequestContextPtr ctx = {}) = 0;
03282
03284     virtual AsyncResult * listBusinessInvitationsAsync(
03285         bool includeRequestedInvitations,
03286         IRequestContextPtr ctx = {}) = 0;
03287
03298     virtual AccountLimits getAccountLimits(
03299         ServiceLevel serviceLevel,
03300         IRequestContextPtr ctx = {}) = 0;
03301
03303     virtual AsyncResult * getAccountLimitsAsync(
03304         ServiceLevel serviceLevel,
03305         IRequestContextPtr ctx = {}) = 0;
03306
03307 };
03308
03309 using IUserStorePtr = std::shared_ptr<IUserStore>;
03310
03312
03313 QEVERCLOUD_EXPORT INoteStore * newNoteStore(
03314     QString noteStoreUrl = {},
03315     IRequestContextPtr ctx = {},
03316     QObject * parent = nullptr,
03317     IRetryPolicyPtr retryPolicy = {});
03318
03319 QEVERCLOUD_EXPORT IUserStore * newUserStore(
03320     QString userStoreUrl = {},
03321     IRequestContextPtr ctx = {},
03322     QObject * parent = nullptr,
03323     IRetryPolicyPtr retryPolicy = {});
03324
03325 } // namespace qevercloud
03326
03327 Q_DECLARE_METATYPE(QList<qevercloud::Notebook>)
03328 Q_DECLARE_METATYPE(QList<qevercloud::Tag>)
03329 Q_DECLARE_METATYPE(QList<qevercloud::SavedSearch>)
03330 Q_DECLARE_METATYPE(QList<qevercloud::NoteVersionId>)
03331 Q_DECLARE_METATYPE(QList<qevercloud::SharedNotebook>)
03332 Q_DECLARE_METATYPE(QList<qevercloud::LinkedNotebook>)
03333 Q_DECLARE_METATYPE(QList<qevercloud::BusinessInvitation>)
03334 Q_DECLARE_METATYPE(QList<qevercloud::UserProfile>)
03335
03336 #endif // QEVERCLOUD_GENERATED_SERVICES_H
```

8.21 Types.h File Reference

```
#include "../Export.h"
#include "../Optional.h"
#include "../Printable.h"
#include "EDAMErrorCode.h"
#include <QByteArray>
#include <QDateTime>
#include <QHash>
#include <QList>
#include <QMap>
#include <QMetaType>
#include <QSet>
#include <QStringList>
#include <QVariant>
```

Classes

- class qevercloud::EverCloudLocalData

The EverCloudLocalData class contains several data elements which are not synchronized with Evernote service but which are nevertheless useful in applications using QEverCloud to implement feature rich full sync Evernote clients. Values of this class' types are contained within QEverCloud types corresponding to actual Evernote API types.

- struct qevercloud::SyncState
- struct qevercloud::SyncChunkFilter
- struct qevercloud::NoteFilter
- struct qevercloud::NotesMetadataResultSpec
- struct qevercloud::NoteCollectionCounts
- struct qevercloud::NoteResultSpec
- struct qevercloud::NoteVersionId
- struct qevercloud::RelatedQuery
- struct qevercloud::RelatedResultSpec
- struct qevercloud::ShareRelationshipRestrictions
- struct qevercloud::MemberShareRelationship
- struct qevercloud::NoteShareRelationshipRestrictions
- struct qevercloud::NoteMemberShareRelationship
- struct qevercloud::NoteInvitationShareRelationship
- struct qevercloud::NoteShareRelationships
- struct qevercloud::ManageNoteSharesParameters
- struct qevercloud::Data
- struct qevercloud::UserAttributes
- struct qevercloud::BusinessUserAttributes
- struct qevercloud::Accounting
- struct qevercloud::BusinessUserInfo
- struct qevercloud::AccountLimits
- struct qevercloud::User
- struct qevercloud::Contact
- struct qevercloud::Identity
- struct qevercloud::Tag
- struct qevercloud::LazyMap
- struct qevercloud::ResourceAttributes
- struct qevercloud::Resource
- struct qevercloud::NoteAttributes
- struct qevercloud::SharedNote
- struct qevercloud::NoteRestrictions
- struct qevercloud::NoteLimits
- struct qevercloud::Note
- struct qevercloud::Publishing
- struct qevercloud::BusinessNotebook
- struct qevercloud::SavedSearchScope
- struct qevercloud::SavedSearch
- struct qevercloud::SharedNotebookRecipientSettings
- struct qevercloud::NotebookRecipientSettings
- struct qevercloud::SharedNotebook
- struct qevercloud::CanMoveToContainerRestrictions
- struct qevercloud::NotebookRestrictions
- struct qevercloud::Notebook
- struct qevercloud::LinkedNotebook
- struct qevercloud::NotebookDescriptor
- struct qevercloud::UserProfile
- struct qevercloud::RelatedContentImage
- struct qevercloud::RelatedContent
- struct qevercloud::BusinessInvitation
- struct qevercloud::UserIdentity
- struct qevercloud::PublicUserInfo

- struct qevercloud::UserUrls
- struct qevercloud::AuthenticationResult
- struct qevercloud::BootstrapSettings
- struct qevercloud::BootstrapProfile
- struct qevercloud::BootstrapInfo
- class qevercloud::EDAMUserException
- class qevercloud::EDAMSystemException
- class qevercloud::EDAMNotFoundException
- class qevercloud::EDAMInvalidContactsException
- struct qevercloud::SyncChunk
- struct qevercloud::NoteList
- struct qevercloud::NoteMetadata
- struct qevercloud::NotesMetadataList
- struct qevercloud::NoteEmailParameters
- struct qevercloud::RelatedResult
- struct qevercloud::UpdateNoteIfUsnMatchesResult
- struct qevercloud::InvitationShareRelationship
- struct qevercloud::ShareRelationships
- struct qevercloud::ManageNotebookSharesParameters
- struct qevercloud::ManageNotebookSharesError
- struct qevercloud::ManageNotebookSharesResult
- struct qevercloud::SharedNoteTemplate
- struct qevercloud::NotebookShareTemplate
- struct qevercloud::CreateOrUpdateNotebookSharesResult
- struct qevercloud::ManageNoteSharesError
- struct qevercloud::ManageNoteSharesResult

Namespaces

- namespace qevercloud

Typedefs

- using qevercloud::InvalidationSequenceNumber = qint64
- using qevercloud::IdentityID = qint64
- using qevercloud::UserID = qint32
- using qevercloud::Guid = QString
- using qevercloud::Timestamp = qint64
- using qevercloud::MessageEventID = qint64
- using qevercloud::MessageThreadID = qint64

8.22 Types.h

Go to the documentation of this file.

```
00001
00012 #ifndef QEVERCLOUD_GENERATED_TYPES_H
00013 #define QEVERCLOUD_GENERATED_TYPES_H
00014
00015 #include "../Export.h"
00016
00017 #include "../Optional.h"
00018 #include "../Printable.h"
00019 #include "EDAMErrorCode.h"
00020 #include <QByteArray>
00021 #include <QDateTime>
```

```
00022 #include <QHash>
00023 #include <QList>
00024 #include <QMap>
00025 #include <QMetaType>
00026 #include <QMetaType>
00027 #include <QSet>
00028 #include <QStringList>
00029 #include <QVariant>
00030
00031 namespace qevercloud {
00032
00037 using InvalidationSequenceNumber = qint64;
00038
00042 using IdentityID = qint64;
00043
00050 using UserID = qint32;
00051
00062 using Guid = QString;
00063
00081 using Timestamp = qint64;
00082
00086 using MessageEventID = qint64;
00087
00091 using MessageThreadID = qint64;
00092
00093
00102 class QEVERCLOUD_EXPORT EverCloudLocalData: public Printable
00103 {
00104     Q_GADGET
00105 public:
00106     EverCloudLocalData();
00107     virtual ~EverCloudLocalData() noexcept override;
00108
00109     virtual void print(QTextStream & strm) const override;
00110
00111     bool operator==(const EverCloudLocalData & other) const;
00112     bool operator!=(const EverCloudLocalData & other) const;
00121     QString id;
00122
00128     bool dirty = false;
00129
00135     bool local = false;
00136
00143     bool favorited = false;
00144
00149     QHash<QString, QVariant> dict;
00150
00151     // Properties declaration for meta-object system
00152     Q_PROPERTY(QString id MEMBER id USER true)
00153     Q_PROPERTY(bool dirty MEMBER dirty)
00154     Q_PROPERTY(bool local MEMBER local)
00155     Q_PROPERTY(bool favorited MEMBER favorited)
00156
00157     using Dict = QHash<QString, QVariant>;
00158     Q_PROPERTY(Dict dict MEMBER dict)
00159 };
00160
00165 struct QEVERCLOUD_EXPORT SyncState: public Printable
00166 {
00167 private:
00168     Q_GADGET
00169 public:
00173     EverCloudLocalData localData;
00174
00178     Timestamp currentTime = 0;
00186     Timestamp fullSyncBefore = 0;
00195     qint32 updateCount = 0;
00204     Optional<qint64> uploaded;
00224     Optional<Timestamp> userLastUpdated;
00232     Optional<MessageEventID> userMaxMessageEventId;
00233
00234     virtual void print(QTextStream & strm) const override;
00235
00236     bool operator==(const SyncState & other) const
00237     {
00238         return (currentTime == other.currentTime)
00239             && (fullSyncBefore == other.fullSyncBefore)
00240             && (updateCount == other.updateCount)
00241             && uploaded.isEqual(other.uploaded)
00242             && userLastUpdated.isEqual(other.userLastUpdated)
00243             && userMaxMessageEventId.isEqual(other.userMaxMessageEventId)
00244         ;
00245     }
00246
00247     bool operator!=(const SyncState & other) const
00248     {
00249         return !(*this == other);
```

```
00250     }
00251
00252     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
00253     Q_PROPERTY(Timestamp currentTime MEMBER currentTime)
00254     Q_PROPERTY(Timestamp fullSyncBefore MEMBER fullSyncBefore)
00255     Q_PROPERTY(qint32 updateCount MEMBER updateCount)
00256     Q_PROPERTY(Optional<qint64> uploaded MEMBER uploaded)
00257     Q_PROPERTY(Optional<Timestamp> userLastUpdated MEMBER userLastUpdated)
00258     Q_PROPERTY(Optional<MessageEventID> userMaxMessageEventId MEMBER userMaxMessageEventId)
00259 };
00260
00268 struct QEVERCLOUD_EXPORT SyncChunkFilter: public Printable
00269 {
00270 private:
00271     Q_GADGET
00272 public:
00276     EverCloudLocalData localData;
00277
00281     Optional<bool> includeNotes;
00287     Optional<bool> includeNoteResources;
00293     Optional<bool> includeNoteAttributes;
00297     Optional<bool> includeNotebooks;
00301     Optional<bool> includeTags;
00305     Optional<bool> includeSearches;
00313     Optional<bool> includeResources;
00317     Optional<bool> includeLinkedNotebooks;
00324     Optional<bool> includeExpunged;
00330     Optional<bool> includeNoteApplicationDataFullMap;
00337     Optional<bool> includeResourceApplicationDataFullMap;
00344     Optional<bool> includeNoteResourceApplicationDataFullMap;
00350     Optional<bool> includeSharedNotes;
00352     Optional<bool> omitSharedNotebooks;
00358     Optional<QString> requireNoteContentClass;
00376     Optional<QSet<QString>> notebookGuids;
00377
00378     virtual void print(QTextStream & strm) const override;
00379
00380     bool operator==(const SyncChunkFilter & other) const
00381     {
00382         return includeNotes.isEqual(other.includeNotes)
00383             && includeNoteResources.isEqual(other.includeNoteResources)
00384             && includeNoteAttributes.isEqual(other.includeNoteAttributes)
00385             && includeNotebooks.isEqual(other.includeNotebooks)
00386             && includeTags.isEqual(other.includeTags)
00387             && includeSearches.isEqual(other.includeSearches)
00388             && includeResources.isEqual(other.includeResources)
00389             && includeLinkedNotebooks.isEqual(other.includeLinkedNotebooks)
00390             && includeExpunged.isEqual(other.includeExpunged)
00391             && includeNoteApplicationDataFullMap.isEqual(other.includeNoteApplicationDataFullMap)
00392             &&
    includeResourceApplicationDataFullMap.isEqual(other.includeResourceApplicationDataFullMap)
00393             &&
    includeNoteResourceApplicationDataFullMap.isEqual(other.includeNoteResourceApplicationDataFullMap)
00394             && includeSharedNotes.isEqual(other.includeSharedNotes)
00395             && omitSharedNotebooks.isEqual(other.omitSharedNotebooks)
00396             && requireNoteContentClass.isEqual(other.requireNoteContentClass)
00397             && notebookGuids.isEqual(other.notebookGuids)
00398         ;
00399     }
00400
00401     bool operator!=(const SyncChunkFilter & other) const
00402     {
00403         return !(*this == other);
00404     }
00405
00406     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
00407     Q_PROPERTY(Optional<bool> includeNotes MEMBER includeNotes)
00408     Q_PROPERTY(Optional<bool> includeNoteResources MEMBER includeNoteResources)
00409     Q_PROPERTY(Optional<bool> includeNoteAttributes MEMBER includeNoteAttributes)
00410     Q_PROPERTY(Optional<bool> includeNotebooks MEMBER includeNotebooks)
00411     Q_PROPERTY(Optional<bool> includeTags MEMBER includeTags)
00412     Q_PROPERTY(Optional<bool> includeSearches MEMBER includeSearches)
00413     Q_PROPERTY(Optional<bool> includeResources MEMBER includeResources)
00414     Q_PROPERTY(Optional<bool> includeLinkedNotebooks MEMBER includeLinkedNotebooks)
00415     Q_PROPERTY(Optional<bool> includeExpunged MEMBER includeExpunged)
00416     Q_PROPERTY(Optional<bool> includeNoteApplicationDataFullMap MEMBER
    includeNoteApplicationDataFullMap)
00417     Q_PROPERTY(Optional<bool> includeResourceApplicationDataFullMap MEMBER
    includeResourceApplicationDataFullMap)
00418     Q_PROPERTY(Optional<bool> includeNoteResourceApplicationDataFullMap MEMBER
    includeNoteResourceApplicationDataFullMap)
00419     Q_PROPERTY(Optional<bool> includeSharedNotes MEMBER includeSharedNotes)
00420     Q_PROPERTY(Optional<bool> omitSharedNotebooks MEMBER omitSharedNotebooks)
00421     Q_PROPERTY(Optional<QString> requireNoteContentClass MEMBER requireNoteContentClass)
00422     Q_PROPERTY(Optional<QSet<QString» notebookGuids MEMBER notebookGuids)
00423 };
00424
```

```
00431 struct QEVERCLOUD_EXPORT NoteFilter: public Printable
00432 {
00433 private:
00434     Q_GADGET
00435 public:
00439     EverCloudLocalData localData;
00440
00445     Optional<qint32> order;
00450     Optional<bool> ascending;
00455     Optional<QString> words;
00460     Optional<Guid> notebookGuid;
00465     Optional<QList<Guid>> tagGuids;
00475     Optional<QString> timeZone;
00481     Optional<bool> inactive;
00488     Optional<QString> emphasized;
00494     Optional<bool> includeAllReadableNotebooks;
00500     Optional<bool> includeAllReadableWorkspaces;
00506     Optional<QString> context;
00511     Optional<QString> rawWords;
00518     Optional<QByteArray> searchContextBytes;
00519
00520     virtual void print(QTextStream & strm) const override;
00521
00522     bool operator==(const NoteFilter & other) const
00523     {
00524         return order.isEqual(other.order)
00525             && ascending.isEqual(other.ascending)
00526             && words.isEqual(other.words)
00527             && notebookGuid.isEqual(other.notebookGuid)
00528             && tagGuids.isEqual(other.tagGuids)
00529             && timeZone.isEqual(other.timeZone)
00530             && inactive.isEqual(other.inactive)
00531             && emphasized.isEqual(other.emphasized)
00532             && includeAllReadableNotebooks.isEqual(other.includeAllReadableNotebooks)
00533             && includeAllReadableWorkspaces.isEqual(other.includeAllReadableWorkspaces)
00534             && context.isEqual(other.context)
00535             && rawWords.isEqual(other.rawWords)
00536             && searchContextBytes.isEqual(other.searchContextBytes)
00537         ;
00538     }
00539
00540     bool operator!=(const NoteFilter & other) const
00541     {
00542         return !(*this == other);
00543     }
00544
00545     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
00546     Q_PROPERTY(Optional<qint32> order MEMBER order)
00547     Q_PROPERTY(Optional<bool> ascending MEMBER ascending)
00548     Q_PROPERTY(Optional<QString> words MEMBER words)
00549     Q_PROPERTY(Optional<Guid> notebookGuid MEMBER notebookGuid)
00550     Q_PROPERTY(Optional<QList<Guid>> tagGuids MEMBER tagGuids)
00551     Q_PROPERTY(Optional<QString> timeZone MEMBER timeZone)
00552     Q_PROPERTY(Optional<bool> inactive MEMBER inactive)
00553     Q_PROPERTY(Optional<QString> emphasized MEMBER emphasized)
00554     Q_PROPERTY(Optional<bool> includeAllReadableNotebooks MEMBER includeAllReadableNotebooks)
00555     Q_PROPERTY(Optional<bool> includeAllReadableWorkspaces MEMBER includeAllReadableWorkspaces)
00556     Q_PROPERTY(Optional<QString> context MEMBER context)
00557     Q_PROPERTY(Optional<QString> rawWords MEMBER rawWords)
00558     Q_PROPERTY(Optional<QByteArray> searchContextBytes MEMBER searchContextBytes)
00559 };
00560
00574 struct QEVERCLOUD_EXPORT NotesMetadataResultSpec: public Printable
00575 {
00576 private:
00577     Q_GADGET
00578 public:
00582     EverCloudLocalData localData;
00583
00585     Optional<bool> includeTitle;
00587     Optional<bool> includeContentLength;
00589     Optional<bool> includeCreated;
00591     Optional<bool> includeUpdated;
00593     Optional<bool> includeDeleted;
00595     Optional<bool> includeUpdateSequenceNum;
00597     Optional<bool> includeNotebookGuid;
00599     Optional<bool> includeTagGuids;
00601     Optional<bool> includeAttributes;
00603     Optional<bool> includeLargestResourceMime;
00605     Optional<bool> includeLargestResourceSize;
00606
00607     virtual void print(QTextStream & strm) const override;
00608
00609     bool operator==(const NotesMetadataResultSpec & other) const
00610     {
00611         return includeTitle.isEqual(other.includeTitle)
00612             && includeContentLength.isEqual(other.includeContentLength)
```

```
00613             && includeCreated.isEqual(other.includeCreated)
00614             && includeUpdated.isEqual(other.includeUpdated)
00615             && includeDeleted.isEqual(other.includeDeleted)
00616             && includeUpdateSequenceNum.isEqual(other.includeUpdateSequenceNum)
00617             && includeNotebookGuid.isEqual(other.includeNotebookGuid)
00618             && includeTagGuids.isEqual(other.includeTagGuids)
00619             && includeAttributes.isEqual(other.includeAttributes)
00620             && includeLargestResourceMime.isEqual(other.includeLargestResourceMime)
00621             && includeLargestResourceSize.isEqual(other.includeLargestResourceSize)
00622         ;
00623     }
00624
00625     bool operator!=(const NotesMetadataResultSpec & other) const
00626     {
00627         return !(*this == other);
00628     }
00629
00630     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
00631     Q_PROPERTY(Optional<bool> includeTitle MEMBER includeTitle)
00632     Q_PROPERTY(Optional<bool> includeContentLength MEMBER includeContentLength)
00633     Q_PROPERTY(Optional<bool> includeCreated MEMBER includeCreated)
00634     Q_PROPERTY(Optional<bool> includeUpdated MEMBER includeUpdated)
00635     Q_PROPERTY(Optional<bool> includeDeleted MEMBER includeDeleted)
00636     Q_PROPERTY(Optional<bool> includeUpdateSequenceNum MEMBER includeUpdateSequenceNum)
00637     Q_PROPERTY(Optional<bool> includeNotebookGuid MEMBER includeNotebookGuid)
00638     Q_PROPERTY(Optional<bool> includeTagGuids MEMBER includeTagGuids)
00639     Q_PROPERTY(Optional<bool> includeAttributes MEMBER includeAttributes)
00640     Q_PROPERTY(Optional<bool> includeLargestResourceMime MEMBER includeLargestResourceMime)
00641     Q_PROPERTY(Optional<bool> includeLargestResourceSize MEMBER includeLargestResourceSize)
00642 };
00643
00649 struct QEVERCLOUD_EXPORT NoteCollectionCounts: public Printable
00650 {
00651 private:
00652     Q_GADGET
00653 public:
00657     EverCloudLocalData localData;
00658
00663     Optional<QMap<Guid, qint32>> notebookCounts;
00668     Optional<QMap<Guid, qint32>> tagCounts;
00675     Optional<qint32> trashCount;
00676
00677     virtual void print(QTextStream & strm) const override;
00678
00679     bool operator==(const NoteCollectionCounts & other) const
00680     {
00681         return notebookCounts.isEqual(other.notebookCounts)
00682             && tagCounts.isEqual(other.tagCounts)
00683             && trashCount.isEqual(other.trashCount)
00684         ;
00685     }
00686
00687     bool operator!=(const NoteCollectionCounts & other) const
00688     {
00689         return !(*this == other);
00690     }
00691
00692     using TagCounts = QMap<Guid, qint32>;
00693
00694     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
00695     Q_PROPERTY(Optional<TagCounts> notebookCounts MEMBER notebookCounts)
00696     Q_PROPERTY(Optional<TagCounts> tagCounts MEMBER tagCounts)
00697     Q_PROPERTY(Optional<qint32> trashCount MEMBER trashCount)
00698 };
00699
00710 struct QEVERCLOUD_EXPORT NoteResultSpec: public Printable
00711 {
00712 private:
00713     Q_GADGET
00714 public:
00718     EverCloudLocalData localData;
00719
00723     Optional<bool> includeContent;
00728     Optional<bool> includeResourcesData;
00733     Optional<bool> includeResourcesRecognition;
00738     Optional<bool> includeResourcesAlternateData;
00742     Optional<bool> includeSharedNotes;
00746     Optional<bool> includeNoteAppDataValues;
00750     Optional<bool> includeResourceAppDataValues;
00754     Optional<bool> includeAccountLimits;
00755
00756     virtual void print(QTextStream & strm) const override;
00757
00758     bool operator==(const NoteResultSpec & other) const
00759     {
00760         return includeContent.isEqual(other.includeContent)
00761             && includeResourcesData.isEqual(other.includeResourcesData)
```

```
00762             && includeResourcesRecognition.isEqual(other.includeResourcesRecognition)
00763             && includeResourcesAlternateData.isEqual(other.includeResourcesAlternateData)
00764             && includeSharedNotes.isEqual(other.includeSharedNotes)
00765             && includeNoteAppDataValues.isEqual(other.includeNoteAppDataValues)
00766             && includeResourceAppDataValues.isEqual(other.includeResourceAppDataValues)
00767             && includeAccountLimits.isEqual(other.includeAccountLimits)
00768         ;
00769     }
00770
00771     bool operator!=(const NoteResultSpec & other) const
00772     {
00773         return !(*this == other);
00774     }
00775
00776     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
00777     Q_PROPERTY(Optional<bool> includeContent MEMBER includeContent)
00778     Q_PROPERTY(Optional<bool> includeResourcesData MEMBER includeResourcesData)
00779     Q_PROPERTY(Optional<bool> includeResourcesRecognition MEMBER includeResourcesRecognition)
00780     Q_PROPERTY(Optional<bool> includeResourcesAlternateData MEMBER includeResourcesAlternateData)
00781     Q_PROPERTY(Optional<bool> includeSharedNotes MEMBER includeSharedNotes)
00782     Q_PROPERTY(Optional<bool> includeNoteAppDataValues MEMBER includeNoteAppDataValues)
00783     Q_PROPERTY(Optional<bool> includeResourceAppDataValues MEMBER includeResourceAppDataValues)
00784     Q_PROPERTY(Optional<bool> includeAccountLimits MEMBER includeAccountLimits)
00785 };
00786
00793 struct QEVERCLOUD_EXPORT NoteVersionId: public Printable
00794 {
00795 private:
00796     Q_GADGET
00797 public:
00801     EverCloudLocalData localData;
00802
00808     qint32 updateSequenceNum = 0;
00816     Timestamp updated = 0;
00821     Timestamp saved = 0;
00826     QString title;
00831     Optional<UserID> lastEditorId;
00832
00833     virtual void print(QTextStream & strm) const override;
00834
00835     bool operator==(const NoteVersionId & other) const
00836     {
00837         return (updateSequenceNum == other.updateSequenceNum)
00838             && (updated == other.updated)
00839             && (saved == other.saved)
00840             && (title == other.title)
00841             && lastEditorId.isEqual(other.lastEditorId)
00842         ;
00843     }
00844
00845     bool operator!=(const NoteVersionId & other) const
00846     {
00847         return !(*this == other);
00848     }
00849
00850     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
00851     Q_PROPERTY(qint32 updateSequenceNum MEMBER updateSequenceNum)
00852     Q_PROPERTY(Timestamp updated MEMBER updated)
00853     Q_PROPERTY(Timestamp saved MEMBER saved)
00854     Q_PROPERTY(QString title MEMBER title)
00855     Q_PROPERTY(Optional<UserID> lastEditorId MEMBER lastEditorId)
00856 };
00857
00866 struct QEVERCLOUD_EXPORT RelatedQuery: public Printable
00867 {
00868 private:
00869     Q_GADGET
00870 public:
00874     EverCloudLocalData localData;
00875
00880     Optional<QString> noteGuid;
00886     Optional<QString> plainText;
00893     Optional<NoteFilter> filter;
00898     Optional<QString> referenceUri;
00904     Optional<QString> context;
00916     Optional<QString> cacheKey;
00917
00918     virtual void print(QTextStream & strm) const override;
00919
00920     bool operator==(const RelatedQuery & other) const
00921     {
00922         return noteGuid.isEqual(other.noteGuid)
00923             && plainText.isEqual(other.plainText)
00924             && filter.isEqual(other.filter)
00925             && referenceUri.isEqual(other.referenceUri)
00926             && context.isEqual(other.context)
00927             && cacheKey.isEqual(other.cacheKey)
```

```
00928         ;
00929     }
00930
00931     bool operator!=(const RelatedQuery & other) const
00932     {
00933         return !(*this == other);
00934     }
00935
00936     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
00937     Q_PROPERTY(Optional<QString> noteGuid MEMBER noteGuid)
00938     Q_PROPERTY(Optional<QString> plainText MEMBER plainText)
00939     Q_PROPERTY(Optional<NoteFilter> filter MEMBER filter)
00940     Q_PROPERTY(Optional<QString> referenceUri MEMBER referenceUri)
00941     Q_PROPERTY(Optional<QString> context MEMBER context)
00942     Q_PROPERTY(Optional<QString> cacheKey MEMBER cacheKey)
00943 };
00944
00952 struct QEVERCLOUD_EXPORT RelatedResultSpec: public Printable
00953 {
00954 private:
00955     Q_GADGET
00956 public:
00960     EverCloudLocalData localData;
00961
00968     Optional<qint32> maxNotes;
00975     Optional<qint32> maxNotebooks;
00982     Optional<qint32> maxTags;
00989     Optional<bool> writableNotebooksOnly;
00995     Optional<bool> includeContainingNotebooks;
01001     Optional<bool> includeDebugInfo;
01008     Optional<qint32> maxExperts;
01014     Optional<qint32> maxRelatedContent;
01018     Optional<QSet<RelatedContentType>> relatedContentTypes;
01019
01020     virtual void print(QTextStream & strm) const override;
01021
01022     bool operator==(const RelatedResultSpec & other) const
01023     {
01024         return maxNotes.isEqual(other.maxNotes)
01025             && maxNotebooks.isEqual(other.maxNotebooks)
01026             && maxTags.isEqual(other.maxTags)
01027             && writableNotebooksOnly.isEqual(other.writableNotebooksOnly)
01028             && includeContainingNotebooks.isEqual(other.includeContainingNotebooks)
01029             && includeDebugInfo.isEqual(other.includeDebugInfo)
01030             && maxExperts.isEqual(other.maxExperts)
01031             && maxRelatedContent.isEqual(other.maxRelatedContent)
01032             && relatedContentTypes.isEqual(other.relatedContentTypes)
01033         ;
01034     }
01035
01036     bool operator!=(const RelatedResultSpec & other) const
01037     {
01038         return !(*this == other);
01039     }
01040
01041     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
01042     Q_PROPERTY(Optional<qint32> maxNotes MEMBER maxNotes)
01043     Q_PROPERTY(Optional<qint32> maxNotebooks MEMBER maxNotebooks)
01044     Q_PROPERTY(Optional<qint32> maxTags MEMBER maxTags)
01045     Q_PROPERTY(Optional<bool> writableNotebooksOnly MEMBER writableNotebooksOnly)
01046     Q_PROPERTY(Optional<bool> includeContainingNotebooks MEMBER includeContainingNotebooks)
01047     Q_PROPERTY(Optional<bool> includeDebugInfo MEMBER includeDebugInfo)
01048     Q_PROPERTY(Optional<qint32> maxExperts MEMBER maxExperts)
01049     Q_PROPERTY(Optional<qint32> maxRelatedContent MEMBER maxRelatedContent)
01050     Q_PROPERTY(Optional<QSet<RelatedContentType» relatedContentTypes MEMBER relatedContentTypes)
01051 };
01052
01054 struct QEVERCLOUD_EXPORT ShareRelationshipRestrictions: public Printable
01055 {
01056 private:
01057     Q_GADGET
01058 public:
01062     EverCloudLocalData localData;
01063
01065     Optional<bool> noSetReadOnly;
01067     Optional<bool> noSetReadPlusActivity;
01069     Optional<bool> noSetModify;
01071     Optional<bool> noSetFullAccess;
01072
01073     virtual void print(QTextStream & strm) const override;
01074
01075     bool operator==(const ShareRelationshipRestrictions & other) const
01076     {
01077         return noSetReadOnly.isEqual(other.noSetReadOnly)
01078             && noSetReadPlusActivity.isEqual(other.noSetReadPlusActivity)
01079             && noSetModify.isEqual(other.noSetModify)
01080             && noSetFullAccess.isEqual(other.noSetFullAccess)
```

```
01081         ;
01082     }
01083
01084     bool operator!=(const ShareRelationshipRestrictions & other) const
01085     {
01086         return !(*this == other);
01087     }
01088
01089     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
01090     Q_PROPERTY(Optional<bool> noSetReadOnly MEMBER noSetReadOnly)
01091     Q_PROPERTY(Optional<bool> noSetReadPlusActivity MEMBER noSetReadPlusActivity)
01092     Q_PROPERTY(Optional<bool> noSetModify MEMBER noSetModify)
01093     Q_PROPERTY(Optional<bool> noSetFullAccess MEMBER noSetFullAccess)
01094 };
01095
01101 struct QEVERCLOUD_EXPORT MemberShareRelationship: public Printable
01102 {
01103 private:
01104     Q_GADGET
01105 public:
01109     EverCloudLocalData localData;
01110
01115     Optional<QString> displayName;
01119     Optional<UserID> recipientUserId;
01127     Optional<ShareRelationshipPrivilegeLevel> bestPrivilege;
01137     Optional<ShareRelationshipPrivilegeLevel> individualPrivilege;
01142     Optional<ShareRelationshipRestrictions> restrictions;
01151     Optional<UserID> sharerUserId;
01152
01153     virtual void print(QTextStream & strm) const override;
01154
01155     bool operator==(const MemberShareRelationship & other) const
01156     {
01157         return displayName.isEqual(other.displayName)
01158             && recipientUserId.isEqual(other.recipientUserId)
01159             && bestPrivilege.isEqual(other.bestPrivilege)
01160             && individualPrivilege.isEqual(other.individualPrivilege)
01161             && restrictions.isEqual(other.restrictions)
01162             && sharerUserId.isEqual(other.sharerUserId)
01163         ;
01164     }
01165
01166     bool operator!=(const MemberShareRelationship & other) const
01167     {
01168         return !(*this == other);
01169     }
01170
01171     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
01172     Q_PROPERTY(Optional<QString> displayName MEMBER displayName)
01173     Q_PROPERTY(Optional<UserID> recipientUserId MEMBER recipientUserId)
01174     Q_PROPERTY(Optional<ShareRelationshipPrivilegeLevel> bestPrivilege MEMBER bestPrivilege)
01175     Q_PROPERTY(Optional<ShareRelationshipPrivilegeLevel> individualPrivilege MEMBER
    individualPrivilege)
01176     Q_PROPERTY(Optional<ShareRelationshipRestrictions> restrictions MEMBER restrictions)
01177     Q_PROPERTY(Optional<UserID> sharerUserId MEMBER sharerUserId)
01178 };
01179
01186 struct QEVERCLOUD_EXPORT NoteShareRelationshipRestrictions: public Printable
01187 {
01188 private:
01189     Q_GADGET
01190 public:
01194     EverCloudLocalData localData;
01195
01200     Optional<bool> noSetReadNote;
01205     Optional<bool> noSetModifyNote;
01210     Optional<bool> noSetFullAccess;
01211
01212     virtual void print(QTextStream & strm) const override;
01213
01214     bool operator==(const NoteShareRelationshipRestrictions & other) const
01215     {
01216         return noSetReadNote.isEqual(other.noSetReadNote)
01217             && noSetModifyNote.isEqual(other.noSetModifyNote)
01218             && noSetFullAccess.isEqual(other.noSetFullAccess)
01219         ;
01220     }
01221
01222     bool operator!=(const NoteShareRelationshipRestrictions & other) const
01223     {
01224         return !(*this == other);
01225     }
01226
01227     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
01228     Q_PROPERTY(Optional<bool> noSetReadNote MEMBER noSetReadNote)
01229     Q_PROPERTY(Optional<bool> noSetModifyNote MEMBER noSetModifyNote)
01230     Q_PROPERTY(Optional<bool> noSetFullAccess MEMBER noSetFullAccess)
```

```
01231 };
01232
01238 struct QEVERCLOUD_EXPORT NoteMemberShareRelationship: public Printable
01239 {
01240 private:
01241     Q_GADGET
01242 public:
01246     EverCloudLocalData localData;
01247
01252     Optional<QString> displayName;
01256     Optional<UserID> recipientUserId;
01264     Optional<SharedNotePrivilegeLevel> privilege;
01271     Optional<NoteShareRelationshipRestrictions> restrictions;
01277     Optional<UserID> sharerUserId;
01278
01279     virtual void print(QTextStream & strm) const override;
01280
01281     bool operator==(const NoteMemberShareRelationship & other) const
01282     {
01283         return displayName.isEqual(other.displayName)
01284             && recipientUserId.isEqual(other.recipientUserId)
01285             && privilege.isEqual(other.privilege)
01286             && restrictions.isEqual(other.restrictions)
01287             && sharerUserId.isEqual(other.sharerUserId)
01288         ;
01289     }
01290
01291     bool operator!=(const NoteMemberShareRelationship & other) const
01292     {
01293         return !(*this == other);
01294     }
01295
01296     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
01297     Q_PROPERTY(Optional<QString> displayName MEMBER displayName)
01298     Q_PROPERTY(Optional<UserID> recipientUserId MEMBER recipientUserId)
01299     Q_PROPERTY(Optional<SharedNotePrivilegeLevel> privilege MEMBER privilege)
01300     Q_PROPERTY(Optional<NoteShareRelationshipRestrictions> restrictions MEMBER restrictions)
01301     Q_PROPERTY(Optional<UserID> sharerUserId MEMBER sharerUserId)
01302 };
01303
01309 struct QEVERCLOUD_EXPORT NoteInvitationShareRelationship: public Printable
01310 {
01311 private:
01312     Q_GADGET
01313 public:
01317     EverCloudLocalData localData;
01318
01323     Optional<QString> displayName;
01331     Optional<IdentityID> recipientIdentityId;
01337     Optional<SharedNotePrivilegeLevel> privilege;
01343     Optional<UserID> sharerUserId;
01344
01345     virtual void print(QTextStream & strm) const override;
01346
01347     bool operator==(const NoteInvitationShareRelationship & other) const
01348     {
01349         return displayName.isEqual(other.displayName)
01350             && recipientIdentityId.isEqual(other.recipientIdentityId)
01351             && privilege.isEqual(other.privilege)
01352             && sharerUserId.isEqual(other.sharerUserId)
01353         ;
01354     }
01355
01356     bool operator!=(const NoteInvitationShareRelationship & other) const
01357     {
01358         return !(*this == other);
01359     }
01360
01361     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
01362     Q_PROPERTY(Optional<QString> displayName MEMBER displayName)
01363     Q_PROPERTY(Optional<IdentityID> recipientIdentityId MEMBER recipientIdentityId)
01364     Q_PROPERTY(Optional<SharedNotePrivilegeLevel> privilege MEMBER privilege)
01365     Q_PROPERTY(Optional<UserID> sharerUserId MEMBER sharerUserId)
01366 };
01367
01375 struct QEVERCLOUD_EXPORT NoteShareRelationships: public Printable
01376 {
01377 private:
01378     Q_GADGET
01379 public:
01383     EverCloudLocalData localData;
01384
01389     Optional<QList<NoteInvitationShareRelationship>> invitations;
01395     Optional<QList<NoteMemberShareRelationship>> memberships;
01397     Optional<NoteShareRelationshipRestrictions> invitationRestrictions;
01398
01399     virtual void print(QTextStream & strm) const override;
```

```
01400
01401     bool operator==(const NoteShareRelationships & other) const
01402     {
01403         return invitations.isEqual(other.invitations)
01404             && memberships.isEqual(other.memberships)
01405             && invitationRestrictions.isEqual(other.invitationRestrictions)
01406         ;
01407     }
01408
01409     bool operator!=(const NoteShareRelationships & other) const
01410     {
01411         return !(*this == other);
01412     }
01413
01414     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
01415     Q_PROPERTY(Optional<QList<NoteInvitationShareRelationship» invitations MEMBER invitations)
01416     Q_PROPERTY(Optional<QList<NoteMemberShareRelationship» memberships MEMBER memberships)
01417     Q_PROPERTY(Optional<NoteShareRelationshipRestrictions> invitationRestrictions MEMBER
      invitationRestrictions)
01418 };
01419
01430 struct QEVERCLOUD_EXPORT ManageNoteSharesParameters: public Printable
01431 {
01432 private:
01433     Q_GADGET
01434 public:
01438     EverCloudLocalData localData;
01439
01443     Optional<QString> noteGuid;
01451     Optional<QList<NoteMemberShareRelationship>> membershipsToUpdate;
01459     Optional<QList<NoteInvitationShareRelationship>> invitationsToUpdate;
01463     Optional<QList<UserID>> membershipsToUnshare;
01467     Optional<QList<IdentityID>> invitationsToUnshare;
01468
01469     virtual void print(QTextStream & strm) const override;
01470
01471     bool operator==(const ManageNoteSharesParameters & other) const
01472     {
01473         return noteGuid.isEqual(other.noteGuid)
01474             && membershipsToUpdate.isEqual(other.membershipsToUpdate)
01475             && invitationsToUpdate.isEqual(other.invitationsToUpdate)
01476             && membershipsToUnshare.isEqual(other.membershipsToUnshare)
01477             && invitationsToUnshare.isEqual(other.invitationsToUnshare)
01478         ;
01479     }
01480
01481     bool operator!=(const ManageNoteSharesParameters & other) const
01482     {
01483         return !(*this == other);
01484     }
01485
01486     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
01487     Q_PROPERTY(Optional<QString> noteGuid MEMBER noteGuid)
01488     Q_PROPERTY(Optional<QList<NoteMemberShareRelationship» membershipsToUpdate MEMBER
      membershipsToUpdate)
01489     Q_PROPERTY(Optional<QList<NoteInvitationShareRelationship» invitationsToUpdate MEMBER
      invitationsToUpdate)
01490     Q_PROPERTY(Optional<QList<UserID» membershipsToUnshare MEMBER membershipsToUnshare)
01491     Q_PROPERTY(Optional<QList<IdentityID» invitationsToUnshare MEMBER invitationsToUnshare)
01492 };
01493
01503 struct QEVERCLOUD_EXPORT Data: public Printable
01504 {
01505 private:
01506     Q_GADGET
01507 public:
01511     EverCloudLocalData localData;
01512
01518     Optional<QByteArray> bodyHash;
01522     Optional<qint32> size;
01530     Optional<QByteArray> body;
01531
01532     virtual void print(QTextStream & strm) const override;
01533
01534     bool operator==(const Data & other) const
01535     {
01536         return bodyHash.isEqual(other.bodyHash)
01537             && size.isEqual(other.size)
01538             && body.isEqual(other.body)
01539         ;
01540     }
01541
01542     bool operator!=(const Data & other) const
01543     {
01544         return !(*this == other);
01545     }
01546
```

```
01547     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
01548     Q_PROPERTY(Optional<QByteArray> bodyHash MEMBER bodyHash)
01549     Q_PROPERTY(Optional<qint32> size MEMBER size)
01550     Q_PROPERTY(Optional<QByteArray> body MEMBER body)
01551 };
01552
01558 struct QEVERCLOUD_EXPORT UserAttributes: public Printable
01559 {
01560 private:
01561     Q_GADGET
01562 public:
01566     EverCloudLocalData localData;
01567
01574     Optional<QString> defaultLocationName;
01579     Optional<double> defaultLatitude;
01584     Optional<double> defaultLongitude;
01590     Optional<bool> preactivation;
01597     Optional<QStringList> viewedPromotions;
01606     Optional<QString> incomingEmailAddress;
01616     Optional<QStringList> recentMailedAddresses;
01622     Optional<QString> comments;
01628     Optional<Timestamp> dateAgreedToTermsOfService;
01633     Optional<qint32> maxReferrals;
01637     Optional<qint32> referralCount;
01642     Optional<QString> refererCode;
01648     Optional<Timestamp> sentEmailDate;
01654     Optional<qint32> sentEmailCount;
01660     Optional<qint32> dailyEmailLimit;
01666     Optional<Timestamp> emailOptOutDate;
01673     Optional<Timestamp> partnerEmailOptInDate;
01680     Optional<QString> preferredLanguage;
01685     Optional<QString> preferredCountry;
01690     Optional<bool> clipFullPage;
01696     Optional<QString> twitterUserName;
01701     Optional<QString> twitterId;
01706     Optional<QString> groupName;
01714     Optional<QString> recognitionLanguage;
01716     Optional<QString> referralProof;
01718     Optional<bool> educationalDiscount;
01722     Optional<QString> businessAddress;
01727     Optional<bool> hideSponsorBilling;
01732     Optional<bool> useEmailAutoFiling;
01740     Optional<ReminderEmailConfig> reminderEmailConfig;
01746     Optional<Timestamp> emailAddressLastConfirmed;
01752     Optional<Timestamp> passwordUpdated;
01754     Optional<bool> salesforcePushEnabled;
01759     Optional<bool> shouldLogClientEvent;
01764     Optional<bool> optOutMachineLearning;
01765
01766     virtual void print(QTextStream & strm) const override;
01767
01768     bool operator==(const UserAttributes & other) const
01769     {
01770         return defaultLocationName.isEqual(other.defaultLocationName)
01771             && defaultLatitude.isEqual(other.defaultLatitude)
01772             && defaultLongitude.isEqual(other.defaultLongitude)
01773             && preactivation.isEqual(other.preactivation)
01774             && viewedPromotions.isEqual(other.viewedPromotions)
01775             && incomingEmailAddress.isEqual(other.incomingEmailAddress)
01776             && recentMailedAddresses.isEqual(other.recentMailedAddresses)
01777             && comments.isEqual(other.comments)
01778             && dateAgreedToTermsOfService.isEqual(other.dateAgreedToTermsOfService)
01779             && maxReferrals.isEqual(other.maxReferrals)
01780             && referralCount.isEqual(other.referralCount)
01781             && refererCode.isEqual(other.refererCode)
01782             && sentEmailDate.isEqual(other.sentEmailDate)
01783             && sentEmailCount.isEqual(other.sentEmailCount)
01784             && dailyEmailLimit.isEqual(other.dailyEmailLimit)
01785             && emailOptOutDate.isEqual(other.emailOptOutDate)
01786             && partnerEmailOptInDate.isEqual(other.partnerEmailOptInDate)
01787             && preferredLanguage.isEqual(other.preferredLanguage)
01788             && preferredCountry.isEqual(other.preferredCountry)
01789             && clipFullPage.isEqual(other.clipFullPage)
01790             && twitterUserName.isEqual(other.twitterUserName)
01791             && twitterId.isEqual(other.twitterId)
01792             && groupName.isEqual(other.groupName)
01793             && recognitionLanguage.isEqual(other.recognitionLanguage)
01794             && referralProof.isEqual(other.referralProof)
01795             && educationalDiscount.isEqual(other.educationalDiscount)
01796             && businessAddress.isEqual(other.businessAddress)
01797             && hideSponsorBilling.isEqual(other.hideSponsorBilling)
01798             && useEmailAutoFiling.isEqual(other.useEmailAutoFiling)
01799             && reminderEmailConfig.isEqual(other.reminderEmailConfig)
01800             && emailAddressLastConfirmed.isEqual(other.emailAddressLastConfirmed)
01801             && passwordUpdated.isEqual(other.passwordUpdated)
01802             && salesforcePushEnabled.isEqual(other.salesforcePushEnabled)
01803             && shouldLogClientEvent.isEqual(other.shouldLogClientEvent)
```

```
01804             && optOutMachineLearning.isEqual(other.optOutMachineLearning)
01805         ;
01806     }
01807
01808     bool operator!=(const UserAttributes & other) const
01809     {
01810         return !(*this == other);
01811     }
01812
01813     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
01814     Q_PROPERTY(Optional<QString> defaultLocationName MEMBER defaultLocationName)
01815     Q_PROPERTY(Optional<double> defaultLatitude MEMBER defaultLatitude)
01816     Q_PROPERTY(Optional<double> defaultLongitude MEMBER defaultLongitude)
01817     Q_PROPERTY(Optional<bool> preactivation MEMBER preactivation)
01818     Q_PROPERTY(Optional<QStringList> viewedPromotions MEMBER viewedPromotions)
01819     Q_PROPERTY(Optional<QString> incomingEmailAddress MEMBER incomingEmailAddress)
01820     Q_PROPERTY(Optional<QStringList> recentMailedAddresses MEMBER recentMailedAddresses)
01821     Q_PROPERTY(Optional<QString> comments MEMBER comments)
01822     Q_PROPERTY(Optional<Timestamp> dateAgreedToTermsOfService MEMBER dateAgreedToTermsOfService)
01823     Q_PROPERTY(Optional<qint32> maxReferrals MEMBER maxReferrals)
01824     Q_PROPERTY(Optional<qint32> referralCount MEMBER referralCount)
01825     Q_PROPERTY(Optional<QString> refererCode MEMBER refererCode)
01826     Q_PROPERTY(Optional<Timestamp> sentEmailDate MEMBER sentEmailDate)
01827     Q_PROPERTY(Optional<qint32> sentEmailCount MEMBER sentEmailCount)
01828     Q_PROPERTY(Optional<qint32> dailyEmailLimit MEMBER dailyEmailLimit)
01829     Q_PROPERTY(Optional<Timestamp> emailOptOutDate MEMBER emailOptOutDate)
01830     Q_PROPERTY(Optional<Timestamp> partnerEmailOptInDate MEMBER partnerEmailOptInDate)
01831     Q_PROPERTY(Optional<QString> preferredLanguage MEMBER preferredLanguage)
01832     Q_PROPERTY(Optional<QString> preferredCountry MEMBER preferredCountry)
01833     Q_PROPERTY(Optional<bool> clipFullPage MEMBER clipFullPage)
01834     Q_PROPERTY(Optional<QString> twitterUserName MEMBER twitterUserName)
01835     Q_PROPERTY(Optional<QString> twitterId MEMBER twitterId)
01836     Q_PROPERTY(Optional<QString> groupName MEMBER groupName)
01837     Q_PROPERTY(Optional<QString> recognitionLanguage MEMBER recognitionLanguage)
01838     Q_PROPERTY(Optional<QString> referralProof MEMBER referralProof)
01839     Q_PROPERTY(Optional<bool> educationalDiscount MEMBER educationalDiscount)
01840     Q_PROPERTY(Optional<QString> businessAddress MEMBER businessAddress)
01841     Q_PROPERTY(Optional<bool> hideSponsorBilling MEMBER hideSponsorBilling)
01842     Q_PROPERTY(Optional<bool> useEmailAutoFiling MEMBER useEmailAutoFiling)
01843     Q_PROPERTY(Optional<ReminderEmailConfig> reminderEmailConfig MEMBER reminderEmailConfig)
01844     Q_PROPERTY(Optional<Timestamp> emailAddressLastConfirmed MEMBER emailAddressLastConfirmed)
01845     Q_PROPERTY(Optional<Timestamp> passwordUpdated MEMBER passwordUpdated)
01846     Q_PROPERTY(Optional<bool> salesforcePushEnabled MEMBER salesforcePushEnabled)
01847     Q_PROPERTY(Optional<bool> shouldLogClientEvent MEMBER shouldLogClientEvent)
01848     Q_PROPERTY(Optional<bool> optOutMachineLearning MEMBER optOutMachineLearning)
01849 };
01850
01856 struct QEVERCLOUD_EXPORT BusinessUserAttributes: public Printable
01857 {
01858 private:
01859     Q_GADGET
01860 public:
01864     EverCloudLocalData localData;
01865
01869     Optional<QString> title;
01873     Optional<QString> location;
01877     Optional<QString> department;
01881     Optional<QString> mobilePhone;
01886     Optional<QString> linkedInProfileUrl;
01890     Optional<QString> workPhone;
01894     Optional<Timestamp> companyStartDate;
01895
01896     virtual void print(QTextStream & strm) const override;
01897
01898     bool operator==(const BusinessUserAttributes & other) const
01899     {
01900         return title.isEqual(other.title)
01901             && location.isEqual(other.location)
01902             && department.isEqual(other.department)
01903             && mobilePhone.isEqual(other.mobilePhone)
01904             && linkedInProfileUrl.isEqual(other.linkedInProfileUrl)
01905             && workPhone.isEqual(other.workPhone)
01906             && companyStartDate.isEqual(other.companyStartDate)
01907         ;
01908     }
01909
01910     bool operator!=(const BusinessUserAttributes & other) const
01911     {
01912         return !(*this == other);
01913     }
01914
01915     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
01916     Q_PROPERTY(Optional<QString> title MEMBER title)
01917     Q_PROPERTY(Optional<QString> location MEMBER location)
01918     Q_PROPERTY(Optional<QString> department MEMBER department)
01919     Q_PROPERTY(Optional<QString> mobilePhone MEMBER mobilePhone)
01920     Q_PROPERTY(Optional<QString> linkedInProfileUrl MEMBER linkedInProfileUrl)
```

```
01921     Q_PROPERTY(Optional<QString> workPhone MEMBER workPhone)
01922     Q_PROPERTY(Optional<Timestamp> companyStartDate MEMBER companyStartDate)
01923 };
01924
01929 struct QEVERCLOUD_EXPORT Accounting: public Printable
01930 {
01931 private:
01932     Q_GADGET
01933 public:
01937     EverCloudLocalData localData;
01938
01945     Optional<Timestamp> uploadLimitEnd;
01951     Optional<qint64> uploadLimitNextMonth;
01956     Optional<PremiumOrderStatus> premiumServiceStatus;
01961     Optional<QString> premiumOrderNumber;
01966     Optional<QString> premiumCommerceService;
01972     Optional<Timestamp> premiumServiceStart;
01977     Optional<QString> premiumServiceSKU;
01982     Optional<Timestamp> lastSuccessfulCharge;
01987     Optional<Timestamp> lastFailedCharge;
01991     Optional<QString> lastFailedChargeReason;
01996     Optional<Timestamp> nextPaymentDue;
02001     Optional<Timestamp> premiumLockUntil;
02005     Optional<Timestamp> updated;
02010     Optional<QString> premiumSubscriptionNumber;
02014     Optional<Timestamp> lastRequestedCharge;
02018     Optional<QString> currency;
02022     Optional<qint32> unitPrice;
02026     Optional<qint32> businessId;
02030     Optional<QString> businessName;
02034     Optional<BusinessUserRole> businessRole;
02039     Optional<qint32> unitDiscount;
02044     Optional<Timestamp> nextChargeDate;
02046     Optional<qint32> availablePoints;
02047
02048     virtual void print(QTextStream & strm) const override;
02049
02050     bool operator==(const Accounting & other) const
02051     {
02052         return uploadLimitEnd.isEqual(other.uploadLimitEnd)
02053             && uploadLimitNextMonth.isEqual(other.uploadLimitNextMonth)
02054             && premiumServiceStatus.isEqual(other.premiumServiceStatus)
02055             && premiumOrderNumber.isEqual(other.premiumOrderNumber)
02056             && premiumCommerceService.isEqual(other.premiumCommerceService)
02057             && premiumServiceStart.isEqual(other.premiumServiceStart)
02058             && premiumServiceSKU.isEqual(other.premiumServiceSKU)
02059             && lastSuccessfulCharge.isEqual(other.lastSuccessfulCharge)
02060             && lastFailedCharge.isEqual(other.lastFailedCharge)
02061             && lastFailedChargeReason.isEqual(other.lastFailedChargeReason)
02062             && nextPaymentDue.isEqual(other.nextPaymentDue)
02063             && premiumLockUntil.isEqual(other.premiumLockUntil)
02064             && updated.isEqual(other.updated)
02065             && premiumSubscriptionNumber.isEqual(other.premiumSubscriptionNumber)
02066             && lastRequestedCharge.isEqual(other.lastRequestedCharge)
02067             && currency.isEqual(other.currency)
02068             && unitPrice.isEqual(other.unitPrice)
02069             && businessId.isEqual(other.businessId)
02070             && businessName.isEqual(other.businessName)
02071             && businessRole.isEqual(other.businessRole)
02072             && unitDiscount.isEqual(other.unitDiscount)
02073             && nextChargeDate.isEqual(other.nextChargeDate)
02074             && availablePoints.isEqual(other.availablePoints)
02075         ;
02076     }
02077
02078     bool operator!=(const Accounting & other) const
02079     {
02080         return !(*this == other);
02081     }
02082
02083     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
02084     Q_PROPERTY(Optional<Timestamp> uploadLimitEnd MEMBER uploadLimitEnd)
02085     Q_PROPERTY(Optional<qint64> uploadLimitNextMonth MEMBER uploadLimitNextMonth)
02086     Q_PROPERTY(Optional<PremiumOrderStatus> premiumServiceStatus MEMBER premiumServiceStatus)
02087     Q_PROPERTY(Optional<QString> premiumOrderNumber MEMBER premiumOrderNumber)
02088     Q_PROPERTY(Optional<QString> premiumCommerceService MEMBER premiumCommerceService)
02089     Q_PROPERTY(Optional<Timestamp> premiumServiceStart MEMBER premiumServiceStart)
02090     Q_PROPERTY(Optional<QString> premiumServiceSKU MEMBER premiumServiceSKU)
02091     Q_PROPERTY(Optional<Timestamp> lastSuccessfulCharge MEMBER lastSuccessfulCharge)
02092     Q_PROPERTY(Optional<Timestamp> lastFailedCharge MEMBER lastFailedCharge)
02093     Q_PROPERTY(Optional<QString> lastFailedChargeReason MEMBER lastFailedChargeReason)
02094     Q_PROPERTY(Optional<Timestamp> nextPaymentDue MEMBER nextPaymentDue)
02095     Q_PROPERTY(Optional<Timestamp> premiumLockUntil MEMBER premiumLockUntil)
02096     Q_PROPERTY(Optional<Timestamp> updated MEMBER updated)
02097     Q_PROPERTY(Optional<QString> premiumSubscriptionNumber MEMBER premiumSubscriptionNumber)
02098     Q_PROPERTY(Optional<Timestamp> lastRequestedCharge MEMBER lastRequestedCharge)
02099     Q_PROPERTY(Optional<QString> currency MEMBER currency)
```

```
02100     Q_PROPERTY(Optional<qint32> unitPrice MEMBER unitPrice)
02101     Q_PROPERTY(Optional<qint32> businessId MEMBER businessId)
02102     Q_PROPERTY(Optional<QString> businessName MEMBER businessName)
02103     Q_PROPERTY(Optional<BusinessUserRole> businessRole MEMBER businessRole)
02104     Q_PROPERTY(Optional<qint32> unitDiscount MEMBER unitDiscount)
02105     Q_PROPERTY(Optional<Timestamp> nextChargeDate MEMBER nextChargeDate)
02106     Q_PROPERTY(Optional<qint32> availablePoints MEMBER availablePoints)
02107 };
02108
02114 struct QEVERCLOUD_EXPORT BusinessUserInfo: public Printable
02115 {
02116 private:
02117     Q_GADGET
02118 public:
02122     EverCloudLocalData localData;
02123
02130     Optional<qint32> businessId;
02132     Optional<QString> businessName;
02137     Optional<BusinessUserRole> role;
02145     Optional<QString> email;
02149     Optional<Timestamp> updated;
02150
02151     virtual void print(QTextStream & strm) const override;
02152
02153     bool operator==(const BusinessUserInfo & other) const
02154     {
02155         return businessId.isEqual(other.businessId)
02156             && businessName.isEqual(other.businessName)
02157             && role.isEqual(other.role)
02158             && email.isEqual(other.email)
02159             && updated.isEqual(other.updated)
02160         ;
02161     }
02162
02163     bool operator!=(const BusinessUserInfo & other) const
02164     {
02165         return !(*this == other);
02166     }
02167
02168     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
02169     Q_PROPERTY(Optional<qint32> businessId MEMBER businessId)
02170     Q_PROPERTY(Optional<QString> businessName MEMBER businessName)
02171     Q_PROPERTY(Optional<BusinessUserRole> role MEMBER role)
02172     Q_PROPERTY(Optional<QString> email MEMBER email)
02173     Q_PROPERTY(Optional<Timestamp> updated MEMBER updated)
02174 };
02175
02179 struct QEVERCLOUD_EXPORT AccountLimits: public Printable
02180 {
02181 private:
02182     Q_GADGET
02183 public:
02187     EverCloudLocalData localData;
02188
02194     Optional<qint32> userMailLimitDaily;
02201     Optional<qint64> noteSizeMax;
02205     Optional<qint64> resourceSizeMax;
02209     Optional<qint32> userLinkedNotebookMax;
02218     Optional<qint64> uploadLimit;
02222     Optional<qint32> userNoteCountMax;
02226     Optional<qint32> userNotebookCountMax;
02230     Optional<qint32> userTagCountMax;
02234     Optional<qint32> noteTagCountMax;
02238     Optional<qint32> userSavedSearchesMax;
02242     Optional<qint32> noteResourceCountMax;
02243
02244     virtual void print(QTextStream & strm) const override;
02245
02246     bool operator==(const AccountLimits & other) const
02247     {
02248         return userMailLimitDaily.isEqual(other.userMailLimitDaily)
02249             && noteSizeMax.isEqual(other.noteSizeMax)
02250             && resourceSizeMax.isEqual(other.resourceSizeMax)
02251             && userLinkedNotebookMax.isEqual(other.userLinkedNotebookMax)
02252             && uploadLimit.isEqual(other.uploadLimit)
02253             && userNoteCountMax.isEqual(other.userNoteCountMax)
02254             && userNotebookCountMax.isEqual(other.userNotebookCountMax)
02255             && userTagCountMax.isEqual(other.userTagCountMax)
02256             && noteTagCountMax.isEqual(other.noteTagCountMax)
02257             && userSavedSearchesMax.isEqual(other.userSavedSearchesMax)
02258             && noteResourceCountMax.isEqual(other.noteResourceCountMax)
02259         ;
02260     }
02261
02262     bool operator!=(const AccountLimits & other) const
02263     {
02264         return !(*this == other);
```

```
02265     }
02266
02267     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
02268     Q_PROPERTY(Optional<qint32> userMailLimitDaily MEMBER userMailLimitDaily)
02269     Q_PROPERTY(Optional<qint64> noteSizeMax MEMBER noteSizeMax)
02270     Q_PROPERTY(Optional<qint64> resourceSizeMax MEMBER resourceSizeMax)
02271     Q_PROPERTY(Optional<qint32> userLinkedNotebookMax MEMBER userLinkedNotebookMax)
02272     Q_PROPERTY(Optional<qint64> uploadLimit MEMBER uploadLimit)
02273     Q_PROPERTY(Optional<qint32> userNoteCountMax MEMBER userNoteCountMax)
02274     Q_PROPERTY(Optional<qint32> userNotebookCountMax MEMBER userNotebookCountMax)
02275     Q_PROPERTY(Optional<qint32> userTagCountMax MEMBER userTagCountMax)
02276     Q_PROPERTY(Optional<qint32> noteTagCountMax MEMBER noteTagCountMax)
02277     Q_PROPERTY(Optional<qint32> userSavedSearchesMax MEMBER userSavedSearchesMax)
02278     Q_PROPERTY(Optional<qint32> noteResourceCountMax MEMBER noteResourceCountMax)
02279 };
02280
02284 struct QEVERCLOUD_EXPORT User: public Printable
02285 {
02286 private:
02287     Q_GADGET
02288 public:
02292     EverCloudLocalData localData;
02293
02298     Optional<UserID> id;
02309     Optional<QString> username;
02319     Optional<QString> email;
02331     Optional<QString> name;
02343     Optional<QString> timezone;
02345     Optional<PrivilegeLevel> privilege;
02350     Optional<ServiceLevel> serviceLevel;
02355     Optional<Timestamp> created;
02360     Optional<Timestamp> updated;
02366     Optional<Timestamp> deleted;
02371     Optional<bool> active;
02375     Optional<QString> shardId;
02380     Optional<UserAttributes> attributes;
02384     Optional<Accounting> accounting;
02390     Optional<BusinessUserInfo> businessUserInfo;
02397     Optional<QString> photoUrl;
02403     Optional<Timestamp> photoLastUpdated;
02407     Optional<AccountLimits> accountLimits;
02408
02409     virtual void print(QTextStream & strm) const override;
02410
02411     bool operator==(const User & other) const
02412     {
02413         return id.isEqual(other.id)
02414             && username.isEqual(other.username)
02415             && email.isEqual(other.email)
02416             && name.isEqual(other.name)
02417             && timezone.isEqual(other.timezone)
02418             && privilege.isEqual(other.privilege)
02419             && serviceLevel.isEqual(other.serviceLevel)
02420             && created.isEqual(other.created)
02421             && updated.isEqual(other.updated)
02422             && deleted.isEqual(other.deleted)
02423             && active.isEqual(other.active)
02424             && shardId.isEqual(other.shardId)
02425             && attributes.isEqual(other.attributes)
02426             && accounting.isEqual(other.accounting)
02427             && businessUserInfo.isEqual(other.businessUserInfo)
02428             && photoUrl.isEqual(other.photoUrl)
02429             && photoLastUpdated.isEqual(other.photoLastUpdated)
02430             && accountLimits.isEqual(other.accountLimits)
02431         ;
02432     }
02433
02434     bool operator!=(const User & other) const
02435     {
02436         return !(*this == other);
02437     }
02438
02439     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
02440     Q_PROPERTY(Optional<UserID> id MEMBER id)
02441     Q_PROPERTY(Optional<QString> username MEMBER username)
02442     Q_PROPERTY(Optional<QString> email MEMBER email)
02443     Q_PROPERTY(Optional<QString> name MEMBER name)
02444     Q_PROPERTY(Optional<QString> timezone MEMBER timezone)
02445     Q_PROPERTY(Optional<PrivilegeLevel> privilege MEMBER privilege)
02446     Q_PROPERTY(Optional<ServiceLevel> serviceLevel MEMBER serviceLevel)
02447     Q_PROPERTY(Optional<Timestamp> created MEMBER created)
02448     Q_PROPERTY(Optional<Timestamp> updated MEMBER updated)
02449     Q_PROPERTY(Optional<Timestamp> deleted MEMBER deleted)
02450     Q_PROPERTY(Optional<bool> active MEMBER active)
02451     Q_PROPERTY(Optional<QString> shardId MEMBER shardId)
02452     Q_PROPERTY(Optional<UserAttributes> attributes MEMBER attributes)
02453     Q_PROPERTY(Optional<Accounting> accounting MEMBER accounting)
```

```
02454     Q_PROPERTY(Optional<BusinessUserInfo> businessUserInfo MEMBER businessUserInfo)
02455     Q_PROPERTY(Optional<QString> photoUrl MEMBER photoUrl)
02456     Q_PROPERTY(Optional<Timestamp> photoLastUpdated MEMBER photoLastUpdated)
02457     Q_PROPERTY(Optional<AccountLimits> accountLimits MEMBER accountLimits)
02458 };
02459
02465 struct QEVERCLOUD_EXPORT Contact: public Printable
02466 {
02467 private:
02468     Q_GADGET
02469 public:
02473     EverCloudLocalData localData;
02474
02479     Optional<QString> name;
02483     Optional<QString> id;
02487     Optional<ContactType> type;
02492     Optional<QString> photoUrl;
02498     Optional<Timestamp> photoLastUpdated;
02506     Optional<QByteArray> messagingPermit;
02513     Optional<Timestamp> messagingPermitExpires;
02514
02515     virtual void print(QTextStream & strm) const override;
02516
02517     bool operator==(const Contact & other) const
02518     {
02519         return name.isEqual(other.name)
02520             && id.isEqual(other.id)
02521             && type.isEqual(other.type)
02522             && photoUrl.isEqual(other.photoUrl)
02523             && photoLastUpdated.isEqual(other.photoLastUpdated)
02524             && messagingPermit.isEqual(other.messagingPermit)
02525             && messagingPermitExpires.isEqual(other.messagingPermitExpires)
02526         ;
02527     }
02528
02529     bool operator!=(const Contact & other) const
02530     {
02531         return !(*this == other);
02532     }
02533
02534     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
02535     Q_PROPERTY(Optional<QString> name MEMBER name)
02536     Q_PROPERTY(Optional<QString> id MEMBER id)
02537     Q_PROPERTY(Optional<ContactType> type MEMBER type)
02538     Q_PROPERTY(Optional<QString> photoUrl MEMBER photoUrl)
02539     Q_PROPERTY(Optional<Timestamp> photoLastUpdated MEMBER photoLastUpdated)
02540     Q_PROPERTY(Optional<QByteArray> messagingPermit MEMBER messagingPermit)
02541     Q_PROPERTY(Optional<Timestamp> messagingPermitExpires MEMBER messagingPermitExpires)
02542 };
02543
02549 struct QEVERCLOUD_EXPORT Identity: public Printable
02550 {
02551 private:
02552     Q_GADGET
02553 public:
02557     EverCloudLocalData localData;
02558
02562     IdentityID id = 0;
02564     Optional<Contact> contact;
02570     Optional<UserID> userId;
02578     Optional<bool> deactivated;
02583     Optional<bool> sameBusiness;
02587     Optional<bool> blocked;
02604     Optional<bool> userConnected;
02609     Optional<MessageEventID> eventId;
02610
02611     virtual void print(QTextStream & strm) const override;
02612
02613     bool operator==(const Identity & other) const
02614     {
02615         return (id == other.id)
02616             && contact.isEqual(other.contact)
02617             && userId.isEqual(other.userId)
02618             && deactivated.isEqual(other.deactivated)
02619             && sameBusiness.isEqual(other.sameBusiness)
02620             && blocked.isEqual(other.blocked)
02621             && userConnected.isEqual(other.userConnected)
02622             && eventId.isEqual(other.eventId)
02623         ;
02624     }
02625
02626     bool operator!=(const Identity & other) const
02627     {
02628         return !(*this == other);
02629     }
02630
02631     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
```

```
02632     Q_PROPERTY(IdentityID id MEMBER id)
02633     Q_PROPERTY(Optional<Contact> contact MEMBER contact)
02634     Q_PROPERTY(Optional<UserID> userId MEMBER userId)
02635     Q_PROPERTY(Optional<bool> deactivated MEMBER deactivated)
02636     Q_PROPERTY(Optional<bool> sameBusiness MEMBER sameBusiness)
02637     Q_PROPERTY(Optional<bool> blocked MEMBER blocked)
02638     Q_PROPERTY(Optional<bool> userConnected MEMBER userConnected)
02639     Q_PROPERTY(Optional<MessageEventID> eventId MEMBER eventId)
02640 };
02641
02646 struct QEVERCLOUD_EXPORT Tag: public Printable
02647 {
02648 private:
02649     Q_GADGET
02650 public:
02654     EverCloudLocalData localData;
02655
02664     Optional<Guid> guid;
02677     Optional<QString> name;
02689     Optional<Guid> parentGuid;
02696     Optional<qint32> updateSequenceNum;
02697
02698     virtual void print(QTextStream & strm) const override;
02699
02700     bool operator==(const Tag & other) const
02701     {
02702         return guid.isEqual(other.guid)
02703             && name.isEqual(other.name)
02704             && parentGuid.isEqual(other.parentGuid)
02705             && updateSequenceNum.isEqual(other.updateSequenceNum)
02706         ;
02707     }
02708
02709     bool operator!=(const Tag & other) const
02710     {
02711         return !(*this == other);
02712     }
02713
02714     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
02715     Q_PROPERTY(Optional<Guid> guid MEMBER guid)
02716     Q_PROPERTY(Optional<QString> name MEMBER name)
02717     Q_PROPERTY(Optional<Guid> parentGuid MEMBER parentGuid)
02718     Q_PROPERTY(Optional<qint32> updateSequenceNum MEMBER updateSequenceNum)
02719 };
02720
02740 struct QEVERCLOUD_EXPORT LazyMap: public Printable
02741 {
02742 private:
02743     Q_GADGET
02744 public:
02748     EverCloudLocalData localData;
02749
02754     Optional<QSet<QString>> keysOnly;
02758     Optional<QMap<QString, QString>> fullMap;
02759
02760     virtual void print(QTextStream & strm) const override;
02761
02762     bool operator==(const LazyMap & other) const
02763     {
02764         return keysOnly.isEqual(other.keysOnly)
02765             && fullMap.isEqual(other.fullMap)
02766         ;
02767     }
02768
02769     bool operator!=(const LazyMap & other) const
02770     {
02771         return !(*this == other);
02772     }
02773
02774     using FullMap = QMap<QString, QString>;
02775
02776     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
02777     Q_PROPERTY(Optional<QSet<QString» keysOnly MEMBER keysOnly)
02778     Q_PROPERTY(Optional<FullMap> fullMap MEMBER fullMap)
02779 };
02780
02784 struct QEVERCLOUD_EXPORT ResourceAttributes: public Printable
02785 {
02786 private:
02787     Q_GADGET
02788 public:
02792     EverCloudLocalData localData;
02793
02799     Optional<QString> sourceURL;
02804     Optional<Timestamp> timestamp;
02808     Optional<double> latitude;
02812     Optional<double> longitude;
```

```
02816     Optional<double> altitude;
02823     Optional<QString> cameraMake;
02830     Optional<QString> cameraModel;
02836     Optional<bool> clientWillIndex;
02841     Optional<QString> recoType;
02847     Optional<QString> fileName;
02852     Optional<bool> attachment;
02869     Optional<LazyMap> applicationData;
02870
02871     virtual void print(QTextStream & strm) const override;
02872
02873     bool operator==(const ResourceAttributes & other) const
02874     {
02875         return sourceURL.isEqual(other.sourceURL)
02876             && timestamp.isEqual(other.timestamp)
02877             && latitude.isEqual(other.latitude)
02878             && longitude.isEqual(other.longitude)
02879             && altitude.isEqual(other.altitude)
02880             && cameraMake.isEqual(other.cameraMake)
02881             && cameraModel.isEqual(other.cameraModel)
02882             && clientWillIndex.isEqual(other.clientWillIndex)
02883             && recoType.isEqual(other.recoType)
02884             && fileName.isEqual(other.fileName)
02885             && attachment.isEqual(other.attachment)
02886             && applicationData.isEqual(other.applicationData)
02887         ;
02888     }
02889
02890     bool operator!=(const ResourceAttributes & other) const
02891     {
02892         return !(*this == other);
02893     }
02894
02895     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
02896     Q_PROPERTY(Optional<QString> sourceURL MEMBER sourceURL)
02897     Q_PROPERTY(Optional<Timestamp> timestamp MEMBER timestamp)
02898     Q_PROPERTY(Optional<double> latitude MEMBER latitude)
02899     Q_PROPERTY(Optional<double> longitude MEMBER longitude)
02900     Q_PROPERTY(Optional<double> altitude MEMBER altitude)
02901     Q_PROPERTY(Optional<QString> cameraMake MEMBER cameraMake)
02902     Q_PROPERTY(Optional<QString> cameraModel MEMBER cameraModel)
02903     Q_PROPERTY(Optional<bool> clientWillIndex MEMBER clientWillIndex)
02904     Q_PROPERTY(Optional<QString> recoType MEMBER recoType)
02905     Q_PROPERTY(Optional<QString> fileName MEMBER fileName)
02906     Q_PROPERTY(Optional<bool> attachment MEMBER attachment)
02907     Q_PROPERTY(Optional<LazyMap> applicationData MEMBER applicationData)
02908 };
02909
02914 struct QEVERCLOUD_EXPORT Resource: public Printable
02915 {
02916 private:
02917     Q_GADGET
02918 public:
02922     EverCloudLocalData localData;
02923
02933     Optional<Guid> guid;
02943     Optional<Guid> noteGuid;
02949     Optional<Data> data;
02957     Optional<QString> mime;
02962     Optional<qint16> width;
02967     Optional<qint16> height;
02971     Optional<qint16> duration;
02975     Optional<bool> active;
02980     Optional<Data> recognition;
02984     Optional<ResourceAttributes> attributes;
02991     Optional<qint32> updateSequenceNum;
02999     Optional<Data> alternateData;
03000
03001     virtual void print(QTextStream & strm) const override;
03002
03003     bool operator==(const Resource & other) const
03004     {
03005         return guid.isEqual(other.guid)
03006             && noteGuid.isEqual(other.noteGuid)
03007             && data.isEqual(other.data)
03008             && mime.isEqual(other.mime)
03009             && width.isEqual(other.width)
03010             && height.isEqual(other.height)
03011             && duration.isEqual(other.duration)
03012             && active.isEqual(other.active)
03013             && recognition.isEqual(other.recognition)
03014             && attributes.isEqual(other.attributes)
03015             && updateSequenceNum.isEqual(other.updateSequenceNum)
03016             && alternateData.isEqual(other.alternateData)
03017         ;
03018     }
03019
```

```
03020     bool operator!=(const Resource & other) const
03021     {
03022         return !(*this == other);
03023     }
03024
03025     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
03026     Q_PROPERTY(Optional<Guid> guid MEMBER guid)
03027     Q_PROPERTY(Optional<Guid> noteGuid MEMBER noteGuid)
03028     Q_PROPERTY(Optional<Data> data MEMBER data)
03029     Q_PROPERTY(Optional<QString> mime MEMBER mime)
03030     Q_PROPERTY(Optional<qint16> width MEMBER width)
03031     Q_PROPERTY(Optional<qint16> height MEMBER height)
03032     Q_PROPERTY(Optional<qint16> duration MEMBER duration)
03033     Q_PROPERTY(Optional<bool> active MEMBER active)
03034     Q_PROPERTY(Optional<Data> recognition MEMBER recognition)
03035     Q_PROPERTY(Optional<ResourceAttributes> attributes MEMBER attributes)
03036     Q_PROPERTY(Optional<qint32> updateSequenceNum MEMBER updateSequenceNum)
03037     Q_PROPERTY(Optional<Data> alternateData MEMBER alternateData)
03038 };
03039
03043 struct QEVERCLOUD_EXPORT NoteAttributes: public Printable
03044 {
03045 private:
03046     Q_GADGET
03047 public:
03051     EverCloudLocalData localData;
03052
03056     Optional<Timestamp> subjectDate;
03060     Optional<double> latitude;
03064     Optional<double> longitude;
03068     Optional<double> altitude;
03074     Optional<QString> author;
03081     Optional<QString> source;
03088     Optional<QString> sourceURL;
03096     Optional<QString> sourceApplication;
03104     Optional<Timestamp> shareDate;
03131     Optional<qint64> reminderOrder;
03138     Optional<Timestamp> reminderDoneTime;
03149     Optional<Timestamp> reminderTime;
03161     Optional<QString> placeName;
03184     Optional<QString> contentClass;
03201     Optional<LazyMap> applicationData;
03213     Optional<QString> lastEditedBy;
03219     Optional<QMap<QString, QString>> classifications;
03223     Optional<UserID> creatorId;
03227     Optional<UserID> lastEditorId;
03236     Optional<bool> sharedWithBusiness;
03245     Optional<Guid> conflictSourceNoteGuid;
03258     Optional<qint32> noteTitleQuality;
03259
03260     virtual void print(QTextStream & strm) const override;
03261
03262     bool operator==(const NoteAttributes & other) const
03263     {
03264         return subjectDate.isEqual(other.subjectDate)
03265             && latitude.isEqual(other.latitude)
03266             && longitude.isEqual(other.longitude)
03267             && altitude.isEqual(other.altitude)
03268             && author.isEqual(other.author)
03269             && source.isEqual(other.source)
03270             && sourceURL.isEqual(other.sourceURL)
03271             && sourceApplication.isEqual(other.sourceApplication)
03272             && shareDate.isEqual(other.shareDate)
03273             && reminderOrder.isEqual(other.reminderOrder)
03274             && reminderDoneTime.isEqual(other.reminderDoneTime)
03275             && reminderTime.isEqual(other.reminderTime)
03276             && placeName.isEqual(other.placeName)
03277             && contentClass.isEqual(other.contentClass)
03278             && applicationData.isEqual(other.applicationData)
03279             && lastEditedBy.isEqual(other.lastEditedBy)
03280             && classifications.isEqual(other.classifications)
03281             && creatorId.isEqual(other.creatorId)
03282             && lastEditorId.isEqual(other.lastEditorId)
03283             && sharedWithBusiness.isEqual(other.sharedWithBusiness)
03284             && conflictSourceNoteGuid.isEqual(other.conflictSourceNoteGuid)
03285             && noteTitleQuality.isEqual(other.noteTitleQuality)
03286         ;
03287     }
03288
03289     bool operator!=(const NoteAttributes & other) const
03290     {
03291         return !(*this == other);
03292     }
03293
03294     using Classifications = QMap<QString, QString>;
03295
03296     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
```

```
03297     Q_PROPERTY(Optional<Timestamp> subjectDate MEMBER subjectDate)
03298     Q_PROPERTY(Optional<double> latitude MEMBER latitude)
03299     Q_PROPERTY(Optional<double> longitude MEMBER longitude)
03300     Q_PROPERTY(Optional<double> altitude MEMBER altitude)
03301     Q_PROPERTY(Optional<QString> author MEMBER author)
03302     Q_PROPERTY(Optional<QString> source MEMBER source)
03303     Q_PROPERTY(Optional<QString> sourceURL MEMBER sourceURL)
03304     Q_PROPERTY(Optional<QString> sourceApplication MEMBER sourceApplication)
03305     Q_PROPERTY(Optional<Timestamp> shareDate MEMBER shareDate)
03306     Q_PROPERTY(Optional<qint64> reminderOrder MEMBER reminderOrder)
03307     Q_PROPERTY(Optional<Timestamp> reminderDoneTime MEMBER reminderDoneTime)
03308     Q_PROPERTY(Optional<Timestamp> reminderTime MEMBER reminderTime)
03309     Q_PROPERTY(Optional<QString> placeName MEMBER placeName)
03310     Q_PROPERTY(Optional<QString> contentClass MEMBER contentClass)
03311     Q_PROPERTY(Optional<LazyMap> applicationData MEMBER applicationData)
03312     Q_PROPERTY(Optional<QString> lastEditedBy MEMBER lastEditedBy)
03313     Q_PROPERTY(Optional<Classifications> classifications MEMBER classifications)
03314     Q_PROPERTY(Optional<UserID> creatorId MEMBER creatorId)
03315     Q_PROPERTY(Optional<UserID> lastEditorId MEMBER lastEditorId)
03316     Q_PROPERTY(Optional<bool> sharedWithBusiness MEMBER sharedWithBusiness)
03317     Q_PROPERTY(Optional<Guid> conflictSourceNoteGuid MEMBER conflictSourceNoteGuid)
03318     Q_PROPERTY(Optional<qint32> noteTitleQuality MEMBER noteTitleQuality)
03319 };
03320
03327 struct QEVERCLOUD_EXPORT SharedNote: public Printable
03328 {
03329 private:
03330     Q_GADGET
03331 public:
03335     EverCloudLocalData localData;
03336
03340     Optional<UserID> sharerUserID;
03347     Optional<Identity> recipientIdentity;
03351     Optional<SharedNotePrivilegeLevel> privilege;
03355     Optional<Timestamp> serviceCreated;
03359     Optional<Timestamp> serviceUpdated;
03363     Optional<Timestamp> serviceAssigned;
03364
03365     virtual void print(QTextStream & strm) const override;
03366
03367     bool operator==(const SharedNote & other) const
03368     {
03369         return sharerUserID.isEqual(other.sharerUserID)
03370             && recipientIdentity.isEqual(other.recipientIdentity)
03371             && privilege.isEqual(other.privilege)
03372             && serviceCreated.isEqual(other.serviceCreated)
03373             && serviceUpdated.isEqual(other.serviceUpdated)
03374             && serviceAssigned.isEqual(other.serviceAssigned)
03375         ;
03376     }
03377
03378     bool operator!=(const SharedNote & other) const
03379     {
03380         return !(*this == other);
03381     }
03382
03383     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
03384     Q_PROPERTY(Optional<UserID> sharerUserID MEMBER sharerUserID)
03385     Q_PROPERTY(Optional<Identity> recipientIdentity MEMBER recipientIdentity)
03386     Q_PROPERTY(Optional<SharedNotePrivilegeLevel> privilege MEMBER privilege)
03387     Q_PROPERTY(Optional<Timestamp> serviceCreated MEMBER serviceCreated)
03388     Q_PROPERTY(Optional<Timestamp> serviceUpdated MEMBER serviceUpdated)
03389     Q_PROPERTY(Optional<Timestamp> serviceAssigned MEMBER serviceAssigned)
03390 };
03391
03429 struct QEVERCLOUD_EXPORT NoteRestrictions: public Printable
03430 {
03431 private:
03432     Q_GADGET
03433 public:
03437     EverCloudLocalData localData;
03438
03442     Optional<bool> noUpdateTitle;
03444     Optional<bool> noUpdateContent;
03448     Optional<bool> noEmail;
03453     Optional<bool> noShare;
03457     Optional<bool> noSharePublicly;
03458
03459     virtual void print(QTextStream & strm) const override;
03460
03461     bool operator==(const NoteRestrictions & other) const
03462     {
03463         return noUpdateTitle.isEqual(other.noUpdateTitle)
03464             && noUpdateContent.isEqual(other.noUpdateContent)
03465             && noEmail.isEqual(other.noEmail)
03466             && noShare.isEqual(other.noShare)
03467             && noSharePublicly.isEqual(other.noSharePublicly)
```

```
03468         ;
03469     }
03470
03471     bool operator!=(const NoteRestrictions & other) const
03472     {
03473         return !(*this == other);
03474     }
03475
03476     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
03477     Q_PROPERTY(Optional<bool> noUpdateTitle MEMBER noUpdateTitle)
03478     Q_PROPERTY(Optional<bool> noUpdateContent MEMBER noUpdateContent)
03479     Q_PROPERTY(Optional<bool> noEmail MEMBER noEmail)
03480     Q_PROPERTY(Optional<bool> noShare MEMBER noShare)
03481     Q_PROPERTY(Optional<bool> noSharePublicly MEMBER noSharePublicly)
03482 };
03483
03492 struct QEVERCLOUD_EXPORT NoteLimits: public Printable
03493 {
03494 private:
03495     Q_GADGET
03496 public:
03500     EverCloudLocalData localData;
03501
03503     Optional<qint32> noteResourceCountMax;
03505     Optional<qint64> uploadLimit;
03507     Optional<qint64> resourceSizeMax;
03509     Optional<qint64> noteSizeMax;
03511     Optional<qint64> uploaded;
03512
03513     virtual void print(QTextStream & strm) const override;
03514
03515     bool operator==(const NoteLimits & other) const
03516     {
03517         return noteResourceCountMax.isEqual(other.noteResourceCountMax)
03518             && uploadLimit.isEqual(other.uploadLimit)
03519             && resourceSizeMax.isEqual(other.resourceSizeMax)
03520             && noteSizeMax.isEqual(other.noteSizeMax)
03521             && uploaded.isEqual(other.uploaded)
03522         ;
03523     }
03524
03525     bool operator!=(const NoteLimits & other) const
03526     {
03527         return !(*this == other);
03528     }
03529
03530     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
03531     Q_PROPERTY(Optional<qint32> noteResourceCountMax MEMBER noteResourceCountMax)
03532     Q_PROPERTY(Optional<qint64> uploadLimit MEMBER uploadLimit)
03533     Q_PROPERTY(Optional<qint64> resourceSizeMax MEMBER resourceSizeMax)
03534     Q_PROPERTY(Optional<qint64> noteSizeMax MEMBER noteSizeMax)
03535     Q_PROPERTY(Optional<qint64> uploaded MEMBER uploaded)
03536 };
03537
03542 struct QEVERCLOUD_EXPORT Note: public Printable
03543 {
03544 private:
03545     Q_GADGET
03546 public:
03550     EverCloudLocalData localData;
03551
03560     Optional<Guid> guid;
03568     Optional<QString> title;
03579     Optional<QString> content;
03587     Optional<QByteArray> contentHash;
03593     Optional<qint32> contentLength;
03603     Optional<Timestamp> created;
03610     Optional<Timestamp> updated;
03618     Optional<Timestamp> deleted;
03623     Optional<bool> active;
03630     Optional<qint32> updateSequenceNum;
03640     Optional<QString> notebookGuid;
03651     Optional<QList<Guid>> tagGuids;
03661     Optional<QList<Resource>> resources;
03667     Optional<NoteAttributes> attributes;
03675     Optional<QStringList> tagNames;
03682     Optional<QList<SharedNote>> sharedNotes;
03691     Optional<NoteRestrictions> restrictions;
03693     Optional<NoteLimits> limits;
03694
03695     virtual void print(QTextStream & strm) const override;
03696
03697     bool operator==(const Note & other) const
03698     {
03699         return guid.isEqual(other.guid)
03700             && title.isEqual(other.title)
03701             && content.isEqual(other.content)
```

```
03702             && contentHash.isEqual(other.contentHash)
03703             && contentLength.isEqual(other.contentLength)
03704             && created.isEqual(other.created)
03705             && updated.isEqual(other.updated)
03706             && deleted.isEqual(other.deleted)
03707             && active.isEqual(other.active)
03708             && updateSequenceNum.isEqual(other.updateSequenceNum)
03709             && notebookGuid.isEqual(other.notebookGuid)
03710             && tagGuids.isEqual(other.tagGuids)
03711             && resources.isEqual(other.resources)
03712             && attributes.isEqual(other.attributes)
03713             && tagNames.isEqual(other.tagNames)
03714             && sharedNotes.isEqual(other.sharedNotes)
03715             && restrictions.isEqual(other.restrictions)
03716             && limits.isEqual(other.limits)
03717         ;
03718     }
03719
03720     bool operator!=(const Note & other) const
03721     {
03722         return !(*this == other);
03723     }
03724
03725     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
03726     Q_PROPERTY(Optional<Guid> guid MEMBER guid)
03727     Q_PROPERTY(Optional<QString> title MEMBER title)
03728     Q_PROPERTY(Optional<QString> content MEMBER content)
03729     Q_PROPERTY(Optional<QByteArray> contentHash MEMBER contentHash)
03730     Q_PROPERTY(Optional<qint32> contentLength MEMBER contentLength)
03731     Q_PROPERTY(Optional<Timestamp> created MEMBER created)
03732     Q_PROPERTY(Optional<Timestamp> updated MEMBER updated)
03733     Q_PROPERTY(Optional<Timestamp> deleted MEMBER deleted)
03734     Q_PROPERTY(Optional<bool> active MEMBER active)
03735     Q_PROPERTY(Optional<qint32> updateSequenceNum MEMBER updateSequenceNum)
03736     Q_PROPERTY(Optional<QString> notebookGuid MEMBER notebookGuid)
03737     Q_PROPERTY(Optional<QList<Guid» tagGuids MEMBER tagGuids)
03738     Q_PROPERTY(Optional<QList<Resource» resources MEMBER resources)
03739     Q_PROPERTY(Optional<NoteAttributes> attributes MEMBER attributes)
03740     Q_PROPERTY(Optional<QStringList> tagNames MEMBER tagNames)
03741     Q_PROPERTY(Optional<QList<SharedNote» sharedNotes MEMBER sharedNotes)
03742     Q_PROPERTY(Optional<NoteRestrictions> restrictions MEMBER restrictions)
03743     Q_PROPERTY(Optional<NoteLimits> limits MEMBER limits)
03744 };
03745
03751 struct QEVERCLOUD_EXPORT Publishing: public Printable
03752 {
03753 private:
03754     Q_GADGET
03755 public:
03759     EverCloudLocalData localData;
03760
03772     Optional<QString> uri;
03777     Optional<NoteSortOrder> order;
03783     Optional<bool> ascending;
03794     Optional<QString> publicDescription;
03795
03796     virtual void print(QTextStream & strm) const override;
03797
03798     bool operator==(const Publishing & other) const
03799     {
03800         return uri.isEqual(other.uri)
03801             && order.isEqual(other.order)
03802             && ascending.isEqual(other.ascending)
03803             && publicDescription.isEqual(other.publicDescription)
03804         ;
03805     }
03806
03807     bool operator!=(const Publishing & other) const
03808     {
03809         return !(*this == other);
03810     }
03811
03812     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
03813     Q_PROPERTY(Optional<QString> uri MEMBER uri)
03814     Q_PROPERTY(Optional<NoteSortOrder> order MEMBER order)
03815     Q_PROPERTY(Optional<bool> ascending MEMBER ascending)
03816     Q_PROPERTY(Optional<QString> publicDescription MEMBER publicDescription)
03817 };
03818
03826 struct QEVERCLOUD_EXPORT BusinessNotebook: public Printable
03827 {
03828 private:
03829     Q_GADGET
03830 public:
03834     EverCloudLocalData localData;
03835
03846     Optional<QString> notebookDescription;
```

```
03851     Optional<SharedNotebookPrivilegeLevel> privilege;
03856     Optional<bool> recommended;
03857
03858     virtual void print(QTextStream & strm) const override;
03859
03860     bool operator==(const BusinessNotebook & other) const
03861     {
03862         return notebookDescription.isEqual(other.notebookDescription)
03863             && privilege.isEqual(other.privilege)
03864             && recommended.isEqual(other.recommended)
03865         ;
03866     }
03867
03868     bool operator!=(const BusinessNotebook & other) const
03869     {
03870         return !(*this == other);
03871     }
03872
03873     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
03874     Q_PROPERTY(Optional<QString> notebookDescription MEMBER notebookDescription)
03875     Q_PROPERTY(Optional<SharedNotebookPrivilegeLevel> privilege MEMBER privilege)
03876     Q_PROPERTY(Optional<bool> recommended MEMBER recommended)
03877 };
03878
03883 struct QEVERCLOUD_EXPORT SavedSearchScope: public Printable
03884 {
03885 private:
03886     Q_GADGET
03887 public:
03891     EverCloudLocalData localData;
03892
03896     Optional<bool> includeAccount;
03901     Optional<bool> includePersonalLinkedNotebooks;
03907     Optional<bool> includeBusinessLinkedNotebooks;
03908
03909     virtual void print(QTextStream & strm) const override;
03910
03911     bool operator==(const SavedSearchScope & other) const
03912     {
03913         return includeAccount.isEqual(other.includeAccount)
03914             && includePersonalLinkedNotebooks.isEqual(other.includePersonalLinkedNotebooks)
03915             && includeBusinessLinkedNotebooks.isEqual(other.includeBusinessLinkedNotebooks)
03916         ;
03917     }
03918
03919     bool operator!=(const SavedSearchScope & other) const
03920     {
03921         return !(*this == other);
03922     }
03923
03924     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
03925     Q_PROPERTY(Optional<bool> includeAccount MEMBER includeAccount)
03926     Q_PROPERTY(Optional<bool> includePersonalLinkedNotebooks MEMBER includePersonalLinkedNotebooks)
03927     Q_PROPERTY(Optional<bool> includeBusinessLinkedNotebooks MEMBER includeBusinessLinkedNotebooks)
03928 };
03929
03933 struct QEVERCLOUD_EXPORT SavedSearch: public Printable
03934 {
03935 private:
03936     Q_GADGET
03937 public:
03941     EverCloudLocalData localData;
03942
03951     Optional<Guid> guid;
03961     Optional<QString> name;
03967     Optional<QString> query;
03972     Optional<QueryFormat> format;
03979     Optional<qint32> updateSequenceNum;
03993     Optional<SavedSearchScope> scope;
03994
03995     virtual void print(QTextStream & strm) const override;
03996
03997     bool operator==(const SavedSearch & other) const
03998     {
03999         return guid.isEqual(other.guid)
04000             && name.isEqual(other.name)
04001             && query.isEqual(other.query)
04002             && format.isEqual(other.format)
04003             && updateSequenceNum.isEqual(other.updateSequenceNum)
04004             && scope.isEqual(other.scope)
04005         ;
04006     }
04007
04008     bool operator!=(const SavedSearch & other) const
04009     {
04010         return !(*this == other);
04011     }
```

```
04012
04013     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
04014     Q_PROPERTY(Optional<Guid> guid MEMBER guid)
04015     Q_PROPERTY(Optional<QString> name MEMBER name)
04016     Q_PROPERTY(Optional<QString> query MEMBER query)
04017     Q_PROPERTY(Optional<QueryFormat> format MEMBER format)
04018     Q_PROPERTY(Optional<qint32> updateSequenceNum MEMBER updateSequenceNum)
04019     Q_PROPERTY(Optional<SavedSearchScope> scope MEMBER scope)
04020 };
04021
04036 struct QEVERCLOUD_EXPORT SharedNotebookRecipientSettings: public Printable
04037 {
04038 private:
04039     Q_GADGET
04040 public:
04044     EverCloudLocalData localData;
04045
04052     Optional<bool> reminderNotifyEmail;
04059     Optional<bool> reminderNotifyInApp;
04060
04061     virtual void print(QTextStream & strm) const override;
04062
04063     bool operator==(const SharedNotebookRecipientSettings & other) const
04064     {
04065         return reminderNotifyEmail.isEqual(other.reminderNotifyEmail)
04066             && reminderNotifyInApp.isEqual(other.reminderNotifyInApp)
04067         ;
04068     }
04069
04070     bool operator!=(const SharedNotebookRecipientSettings & other) const
04071     {
04072         return !(*this == other);
04073     }
04074
04075     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
04076     Q_PROPERTY(Optional<bool> reminderNotifyEmail MEMBER reminderNotifyEmail)
04077     Q_PROPERTY(Optional<bool> reminderNotifyInApp MEMBER reminderNotifyInApp)
04078 };
04079
04090 struct QEVERCLOUD_EXPORT NotebookRecipientSettings: public Printable
04091 {
04092 private:
04093     Q_GADGET
04094 public:
04098     EverCloudLocalData localData;
04099
04107     Optional<bool> reminderNotifyEmail;
04114     Optional<bool> reminderNotifyInApp;
04120     Optional<bool> inMyList;
04126     Optional<QString> stack;
04132     Optional<RecipientStatus> recipientStatus;
04133
04134     virtual void print(QTextStream & strm) const override;
04135
04136     bool operator==(const NotebookRecipientSettings & other) const
04137     {
04138         return reminderNotifyEmail.isEqual(other.reminderNotifyEmail)
04139             && reminderNotifyInApp.isEqual(other.reminderNotifyInApp)
04140             && inMyList.isEqual(other.inMyList)
04141             && stack.isEqual(other.stack)
04142             && recipientStatus.isEqual(other.recipientStatus)
04143         ;
04144     }
04145
04146     bool operator!=(const NotebookRecipientSettings & other) const
04147     {
04148         return !(*this == other);
04149     }
04150
04151     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
04152     Q_PROPERTY(Optional<bool> reminderNotifyEmail MEMBER reminderNotifyEmail)
04153     Q_PROPERTY(Optional<bool> reminderNotifyInApp MEMBER reminderNotifyInApp)
04154     Q_PROPERTY(Optional<bool> inMyList MEMBER inMyList)
04155     Q_PROPERTY(Optional<QString> stack MEMBER stack)
04156     Q_PROPERTY(Optional<RecipientStatus> recipientStatus MEMBER recipientStatus)
04157 };
04158
04163 struct QEVERCLOUD_EXPORT SharedNotebook: public Printable
04164 {
04165 private:
04166     Q_GADGET
04167 public:
04171     EverCloudLocalData localData;
04172
04176     Optional<qint64> id;
04180     Optional<UserID> userId;
04184     Optional<Guid> notebookGuid;
```

```
04191     Optional<QString> email;
04196     Optional<IdentityID> recipientIdentityId;
04200     Optional<bool> notebookModifiable;
04205     Optional<Timestamp> serviceCreated;
04214     Optional<Timestamp> serviceUpdated;
04222     Optional<QString> globalId;
04228     Optional<QString> username;
04233     Optional<SharedNotebookPrivilegeLevel> privilege;
04240     Optional<SharedNotebookRecipientSettings> recipientSettings;
04249     Optional<UserID> sharerUserId;
04257     Optional<QString> recipientUsername;
04266     Optional<UserID> recipientUserId;
04272     Optional<Timestamp> serviceAssigned;
04273
04274     virtual void print(QTextStream & strm) const override;
04275
04276     bool operator==(const SharedNotebook & other) const
04277     {
04278         return id.isEqual(other.id)
04279             && userId.isEqual(other.userId)
04280             && notebookGuid.isEqual(other.notebookGuid)
04281             && email.isEqual(other.email)
04282             && recipientIdentityId.isEqual(other.recipientIdentityId)
04283             && notebookModifiable.isEqual(other.notebookModifiable)
04284             && serviceCreated.isEqual(other.serviceCreated)
04285             && serviceUpdated.isEqual(other.serviceUpdated)
04286             && globalId.isEqual(other.globalId)
04287             && username.isEqual(other.username)
04288             && privilege.isEqual(other.privilege)
04289             && recipientSettings.isEqual(other.recipientSettings)
04290             && sharerUserId.isEqual(other.sharerUserId)
04291             && recipientUsername.isEqual(other.recipientUsername)
04292             && recipientUserId.isEqual(other.recipientUserId)
04293             && serviceAssigned.isEqual(other.serviceAssigned)
04294         ;
04295     }
04296
04297     bool operator!=(const SharedNotebook & other) const
04298     {
04299         return !(*this == other);
04300     }
04301
04302     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
04303     Q_PROPERTY(Optional<qint64> id MEMBER id)
04304     Q_PROPERTY(Optional<UserID> userId MEMBER userId)
04305     Q_PROPERTY(Optional<Guid> notebookGuid MEMBER notebookGuid)
04306     Q_PROPERTY(Optional<QString> email MEMBER email)
04307     Q_PROPERTY(Optional<IdentityID> recipientIdentityId MEMBER recipientIdentityId)
04308     Q_PROPERTY(Optional<bool> notebookModifiable MEMBER notebookModifiable)
04309     Q_PROPERTY(Optional<Timestamp> serviceCreated MEMBER serviceCreated)
04310     Q_PROPERTY(Optional<Timestamp> serviceUpdated MEMBER serviceUpdated)
04311     Q_PROPERTY(Optional<QString> globalId MEMBER globalId)
04312     Q_PROPERTY(Optional<QString> username MEMBER username)
04313     Q_PROPERTY(Optional<SharedNotebookPrivilegeLevel> privilege MEMBER privilege)
04314     Q_PROPERTY(Optional<SharedNotebookRecipientSettings> recipientSettings MEMBER recipientSettings)
04315     Q_PROPERTY(Optional<UserID> sharerUserId MEMBER sharerUserId)
04316     Q_PROPERTY(Optional<QString> recipientUsername MEMBER recipientUsername)
04317     Q_PROPERTY(Optional<UserID> recipientUserId MEMBER recipientUserId)
04318     Q_PROPERTY(Optional<Timestamp> serviceAssigned MEMBER serviceAssigned)
04319 };
04320
04324 struct QEVERCLOUD_EXPORT CanMoveToContainerRestrictions: public Printable
04325 {
04326 private:
04327     Q_GADGET
04328 public:
04332     EverCloudLocalData localData;
04333
04335     Optional<CanMoveToContainerStatus> canMoveToContainer;
04336
04337     virtual void print(QTextStream & strm) const override;
04338
04339     bool operator==(const CanMoveToContainerRestrictions & other) const
04340     {
04341         return canMoveToContainer.isEqual(other.canMoveToContainer)
04342         ;
04343     }
04344
04345     bool operator!=(const CanMoveToContainerRestrictions & other) const
04346     {
04347         return !(*this == other);
04348     }
04349
04350     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
04351     Q_PROPERTY(Optional<CanMoveToContainerStatus> canMoveToContainer MEMBER canMoveToContainer)
04352 };
04353
```

```
04380 struct QEVERCLOUD_EXPORT NotebookRestrictions: public Printable
04381 {
04382 private:
04383     Q_GADGET
04384 public:
04388     EverCloudLocalData localData;
04389
04394     Optional<bool> noReadNotes;
04398     Optional<bool> noCreateNotes;
04402     Optional<bool> noUpdateNotes;
04406     Optional<bool> noExpungeNotes;
04411     Optional<bool> noShareNotes;
04417     Optional<bool> noEmailNotes;
04422     Optional<bool> noSendMessageToRecipients;
04427     Optional<bool> noUpdateNotebook;
04432     Optional<bool> noExpungeNotebook;
04437     Optional<bool> noSetDefaultNotebook;
04442     Optional<bool> noSetNotebookStack;
04447     Optional<bool> noPublishToPublic;
04451     Optional<bool> noPublishToBusinessLibrary;
04456     Optional<bool> noCreateTags;
04460     Optional<bool> noUpdateTags;
04464     Optional<bool> noExpungeTags;
04469     Optional<bool> noSetParentTag;
04473     Optional<bool> noCreateSharedNotebooks;
04480     Optional<SharedNotebookInstanceRestrictions> updateWhichSharedNotebookRestrictions;
04487     Optional<SharedNotebookInstanceRestrictions> expungeWhichSharedNotebookRestrictions;
04492     Optional<bool> noShareNotesWithBusiness;
04496     Optional<bool> noRenameNotebook;
04501     Optional<bool> noSetInMyList;
04503     Optional<bool> noChangeContact;
04508     Optional<CanMoveToContainerRestrictions> canMoveToContainerRestrictions;
04510     Optional<bool> noSetReminderNotifyEmail;
04512     Optional<bool> noSetReminderNotifyInApp;
04514     Optional<bool> noSetRecipientSettingsStack;
04518     Optional<bool> noCanMoveNote;
04519
04520     virtual void print(QTextStream & strm) const override;
04521
04522     bool operator==(const NotebookRestrictions & other) const
04523     {
04524         return noReadNotes.isEqual(other.noReadNotes)
04525             && noCreateNotes.isEqual(other.noCreateNotes)
04526             && noUpdateNotes.isEqual(other.noUpdateNotes)
04527             && noExpungeNotes.isEqual(other.noExpungeNotes)
04528             && noShareNotes.isEqual(other.noShareNotes)
04529             && noEmailNotes.isEqual(other.noEmailNotes)
04530             && noSendMessageToRecipients.isEqual(other.noSendMessageToRecipients)
04531             && noUpdateNotebook.isEqual(other.noUpdateNotebook)
04532             && noExpungeNotebook.isEqual(other.noExpungeNotebook)
04533             && noSetDefaultNotebook.isEqual(other.noSetDefaultNotebook)
04534             && noSetNotebookStack.isEqual(other.noSetNotebookStack)
04535             && noPublishToPublic.isEqual(other.noPublishToPublic)
04536             && noPublishToBusinessLibrary.isEqual(other.noPublishToBusinessLibrary)
04537             && noCreateTags.isEqual(other.noCreateTags)
04538             && noUpdateTags.isEqual(other.noUpdateTags)
04539             && noExpungeTags.isEqual(other.noExpungeTags)
04540             && noSetParentTag.isEqual(other.noSetParentTag)
04541             && noCreateSharedNotebooks.isEqual(other.noCreateSharedNotebooks)
04542             &&
    updateWhichSharedNotebookRestrictions.isEqual(other.updateWhichSharedNotebookRestrictions)
04543             &&
    expungeWhichSharedNotebookRestrictions.isEqual(other.expungeWhichSharedNotebookRestrictions)
04544             && noShareNotesWithBusiness.isEqual(other.noShareNotesWithBusiness)
04545             && noRenameNotebook.isEqual(other.noRenameNotebook)
04546             && noSetInMyList.isEqual(other.noSetInMyList)
04547             && noChangeContact.isEqual(other.noChangeContact)
04548             && canMoveToContainerRestrictions.isEqual(other.canMoveToContainerRestrictions)
04549             && noSetReminderNotifyEmail.isEqual(other.noSetReminderNotifyEmail)
04550             && noSetReminderNotifyInApp.isEqual(other.noSetReminderNotifyInApp)
04551             && noSetRecipientSettingsStack.isEqual(other.noSetRecipientSettingsStack)
04552             && noCanMoveNote.isEqual(other.noCanMoveNote)
04553         ;
04554     }
04555
04556     bool operator!=(const NotebookRestrictions & other) const
04557     {
04558         return !(*this == other);
04559     }
04560
04561     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
04562     Q_PROPERTY(Optional<bool> noReadNotes MEMBER noReadNotes)
04563     Q_PROPERTY(Optional<bool> noCreateNotes MEMBER noCreateNotes)
04564     Q_PROPERTY(Optional<bool> noUpdateNotes MEMBER noUpdateNotes)
04565     Q_PROPERTY(Optional<bool> noExpungeNotes MEMBER noExpungeNotes)
04566     Q_PROPERTY(Optional<bool> noShareNotes MEMBER noShareNotes)
04567     Q_PROPERTY(Optional<bool> noEmailNotes MEMBER noEmailNotes)
```

```
04568     Q_PROPERTY(Optional<bool> noSendMessageToRecipients MEMBER noSendMessageToRecipients)
04569     Q_PROPERTY(Optional<bool> noUpdateNotebook MEMBER noUpdateNotebook)
04570     Q_PROPERTY(Optional<bool> noExpungeNotebook MEMBER noExpungeNotebook)
04571     Q_PROPERTY(Optional<bool> noSetDefaultNotebook MEMBER noSetDefaultNotebook)
04572     Q_PROPERTY(Optional<bool> noSetNotebookStack MEMBER noSetNotebookStack)
04573     Q_PROPERTY(Optional<bool> noPublishToPublic MEMBER noPublishToPublic)
04574     Q_PROPERTY(Optional<bool> noPublishToBusinessLibrary MEMBER noPublishToBusinessLibrary)
04575     Q_PROPERTY(Optional<bool> noCreateTags MEMBER noCreateTags)
04576     Q_PROPERTY(Optional<bool> noUpdateTags MEMBER noUpdateTags)
04577     Q_PROPERTY(Optional<bool> noExpungeTags MEMBER noExpungeTags)
04578     Q_PROPERTY(Optional<bool> noSetParentTag MEMBER noSetParentTag)
04579     Q_PROPERTY(Optional<bool> noCreateSharedNotebooks MEMBER noCreateSharedNotebooks)
04580     Q_PROPERTY(Optional<SharedNotebookInstanceRestrictions> updateWhichSharedNotebookRestrictions
      MEMBER updateWhichSharedNotebookRestrictions)
04581     Q_PROPERTY(Optional<SharedNotebookInstanceRestrictions> expungeWhichSharedNotebookRestrictions
      MEMBER expungeWhichSharedNotebookRestrictions)
04582     Q_PROPERTY(Optional<bool> noShareNotesWithBusiness MEMBER noShareNotesWithBusiness)
04583     Q_PROPERTY(Optional<bool> noRenameNotebook MEMBER noRenameNotebook)
04584     Q_PROPERTY(Optional<bool> noSetInMyList MEMBER noSetInMyList)
04585     Q_PROPERTY(Optional<bool> noChangeContact MEMBER noChangeContact)
04586     Q_PROPERTY(Optional<CanMoveToContainerRestrictions> canMoveToContainerRestrictions MEMBER
      canMoveToContainerRestrictions)
04587     Q_PROPERTY(Optional<bool> noSetReminderNotifyEmail MEMBER noSetReminderNotifyEmail)
04588     Q_PROPERTY(Optional<bool> noSetReminderNotifyInApp MEMBER noSetReminderNotifyInApp)
04589     Q_PROPERTY(Optional<bool> noSetRecipientSettingsStack MEMBER noSetRecipientSettingsStack)
04590     Q_PROPERTY(Optional<bool> noCanMoveNote MEMBER noCanMoveNote)
04591 };
04592
04596 struct QEVERCLOUD_EXPORT Notebook: public Printable
04597 {
04598 private:
04599     Q_GADGET
04600 public:
04604     EverCloudLocalData localData;
04605
04613     Optional<Guid> guid;
04624     Optional<QString> name;
04631     Optional<qint32> updateSequenceNum;
04644     Optional<bool> defaultNotebook;
04651     Optional<Timestamp> serviceCreated;
04658     Optional<Timestamp> serviceUpdated;
04668     Optional<Publishing> publishing;
04677     Optional<bool> published;
04685     Optional<QString> stack;
04689     Optional<QList<qint64>> sharedNotebookIds;
04699     Optional<QList<SharedNotebook>> sharedNotebooks;
04706     Optional<BusinessNotebook> businessNotebook;
04716     Optional<User> contact;
04718     Optional<NotebookRestrictions> restrictions;
04724     Optional<NotebookRecipientSettings> recipientSettings;
04725
04726     virtual void print(QTextStream & strm) const override;
04727
04728     bool operator==(const Notebook & other) const
04729     {
04730         return guid.isEqual(other.guid)
04731             && name.isEqual(other.name)
04732             && updateSequenceNum.isEqual(other.updateSequenceNum)
04733             && defaultNotebook.isEqual(other.defaultNotebook)
04734             && serviceCreated.isEqual(other.serviceCreated)
04735             && serviceUpdated.isEqual(other.serviceUpdated)
04736             && publishing.isEqual(other.publishing)
04737             && published.isEqual(other.published)
04738             && stack.isEqual(other.stack)
04739             && sharedNotebookIds.isEqual(other.sharedNotebookIds)
04740             && sharedNotebooks.isEqual(other.sharedNotebooks)
04741             && businessNotebook.isEqual(other.businessNotebook)
04742             && contact.isEqual(other.contact)
04743             && restrictions.isEqual(other.restrictions)
04744             && recipientSettings.isEqual(other.recipientSettings)
04745         ;
04746     }
04747
04748     bool operator!=(const Notebook & other) const
04749     {
04750         return !(*this == other);
04751     }
04752
04753     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
04754     Q_PROPERTY(Optional<Guid> guid MEMBER guid)
04755     Q_PROPERTY(Optional<QString> name MEMBER name)
04756     Q_PROPERTY(Optional<qint32> updateSequenceNum MEMBER updateSequenceNum)
04757     Q_PROPERTY(Optional<bool> defaultNotebook MEMBER defaultNotebook)
04758     Q_PROPERTY(Optional<Timestamp> serviceCreated MEMBER serviceCreated)
04759     Q_PROPERTY(Optional<Timestamp> serviceUpdated MEMBER serviceUpdated)
04760     Q_PROPERTY(Optional<Publishing> publishing MEMBER publishing)
04761     Q_PROPERTY(Optional<bool> published MEMBER published)
```

```
04762     Q_PROPERTY(Optional<QString> stack MEMBER stack)
04763     Q_PROPERTY(Optional<QList<qint64» sharedNotebookIds MEMBER sharedNotebookIds)
04764     Q_PROPERTY(Optional<QList<SharedNotebook» sharedNotebooks MEMBER sharedNotebooks)
04765     Q_PROPERTY(Optional<BusinessNotebook> businessNotebook MEMBER businessNotebook)
04766     Q_PROPERTY(Optional<User> contact MEMBER contact)
04767     Q_PROPERTY(Optional<NotebookRestrictions> restrictions MEMBER restrictions)
04768     Q_PROPERTY(Optional<NotebookRecipientSettings> recipientSettings MEMBER recipientSettings)
04769 };
04770
04776 struct QEVERCLOUD_EXPORT LinkedNotebook: public Printable
04777 {
04778 private:
04779     Q_GADGET
04780 public:
04784     EverCloudLocalData localData;
04785
04789     Optional<QString> shareName;
04793     Optional<QString> username;
04800     Optional<QString> shardId;
04808     Optional<QString> sharedNotebookGlobalId;
04810     Optional<QString> uri;
04820     Optional<Guid> guid;
04827     Optional<qint32> updateSequenceNum;
04834     Optional<QString> noteStoreUrl;
04844     Optional<QString> webApiUrlPrefix;
04853     Optional<QString> stack;
04858     Optional<qint32> businessId;
04859
04860     virtual void print(QTextStream & strm) const override;
04861
04862     bool operator==(const LinkedNotebook & other) const
04863     {
04864         return shareName.isEqual(other.shareName)
04865             && username.isEqual(other.username)
04866             && shardId.isEqual(other.shardId)
04867             && sharedNotebookGlobalId.isEqual(other.sharedNotebookGlobalId)
04868             && uri.isEqual(other.uri)
04869             && guid.isEqual(other.guid)
04870             && updateSequenceNum.isEqual(other.updateSequenceNum)
04871             && noteStoreUrl.isEqual(other.noteStoreUrl)
04872             && webApiUrlPrefix.isEqual(other.webApiUrlPrefix)
04873             && stack.isEqual(other.stack)
04874             && businessId.isEqual(other.businessId)
04875         ;
04876     }
04877
04878     bool operator!=(const LinkedNotebook & other) const
04879     {
04880         return !(*this == other);
04881     }
04882
04883     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
04884     Q_PROPERTY(Optional<QString> shareName MEMBER shareName)
04885     Q_PROPERTY(Optional<QString> username MEMBER username)
04886     Q_PROPERTY(Optional<QString> shardId MEMBER shardId)
04887     Q_PROPERTY(Optional<QString> sharedNotebookGlobalId MEMBER sharedNotebookGlobalId)
04888     Q_PROPERTY(Optional<QString> uri MEMBER uri)
04889     Q_PROPERTY(Optional<Guid> guid MEMBER guid)
04890     Q_PROPERTY(Optional<qint32> updateSequenceNum MEMBER updateSequenceNum)
04891     Q_PROPERTY(Optional<QString> noteStoreUrl MEMBER noteStoreUrl)
04892     Q_PROPERTY(Optional<QString> webApiUrlPrefix MEMBER webApiUrlPrefix)
04893     Q_PROPERTY(Optional<QString> stack MEMBER stack)
04894     Q_PROPERTY(Optional<qint32> businessId MEMBER businessId)
04895 };
04896
04902 struct QEVERCLOUD_EXPORT NotebookDescriptor: public Printable
04903 {
04904 private:
04905     Q_GADGET
04906 public:
04910     EverCloudLocalData localData;
04911
04915     Optional<Guid> guid;
04920     Optional<QString> notebookDisplayName;
04924     Optional<QString> contactName;
04929     Optional<bool> hasSharedNotebook;
04933     Optional<qint32> joinedUserCount;
04934
04935     virtual void print(QTextStream & strm) const override;
04936
04937     bool operator==(const NotebookDescriptor & other) const
04938     {
04939         return guid.isEqual(other.guid)
04940             && notebookDisplayName.isEqual(other.notebookDisplayName)
04941             && contactName.isEqual(other.contactName)
04942             && hasSharedNotebook.isEqual(other.hasSharedNotebook)
04943             && joinedUserCount.isEqual(other.joinedUserCount)
```

```
04944         ;
04945     }
04946
04947     bool operator!=(const NotebookDescriptor & other) const
04948     {
04949         return !(*this == other);
04950     }
04951
04952     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
04953     Q_PROPERTY(Optional<Guid> guid MEMBER guid)
04954     Q_PROPERTY(Optional<QString> notebookDisplayName MEMBER notebookDisplayName)
04955     Q_PROPERTY(Optional<QString> contactName MEMBER contactName)
04956     Q_PROPERTY(Optional<bool> hasSharedNotebook MEMBER hasSharedNotebook)
04957     Q_PROPERTY(Optional<qint32> joinedUserCount MEMBER joinedUserCount)
04958 };
04959
04964 struct QEVERCLOUD_EXPORT UserProfile: public Printable
04965 {
04966 private:
04967     Q_GADGET
04968 public:
04972     EverCloudLocalData localData;
04973
04977     Optional<UserID> id;
04981     Optional<QString> name;
04986     Optional<QString> email;
04990     Optional<QString> username;
04994     Optional<BusinessUserAttributes> attributes;
04998     Optional<Timestamp> joined;
05002     Optional<Timestamp> photoLastUpdated;
05006     Optional<QString> photoUrl;
05010     Optional<BusinessUserRole> role;
05014     Optional<BusinessUserStatus> status;
05015
05016     virtual void print(QTextStream & strm) const override;
05017
05018     bool operator==(const UserProfile & other) const
05019     {
05020         return id.isEqual(other.id)
05021             && name.isEqual(other.name)
05022             && email.isEqual(other.email)
05023             && username.isEqual(other.username)
05024             && attributes.isEqual(other.attributes)
05025             && joined.isEqual(other.joined)
05026             && photoLastUpdated.isEqual(other.photoLastUpdated)
05027             && photoUrl.isEqual(other.photoUrl)
05028             && role.isEqual(other.role)
05029             && status.isEqual(other.status)
05030         ;
05031     }
05032
05033     bool operator!=(const UserProfile & other) const
05034     {
05035         return !(*this == other);
05036     }
05037
05038     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
05039     Q_PROPERTY(Optional<UserID> id MEMBER id)
05040     Q_PROPERTY(Optional<QString> name MEMBER name)
05041     Q_PROPERTY(Optional<QString> email MEMBER email)
05042     Q_PROPERTY(Optional<QString> username MEMBER username)
05043     Q_PROPERTY(Optional<BusinessUserAttributes> attributes MEMBER attributes)
05044     Q_PROPERTY(Optional<Timestamp> joined MEMBER joined)
05045     Q_PROPERTY(Optional<Timestamp> photoLastUpdated MEMBER photoLastUpdated)
05046     Q_PROPERTY(Optional<QString> photoUrl MEMBER photoUrl)
05047     Q_PROPERTY(Optional<BusinessUserRole> role MEMBER role)
05048     Q_PROPERTY(Optional<BusinessUserStatus> status MEMBER status)
05049 };
05050
05057 struct QEVERCLOUD_EXPORT RelatedContentImage: public Printable
05058 {
05059 private:
05060     Q_GADGET
05061 public:
05065     EverCloudLocalData localData;
05066
05070     Optional<QString> url;
05074     Optional<qint32> width;
05078     Optional<qint32> height;
05082     Optional<double> pixelRatio;
05086     Optional<qint32> fileSize;
05087
05088     virtual void print(QTextStream & strm) const override;
05089
05090     bool operator==(const RelatedContentImage & other) const
05091     {
05092         return url.isEqual(other.url)
```

```
05093             && width.isEqual(other.width)
05094             && height.isEqual(other.height)
05095             && pixelRatio.isEqual(other.pixelRatio)
05096             && fileSize.isEqual(other.fileSize)
05097         ;
05098     }
05099
05100     bool operator!=(const RelatedContentImage & other) const
05101     {
05102         return !(*this == other);
05103     }
05104
05105     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
05106     Q_PROPERTY(Optional<QString> url MEMBER url)
05107     Q_PROPERTY(Optional<qint32> width MEMBER width)
05108     Q_PROPERTY(Optional<qint32> height MEMBER height)
05109     Q_PROPERTY(Optional<double> pixelRatio MEMBER pixelRatio)
05110     Q_PROPERTY(Optional<qint32> fileSize MEMBER fileSize)
05111 };
05112
05118 struct QEVERCLOUD_EXPORT RelatedContent: public Printable
05119 {
05120 private:
05121     Q_GADGET
05122 public:
05126     EverCloudLocalData localData;
05127
05131     Optional<QString> contentId;
05135     Optional<QString> title;
05139     Optional<QString> url;
05143     Optional<QString> sourceId;
05147     Optional<QString> sourceUrl;
05151     Optional<QString> sourceFaviconUrl;
05155     Optional<QString> sourceName;
05159     Optional<Timestamp> date;
05164     Optional<QString> teaser;
05168     Optional<QList<RelatedContentImage>> thumbnails;
05172     Optional<RelatedContentType> contentType;
05177     Optional<RelatedContentAccess> accessType;
05183     Optional<QString> visibleUrl;
05189     Optional<QString> clipUrl;
05194     Optional<Contact> contact;
05198     Optional<QStringList> authors;
05199
05200     virtual void print(QTextStream & strm) const override;
05201
05202     bool operator==(const RelatedContent & other) const
05203     {
05204         return contentId.isEqual(other.contentId)
05205             && title.isEqual(other.title)
05206             && url.isEqual(other.url)
05207             && sourceId.isEqual(other.sourceId)
05208             && sourceUrl.isEqual(other.sourceUrl)
05209             && sourceFaviconUrl.isEqual(other.sourceFaviconUrl)
05210             && sourceName.isEqual(other.sourceName)
05211             && date.isEqual(other.date)
05212             && teaser.isEqual(other.teaser)
05213             && thumbnails.isEqual(other.thumbnails)
05214             && contentType.isEqual(other.contentType)
05215             && accessType.isEqual(other.accessType)
05216             && visibleUrl.isEqual(other.visibleUrl)
05217             && clipUrl.isEqual(other.clipUrl)
05218             && contact.isEqual(other.contact)
05219             && authors.isEqual(other.authors)
05220         ;
05221     }
05222
05223     bool operator!=(const RelatedContent & other) const
05224     {
05225         return !(*this == other);
05226     }
05227
05228     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
05229     Q_PROPERTY(Optional<QString> contentId MEMBER contentId)
05230     Q_PROPERTY(Optional<QString> title MEMBER title)
05231     Q_PROPERTY(Optional<QString> url MEMBER url)
05232     Q_PROPERTY(Optional<QString> sourceId MEMBER sourceId)
05233     Q_PROPERTY(Optional<QString> sourceUrl MEMBER sourceUrl)
05234     Q_PROPERTY(Optional<QString> sourceFaviconUrl MEMBER sourceFaviconUrl)
05235     Q_PROPERTY(Optional<QString> sourceName MEMBER sourceName)
05236     Q_PROPERTY(Optional<Timestamp> date MEMBER date)
05237     Q_PROPERTY(Optional<QString> teaser MEMBER teaser)
05238     Q_PROPERTY(Optional<QList<RelatedContentImage» thumbnails MEMBER thumbnails)
05239     Q_PROPERTY(Optional<RelatedContentType> contentType MEMBER contentType)
05240     Q_PROPERTY(Optional<RelatedContentAccess> accessType MEMBER accessType)
05241     Q_PROPERTY(Optional<QString> visibleUrl MEMBER visibleUrl)
05242     Q_PROPERTY(Optional<QString> clipUrl MEMBER clipUrl)
```

```
05243     Q_PROPERTY(Optional<Contact> contact MEMBER contact)
05244     Q_PROPERTY(Optional<QStringList> authors MEMBER authors)
05245 };
05246
05251 struct QEVERCLOUD_EXPORT BusinessInvitation: public Printable
05252 {
05253 private:
05254     Q_GADGET
05255 public:
05259     EverCloudLocalData localData;
05260
05264     Optional<qint32> businessId;
05268     Optional<QString> email;
05272     Optional<BusinessUserRole> role;
05276     Optional<BusinessInvitationStatus> status;
05282     Optional<UserID> requesterId;
05287     Optional<bool> fromWorkChat;
05291     Optional<Timestamp> created;
05295     Optional<Timestamp> mostRecentReminder;
05296
05297     virtual void print(QTextStream & strm) const override;
05298
05299     bool operator==(const BusinessInvitation & other) const
05300     {
05301         return businessId.isEqual(other.businessId)
05302             && email.isEqual(other.email)
05303             && role.isEqual(other.role)
05304             && status.isEqual(other.status)
05305             && requesterId.isEqual(other.requesterId)
05306             && fromWorkChat.isEqual(other.fromWorkChat)
05307             && created.isEqual(other.created)
05308             && mostRecentReminder.isEqual(other.mostRecentReminder)
05309         ;
05310     }
05311
05312     bool operator!=(const BusinessInvitation & other) const
05313     {
05314         return !(*this == other);
05315     }
05316
05317     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
05318     Q_PROPERTY(Optional<qint32> businessId MEMBER businessId)
05319     Q_PROPERTY(Optional<QString> email MEMBER email)
05320     Q_PROPERTY(Optional<BusinessUserRole> role MEMBER role)
05321     Q_PROPERTY(Optional<BusinessInvitationStatus> status MEMBER status)
05322     Q_PROPERTY(Optional<UserID> requesterId MEMBER requesterId)
05323     Q_PROPERTY(Optional<bool> fromWorkChat MEMBER fromWorkChat)
05324     Q_PROPERTY(Optional<Timestamp> created MEMBER created)
05325     Q_PROPERTY(Optional<Timestamp> mostRecentReminder MEMBER mostRecentReminder)
05326 };
05327
05357 struct QEVERCLOUD_EXPORT UserIdentity: public Printable
05358 {
05359 private:
05360     Q_GADGET
05361 public:
05365     EverCloudLocalData localData;
05366
05368     Optional<UserIdentityType> type;
05370     Optional<QString> stringIdentifier;
05372     Optional<qint64> longIdentifier;
05373
05374     virtual void print(QTextStream & strm) const override;
05375
05376     bool operator==(const UserIdentity & other) const
05377     {
05378         return type.isEqual(other.type)
05379             && stringIdentifier.isEqual(other.stringIdentifier)
05380             && longIdentifier.isEqual(other.longIdentifier)
05381         ;
05382     }
05383
05384     bool operator!=(const UserIdentity & other) const
05385     {
05386         return !(*this == other);
05387     }
05388
05389     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
05390     Q_PROPERTY(Optional<UserIdentityType> type MEMBER type)
05391     Q_PROPERTY(Optional<QString> stringIdentifier MEMBER stringIdentifier)
05392     Q_PROPERTY(Optional<qint64> longIdentifier MEMBER longIdentifier)
05393 };
05394
05399 struct QEVERCLOUD_EXPORT PublicUserInfo: public Printable
05400 {
05401 private:
05402     Q_GADGET
```

```
05403 public:
05407     EverCloudLocalData localData;
05408
05412     UserID userId = 0;
05416     Optional<ServiceLevel> serviceLevel;
05418     Optional<QString> username;
05425     Optional<QString> noteStoreUrl;
05435     Optional<QString> webApiUrlPrefix;
05436
05437     virtual void print(QTextStream & strm) const override;
05438
05439     bool operator==(const PublicUserInfo & other) const
05440     {
05441         return (userId == other.userId)
05442             && serviceLevel.isEqual(other.serviceLevel)
05443             && username.isEqual(other.username)
05444             && noteStoreUrl.isEqual(other.noteStoreUrl)
05445             && webApiUrlPrefix.isEqual(other.webApiUrlPrefix)
05446         ;
05447     }
05448
05449     bool operator!=(const PublicUserInfo & other) const
05450     {
05451         return !(*this == other);
05452     }
05453
05454     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
05455     Q_PROPERTY(UserID userId MEMBER userId)
05456     Q_PROPERTY(Optional<ServiceLevel> serviceLevel MEMBER serviceLevel)
05457     Q_PROPERTY(Optional<QString> username MEMBER username)
05458     Q_PROPERTY(Optional<QString> noteStoreUrl MEMBER noteStoreUrl)
05459     Q_PROPERTY(Optional<QString> webApiUrlPrefix MEMBER webApiUrlPrefix)
05460 };
05461
05464 struct QEVERCLOUD_EXPORT UserUrls: public Printable
05465 {
05466 private:
05467     Q_GADGET
05468 public:
05472     EverCloudLocalData localData;
05473
05480     Optional<QString> noteStoreUrl;
05490     Optional<QString> webApiUrlPrefix;
05497     Optional<QString> userStoreUrl;
05504     Optional<QString> utilityUrl;
05511     Optional<QString> messageStoreUrl;
05516     Optional<QString> userWebSocketUrl;
05517
05518     virtual void print(QTextStream & strm) const override;
05519
05520     bool operator==(const UserUrls & other) const
05521     {
05522         return noteStoreUrl.isEqual(other.noteStoreUrl)
05523             && webApiUrlPrefix.isEqual(other.webApiUrlPrefix)
05524             && userStoreUrl.isEqual(other.userStoreUrl)
05525             && utilityUrl.isEqual(other.utilityUrl)
05526             && messageStoreUrl.isEqual(other.messageStoreUrl)
05527             && userWebSocketUrl.isEqual(other.userWebSocketUrl)
05528         ;
05529     }
05530
05531     bool operator!=(const UserUrls & other) const
05532     {
05533         return !(*this == other);
05534     }
05535
05536     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
05537     Q_PROPERTY(Optional<QString> noteStoreUrl MEMBER noteStoreUrl)
05538     Q_PROPERTY(Optional<QString> webApiUrlPrefix MEMBER webApiUrlPrefix)
05539     Q_PROPERTY(Optional<QString> userStoreUrl MEMBER userStoreUrl)
05540     Q_PROPERTY(Optional<QString> utilityUrl MEMBER utilityUrl)
05541     Q_PROPERTY(Optional<QString> messageStoreUrl MEMBER messageStoreUrl)
05542     Q_PROPERTY(Optional<QString> userWebSocketUrl MEMBER userWebSocketUrl)
05543 };
05544
05549 struct QEVERCLOUD_EXPORT AuthenticationResult: public Printable
05550 {
05551 private:
05552     Q_GADGET
05553 public:
05557     EverCloudLocalData localData;
05558
05563     Timestamp currentTime = 0;
05568     QString authenticationToken;
05575     Timestamp expiration = 0;
05581     Optional<User> user;
05587     Optional<PublicUserInfo> publicUserInfo;
```

```
05591     Optional<QString> noteStoreUrl;
05595     Optional<QString> webApiUrlPrefix;
05603     Optional<bool> secondFactorRequired;
05611     Optional<QString> secondFactorDeliveryHint;
05616     Optional<UserUrls> urls;
05617
05618     virtual void print(QTextStream & strm) const override;
05619
05620     bool operator==(const AuthenticationResult & other) const
05621     {
05622         return (currentTime == other.currentTime)
05623             && (authenticationToken == other.authenticationToken)
05624             && (expiration == other.expiration)
05625             && user.isEqual(other.user)
05626             && publicUserInfo.isEqual(other.publicUserInfo)
05627             && noteStoreUrl.isEqual(other.noteStoreUrl)
05628             && webApiUrlPrefix.isEqual(other.webApiUrlPrefix)
05629             && secondFactorRequired.isEqual(other.secondFactorRequired)
05630             && secondFactorDeliveryHint.isEqual(other.secondFactorDeliveryHint)
05631             && urls.isEqual(other.urls)
05632         ;
05633     }
05634
05635     bool operator!=(const AuthenticationResult & other) const
05636     {
05637         return !(*this == other);
05638     }
05639
05640     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
05641     Q_PROPERTY(Timestamp currentTime MEMBER currentTime)
05642     Q_PROPERTY(QString authenticationToken MEMBER authenticationToken)
05643     Q_PROPERTY(Timestamp expiration MEMBER expiration)
05644     Q_PROPERTY(Optional<User> user MEMBER user)
05645     Q_PROPERTY(Optional<PublicUserInfo> publicUserInfo MEMBER publicUserInfo)
05646     Q_PROPERTY(Optional<QString> noteStoreUrl MEMBER noteStoreUrl)
05647     Q_PROPERTY(Optional<QString> webApiUrlPrefix MEMBER webApiUrlPrefix)
05648     Q_PROPERTY(Optional<bool> secondFactorRequired MEMBER secondFactorRequired)
05649     Q_PROPERTY(Optional<QString> secondFactorDeliveryHint MEMBER secondFactorDeliveryHint)
05650     Q_PROPERTY(Optional<UserUrls> urls MEMBER urls)
05651 };
05652
05656 struct QEVERCLOUD_EXPORT BootstrapSettings: public Printable
05657 {
05658 private:
05659     Q_GADGET
05660 public:
05664     EverCloudLocalData localData;
05665
05676     QString serviceHost;
05682     QString marketingUrl;
05686     QString supportUrl;
05691     QString accountEmailDomain;
05695     Optional<bool> enableFacebookSharing;
05699     Optional<bool> enableGiftSubscriptions;
05703     Optional<bool> enableSupportTickets;
05707     Optional<bool> enableSharedNotebooks;
05711     Optional<bool> enableSingleNoteSharing;
05715     Optional<bool> enableSponsoredAccounts;
05719     Optional<bool> enableTwitterSharing;
05721     Optional<bool> enableLinkedInSharing;
05723     Optional<bool> enablePublicNotebooks;
05728     Optional<bool> enableGoogle;
05729
05730     virtual void print(QTextStream & strm) const override;
05731
05732     bool operator==(const BootstrapSettings & other) const
05733     {
05734         return (serviceHost == other.serviceHost)
05735             && (marketingUrl == other.marketingUrl)
05736             && (supportUrl == other.supportUrl)
05737             && (accountEmailDomain == other.accountEmailDomain)
05738             && enableFacebookSharing.isEqual(other.enableFacebookSharing)
05739             && enableGiftSubscriptions.isEqual(other.enableGiftSubscriptions)
05740             && enableSupportTickets.isEqual(other.enableSupportTickets)
05741             && enableSharedNotebooks.isEqual(other.enableSharedNotebooks)
05742             && enableSingleNoteSharing.isEqual(other.enableSingleNoteSharing)
05743             && enableSponsoredAccounts.isEqual(other.enableSponsoredAccounts)
05744             && enableTwitterSharing.isEqual(other.enableTwitterSharing)
05745             && enableLinkedInSharing.isEqual(other.enableLinkedInSharing)
05746             && enablePublicNotebooks.isEqual(other.enablePublicNotebooks)
05747             && enableGoogle.isEqual(other.enableGoogle)
05748         ;
05749     }
05750
05751     bool operator!=(const BootstrapSettings & other) const
05752     {
05753         return !(*this == other);
```

```
05754     }
05755
05756     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
05757     Q_PROPERTY(QString serviceHost MEMBER serviceHost)
05758     Q_PROPERTY(QString marketingUrl MEMBER marketingUrl)
05759     Q_PROPERTY(QString supportUrl MEMBER supportUrl)
05760     Q_PROPERTY(QString accountEmailDomain MEMBER accountEmailDomain)
05761     Q_PROPERTY(Optional<bool> enableFacebookSharing MEMBER enableFacebookSharing)
05762     Q_PROPERTY(Optional<bool> enableGiftSubscriptions MEMBER enableGiftSubscriptions)
05763     Q_PROPERTY(Optional<bool> enableSupportTickets MEMBER enableSupportTickets)
05764     Q_PROPERTY(Optional<bool> enableSharedNotebooks MEMBER enableSharedNotebooks)
05765     Q_PROPERTY(Optional<bool> enableSingleNoteSharing MEMBER enableSingleNoteSharing)
05766     Q_PROPERTY(Optional<bool> enableSponsoredAccounts MEMBER enableSponsoredAccounts)
05767     Q_PROPERTY(Optional<bool> enableTwitterSharing MEMBER enableTwitterSharing)
05768     Q_PROPERTY(Optional<bool> enableLinkedInSharing MEMBER enableLinkedInSharing)
05769     Q_PROPERTY(Optional<bool> enablePublicNotebooks MEMBER enablePublicNotebooks)
05770     Q_PROPERTY(Optional<bool> enableGoogle MEMBER enableGoogle)
05771 };
05772
05776 struct QEVERCLOUD_EXPORT BootstrapProfile: public Printable
05777 {
05778 private:
05779     Q_GADGET
05780 public:
05784     EverCloudLocalData localData;
05785
05790     QString name;
05794     BootstrapSettings settings;
05795
05796     virtual void print(QTextStream & strm) const override;
05797
05798     bool operator==(const BootstrapProfile & other) const
05799     {
05800         return (name == other.name)
05801             && (settings == other.settings)
05802         ;
05803     }
05804
05805     bool operator!=(const BootstrapProfile & other) const
05806     {
05807         return !(*this == other);
05808     }
05809
05810     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
05811     Q_PROPERTY(QString name MEMBER name)
05812     Q_PROPERTY(BootstrapSettings settings MEMBER settings)
05813 };
05814
05818 struct QEVERCLOUD_EXPORT BootstrapInfo: public Printable
05819 {
05820 private:
05821     Q_GADGET
05822 public:
05826     EverCloudLocalData localData;
05827
05832     QList<BootstrapProfile> profiles;
05833
05834     virtual void print(QTextStream & strm) const override;
05835
05836     bool operator==(const BootstrapInfo & other) const
05837     {
05838         return (profiles == other.profiles)
05839         ;
05840     }
05841
05842     bool operator!=(const BootstrapInfo & other) const
05843     {
05844         return !(*this == other);
05845     }
05846
05847     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
05848     Q_PROPERTY(QList<BootstrapProfile> profiles MEMBER profiles)
05849 };
05850
05869 class QEVERCLOUD_EXPORT EDAMUserException: public EvernoteException, public Printable
05870 {
05871     Q_GADGET
05872 public:
05873     EDAMErrorCode errorCode;
05874     Optional<QString> parameter;
05875
05876     EDAMUserException();
05877     virtual ~EDAMUserException() noexcept override;
05878
05879     EDAMUserException(const EDAMUserException & other);
05880     const char * what() const noexcept override;
05881     virtual EverCloudExceptionDataPtr exceptionData() const override;
```

```
05882
05883     virtual void print(QTextStream & strm) const override;
05884
05885     bool operator==(const EDAMUserException & other) const
05886     {
05887         return (errorCode == other.errorCode)
05888             && parameter.isEqual(other.parameter)
05889         ;
05890     }
05891
05892     bool operator!=(const EDAMUserException & other) const
05893     {
05894         return !(*this == other);
05895     }
05896
05897     Q_PROPERTY(EDAMErrorCode errorCode MEMBER errorCode)
05898     Q_PROPERTY(Optional<QString> parameter MEMBER parameter)
05899 };
05900
05915 class QEVERCLOUD_EXPORT EDAMSystemException: public EvernoteException, public Printable
05916 {
05917     Q_GADGET
05918 public:
05919     EDAMErrorCode errorCode;
05920     Optional<QString> message;
05921     Optional<qint32> rateLimitDuration;
05922
05923     EDAMSystemException();
05924     virtual ~EDAMSystemException() noexcept override;
05925
05926     EDAMSystemException(const EDAMSystemException & other);
05927     const char * what() const noexcept override;
05928     virtual EverCloudExceptionDataPtr exceptionData() const override;
05929
05930     virtual void print(QTextStream & strm) const override;
05931
05932     bool operator==(const EDAMSystemException & other) const
05933     {
05934         return (errorCode == other.errorCode)
05935             && message.isEqual(other.message)
05936             && rateLimitDuration.isEqual(other.rateLimitDuration)
05937         ;
05938     }
05939
05940     bool operator!=(const EDAMSystemException & other) const
05941     {
05942         return !(*this == other);
05943     }
05944
05945     Q_PROPERTY(EDAMErrorCode errorCode MEMBER errorCode)
05946     Q_PROPERTY(Optional<QString> message MEMBER message)
05947     Q_PROPERTY(Optional<qint32> rateLimitDuration MEMBER rateLimitDuration)
05948 };
05949
05963 class QEVERCLOUD_EXPORT EDAMNotFoundException: public EvernoteException, public Printable
05964 {
05965     Q_GADGET
05966 public:
05967     Optional<QString> identifier;
05968     Optional<QString> key;
05969
05970     EDAMNotFoundException();
05971     virtual ~EDAMNotFoundException() noexcept override;
05972
05973     EDAMNotFoundException(const EDAMNotFoundException & other);
05974     const char * what() const noexcept override;
05975     virtual EverCloudExceptionDataPtr exceptionData() const override;
05976
05977     virtual void print(QTextStream & strm) const override;
05978
05979     bool operator==(const EDAMNotFoundException & other) const
05980     {
05981         return identifier.isEqual(other.identifier)
05982             && key.isEqual(other.key)
05983         ;
05984     }
05985
05986     bool operator!=(const EDAMNotFoundException & other) const
05987     {
05988         return !(*this == other);
05989     }
05990
05991     Q_PROPERTY(Optional<QString> identifier MEMBER identifier)
05992     Q_PROPERTY(Optional<QString> key MEMBER key)
05993 };
05994
06017 class QEVERCLOUD_EXPORT EDAMInvalidContactsException: public EvernoteException, public Printable
```

```
06018 {
06019     Q_GADGET
06020 public:
06021     QList<Contact> contacts;
06022     Optional<QString> parameter;
06023     Optional<QList<EDAMInvalidContactReason>> reasons;
06024
06025     EDAMInvalidContactsException();
06026     virtual ~EDAMInvalidContactsException() noexcept override;
06027
06028     EDAMInvalidContactsException(const EDAMInvalidContactsException & other);
06029     const char * what() const noexcept override;
06030     virtual EverCloudExceptionDataPtr exceptionData() const override;
06031
06032     virtual void print(QTextStream & strm) const override;
06033
06034     bool operator==(const EDAMInvalidContactsException & other) const
06035     {
06036         return (contacts == other.contacts)
06037             && parameter.isEqual(other.parameter)
06038             && reasons.isEqual(other.reasons)
06039         ;
06040     }
06041
06042     bool operator!=(const EDAMInvalidContactsException & other) const
06043     {
06044         return !(*this == other);
06045     }
06046
06047     Q_PROPERTY(QList<Contact> contacts MEMBER contacts)
06048     Q_PROPERTY(Optional<QString> parameter MEMBER parameter)
06049     Q_PROPERTY(Optional<QList<EDAMInvalidContactReason» reasons MEMBER reasons)
06050 };
06051
06063 struct QEVERCLOUD_EXPORT SyncChunk: public Printable
06064 {
06065 private:
06066     Q_GADGET
06067 public:
06071     EverCloudLocalData localData;
06072
06076     Timestamp currentTime = 0;
06082     Optional<qint32> chunkHighUSN;
06090     qint32 updateCount = 0;
06098     Optional<QList<Note>> notes;
06103     Optional<QList<Notebook>> notebooks;
06108     Optional<QList<Tag>> tags;
06113     Optional<QList<SavedSearch>> searches;
06120     Optional<QList<Resource>> resources;
06125     Optional<QList<Guid>> expungedNotes;
06132     Optional<QList<Guid>> expungedNotebooks;
06137     Optional<QList<Guid>> expungedTags;
06142     Optional<QList<Guid>> expungedSearches;
06147     Optional<QList<LinkedNotebook>> linkedNotebooks;
06152     Optional<QList<Guid>> expungedLinkedNotebooks;
06153
06154     virtual void print(QTextStream & strm) const override;
06155
06156     bool operator==(const SyncChunk & other) const
06157     {
06158         return (currentTime == other.currentTime)
06159             && chunkHighUSN.isEqual(other.chunkHighUSN)
06160             && (updateCount == other.updateCount)
06161             && notes.isEqual(other.notes)
06162             && notebooks.isEqual(other.notebooks)
06163             && tags.isEqual(other.tags)
06164             && searches.isEqual(other.searches)
06165             && resources.isEqual(other.resources)
06166             && expungedNotes.isEqual(other.expungedNotes)
06167             && expungedNotebooks.isEqual(other.expungedNotebooks)
06168             && expungedTags.isEqual(other.expungedTags)
06169             && expungedSearches.isEqual(other.expungedSearches)
06170             && linkedNotebooks.isEqual(other.linkedNotebooks)
06171             && expungedLinkedNotebooks.isEqual(other.expungedLinkedNotebooks)
06172         ;
06173     }
06174
06175     bool operator!=(const SyncChunk & other) const
06176     {
06177         return !(*this == other);
06178     }
06179
06180     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
06181     Q_PROPERTY(Timestamp currentTime MEMBER currentTime)
06182     Q_PROPERTY(Optional<qint32> chunkHighUSN MEMBER chunkHighUSN)
06183     Q_PROPERTY(qint32 updateCount MEMBER updateCount)
06184     Q_PROPERTY(Optional<QList<Note» notes MEMBER notes)
```

```
06185     Q_PROPERTY(Optional<QList<Notebook» notebooks MEMBER notebooks)
06186     Q_PROPERTY(Optional<QList<Tag» tags MEMBER tags)
06187     Q_PROPERTY(Optional<QList<SavedSearch» searches MEMBER searches)
06188     Q_PROPERTY(Optional<QList<Resource» resources MEMBER resources)
06189     Q_PROPERTY(Optional<QList<Guid» expungedNotes MEMBER expungedNotes)
06190     Q_PROPERTY(Optional<QList<Guid» expungedNotebooks MEMBER expungedNotebooks)
06191     Q_PROPERTY(Optional<QList<Guid» expungedTags MEMBER expungedTags)
06192     Q_PROPERTY(Optional<QList<Guid» expungedSearches MEMBER expungedSearches)
06193     Q_PROPERTY(Optional<QList<LinkedNotebook» linkedNotebooks MEMBER linkedNotebooks)
06194     Q_PROPERTY(Optional<QList<Guid» expungedLinkedNotebooks MEMBER expungedLinkedNotebooks)
06195 };
06196
06201 struct QEVERCLOUD_EXPORT NoteList: public Printable
06202 {
06203 private:
06204     Q_GADGET
06205 public:
06209     EverCloudLocalData localData;
06210
06215     qint32 startIndex = 0;
06221     qint32 totalNotes = 0;
06227     QList<Note> notes;
06233     Optional<QStringList> stoppedWords;
06240     Optional<QStringList> searchedWords;
06249     Optional<qint32> updateCount;
06253     Optional<QByteArray> searchContextBytes;
06258     Optional<QString> debugInfo;
06259
06260     virtual void print(QTextStream & strm) const override;
06261
06262     bool operator==(const NoteList & other) const
06263     {
06264         return (startIndex == other.startIndex)
06265             && (totalNotes == other.totalNotes)
06266             && (notes == other.notes)
06267             && stoppedWords.isEqual(other.stoppedWords)
06268             && searchedWords.isEqual(other.searchedWords)
06269             && updateCount.isEqual(other.updateCount)
06270             && searchContextBytes.isEqual(other.searchContextBytes)
06271             && debugInfo.isEqual(other.debugInfo)
06272         ;
06273     }
06274
06275     bool operator!=(const NoteList & other) const
06276     {
06277         return !(*this == other);
06278     }
06279
06280     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
06281     Q_PROPERTY(qint32 startIndex MEMBER startIndex)
06282     Q_PROPERTY(qint32 totalNotes MEMBER totalNotes)
06283     Q_PROPERTY(QList<Note> notes MEMBER notes)
06284     Q_PROPERTY(Optional<QStringList> stoppedWords MEMBER stoppedWords)
06285     Q_PROPERTY(Optional<QStringList> searchedWords MEMBER searchedWords)
06286     Q_PROPERTY(Optional<qint32> updateCount MEMBER updateCount)
06287     Q_PROPERTY(Optional<QByteArray> searchContextBytes MEMBER searchContextBytes)
06288     Q_PROPERTY(Optional<QString> debugInfo MEMBER debugInfo)
06289 };
06290
06301 struct QEVERCLOUD_EXPORT NoteMetadata: public Printable
06302 {
06303 private:
06304     Q_GADGET
06305 public:
06309     EverCloudLocalData localData;
06310
06312     Guid guid;
06314     Optional<QString> title;
06316     Optional<qint32> contentLength;
06318     Optional<Timestamp> created;
06320     Optional<Timestamp> updated;
06322     Optional<Timestamp> deleted;
06324     Optional<qint32> updateSequenceNum;
06326     Optional<QString> notebookGuid;
06328     Optional<QList<Guid>> tagGuids;
06330     Optional<NoteAttributes> attributes;
06336     Optional<QString> largestResourceMime;
06342     Optional<qint32> largestResourceSize;
06343
06344     virtual void print(QTextStream & strm) const override;
06345
06346     bool operator==(const NoteMetadata & other) const
06347     {
06348         return (guid == other.guid)
06349             && title.isEqual(other.title)
06350             && contentLength.isEqual(other.contentLength)
06351             && created.isEqual(other.created)
```

```
06352             && updated.isEqual(other.updated)
06353             && deleted.isEqual(other.deleted)
06354             && updateSequenceNum.isEqual(other.updateSequenceNum)
06355             && notebookGuid.isEqual(other.notebookGuid)
06356             && tagGuids.isEqual(other.tagGuids)
06357             && attributes.isEqual(other.attributes)
06358             && largestResourceMime.isEqual(other.largestResourceMime)
06359             && largestResourceSize.isEqual(other.largestResourceSize)
06360         ;
06361     }
06362
06363     bool operator!=(const NoteMetadata & other) const
06364     {
06365         return !(*this == other);
06366     }
06367
06368     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
06369     Q_PROPERTY(Guid guid MEMBER guid)
06370     Q_PROPERTY(Optional<QString> title MEMBER title)
06371     Q_PROPERTY(Optional<qint32> contentLength MEMBER contentLength)
06372     Q_PROPERTY(Optional<Timestamp> created MEMBER created)
06373     Q_PROPERTY(Optional<Timestamp> updated MEMBER updated)
06374     Q_PROPERTY(Optional<Timestamp> deleted MEMBER deleted)
06375     Q_PROPERTY(Optional<qint32> updateSequenceNum MEMBER updateSequenceNum)
06376     Q_PROPERTY(Optional<QString> notebookGuid MEMBER notebookGuid)
06377     Q_PROPERTY(Optional<QList<Guid» tagGuids MEMBER tagGuids)
06378     Q_PROPERTY(Optional<NoteAttributes> attributes MEMBER attributes)
06379     Q_PROPERTY(Optional<QString> largestResourceMime MEMBER largestResourceMime)
06380     Q_PROPERTY(Optional<qint32> largestResourceSize MEMBER largestResourceSize)
06381 };
06382
06389 struct QEVERCLOUD_EXPORT NotesMetadataList: public Printable
06390 {
06391 private:
06392     Q_GADGET
06393 public:
06397     EverCloudLocalData localData;
06398
06403     qint32 startIndex = 0;
06409     qint32 totalNotes = 0;
06417     QList<NoteMetadata> notes;
06423     Optional<QStringList> stoppedWords;
06430     Optional<QStringList> searchedWords;
06439     Optional<qint32> updateCount;
06443     Optional<QByteArray> searchContextBytes;
06448     Optional<QString> debugInfo;
06449
06450     virtual void print(QTextStream & strm) const override;
06451
06452     bool operator==(const NotesMetadataList & other) const
06453     {
06454         return (startIndex == other.startIndex)
06455             && (totalNotes == other.totalNotes)
06456             && (notes == other.notes)
06457             && stoppedWords.isEqual(other.stoppedWords)
06458             && searchedWords.isEqual(other.searchedWords)
06459             && updateCount.isEqual(other.updateCount)
06460             && searchContextBytes.isEqual(other.searchContextBytes)
06461             && debugInfo.isEqual(other.debugInfo)
06462         ;
06463     }
06464
06465     bool operator!=(const NotesMetadataList & other) const
06466     {
06467         return !(*this == other);
06468     }
06469
06470     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
06471     Q_PROPERTY(qint32 startIndex MEMBER startIndex)
06472     Q_PROPERTY(qint32 totalNotes MEMBER totalNotes)
06473     Q_PROPERTY(QList<NoteMetadata> notes MEMBER notes)
06474     Q_PROPERTY(Optional<QStringList> stoppedWords MEMBER stoppedWords)
06475     Q_PROPERTY(Optional<QStringList> searchedWords MEMBER searchedWords)
06476     Q_PROPERTY(Optional<qint32> updateCount MEMBER updateCount)
06477     Q_PROPERTY(Optional<QByteArray> searchContextBytes MEMBER searchContextBytes)
06478     Q_PROPERTY(Optional<QString> debugInfo MEMBER debugInfo)
06479 };
06480
06486 struct QEVERCLOUD_EXPORT NoteEmailParameters: public Printable
06487 {
06488 private:
06489     Q_GADGET
06490 public:
06494     EverCloudLocalData localData;
06495
06501     Optional<QString> guid;
06508     Optional<Note> note;
```

```
06514     Optional<QStringList> toAddresses;
06520     Optional<QStringList> ccAddresses;
06526     Optional<QString> subject;
06531     Optional<QString> message;
06532
06533     virtual void print(QTextStream & strm) const override;
06534
06535     bool operator==(const NoteEmailParameters & other) const
06536     {
06537         return guid.isEqual(other.guid)
06538             && note.isEqual(other.note)
06539             && toAddresses.isEqual(other.toAddresses)
06540             && ccAddresses.isEqual(other.ccAddresses)
06541             && subject.isEqual(other.subject)
06542             && message.isEqual(other.message)
06543         ;
06544     }
06545
06546     bool operator!=(const NoteEmailParameters & other) const
06547     {
06548         return !(*this == other);
06549     }
06550
06551     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
06552     Q_PROPERTY(Optional<QString> guid MEMBER guid)
06553     Q_PROPERTY(Optional<Note> note MEMBER note)
06554     Q_PROPERTY(Optional<QStringList> toAddresses MEMBER toAddresses)
06555     Q_PROPERTY(Optional<QStringList> ccAddresses MEMBER ccAddresses)
06556     Q_PROPERTY(Optional<QString> subject MEMBER subject)
06557     Q_PROPERTY(Optional<QString> message MEMBER message)
06558 };
06559
06568 struct QEVERCLOUD_EXPORT RelatedResult: public Printable
06569 {
06570 private:
06571     Q_GADGET
06572 public:
06576     EverCloudLocalData localData;
06577
06582     Optional<QList<Note>> notes;
06587     Optional<QList<Notebook>> notebooks;
06592     Optional<QList<Tag>> tags;
06600     Optional<QList<NotebookDescriptor>> containingNotebooks;
06602     Optional<QString> debugInfo;
06607     Optional<QList<UserProfile>> experts;
06612     Optional<QList<RelatedContent>> relatedContent;
06646     Optional<QString> cacheKey;
06655     Optional<qint32> cacheExpires;
06656
06657     virtual void print(QTextStream & strm) const override;
06658
06659     bool operator==(const RelatedResult & other) const
06660     {
06661         return notes.isEqual(other.notes)
06662             && notebooks.isEqual(other.notebooks)
06663             && tags.isEqual(other.tags)
06664             && containingNotebooks.isEqual(other.containingNotebooks)
06665             && debugInfo.isEqual(other.debugInfo)
06666             && experts.isEqual(other.experts)
06667             && relatedContent.isEqual(other.relatedContent)
06668             && cacheKey.isEqual(other.cacheKey)
06669             && cacheExpires.isEqual(other.cacheExpires)
06670         ;
06671     }
06672
06673     bool operator!=(const RelatedResult & other) const
06674     {
06675         return !(*this == other);
06676     }
06677
06678     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
06679     Q_PROPERTY(Optional<QList<Note» notes MEMBER notes)
06680     Q_PROPERTY(Optional<QList<Notebook» notebooks MEMBER notebooks)
06681     Q_PROPERTY(Optional<QList<Tag» tags MEMBER tags)
06682     Q_PROPERTY(Optional<QList<NotebookDescriptor» containingNotebooks MEMBER containingNotebooks)
06683     Q_PROPERTY(Optional<QString> debugInfo MEMBER debugInfo)
06684     Q_PROPERTY(Optional<QList<UserProfile» experts MEMBER experts)
06685     Q_PROPERTY(Optional<QList<RelatedContent» relatedContent MEMBER relatedContent)
06686     Q_PROPERTY(Optional<QString> cacheKey MEMBER cacheKey)
06687     Q_PROPERTY(Optional<qint32> cacheExpires MEMBER cacheExpires)
06688 };
06689
06695 struct QEVERCLOUD_EXPORT UpdateNoteIfUsnMatchesResult: public Printable
06696 {
06697 private:
06698     Q_GADGET
06699 public:
```

```
06703     EverCloudLocalData localData;
06704
06713     Optional<Note> note;
06717     Optional<bool> updated;
06718
06719     virtual void print(QTextStream & strm) const override;
06720
06721     bool operator==(const UpdateNoteIfUsnMatchesResult & other) const
06722     {
06723         return note.isEqual(other.note)
06724             && updated.isEqual(other.updated)
06725         ;
06726     }
06727
06728     bool operator!=(const UpdateNoteIfUsnMatchesResult & other) const
06729     {
06730         return !(*this == other);
06731     }
06732
06733     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
06734     Q_PROPERTY(Optional<Note> note MEMBER note)
06735     Q_PROPERTY(Optional<bool> updated MEMBER updated)
06736 };
06737
06743 struct QEVERCLOUD_EXPORT InvitationShareRelationship: public Printable
06744 {
06745 private:
06746     Q_GADGET
06747 public:
06751     EverCloudLocalData localData;
06752
06757     Optional<QString> displayName;
06766     Optional<UserIdentity> recipientUserIdentity;
06774     Optional<ShareRelationshipPrivilegeLevel> privilege;
06780     Optional<UserID> sharerUserId;
06781
06782     virtual void print(QTextStream & strm) const override;
06783
06784     bool operator==(const InvitationShareRelationship & other) const
06785     {
06786         return displayName.isEqual(other.displayName)
06787             && recipientUserIdentity.isEqual(other.recipientUserIdentity)
06788             && privilege.isEqual(other.privilege)
06789             && sharerUserId.isEqual(other.sharerUserId)
06790         ;
06791     }
06792
06793     bool operator!=(const InvitationShareRelationship & other) const
06794     {
06795         return !(*this == other);
06796     }
06797
06798     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
06799     Q_PROPERTY(Optional<QString> displayName MEMBER displayName)
06800     Q_PROPERTY(Optional<UserIdentity> recipientUserIdentity MEMBER recipientUserIdentity)
06801     Q_PROPERTY(Optional<ShareRelationshipPrivilegeLevel> privilege MEMBER privilege)
06802     Q_PROPERTY(Optional<UserID> sharerUserId MEMBER sharerUserId)
06803 };
06804
06812 struct QEVERCLOUD_EXPORT ShareRelationships: public Printable
06813 {
06814 private:
06815     Q_GADGET
06816 public:
06820     EverCloudLocalData localData;
06821
06826     Optional<QList<InvitationShareRelationship>> invitations;
06832     Optional<QList<MemberShareRelationship>> memberships;
06841     Optional<ShareRelationshipRestrictions> invitationRestrictions;
06842
06843     virtual void print(QTextStream & strm) const override;
06844
06845     bool operator==(const ShareRelationships & other) const
06846     {
06847         return invitations.isEqual(other.invitations)
06848             && memberships.isEqual(other.memberships)
06849             && invitationRestrictions.isEqual(other.invitationRestrictions)
06850         ;
06851     }
06852
06853     bool operator!=(const ShareRelationships & other) const
06854     {
06855         return !(*this == other);
06856     }
06857
06858     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
06859     Q_PROPERTY(Optional<QList<InvitationShareRelationship» invitations MEMBER invitations)
```

```
06860     Q_PROPERTY(Optional<QList<MemberShareRelationship» memberships MEMBER memberships)
06861     Q_PROPERTY(Optional<ShareRelationshipRestrictions> invitationRestrictions MEMBER
      invitationRestrictions)
06862 };
06863
06869 struct QEVERCLOUD_EXPORT ManageNotebookSharesParameters: public Printable
06870 {
06871 private:
06872     Q_GADGET
06873 public:
06877     EverCloudLocalData localData;
06878
06882     Optional<QString> notebookGuid;
06887     Optional<QString> inviteMessage;
06900     Optional<QList<MemberShareRelationship>> membershipsToUpdate;
06914     Optional<QList<InvitationShareRelationship>> invitationsToCreateOrUpdate;
06923     Optional<QList<UserIdentity>> unshares;
06924
06925     virtual void print(QTextStream & strm) const override;
06926
06927     bool operator==(const ManageNotebookSharesParameters & other) const
06928     {
06929         return notebookGuid.isEqual(other.notebookGuid)
06930             && inviteMessage.isEqual(other.inviteMessage)
06931             && membershipsToUpdate.isEqual(other.membershipsToUpdate)
06932             && invitationsToCreateOrUpdate.isEqual(other.invitationsToCreateOrUpdate)
06933             && unshares.isEqual(other.unshares)
06934         ;
06935     }
06936
06937     bool operator!=(const ManageNotebookSharesParameters & other) const
06938     {
06939         return !(*this == other);
06940     }
06941
06942     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
06943     Q_PROPERTY(Optional<QString> notebookGuid MEMBER notebookGuid)
06944     Q_PROPERTY(Optional<QString> inviteMessage MEMBER inviteMessage)
06945     Q_PROPERTY(Optional<QList<MemberShareRelationship» membershipsToUpdate MEMBER membershipsToUpdate)
06946     Q_PROPERTY(Optional<QList<InvitationShareRelationship» invitationsToCreateOrUpdate MEMBER
      invitationsToCreateOrUpdate)
06947     Q_PROPERTY(Optional<QList<UserIdentity» unshares MEMBER unshares)
06948 };
06949
06961 struct QEVERCLOUD_EXPORT ManageNotebookSharesError: public Printable
06962 {
06963 private:
06964     Q_GADGET
06965 public:
06969     EverCloudLocalData localData;
06970
06975     Optional<UserIdentity> userIdentity;
06981     Optional<EDAMUserException> userException;
06987     Optional<EDAMNotFoundException> notFoundException;
06988
06989     virtual void print(QTextStream & strm) const override;
06990
06991     bool operator==(const ManageNotebookSharesError & other) const
06992     {
06993         return userIdentity.isEqual(other.userIdentity)
06994             && userException.isEqual(other.userException)
06995             && notFoundException.isEqual(other.notFoundException)
06996         ;
06997     }
06998
06999     bool operator!=(const ManageNotebookSharesError & other) const
07000     {
07001         return !(*this == other);
07002     }
07003
07004     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
07005     Q_PROPERTY(Optional<UserIdentity> userIdentity MEMBER userIdentity)
07006     Q_PROPERTY(Optional<EDAMUserException> userException MEMBER userException)
07007     Q_PROPERTY(Optional<EDAMNotFoundException> notFoundException MEMBER notFoundException)
07008 };
07009
07014 struct QEVERCLOUD_EXPORT ManageNotebookSharesResult: public Printable
07015 {
07016 private:
07017     Q_GADGET
07018 public:
07022     EverCloudLocalData localData;
07023
07029     Optional<QList<ManageNotebookSharesError>> errors;
07030
07031     virtual void print(QTextStream & strm) const override;
07032
```

```
07033     bool operator==(const ManageNotebookSharesResult & other) const
07034     {
07035         return errors.isEqual(other.errors)
07036         ;
07037     }
07038
07039     bool operator!=(const ManageNotebookSharesResult & other) const
07040     {
07041         return !(*this == other);
07042     }
07043
07044     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
07045     Q_PROPERTY(Optional<QList<ManageNotebookSharesError» errors MEMBER errors)
07046 };
07047
07052 struct QEVERCLOUD_EXPORT SharedNoteTemplate: public Printable
07053 {
07054 private:
07055     Q_GADGET
07056 public:
07060     EverCloudLocalData localData;
07061
07065     Optional<Guid> noteGuid;
07073     Optional<MessageThreadID> recipientThreadId;
07080     Optional<QList<Contact>> recipientContacts;
07084     Optional<SharedNotePrivilegeLevel> privilege;
07085
07086     virtual void print(QTextStream & strm) const override;
07087
07088     bool operator==(const SharedNoteTemplate & other) const
07089     {
07090         return noteGuid.isEqual(other.noteGuid)
07091             && recipientThreadId.isEqual(other.recipientThreadId)
07092             && recipientContacts.isEqual(other.recipientContacts)
07093             && privilege.isEqual(other.privilege)
07094         ;
07095     }
07096
07097     bool operator!=(const SharedNoteTemplate & other) const
07098     {
07099         return !(*this == other);
07100     }
07101
07102     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
07103     Q_PROPERTY(Optional<Guid> noteGuid MEMBER noteGuid)
07104     Q_PROPERTY(Optional<MessageThreadID> recipientThreadId MEMBER recipientThreadId)
07105     Q_PROPERTY(Optional<QList<Contact» recipientContacts MEMBER recipientContacts)
07106     Q_PROPERTY(Optional<SharedNotePrivilegeLevel> privilege MEMBER privilege)
07107 };
07108
07113 struct QEVERCLOUD_EXPORT NotebookShareTemplate: public Printable
07114 {
07115 private:
07116     Q_GADGET
07117 public:
07121     EverCloudLocalData localData;
07122
07126     Optional<Guid> notebookGuid;
07134     Optional<MessageThreadID> recipientThreadId;
07141     Optional<QList<Contact>> recipientContacts;
07145     Optional<SharedNotebookPrivilegeLevel> privilege;
07146
07147     virtual void print(QTextStream & strm) const override;
07148
07149     bool operator==(const NotebookShareTemplate & other) const
07150     {
07151         return notebookGuid.isEqual(other.notebookGuid)
07152             && recipientThreadId.isEqual(other.recipientThreadId)
07153             && recipientContacts.isEqual(other.recipientContacts)
07154             && privilege.isEqual(other.privilege)
07155         ;
07156     }
07157
07158     bool operator!=(const NotebookShareTemplate & other) const
07159     {
07160         return !(*this == other);
07161     }
07162
07163     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
07164     Q_PROPERTY(Optional<Guid> notebookGuid MEMBER notebookGuid)
07165     Q_PROPERTY(Optional<MessageThreadID> recipientThreadId MEMBER recipientThreadId)
07166     Q_PROPERTY(Optional<QList<Contact» recipientContacts MEMBER recipientContacts)
07167     Q_PROPERTY(Optional<SharedNotebookPrivilegeLevel> privilege MEMBER privilege)
07168 };
07169
07174 struct QEVERCLOUD_EXPORT CreateOrUpdateNotebookSharesResult: public Printable
07175 {
```

```
07176 private:
07177     Q_GADGET
07178 public:
07182     EverCloudLocalData localData;
07183
07187     Optional<qint32> updateSequenceNum;
07194     Optional<QList<SharedNotebook>> matchingShares;
07195
07196     virtual void print(QTextStream & strm) const override;
07197
07198     bool operator==(const CreateOrUpdateNotebookSharesResult & other) const
07199     {
07200         return updateSequenceNum.isEqual(other.updateSequenceNum)
07201             && matchingShares.isEqual(other.matchingShares)
07202         ;
07203     }
07204
07205     bool operator!=(const CreateOrUpdateNotebookSharesResult & other) const
07206     {
07207         return !(*this == other);
07208     }
07209
07210     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
07211     Q_PROPERTY(Optional<qint32> updateSequenceNum MEMBER updateSequenceNum)
07212     Q_PROPERTY(Optional<QList<SharedNotebook» matchingShares MEMBER matchingShares)
07213 };
07214
07226 struct QEVERCLOUD_EXPORT ManageNoteSharesError: public Printable
07227 {
07228 private:
07229     Q_GADGET
07230 public:
07234     EverCloudLocalData localData;
07235
07240     Optional<IdentityID> identityID;
07245     Optional<UserID> userID;
07250     Optional<EDAMUserException> userException;
07258     Optional<EDAMNotFoundException> notFoundException;
07259
07260     virtual void print(QTextStream & strm) const override;
07261
07262     bool operator==(const ManageNoteSharesError & other) const
07263     {
07264         return identityID.isEqual(other.identityID)
07265             && userID.isEqual(other.userID)
07266             && userException.isEqual(other.userException)
07267             && notFoundException.isEqual(other.notFoundException)
07268         ;
07269     }
07270
07271     bool operator!=(const ManageNoteSharesError & other) const
07272     {
07273         return !(*this == other);
07274     }
07275
07276     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
07277     Q_PROPERTY(Optional<IdentityID> identityID MEMBER identityID)
07278     Q_PROPERTY(Optional<UserID> userID MEMBER userID)
07279     Q_PROPERTY(Optional<EDAMUserException> userException MEMBER userException)
07280     Q_PROPERTY(Optional<EDAMNotFoundException> notFoundException MEMBER notFoundException)
07281 };
07282
07287 struct QEVERCLOUD_EXPORT ManageNoteSharesResult: public Printable
07288 {
07289 private:
07290     Q_GADGET
07291 public:
07295     EverCloudLocalData localData;
07296
07302     Optional<QList<ManageNoteSharesError>> errors;
07303
07304     virtual void print(QTextStream & strm) const override;
07305
07306     bool operator==(const ManageNoteSharesResult & other) const
07307     {
07308         return errors.isEqual(other.errors)
07309         ;
07310     }
07311
07312     bool operator!=(const ManageNoteSharesResult & other) const
07313     {
07314         return !(*this == other);
07315     }
07316
07317     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
07318     Q_PROPERTY(Optional<QList<ManageNoteSharesError» errors MEMBER errors)
07319 };
```

```
07320
07321 } // namespace qevercloud
07322
07323 Q_DECLARE_METATYPE(qevercloud::EverCloudLocalData)
07324 Q_DECLARE_METATYPE(qevercloud::SyncState)
07325 Q_DECLARE_METATYPE(qevercloud::SyncChunkFilter)
07326 Q_DECLARE_METATYPE(qevercloud::NoteFilter)
07327 Q_DECLARE_METATYPE(qevercloud::NotesMetadataResultSpec)
07328 Q_DECLARE_METATYPE(qevercloud::NoteCollectionCounts)
07329 Q_DECLARE_METATYPE(qevercloud::NoteResultSpec)
07330 Q_DECLARE_METATYPE(qevercloud::NoteVersionId)
07331 Q_DECLARE_METATYPE(qevercloud::RelatedQuery)
07332 Q_DECLARE_METATYPE(qevercloud::RelatedResultSpec)
07333 Q_DECLARE_METATYPE(qevercloud::ShareRelationshipRestrictions)
07334 Q_DECLARE_METATYPE(qevercloud::MemberShareRelationship)
07335 Q_DECLARE_METATYPE(qevercloud::NoteShareRelationshipRestrictions)
07336 Q_DECLARE_METATYPE(qevercloud::NoteMemberShareRelationship)
07337 Q_DECLARE_METATYPE(qevercloud::NoteInvitationShareRelationship)
07338 Q_DECLARE_METATYPE(qevercloud::NoteShareRelationships)
07339 Q_DECLARE_METATYPE(qevercloud::ManageNoteSharesParameters)
07340 Q_DECLARE_METATYPE(qevercloud::Data)
07341 Q_DECLARE_METATYPE(qevercloud::UserAttributes)
07342 Q_DECLARE_METATYPE(qevercloud::BusinessUserAttributes)
07343 Q_DECLARE_METATYPE(qevercloud::Accounting)
07344 Q_DECLARE_METATYPE(qevercloud::BusinessUserInfo)
07345 Q_DECLARE_METATYPE(qevercloud::AccountLimits)
07346 Q_DECLARE_METATYPE(qevercloud::User)
07347 Q_DECLARE_METATYPE(qevercloud::Contact)
07348 Q_DECLARE_METATYPE(qevercloud::Identity)
07349 Q_DECLARE_METATYPE(qevercloud::Tag)
07350 Q_DECLARE_METATYPE(qevercloud::LazyMap)
07351 Q_DECLARE_METATYPE(qevercloud::ResourceAttributes)
07352 Q_DECLARE_METATYPE(qevercloud::Resource)
07353 Q_DECLARE_METATYPE(qevercloud::NoteAttributes)
07354 Q_DECLARE_METATYPE(qevercloud::SharedNote)
07355 Q_DECLARE_METATYPE(qevercloud::NoteRestrictions)
07356 Q_DECLARE_METATYPE(qevercloud::NoteLimits)
07357 Q_DECLARE_METATYPE(qevercloud::Note)
07358 Q_DECLARE_METATYPE(qevercloud::Publishing)
07359 Q_DECLARE_METATYPE(qevercloud::BusinessNotebook)
07360 Q_DECLARE_METATYPE(qevercloud::SavedSearchScope)
07361 Q_DECLARE_METATYPE(qevercloud::SavedSearch)
07362 Q_DECLARE_METATYPE(qevercloud::SharedNotebookRecipientSettings)
07363 Q_DECLARE_METATYPE(qevercloud::NotebookRecipientSettings)
07364 Q_DECLARE_METATYPE(qevercloud::SharedNotebook)
07365 Q_DECLARE_METATYPE(qevercloud::CanMoveToContainerRestrictions)
07366 Q_DECLARE_METATYPE(qevercloud::NotebookRestrictions)
07367 Q_DECLARE_METATYPE(qevercloud::Notebook)
07368 Q_DECLARE_METATYPE(qevercloud::LinkedNotebook)
07369 Q_DECLARE_METATYPE(qevercloud::NotebookDescriptor)
07370 Q_DECLARE_METATYPE(qevercloud::UserProfile)
07371 Q_DECLARE_METATYPE(qevercloud::RelatedContentImage)
07372 Q_DECLARE_METATYPE(qevercloud::RelatedContent)
07373 Q_DECLARE_METATYPE(qevercloud::BusinessInvitation)
07374 Q_DECLARE_METATYPE(qevercloud::UserIdentity)
07375 Q_DECLARE_METATYPE(qevercloud::PublicUserInfo)
07376 Q_DECLARE_METATYPE(qevercloud::UserUrls)
07377 Q_DECLARE_METATYPE(qevercloud::AuthenticationResult)
07378 Q_DECLARE_METATYPE(qevercloud::BootstrapSettings)
07379 Q_DECLARE_METATYPE(qevercloud::BootstrapProfile)
07380 Q_DECLARE_METATYPE(qevercloud::BootstrapInfo)
07381 Q_DECLARE_METATYPE(qevercloud::EDAMUserException)
07382 Q_DECLARE_METATYPE(qevercloud::EDAMSystemException)
07383 Q_DECLARE_METATYPE(qevercloud::EDAMNotFoundException)
07384 Q_DECLARE_METATYPE(qevercloud::EDAMInvalidContactsException)
07385 Q_DECLARE_METATYPE(qevercloud::SyncChunk)
07386 Q_DECLARE_METATYPE(qevercloud::NoteList)
07387 Q_DECLARE_METATYPE(qevercloud::NoteMetadata)
07388 Q_DECLARE_METATYPE(qevercloud::NotesMetadataList)
07389 Q_DECLARE_METATYPE(qevercloud::NoteEmailParameters)
07390 Q_DECLARE_METATYPE(qevercloud::RelatedResult)
07391 Q_DECLARE_METATYPE(qevercloud::UpdateNoteIfUsnMatchesResult)
07392 Q_DECLARE_METATYPE(qevercloud::InvitationShareRelationship)
07393 Q_DECLARE_METATYPE(qevercloud::ShareRelationships)
07394 Q_DECLARE_METATYPE(qevercloud::ManageNotebookSharesParameters)
07395 Q_DECLARE_METATYPE(qevercloud::ManageNotebookSharesError)
07396 Q_DECLARE_METATYPE(qevercloud::ManageNotebookSharesResult)
07397 Q_DECLARE_METATYPE(qevercloud::SharedNoteTemplate)
07398 Q_DECLARE_METATYPE(qevercloud::NotebookShareTemplate)
07399 Q_DECLARE_METATYPE(qevercloud::CreateOrUpdateNotebookSharesResult)
07400 Q_DECLARE_METATYPE(qevercloud::ManageNoteSharesError)
07401 Q_DECLARE_METATYPE(qevercloud::ManageNoteSharesResult)
07402
07403
07404 #endif // QEVERCLOUD_GENERATED_TYPES_H
```

8.23 Globals.h File Reference

```
#include "Export.h"
#include <QNetworkProxy>
```

Namespaces

- namespace qevercloud

Functions

- QEVERCLOUD_EXPORT QNetworkProxy qevercloud::evernoteNetworkProxy ()
- QEVERCLOUD_EXPORT void qevercloud::setEvernoteNetworkProxy (QNetworkProxy proxy)
- QEVERCLOUD_EXPORT void qevercloud::resetEvernoteNetworkProxy ()
- QEVERCLOUD_EXPORT int qevercloud::libraryVersion ()
- QEVERCLOUD_EXPORT void qevercloud::initializeQEverCloud ()

8.24 Globals.h

Go to the documentation of this file.

```
00001
00009 #ifndef QEVERCLOUD_GLOBALS_H
00010 #define QEVERCLOUD_GLOBALS_H
00011
00012 #include "Export.h"
00013
00014 #include <QNetworkProxy>
00015
00019 namespace qevercloud {
00020
00022
00039 QEVERCLOUD_EXPORT QNetworkProxy evernoteNetworkProxy();
00040
00056 QEVERCLOUD_EXPORT void setEvernoteNetworkProxy(QNetworkProxy proxy);
00057
00066 QEVERCLOUD_EXPORT void resetEvernoteNetworkProxy();
00067
00069
00073 Q_DECL_DEPRECATED_X("libraryVersion is deprecated, use qevercloudVersionMajor/Minor/Patch instead")
00074 QEVERCLOUD_EXPORT int libraryVersion();
00075
00077
00082 QEVERCLOUD_EXPORT void initializeQEverCloud();
00083
00084 } // namespace qevercloud
00085
00086 #endif // QEVERCLOUD_GLOBALS_H
```

8.25 Helpers.h File Reference

```
#include <QtGlobal>
#include <QObject>
#include "VersionInfo.h"
```

Classes

- class qevercloud::QAssociativeContainerReferenceWrapper< Container >
- struct qevercloud::QAssociativeContainerReferenceWrapper< Container >::iterator
- class qevercloud::QAssociativeContainerConstReferenceWrapper< Container >
- struct qevercloud::QAssociativeContainerConstReferenceWrapper< Container >::iterator

Namespaces

- namespace qevercloud

Functions

- template<class Container >
  QAssociativeContainerReferenceWrapper< Container > qevercloud::toRange (Container &container)
- template<class Container >
  QAssociativeContainerConstReferenceWrapper< Container > qevercloud::toRange (const Container &container)

8.26 Helpers.h

Go to the documentation of this file.

```
00001
00008 #ifndef QEVERCLOUD_HELPERS_H
00009 #define QEVERCLOUD_HELPERS_H
00010
00011 #include <QtGlobal>
00012 #include <QObject>
00013
00014 #include "VersionInfo.h"
00015
00016 namespace qevercloud {
00017
00019
00020 #if QT_VERSION < QT_VERSION_CHECK(5, 7, 0)
00021
00022 // this adds const to non-const objects (like std::as_const)
00023 template <typename T>
00024 Q_DECL_CONSTEXPR
00025 typename std::add_const<T>::type & qAsConst(T & t) Q_DECL_NOTHROW
00026 {
00027     return t;
00028 }
00029
00030 // prevent rvalue arguments:
00031 template <typename T>
00032 void qAsConst(const T &&) Q_DECL_EQ_DELETE;
00033
00034 #endif // QT_VERSION_CHECK
00035
00037
00038 template <typename Container>
00039 class QAssociativeContainerReferenceWrapper
00040 {
00041 public:
00042     struct iterator
00043     {
00044         typename Container::iterator m_iterator;
00045         iterator(const typename Container::iterator it) :
00046             m_iterator(it)
00047         {}
00048
00049         typename Container::iterator operator*()
00050         {
00051             return m_iterator;
00052         }
00053
00054         iterator & operator++()
00055         {
```

```
00056             ++m_iterator;
00057             return *this;
00058         }
00059
00060         bool operator!=(const iterator & other) const
00061         {
00062             return m_iterator != other.m_iterator;
00063         }
00064     };
00065
00066 public:
00067     QAssociativeContainerReferenceWrapper(Container & container)
00068         : m_container(container)
00069     {}
00070
00071     iterator begin() {
00072         return m_container.begin();
00073     }
00074
00075     iterator end() {
00076         return m_container.end();
00077     }
00078
00079 private:
00080     Container & m_container;
00081 };
00082
00084
00085 template <typename Container>
00086 class QAssociativeContainerConstReferenceWrapper
00087 {
00088 public:
00089     struct iterator
00090     {
00091         typename Container::const_iterator m_iterator;
00092         iterator(const typename Container::const_iterator it) :
00093             m_iterator(it)
00094         {}
00095
00096         typename Container::const_iterator operator*()
00097         {
00098             return m_iterator;
00099         }
00100
00101         iterator & operator++()
00102         {
00103             ++m_iterator;
00104             return *this;
00105         }
00106
00107         bool operator!=(const iterator & other) const
00108         {
00109             return m_iterator != other.m_iterator;
00110         }
00111     };
00112
00113 public:
00114     QAssociativeContainerConstReferenceWrapper(const Container & container)
00115         : m_container(container)
00116     {}
00117
00118     iterator begin() const {
00119         return m_container.begin();
00120     }
00121
00122     iterator end() const {
00123         return m_container.end();
00124     }
00125
00126 private:
00127     const Container & m_container;
00128 };
00129
00131
00132 template <class Container>
00133 QAssociativeContainerReferenceWrapper<Container> toRange(Container & container)
00134 {
00135     return QAssociativeContainerReferenceWrapper<Container>(container);
00136 }
00137
00139
00140 template <class Container>
00141 QAssociativeContainerConstReferenceWrapper<Container> toRange(
00142     const Container & container)
00143 {
00144     return QAssociativeContainerConstReferenceWrapper<Container>(container);
00145 }
```

```
00146
00147 } // namespace qevercloud
00148
00149 #endif // QEVERCLOUD_HELPERS_H
```

8.27 InkNoteImageDownloader.h File Reference

```
#include "AsyncResult.h"
#include "Export.h"
#include "generated/Types.h"
#include <QByteArray>
#include <QString>
#include <QNetworkAccessManager>
```

Classes

- class qevercloud::InkNoteImageDownloader

  the InkNoteImageDownloader class is for downloading the images of ink notes which can be created with the official Evernote client on Windows (only with it, at least at the time of this writing).

Namespaces

- namespace qevercloud

8.28 InkNoteImageDownloader.h

Go to the documentation of this file.

```
00001
00008 #ifndef QEVERCLOD_INK_NOTE_IMAGE_DOWNLOADER_H
00009 #define QEVERCLOD_INK_NOTE_IMAGE_DOWNLOADER_H
00010
00011 #include "AsyncResult.h"
00012 #include "Export.h"
00013
00014 #include "generated/Types.h"
00015
00016 #include <QByteArray>
00017 #include <QString>
00018 #include <QNetworkAccessManager>
00019
00020 namespace qevercloud {
00021
00023 class InkNoteImageDownloaderPrivate;
00042 class QEVERCLOUD_EXPORT InkNoteImageDownloader
00043 {
00044 public:
00051     InkNoteImageDownloader();
00052
00068     InkNoteImageDownloader(
00069         QString host, QString shardId, QString authenticationToken, int width,
00070         int height);
00071
00072     virtual ~InkNoteImageDownloader();
00073
00078     InkNoteImageDownloader & setHost(QString host);
00079
00085     InkNoteImageDownloader & setShardId(QString shardId);
00086
00092     InkNoteImageDownloader & setAuthenticationToken(
00093         QString authenticationToken);
00094
00099     InkNoteImageDownloader & setWidth(int width);
```

```
00100
00105     InkNoteImageDownloader & setHeight(int height);
00106
00132     QByteArray download(
00133         Guid guid, const bool isPublic = false,
00134         const qint64 timeoutMsec = 30000);
00135
00136 private:
00137     InkNoteImageDownloaderPrivate * const d_ptr;
00138     Q_DECLARE_PRIVATE(InkNoteImageDownloader)
00139 };
00140
00141 } // namespace qevercloud
00142
00143 #endif // QEVERCLOD_INK_NOTE_IMAGE_DOWNLOADER_H
```

8.29 Log.h File Reference

```
#include "Export.h"
#include "Helpers.h"
#include <QDateTime>
#include <QDebug>
#include <QObject>
#include <QTextStream>
#include <memory>
```

Classes

- class qevercloud::ILogger

Namespaces

- namespace qevercloud

Macros

- #define __QEVERCLOUD_LOG_BASE(component, level, message)
- #define QEC_TRACE(component, message) __QEVERCLOUD_LOG_BASE(component, LogLevel::Trace, message) \
- #define QEC_DEBUG(component, message) __QEVERCLOUD_LOG_BASE(component, LogLevel::↩ Debug, message) \
- #define QEC_INFO(component, message) __QEVERCLOUD_LOG_BASE(component, LogLevel::Info, message) \
- #define QEC_WARNING(component, message) __QEVERCLOUD_LOG_BASE(component, LogLevel::↩ Warn, message) \
- #define QEC_ERROR(component, message) __QEVERCLOUD_LOG_BASE(component, LogLevel::Error, message) \

Typedefs

- using qevercloud::ILoggerPtr = std::shared_ptr< ILogger >

Enumerations

- enum class qevercloud::LogLevel {
  qevercloud::Trace = 0 , qevercloud::Debug , qevercloud::Info , qevercloud::Warn ,
  qevercloud::Error }

Functions

- QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const LogLevel level)
- QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const LogLevel level)
- QEVERCLOUD_EXPORT ILoggerPtr qevercloud::logger ()
- QEVERCLOUD_EXPORT void qevercloud::setLogger (ILoggerPtr logger)
- QEVERCLOUD_EXPORT ILoggerPtr qevercloud::nullLogger ()
- QEVERCLOUD_EXPORT ILoggerPtr qevercloud::newStdErrLogger (LogLevel level=LogLevel::Warn)

8.29.1 Macro Definition Documentation

8.29.1.1 __QEVERCLOUD_LOG_BASE

```
#define __QEVERCLOUD_LOG_BASE(
                component,
                level,
                message )
```

Value:

```
    {                                                                       \
        auto __qevercloudLogger = ::qevercloud::logger();                   \
        if (__qevercloudLogger->shouldLog(level, component))               \
        {                                                                   \
            QString msg;                                                    \
            QDebug dbg(&msg);                                               \
            dbg.nospace();                                                  \
            dbg.noquote();                                                  \
            dbg « message;                                                 \
            __qevercloudLogger->log(                                        \
                level,                                                      \
                component,                                                  \
                __FILE__,                                                   \
                __LINE__,                                                   \
                QDateTime::currentMSecsSinceEpoch(),                        \
                msg);                                                       \
        }                                                                   \
    }                                                                       \
```

8.29.1.2 QEC_DEBUG

```
#define QEC_DEBUG(
                component,
                message )  __QEVERCLOUD_LOG_BASE(component, LogLevel::Debug, message) \
```

8.29.1.3 QEC_ERROR

```
#define QEC_ERROR(
                component,
                message )  __QEVERCLOUD_LOG_BASE(component, LogLevel::Error, message) \
```

8.29.1.4 QEC_INFO

```
#define QEC_INFO(
                component,
                message )  __QEVERCLOUD_LOG_BASE(component, LogLevel::Info, message) \
```

8.29.1.5 QEC_TRACE

```
#define QEC_TRACE(
                component,
                message )  __QEVERCLOUD_LOG_BASE(component, LogLevel::Trace, message) \
```

8.29.1.6 QEC_WARNING

```
#define QEC_WARNING(
                component,
                message )  __QEVERCLOUD_LOG_BASE(component, LogLevel::Warn, message) \
```

8.30 Log.h

Go to the documentation of this file.

```
00001
00008 #ifndef QEVERCLOUD_LOG_H
00009 #define QEVERCLOUD_LOG_H
00010
00011 #include "Export.h"
00012 #include "Helpers.h"
00013
00014 #include <QDateTime>
00015 #include <QDebug>
00016 #include <QObject>
00017 #include <QTextStream>
00018
00019 #include <memory>
00020
00021 namespace qevercloud {
00022
00024
00025 enum class LogLevel
00026 {
00027     Trace = 0,
00028     Debug,
00029     Info,
00030     Warn,
00031     Error
00032 };
00033
00034 QEVERCLOUD_EXPORT QTextStream & operator<<(
00035     QTextStream & out, const LogLevel level);
00036
00037 QEVERCLOUD_EXPORT QDebug & operator<<(
00038     QDebug & out, const LogLevel level);
00039
00041
00042 class QEVERCLOUD_EXPORT ILogger
00043 {
00044 public:
00045     virtual bool shouldLog(
00046         const LogLevel level, const char * component) const = 0;
00047
00048     virtual void log(
00049         const LogLevel level, const char * component, const char * fileName,
00050         const quint32 lineNumber, const qint64 timestamp,
00051         const QString & message) = 0;
00052
00053     virtual void setLevel(const LogLevel level) = 0;
00054
```

```
00055     virtual LogLevel level() const = 0;
00056 };
00057
00058 using ILoggerPtr = std::shared_ptr<ILogger>;
00059
00061
00062 QEVERCLOUD_EXPORT ILoggerPtr logger();
00063
00064 QEVERCLOUD_EXPORT void setLogger(ILoggerPtr logger);
00065
00066 QEVERCLOUD_EXPORT ILoggerPtr nullLogger();
00067
00068 QEVERCLOUD_EXPORT ILoggerPtr newStdErrLogger(LogLevel level = LogLevel::Warn);
00069
00071
00072 #define __QEVERCLOUD_LOG_BASE(component, level, message)                    \
00073     {                                                                       \
00074         auto __qevercloudLogger = ::qevercloud::logger();                   \
00075         if (__qevercloudLogger->shouldLog(level, component))               \
00076         {                                                                   \
00077             QString msg;                                                    \
00078             QDebug dbg(&msg);                                               \
00079             dbg.nospace();                                                  \
00080             dbg.noquote();                                                  \
00081             dbg << message;                                                \
00082             __qevercloudLogger->log(                                        \
00083                 level,                                                      \
00084                 component,                                                  \
00085                 __FILE__,                                                   \
00086                 __LINE__,                                                   \
00087                 QDateTime::currentMSecsSinceEpoch(),                        \
00088                 msg);                                                       \
00089         }                                                                   \
00090     }                                                                       \
00091 // __QEVERCLOUD_LOG_BASE
00092
00093 #define QEC_TRACE(component, message)                                       \
00094     __QEVERCLOUD_LOG_BASE(component, LogLevel::Trace, message)              \
00095 // QEC_TRACE
00096
00097 #define QEC_DEBUG(component, message)                                       \
00098     __QEVERCLOUD_LOG_BASE(component, LogLevel::Debug, message)              \
00099 // QEC_DEBUG
00100
00101 #define QEC_INFO(component, message)                                        \
00102     __QEVERCLOUD_LOG_BASE(component, LogLevel::Info, message)               \
00103 // QEC_INFO
00104
00105 #define QEC_WARNING(component, message)                                     \
00106     __QEVERCLOUD_LOG_BASE(component, LogLevel::Warn, message)               \
00107 // QEC_WARNING
00108
00109 #define QEC_ERROR(component, message)                                       \
00110     __QEVERCLOUD_LOG_BASE(component, LogLevel::Error, message)              \
00111 // QEC_ERROR
00112
00113 } // namespace qevercloud
00114
00115 #endif // QEVERCLOUD_LOG_H
```

8.31 OAuth.h File Reference

```
#include "Export.h"
#include "Helpers.h"
#include "Printable.h"
#include "generated/Types.h"
#include <QDialog>
#include <QList>
#include <QNetworkCookie>
#include <QString>
```

Classes

- class qevercloud::EvernoteOAuthWebView

The class is tailored specifically for OAuth authorization with Evernote.

- struct qevercloud::EvernoteOAuthWebView::OAuthResult
- class qevercloud::EvernoteOAuthDialog

  Authorizes your app with the Evernote service by means of OAuth authentication.

Namespaces

- namespace qevercloud

Functions

- void qevercloud::setNonceGenerator (quint64(*nonceGenerator)())

  Sets the function to use for nonce generation for OAuth authentication.

8.32 OAuth.h

Go to the documentation of this file.

```
00001
00009 #ifndef QEVERCLOUD_OAUTH_H
00010 #define QEVERCLOUD_OAUTH_H
00011
00012 #include "Export.h"
00013 #include "Helpers.h"
00014 #include "Printable.h"
00015
00016 #include "generated/Types.h"
00017
00018 #include <QDialog>
00019 #include <QList>
00020 #include <QNetworkCookie>
00021 #include <QString>
00022
00023 namespace qevercloud {
00024
00039 void setNonceGenerator(quint64 (*nonceGenerator)());
00040
00042 class EvernoteOAuthWebViewPrivate;
00057 class QEVERCLOUD_EXPORT EvernoteOAuthWebView: public QWidget
00058 {
00059     Q_OBJECT
00060 public:
00061     EvernoteOAuthWebView(QWidget * parent = Q_NULLPTR);
00062
00083     void authenticate(
00084         QString host, QString consumerKey, QString consumerSecret,
00085         const qint64 timeoutMsec = 30000);
00086
00091     bool isSucceeded() const;
00092
00094     QString oauthError() const;
00095
00097     struct QEVERCLOUD_EXPORT OAuthResult: public Printable
00098     {
00099         QString noteStoreUrl;
00101         Timestamp expires;
00102         QString shardId;
00103         UserID userId;
00104         QString webApiUrlPrefix;
00105         QString authenticationToken;
00106
00118         QList<QNetworkCookie> cookies;
00119
00120         virtual void print(QTextStream & strm) const override;
00121     };
00122
00124     OAuthResult oauthResult() const;
00125
00127     void setSizeHint(QSize sizeHint);
00128
00129     QSize sizeHint() const override;
00130
```

```
00131 Q_SIGNALS:
00133     void authenticationFinished(bool success);
00134
00139     void authenticationSuceeded();
00140
00145     void authenticationFailed();
00146
00147 private:
00148     EvernoteOAuthWebViewPrivate * const d_ptr;
00149     Q_DECLARE_PRIVATE(EvernoteOAuthWebView)
00150 };
00151
00153 class EvernoteOAuthDialogPrivate;
00186 class QEVERCLOUD_EXPORT EvernoteOAuthDialog: public QDialog
00187 {
00188     Q_OBJECT
00189 public:
00190     using OAuthResult = EvernoteOAuthWebView::OAuthResult;
00191
00205     EvernoteOAuthDialog(
00206         QString consumerKey, QString consumerSecret,
00207         QString host = QStringLiteral("www.evernote.com"),
00208         QWidget * parent = Q_NULLPTR);
00209
00210     virtual ~EvernoteOAuthDialog() override;
00211
00218     void setWebViewSizeHint(QSize sizeHint);
00219
00223     bool isSucceeded() const;
00224
00229     QString oauthError() const;
00230
00234     OAuthResult oauthResult() const;
00235
00240     virtual int exec() override;
00241
00244     virtual void open() override;
00245
00246 private:
00247     EvernoteOAuthDialogPrivate * const d_ptr;
00248     Q_DECLARE_PRIVATE(EvernoteOAuthDialog)
00249 };
00250
00251 } // namespace qevercloud
00252
00253 #endif // QEVERCLOUD_OAUTH_H
```

8.33 Optional.h File Reference

```
#include "EverCloudException.h"
#include <algorithm>
```

Classes

- class qevercloud::Optional< T >

Namespaces

- namespace qevercloud

8.34 Optional.h

Go to the documentation of this file.

```
00001
00009 #ifndef QEVERCLOUD_OPTIONAL_H
00010 #define QEVERCLOUD_OPTIONAL_H
00011
00012 #include "EverCloudException.h"
00013
00014 #include <algorithm>
00015
00016 namespace qevercloud {
00017
00042 template<typename T>
00043 class Optional
00044 {
00045 public:
00049     Optional() :
00050         m_isSet(false),
00051         m_value(T())
00052     {}
00053
00057     Optional(const Optional & o) :
00058         m_isSet(o.m_isSet),
00059         m_value(o.m_value)
00060     {}
00061
00066     template<typename X>
00067     Optional(const Optional<X> & o) :
00068         m_isSet(o.m_isSet),
00069         m_value(o.m_value)
00070     {}
00071
00075     Optional(const T & value) :
00076         m_isSet(true),
00077         m_value(value)
00078     {}
00079
00083     template<typename X>
00084     Optional(const X & value) :
00085         m_isSet(true),
00086         m_value(value)
00087     {}
00088
00092     Optional & operator=(const Optional & o)
00093     {
00094         m_value = o.m_value;
00095         m_isSet = o.m_isSet;
00096         return *this;
00097     }
00098
00102     template<typename X>
00103     Optional & operator=(const Optional<X> & o)
00104     {
00105         m_value = o.m_value;
00106         m_isSet = o.m_isSet;
00107         return *this;
00108     }
00109
00113     Optional & operator=(const T & value)
00114     {
00115         m_value = value;
00116         m_isSet = true;
00117         return *this;
00118     }
00119
00123     template<typename X>
00124     Optional & operator=(const X & value)
00125     {
00126         m_value = value;
00127         m_isSet = true;
00128         return *this;
00129     }
00130
00136     operator const T&() const
00137     {
00138         if (!m_isSet) {
00139             throw EverCloudException(
00140                 "qevercloud::Optional: nonexistent value access");
00141         }
00142
00143         return m_value;
00144     }
00145
00151     operator T&()
```

```
00152     {
00153         if (!m_isSet) {
00154             throw EverCloudException(
00155                 "qevercloud::Optional: nonexistent value access");
00156         }
00157 
00158         return m_value;
00159     }
00160 
00167     const T & ref() const
00168     {
00169         if (!m_isSet) {
00170             throw EverCloudException(
00171                 "qevercloud::Optional: nonexistent value access");
00172         }
00173 
00174         return m_value;
00175     }
00176 
00204     T & ref()
00205     {
00206         if (!m_isSet) {
00207             throw EverCloudException(
00208                 "qevercloud::Optional: nonexistent value access");
00209         }
00210 
00211         return m_value;
00212     }
00213 
00220     bool isSet() const
00221     {
00222         return m_isSet;
00223     }
00224 
00236     void clear()
00237     {
00238         m_isSet = false;
00239         m_value = T();
00240     }
00241 
00265     Optional & init()
00266     {
00267         m_isSet = true;
00268         m_value = T();
00269         return *this;
00270     }
00271 
00306     T * operator->()
00307     {
00308         if (!m_isSet) {
00309             throw EverCloudException(
00310                 "qevercloud::Optional: nonexistent value access");
00311         }
00312 
00313         return &m_value;
00314     }
00315 
00319     const T * operator->() const
00320     {
00321         if (!m_isSet) {
00322             throw EverCloudException(
00323                 "qevercloud::Optional: nonexistent value access");
00324         }
00325 
00326         return &m_value;
00327     }
00328 
00336     T value(T defaultValue = T()) const
00337     {
00338         return m_isSet ? m_value : defaultValue;
00339     }
00340 
00345     bool isEqual(const Optional<T> & other) const
00346     {
00347         if (m_isSet != other.m_isSet) {
00348             return false;
00349         }
00350 
00351         return !m_isSet || (m_value == other.m_value);
00352     }
00353 
00354     bool operator==(const Optional<T> & other) const
00355     {
00356         return isEqual(other);
00357     }
00358 
00359     bool operator!=(const Optional<T> & other) const
```

```
00360     {
00361         return !operator==(other);
00362     }
00363 
00364     bool operator==(const T & other) const
00365     {
00366         if (!m_isSet) {
00367             return false;
00368         }
00369 
00370         return m_value == other;
00371     }
00372 
00373     bool operator!=(const T & other) const
00374     {
00375         return !operator==(other);
00376     }
00377 
00378     template<typename X> friend class Optional;
00379 
00380     friend void swap(Optional & first, Optional & second)
00381     {
00382         using std::swap;
00383         swap(first.m_isSet, second.m_isSet);
00384         swap(first.m_value, second.m_value);
00385     }
00386 
00387     Optional(Optional && other)
00388     {
00389         swap(*this, other);
00390     }
00391 
00392     Optional & operator=(Optional && other)
00393     {
00394         swap(*this, other);
00395         return *this;
00396     }
00397 
00398     Optional(T && other)
00399     {
00400         using std::swap;
00401         m_isSet = true;
00402         swap(m_value, other);
00403     }
00404 
00405     Optional & operator=(T && other)
00406     {
00407         using std::swap;
00408         m_isSet = true;
00409         swap(m_value, other);
00410         return *this;
00411     }
00412 
00413 private:
00414     bool m_isSet;
00415     T m_value;
00416 };
00417 
00418 } // namespace qevercloud
00419 
00420 #endif // QEVERCLOUD_OPTIONAL_H
```

8.35 Printable.h File Reference

```
#include "Export.h"
#include <QTextStream>
#include <QDebug>
```

Classes

- class qevercloud::Printable

Namespaces

- namespace qevercloud

8.36 Printable.h

Go to the documentation of this file.

```
00001
00008 #ifndef QEVERCLOUD_PRINTABLE_H
00009 #define QEVERCLOUD_PRINTABLE_H
00010
00011 #include "Export.h"
00012
00013 #include <QTextStream>
00014 #include <QDebug>
00015
00016 namespace qevercloud {
00017
00018 class QEVERCLOUD_EXPORT Printable
00019 {
00020 public:
00021     Printable() = default;
00022     virtual ~Printable() = default;
00023
00024     virtual void print(QTextStream & strm) const = 0;
00025
00026     virtual QString toString() const;
00027
00028     friend QEVERCLOUD_EXPORT QTextStream & operator <<(
00029         QTextStream & strm, const Printable & printable);
00030
00031     friend QEVERCLOUD_EXPORT QDebug & operator <<(
00032         QDebug & dbg, const Printable & printable);
00033 };
00034
00035 } // namespace qevercloud
00036
00037 #endif // QEVERCLOUD_PRINTABLE_H
```

8.37 QEverCloud.h File Reference

```
#include "AsyncResult.h"
#include "DurableService.h"
#include "EventLoopFinisher.h"
#include "EverCloudException.h"
#include "Exceptions.h"
#include "Export.h"
#include "Globals.h"
#include "Helpers.h"
#include "InkNoteImageDownloader.h"
#include "Log.h"
#include "Optional.h"
#include "Printable.h"
#include "RequestContext.h"
#include "Thumbnail.h"
#include "VersionInfo.h"
#include "generated/EDAMErrorCode.h"
#include "generated/Constants.h"
#include "generated/Services.h"
#include "generated/Types.h"
```

8.38 QEverCloud.h

Go to the documentation of this file.

```
00001
00009 #ifndef QEVERCLOUD_INFTHEADER_H
00010 #define QEVERCLOUD_INFTHEADER_H
00011
00012 #include "AsyncResult.h"
00013 #include "DurableService.h"
00014 #include "EventLoopFinisher.h"
00015 #include "EverCloudException.h"
00016 #include "Exceptions.h"
00017 #include "Export.h"
00018 #include "Globals.h"
00019 #include "Helpers.h"
00020 #include "InkNoteImageDownloader.h"
00021 #include "Log.h"
00022 #include "Optional.h"
00023 #include "Printable.h"
00024 #include "RequestContext.h"
00025 #include "Thumbnail.h"
00026 #include "VersionInfo.h"
00027 #include "generated/EDAMErrorCode.h"
00028 #include "generated/Constants.h"
00029 #include "generated/Services.h"
00030 #include "generated/Types.h"
00031
00032 #endif // QEVERCLOUD_INFTHEADER_H
```

8.39 QEverCloudOAuth.h File Reference

```
#include "OAuth.h"
#include "QEverCloud.h"
```

8.40 QEverCloudOAuth.h

Go to the documentation of this file.

```
00001
00009 #ifndef QEVERCLOUDOAUTH_INFTHEADER_H
00010 #define QEVERCLOUDOAUTH_INFTHEADER_H
00011
00012 #include "OAuth.h"
00013 #include "QEverCloud.h"
00014
00015 #endif // QEVERCLOUDOAUTH_INFTHEADER_H
```

8.41 RequestContext.h File Reference

```
#include "Export.h"
#include <QDebug>
#include <QList>
#include <QNetworkCookie>
#include <QTextStream>
#include <QUuid>
#include <memory>
```

Classes

- class qevercloud::IRequestContext

Namespaces

- namespace qevercloud

Typedefs

- using qevercloud::IRequestContextPtr = std::shared_ptr< IRequestContext >

Functions

- QEVERCLOUD_EXPORT IRequestContextPtr qevercloud::newRequestContext (QString authentication↩ Token={}, qint64 requestTimeout=DEFAULT_REQUEST_TIMEOUT_MSEC, bool increaseRequestTimeout↩ Exponentially=DEFAULT_REQUEST_TIMEOUT_EXPONENTIAL_INCREASE, qint64 maxRequest↩ Timeout=DEFAULT_MAX_REQUEST_TIMEOUT_MSEC, quint32 maxRequestRetryCount=DEFAULT_↩ MAX_REQUEST_RETRY_COUNT, QList< QNetworkCookie > cookies={})

8.42 RequestContext.h

Go to the documentation of this file.

```
00001
00008 #ifndef QEVERCLOUD_REQUEST_CONTEXT_H
00009 #define QEVERCLOUD_REQUEST_CONTEXT_H
00010
00011 #include "Export.h"
00012
00013 #include <QDebug>
00014 #include <QList>
00015 #include <QNetworkCookie>
00016 #include <QTextStream>
00017 #include <QUuid>
00018
00019 #include <memory>
00020
00021 namespace qevercloud {
00022
00024
00025 static constexpr quint64 DEFAULT_REQUEST_TIMEOUT_MSEC = 10000ull;
00026
00027 static constexpr bool DEFAULT_REQUEST_TIMEOUT_EXPONENTIAL_INCREASE = true;
00028
00029 static constexpr quint64 DEFAULT_MAX_REQUEST_TIMEOUT_MSEC = 600000ull;
00030
00031 static constexpr quint32 DEFAULT_MAX_REQUEST_RETRY_COUNT = 10;
00032
00034
00039 class QEVERCLOUD_EXPORT IRequestContext
00040 {
00041 public:
00043     virtual QUuid requestId() const = 0;
00044
00046     virtual QString authenticationToken() const = 0;
00047
00049     virtual qint64 requestTimeout() const = 0;
00050
00052     virtual bool increaseRequestTimeoutExponentially() const = 0;
00053
00058     virtual qint64 maxRequestTimeout() const = 0;
00059
00061     virtual quint32 maxRequestRetryCount() const = 0;
00062
00064     virtual QList<QNetworkCookie> cookies() const = 0;
00065
00070     virtual IRequestContext * clone() const = 0;
00071
00072     virtual ~IRequestContext() = default;
00073
00074     friend QEVERCLOUD_EXPORT QTextStream & operator<<(
00075         QTextStream & strm, const IRequestContext & ctx);
00076
00077     friend QEVERCLOUD_EXPORT QDebug & operator<<(
00078         QDebug & dbg, const IRequestContext & ctx);
```

```
00079 };
00080
00081 using IRequestContextPtr = std::shared_ptr<IRequestContext>;
00082
00084
00085 QEVERCLOUD_EXPORT IRequestContextPtr newRequestContext(
00086     QString authenticationToken = {},
00087     qint64 requestTimeout = DEFAULT_REQUEST_TIMEOUT_MSEC,
00088     bool increaseRequestTimeoutExponentially = DEFAULT_REQUEST_TIMEOUT_EXPONENTIAL_INCREASE,
00089     qint64 maxRequestTimeout = DEFAULT_MAX_REQUEST_TIMEOUT_MSEC,
00090     quint32 maxRequestRetryCount = DEFAULT_MAX_REQUEST_RETRY_COUNT,
00091     QList<QNetworkCookie> cookies = {});
00092
00093 } // namespace qevercloud
00094
00095 #endif // QEVERCLOUD_REQUEST_CONTEXT_H
```

8.43 Thumbnail.h File Reference

```
#include "AsyncResult.h"
#include "Export.h"
#include "generated/Types.h"
#include <QByteArray>
#include <QNetworkAccessManager>
#include <QString>
#include <utility>
```

Classes

- class qevercloud::Thumbnail

  The class is for downloading thumbnails for notes and resources from Evernote servers.

Namespaces

- namespace qevercloud

8.44 Thumbnail.h

Go to the documentation of this file.

```
00001
00009 #ifndef QEVERCLOUD_THUMBNAIL_H
00010 #define QEVERCLOUD_THUMBNAIL_H
00011
00012 #include "AsyncResult.h"
00013 #include "Export.h"
00014
00015 #include "generated/Types.h"
00016
00017 #include <QByteArray>
00018 #include <QNetworkAccessManager>
00019 #include <QString>
00020
00021 #include <utility>
00022
00023 namespace qevercloud {
00024
00026 class ThumbnailPrivate;
00047 class QEVERCLOUD_EXPORT Thumbnail
00048 {
00049 public:
00055     enum class ImageType
00056     {
```

```
00057         PNG,
00058         JPEG,
00059         GIF,
00060         BMP
00061     };
00062
00063     friend QEVERCLOUD_EXPORT QTextStream & operator<<(
00064         QTextStream & strm, const ImageType imageType);
00065
00066     friend QEVERCLOUD_EXPORT QDebug & operator<<(
00067         QDebug & dbg, const ImageType imageType);
00068
00075     Thumbnail();
00076
00094     Thumbnail(
00095         QString host, QString shardId, QString authenticationToken,
00096         int size = 300, ImageType imageType = ImageType::PNG);
00097
00098     virtual ~Thumbnail();
00099
00104     Thumbnail & setHost(QString host);
00105
00111     Thumbnail & setShardId(QString shardId);
00112
00119     Thumbnail & setAuthenticationToken(QString authenticationToken);
00120
00126     Thumbnail & setSize(int size);
00127
00132     Thumbnail & setImageType(ImageType imageType);
00133
00148     QByteArray download(
00149         Guid guid, const bool isPublic = false, const bool isResourceGuid = false,
00150         const qint64 timeoutMsec = 30000);
00151
00153     AsyncResult * downloadAsync(
00154         Guid guid, const bool isPublic = false, const bool isResourceGuid = false,
00155         const qint64 timeoutMsec = 30000);
00156
00169     std::pair<QNetworkRequest, QByteArray> createPostRequest(
00170         qevercloud::Guid guid, bool isPublic = false, bool isResourceGuid = false);
00171
00172 private:
00173     ThumbnailPrivate * const d_ptr;
00174     Q_DECLARE_PRIVATE(Thumbnail)
00175 };
00176
00177 } // namespace qevercloud
00178
00179 #endif // QEVERCLOUD_THUMBNAIL_H
```

8.45 README.md File Reference

Index